Rebol в примерах

1. Создание GUI

Наберите код ниже в редакторе и сохраните с расширением .r:

```
view layout/size [] 400x300
```

Код программы создаст окно 400 пикселей поперек, и 300 пикселей вниз (пиксели - точки на компьютерном экране). Окно GUI может быть перемещено вокруг экрана, свернуто и закрыто ("X" в правом верхнем углу), точно так же как любая другая программа Windows. В других языках программирования, только создание окна подобно этому может занять несколько страниц кода.

Чтобы добавлять кнопку к вышеупомянутому GUI, наберите следующий код. Обратите внимание, что слово **"button"** добавилось между скобками:

```
view layout/size [button] 400x300
```

Теперь графический интерфейс пользователя имеет универсальную синюю кнопку, по которой вы можете щелкнуть мышью. Чтобы добавить некоторый текст к кнопке, напечатайте следующий код. Обратите внимание, что текст **"Нажми"** добавился после **button:**

```
view layout/size [button "Нажми"] 400x300
```

Чтобы после нажатия на кнопку произошло действие, наберите следующий код, и затем щелкните кнопкой. Обратите внимание, что текст **["Привет Мир"]** добавился после текста кнопки:

```
view layout/size [button "Нажми" [alert "Привет Мир!"]] 400x300
```

Теперь, когда вы щелкаете кнопкой в вашем окошке, компьютер отвечает, выдавая сообщение **"Привет Мир!"**. Давайте добавим больше интерактивности, наберите следующий код, и затем щелкните кнопкой еще раз. Обратите внимание на **"data: request-text"** добавление в начале строки. Этот код запрашивает некоторый текст от пользователя и назначает это слову **"data"**:

```
data: request-text view layout/size [
    button "Нажми" [alert data]] 400x300
```

Код выше разбит на две строки так, чтобы это вместилось на страницу учебника, но это может быть набрано в интерпретатор Ребола как одна строка. Одна строка - все, что требуется, чтобы создать программу, которая получает некоторые данные от пользователя, создает GUI, который ждет пользовательского взаимодействия, и обрабатывает входные данные (отображает это в небольшом диалоговом окне).

Затем, мы сохраним некоторые данные на жесткий диск вашего компьютера. Наберите следующий код, и щелкните **"yes"**, если интерпретатор Ребола просит у вас разрешения

сохранить на жесткий диск. Обратите внимание на запись **"%/c/data.txt data"** добавленный в конец строки. Код запишет данные, полученные от пользователя, в файл на диск C: , названный **"data.txt"**

```
data: request-text view layout/size [
    button "Нажми" [alert data]] 400x300 write %/c/data.txt data
```

Если вы наблюдаете за C: вашего компьютера после закрытия окна, вы увидите, что там теперь существует текстовый файл **(C:\data.txt)** содержащий текст, который вы напечатали в своей программе. (Если Вы работаете в другой операционной системе кроме Windows, вы должны изменить **"C:"** символ в вышеупомянутой строке, чтобы обратиться к корневому каталогу на вашем жестком диске).

С одной только строкой, вы имеете рабочую программу, которая делает что-то полезное. Получает, отображает, и сохраняет некоторые данные от пользователя, используя знакомые взаимодействия GUI. Можно модифицировать код, чтобы хранить телефонные номера, имена пользователя и пароли, или любую другую полезную информацию.

Преимущество Rebol в том, он может иметь дело со всеми типами общих данных. Вы можете легко отображать фотографии и другую графику в ваших GUI, web-страницах, и т.д. Вот код, который загружает изображение с сервера и отображает в GUI:

```
view layout [image load http://rebol.com/view/bay.jpg]
```

Слово **"image"** в скобках отображает изображение в GUI. Слово **"load"** загружает изображение, которое будет отображено.

Rebol позволяет многим встроенным эффектам быть примененным к изображениям. Обратите внимание на код **"effect [effect type]"** в следующих примерах:

```
view layout [image load http://rebol.com/view/bay.jpg effect [Grayscale]]
view layout [image load http://rebol.com/view/bay.jpg effect [Emboss]]
view layout [image load http://rebol.com/view/bay.jpg effect [Flip 1x1]]
```

Вы можете получить данные с сервера и сохранять на вашем диске C:. **"/binary"** модификатор используется всякий раз, когда имеет дело с двоичными данными:

```
write/binary %/c/bay.jpg read/binary http://rebol.com/view/bay.jpg
```

Теперь вы можете читать изображение непосредственно с вашего жесткого диска и отображать в GUI. Наберите код ниже. Обратите внимание на **"view layout"**, и **[image load %/c/bay.jpg]**, файл загружен с диска C:

```
view layout [image load %/c/bay.jpg]
```

2. Больше примеров

Ниже - некоторые более короткие примеры чтения, записи, и управления данными относительно жесткого диска и Internet, и взаимодействия с пользователем. Наберите их в интерпретатор Rebol.

Следующая строка отображает текущую дату и время:

```
print now
```

Слово **"print"** отображает текстовые данные непосредственно в интерпретаторе Rebol. Слово **"now"** обращается к текущей дате и времени.

Следующая строка исполняет некоторые математические вычисления, и отображает результат:

```
print (10 + 12) / 2
```

Следующий код просит, чтобы пользователь выбрал файл на жестком диске:

```
request-file
```

Код ниже позволяет пользователю выбирать цвет:

```
request-color
```

Следующий код спрашивает пользователя:

```
request "Вы согласны?"
```

Позволить пользователю выбирать дату:

```
request-date
```

Код ниже запрашивает имя и пароль пользователя:

```
request-pass
```

Следующий код открывает web-браузер вашего компьютера и отображает указанную web-страницу:

```
browse http://rebol.com
```

Следующий код запускает встроенный текстовый редактор, и открывает файл **c:\test.txt**

```
editor %/c/test.txt
```

Обратите внимание на символ процента **("%")** в примере выше. В Rebol, этот символ используется, чтобы представить все метки файла. Поскольку Rebol может использоваться на многих операционных системах, чтобы обратиться к дискам, путям, и т.д., Rebol использует универсальный формат: **%/drive/path/path/.../file.ext.** Например, **"%/c/Windows/notepad.exe"** обращается к **"C:\Windows\Notepad.exe"** в Windows. Rebol конвертирует тот синтаксис к соответствующему формату операционной системы, так, чтобы ваш код мог быть написан однажды и использоваться на каждой операционной

системе, без последующих изменений.

Следующая строка посылает электронную почту **user@website.com**, содержа текст **"Привет пользователь. Чем занимаетесь? "**. Пробуйте заменить имя пользователя и сайт с вашим собственным адресом электронной почты (Если вы загрузили и выполнили Rebol, вы будете должны выполнить конфигурацию, чтобы послать электронную почту.):

```
send user@website.com "Привет пользователь. Чем занимаетесь?"
```

Строка ниже посылает web-страницу пользователю:

```
send user@website.com read http://www.rebol.com
```

Код ниже отображает содержание почтового ящика пользователя:

```
print read pop://user:pass@website.com
```

Следующая строка загружает единственный файл на сервер, использующий ftp:

```
write/binary ftp://user:pass@website.com read/binary %file
```

Следующее загружает полный каталог файлов на сервер:

```
foreach file load %./ [if not dir? file [write/binary join
    ftp://user:pass@website.com/ file read/binary file]]
```

3. Быстрое сравнение

Чтобы узнать насколько проще идея Rebol, чем другие языки, рассмотрим короткий пример. Самый простой код, чтобы создать основное окно GUI, был представлен ранее:

```
view layout/size [] 400x300
```

Код для того же самого простого примера представлен ниже, на популярном языке программирования **"C++"**.:

```
#include <windows.h>

/*  Declare Windows procedure  */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/*  Make the class name into a global variable  */
char szClassName[ ] = "C_Example";

int WINAPI
WinMain (HINSTANCE hThisInstance,
         HINSTANCE hPrevInstance,
         LPSTR lpszArgument,
         int nFunsterStil)

{
```

```
    HWND hwnd;
    /* This is the handle for our window */
    MSG messages;
    /* Here messages to the application are saved */
    WNDCLASSEX wincl;
    /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    /* This function is called by windows */
    wincl.style = CS_DBLCLKS;
    /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    /* No menu */
    wincl.cbClsExtra = 0;
    /* No extra bytes after the window class */
    wincl.cbWndExtra = 0;
    /* structure or the window instance */
    /* Use Windows's default color as window background */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    /* Register window class. If it fails quit the program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,
            /* Extended possibilites for variation */
           szClassName,
            /* Classname */
           "C_Example",
            /* Title Text */
           WS_OVERLAPPEDWINDOW,
            /* default window */
           CW_USEDEFAULT,
            /* Windows decides the position */
           CW_USEDEFAULT,
            /* where the window ends up on the screen */
           400,
            /* The programs width */
           300,
            /* and height in pixels */
           HWND_DESKTOP,
            /* The window is a child-window to desktop */
           NULL,
            /* No menu */
           hThisInstance,
            /* Program Instance handler */
           NULL
            /* No Window Creation data */
           );

    /* Make the window visible on the screen */
    ShowWindow (hwnd, nFunsterStil);

    /* Run the message loop.
        It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0))
    {
        /* Translate virtual-key messages
            into character messages */
```

```
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }

    /* The program return-value is 0 -
        The value that PostQuitMessage() gave */
    return messages.wParam;
}
/*  This function is called by the Windows
        function DispatchMessage()  */

LRESULT CALLBACK
WindowProcedure (HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    switch (message)
    /* handle the messages */
    {
        case WM_DESTROY:
            PostQuitMessage (0);
                /* send a WM_QUIT to the message queue */
            break;
        default:
            /* for messages that we don't deal with */
            return DefWindowProc (hwnd, message,
                wParam, lParam);
    }

    return 0;
}
```

Вернемся лучвше к Rebol...

4. Понимание переменных и Функций

Все языки программирования используют две важных особенности: переменные и функции. Функции - команды, которые говорят, что компьютеру делать. Они подобны глаголам в разговорном языке. Вы использовали некоторые функции уже в более ранних примерах. Например, функция **"print"**, представляет некоторое действие.

Переменные - имена, присвоенные данным. Они подобны существительным в разговорном языке. Например, **"now"** можно рассматривать как переменную. Это - существительное, обращающееся к части данных.

Функции и переменные должны использоваться в синтаксисе, определенном в соответствии с языком программирования, который вы используете. Вы должны записать команды более определенным способом - используя синтаксис, который неизменно содержит переменные и функции. Например, чтобы вывести картинку с помощью своей программы, необходимо:

1. назначить имя файла картинки для переменной (существительное).
2. использовать функцию (глагол), чтобы отобразить изображение, упомянутое переменной.

3. назначить функциональное действие на кнопку или некоторый другой графический элемент в GUI.

Rebol высокоуровневый язык, поскольку близок к человеческому, но все еще требует определенного синтаксиса и структуры. Вы должны узнать, как использовать переменные и функции в грамматике, которую язык определяет. Это основа программирования.

5. Слова

Если вы хотите дать метку некоторым данным - так, чтобы это могло использоваться в вашей программе, вы должны назначить им слово. Есть много служебных сло, которые

представляют общие действия **("print", "alert", "request-date", и т. Д.)**. Вы видели множество встроенных слов уже в более ранних примерах. Чтобы создавать ваши собственные функции, вы берете предварительно созданные служебные слова и переменные, группируете их вместе в определенном порядке, и назначаете слово на эту совокупность кода. Тогда вы можете обратиться к этой группе действий или связанных данных, используя слово, которое вы назначили для своей функции.

Изучение служебных слов, которые уже определены в языке программирования - необходимость для изучения языка. Слово **"write"**, например, записывает данные на запоминающее устройство (жесткий диск, CD диск, сервер, и т.д.). Это - функция, которая выполняет действие, и ожидает имя для файла, который будет записан на жесткий диск:

```
write %/c/text.txt "Здесь набранный вами текст"
```

Если вы набрали вышеупомянутую строку в неправильном порядке, это не будет работать:

```
write "Здесь набранный вами текст" %/c/text.txt ; WRONG
```

Если вы напечатали это так, Rebol не поймет синтаксис, и вы получите ошибку.

6. GUI, Слова и Грамматика – еще глубже

Вы видели в более ранних примерах, что слова **"view layout"**, сопровождаемые двумя скобками **(" [] ")** могут использоваться, чтобы отобразить GUI в Rebol. Вы можете поместить элементы, которые вы хотите видеть в GUI, в скобки. Rebol содержит слова, которые отображают все обычные графические элементы, используемые в GUI. Пробуйте набрать код ниже:

```
view layout [button]

view layout [field]

view layout [text "Rebol действительно очень прост для программирования"]

view layout [text-list]

view layout [
    button
    field
    text "Rebol действительно очень прост для программирования"
    text-list
    check
]
```

Обратите внимание, что слова могут быть отделены **"незаполненным пространством"** в скобках. Дополнительные пространства, переводы каретки, и другие пустые символы игнорируются интерпретатором. Табуляторы традиционно используются, чтобы выровнять строки в пределах скобок, но они не требуются.

Более наглядные характеристики о графических элементах могут быть включены непосредственно после каждого из их соответствующих слов. Такие модификаторы называют **"facets"**, и они позволяют вам корректировать все характеристики каждого типа графического элемента (размер, цвет, отображение текста), и т.д. Пробуйте набрать код ниже - это то же самое как вышеупомянутый код, с некоторыми дополнительными изменениями:

```
view layout [
    button red "Нажми"
```

```
    field "Введите здесь текст"
    text "Rebol действительно очень прост для программирования" purple
    text-list 400x300 "Строка 1" "Строка 2" "другая строка"
    check yellow
]
```

6.1 Действия

ВАЖНО: Если вы хотите, чтобы графический элемент исполнил действие, помещайте текст в скобки после указания действия. Когда элементом GUI щелкают мышью или иначе, действие будет выполнено. Наберите следующий код, чтобы увидеть, как это работает:

```
view layout [button [alert "Вы щелкнули кнопкой."] ]
view layout [button red "Нажми" [alert "Вы щелкнули кнопкой."]]
view layout [
    text "Некоторые действия.  Попробуйте использовать виджет:"
    button red "Нажми" [alert "Вы щелкнули кнопкой."]
    field 400 "Наберите здесь текст, затем нажмите Enter."
        [alert value]
    text-list 400x300 "Выбрать строку" "Эта строка" "Новая строка"
        [alert value]
    check yellow [alert "Вы щелкнули желтым переключателем."]
    button "Выйти" [quit]
]
```

Обратите внимание, как слово **"value"** обращается к выбранному элементу в текстовом списке, и к тексту, содержавшемуся в текстовом поле.

Дополнительные действия (то есть, вызванные правым щелчком мыши) могут быть включены для любого графического элемента. Только включите их во второй блок, окруженный скобками:

```
view layout [button [alert "левый клик"] [alert "правый клик"]]
```

Вот некоторые другие элементы GUI, используемые в языке Rebol:

```
view layout [
    backcolor white
    h1 "Примеры GUI:"
    box red 500x2
    bar: progress
    slider 200x16 [bar/data: value show bar]
    area "Введите текст"
    drop-down
    across
    toggle "Нажми" "Здесь" [print value]
    rotary "Нажми" "Еще" "И еще" [print value]
    choice "Выбрать" "Пункт 1" "Пункт 2" "Пункт 3" [print value]
    radio radio radio
    led
    arrow
    return
    text "Normal"
    text "Bold" bold
    text "Italic" italic
    text "Underline" underline
    text "Bold italic underline" bold italic underline
    text "Стиль текста Serif" font-name font-serif
    text "Раздельный текст" font [space: 5x0]
    return
    h1 "Заголовок 1"
```

```
    h2 "Заголовок 2"
    h3 "Заголовок 3"
    h4 "Заголовок 4"
    tt "Текст пишущей машинки"
    code "Текст кода"
    below
    text "Большой" font-size 32
    title "Центрированный заголовок" 200
    across
    vtext "Normal"
    vtext "Bold" bold
    vtext "Italic" italic
    vtext "Underline" underline
    vtext "Bold italic underline" bold italic underline
    vtext "Стиль текста Serif" font-name font-serif
    vtext "Раздельный текст" font [space: 5x0]
    return
    vh1 "Видео заголовок 1"
    vh2 "Видео заголовок 2"
    vh3 "Видео заголовок 3"
    vh4 "Видео заголовок 3"
    label "Метка"
    below
    vtext "Большой" font-size 32
    banner "Баннер" 200
]
```

Примеры выше демонстрируют, как Rebol создает GUI, которые могут использоваться, чтобы ввести и отображать данные. Это основная часть строительства типичной современной компьютерной программы! Для подробной информации о дизайне GUI, см. http://rebol.com/docs/easy-vid.html и http://rebol.com/docs/view-guide.html.

7. Создание собственных переменных слов

Точно так же как разговорные языки, языки программирования легки и выразительны. Нет только одного способа создать программу. Вы должны выбрать и использовать ваши собственные слова, и вы должны организовать их. Точно так же как на разговорном языке, вы должны думать на этом языке. В Rebol вы имеете дополнительную способность создать ваши собственные слова языка, чтобы выразить действия и маркировать данные.

Слова создаются и назначаются для переменных и функций в Rebol при помощи двоеточия **(":")** . Вы можете использовать любое слово, которое вы хотите определить к любым данным или действиям. Например, если вы хотите использовать слово **"picture"**, чтобы обратиться к загрузочному модулю в Internet, вы можете сделать следующее:

```
picture: load http://rebol.com/view/bay.jpg
```

Вышеупомянутая строка создает метку, которая может использоваться в вашей программе, где вы хотите читать и использовать файл, расположенный в **http://rebol.com/view/bay.jpg**. Встроенное Rebol слово **"load"** найдет изображение по указанному пути.

Теперь вы можете использовать слово **"picture"**, чтобы обратиться к вышеупомянутому изображению. Отобразите это в GUI, используя следующий код:

```
view layout [image picture]
```

Слова **"view"**, **"layout"**, и **"image"** встраивают в Rebol, и слово **"image"** теперь также допустимо, потому что это было определено для интерпретатора.

Поскольку слово **"picture"** является переменным, вы можете также переопределить и

изменить данные, упомянутые здесь:

```
picture: load http://rebol.com/view/demos/palms.jpg
```

Теперь, когда вы используете слово **"picture"** в вашей программе, оно обращается к разным файлам в разных местах Internet. Запись того же самого кода GUI теперь отображает разные изображения:

```
view layout [image picture]
```

Вы можете также сделать так чтобы слово **"picture"**, обращалось к файлам на вашем жестком диске, или где-нибудь еще, где вы хотели бы:

```
picture: load %/c/bay.jpg
```

Вот еще некоторые примеры создания и использования переменных слов. Напечатайте их в Rebol интерпретатор, чтобы видеть, как они работают, и понимать, как назначенные слова могут обращаться к любым данным. ОБРАТИТЕ ВНИМАНИЕ: после точки с запятой, код игнорируется интерпретатором. Это используется, чтобы включить комментарии в код:

```
acolor: "синий"

alert acolor ; в диалоговом окне текст синий

print acolor ; выводит слово "синий" в Rebol интерпретаторе

anumber: 12

print anumber ; выводит число 12 в Rebol интерпретаторе

computation: (10 + 12) / 2

print computation ; выводит ответ

filename: request-file

print filename ; выводит имя пользователя

chosen-color: request-color

print chosen-color

answer: request "Здесь текст?"

print answer

pick-a-date: request-date

print pick-a-date

userpass: request-pass

print userpass

webpage: http://rebol.com

browse webpage

file: %./test.txt

editor file
```

```
; открывает встроенный в Rebol текстовый редактор и в нем же файл
; указанный выше

email-address: user@webpage.com

message:  "Привет друг. Как дела?"

send email-address message
```

8. Блоки

В Rebol, вы можете использовать слова, чтобы представить множественные части данных вместе, совокупности других переменных слова, и других элементов программирования. Заключите данные в скобках. Наберите код ниже, чтобы видеть, как это работает:

```
somecolors: ["красный" "желтый" "синий" "черный"]
; "somecolors" теперь определенное слово, используется, чтобы представить весь
; блок данных, включенный в скобках.
print somecolors
```

В Rebol, совокупности подобно этому называют **"блоками"**, и они очень важны. Фактически, они - первичный организационный модуль в Rebol и главной структуре, в которой сохранены данные. В Rebol, для любого типа данных может быть назначено слово, и блоки могут содержать любую комбинацию слов и необработанных данных. Вы уже использовали блоки, чтобы отображать элементы GUI и исполнять действия в них. Любая группа слов, окруженных скобками формирует блок, и вы можете назначить слово к этому блоку. Слова, относящиеся к сложным блокам данных могут аналогично группироваться в другие блоки. Это делает работу с очень сложными структурами данных, очень простой в Rebol.

Наберите код ниже, чтобы видеть, как полное GUI может быть сформировано и представлено, используя одну метку слова:

```
gui-layout: [button field text-list]
```

"gui-layout" теперь обращается к полному блоку. Вы можете отобразить это используя **"view layout"**:

```
view layout gui-layout
```

Блоки данных могут содержать более чем несколько строк, и могут быть отделены дополнительным незаполненным пространством. Только заключите элементы в скобки:

```
gui-layout2: [
    button red "Нажми"
    field "Введите текст."
    text " Rebol действительно очень прост для программирования." purple
    text-list 400x300 "строка 1" "строка 2" "другая строка"
    check yellow
]
```

Теперь отобразите это в GUI:

```
view layout gui-layout2
```

Блоки, содержащие в себе другие блоки, содержат данных намного больше, чем в одном слове!

ОБРАТИТЕ ВНИМАНИЕ: Это - стандартная практика, чтобы выровнять составные блоки с последовательными табуляторами. Начальные и конечные скобки типично помещаются на том же самом стандартном уровне. Это обычно в большинстве языков программирования, потому что это делает сложный код более легким для чтения. Например, составной блок ниже:

```
[blue red green [1 2 4 [jan feb march [monday tuesday wednesday]]]]
```

может быть написан немного более ясно как:

```
[blue red green
    [1 2 4
        [jan feb march
            [monday tuesday wednesday]
        ]
    ]
]
```

Сдвиг не требуется, но очень полезно когда имеете дело с большим количеством запутанных структур.

Вот - простой пример таблицы данных, содержит информацию в пределах блока:

```
schedule: [
    ["Андрей" "Понедельник" "3:00 pm"]
    ["Саша" "Вторник" "11:00 am"]
    ["Маша" "Среда" "4:45 pm"]
]
```

Вы можете отобразить, вышеупомянутый блок в GUI используя встроенное Rebol слово **"list"**. Обратите внимание на код ниже - вставлен полный блок списка показанный выше.

```
    vh2 "Дежурные этой недели:"
    list 600x400 [
        across text 150 text 150 text 100
    ] [ data schedule
]
```

Графическое приложения **"базы данных"**, в несколько строк программы... Нет проще!

Встроенные слова помогают вам управлять данными, сохраненными в блоках. Наберите следующий код, чтобы видеть, как работает слово **"sort"**:

```
somecolors: ["красный" "желтый" "синий" "черный"]

sortedcolors: sort somecolors
```

"Sort" сформирована в служебном слове, которое в алфавитном порядке сортирует элементы данного блока. Строка выше создает недавно определенное слово **"sortedcolors"**, и назначает это на сортированный блок слов, содержавшихся в **"somecolors"**.

```
print sortedcolors
; Этот код отображает сортированный блок текста.

print first sortedcolors
; "first" - другое встроенное слово.
; выбирает первый элемент в данном блоке.
find somecolors "красный"
; "find" является сформированным в слове, которое ищет данные в пределах блока.
```

Вы можете легко сохранить блоки данных на ваш жесткий диск, читать их, и выполнять другие операции.

```
write %/c/colors.txt somecolors
; пишет полный блок текста, представленного "somecolors"
; в текстовый файл по имени colors.txt на C: диск.
```

Интересный прием, демонстрирует, как Rebol может легко смешать datatypes в пределах блока:

```
an-image: load http://rebol.com/view/bay.jpg
; загружает изображение из Internet и назначает
; ему слово "an-image".

append sortedcolors an-image
; "append", добавляет загруженное изображение до конца блока данных
; содержа простые текстовые слова, определенные выше.
; Новый блок содержит и текст и  двоичные данные изображения,
; весь блок назначен на единственное слово!
```

Теперь вы можете выбрать элементы из нового блока:

```
print first sortedcolors
; печатает первый элемент в блоке данных - текст "черный".

view layout [image fifth sortedcolors]
; отображает пятый элемент в блоке данных-
; изображение, загруженное выше - в GUI .
```

При работе с последовательными данными в блоках (тип данных называют **"series"** в Rebol):

```
view layout [image sortedcolors/5]
; "sortedcolors/5" - другой способ обратиться к пятому элементу
; в блоке данных.
```

Вышеупомянутое приложение работает другим способом:

```
length-of-block: length? sortedcolors
; встроенный слово "length? " возвращает число
; в блоке (5 в этом регистре).
view layout compose [image sortedcolors/(length-of-block)]
```

Слово **"compose"**, позволяет вставлять переменные в круглые скобки.

Следующие примеры демонстрируют, что дополнительные слова могут использовать последовательный ряд данных в пределах блока:

```
insert sortedcolors "сиреневый"
; добавляет слово "сиреневый" к блоку данных sortedcolors.

remove sortedcolors
; удаляет первый элемент из блока.

head sortedcolors
; устанавливает маркер позиции в начале блока данных.

next sortedcolors
; устанавливает маркер позиции в следующем элементе в блоке данных.

last sortedcolors
; устанавливает маркер позиции в последнем элементе в блоке данных.

back sortedcolors
; устанавливает маркер позиции в предыдущем элементе в блоке данных.

tail sortedcolors
; устанавливает маркер позиции после последнего элемента в блоке данных
```

Вы можете смешать вместе все типы данных в пределах блока, обращается к частям блоков по имени, обращается и изменять данные с помощью встроенных функций. Блоки и переменные слова, назначенные на блоки помогают вам хранить, управлять, и обращаться ко всем данным, с которыми вы будете иметь дело в ваших программах.

9. Служебные слова

Создание новых функций сопоставимо созданию ваших собственных слов глагола на разговорном языке. Только будьте осторожны, чтобы не использовать слова, которые уже определены в языке Rebol, или в вашей текущей программе. Это изменило бы значение существующего слова. Например, вы можете случайно изменить, значение слова **"write"**, чтобы обратиться к изображению на жестком диске, печатая следующее:

```
write: read %/c/bay.jpg
; *** не используйте это - это изменит значение слова
; "write" в Rebol интерпретаторе (только для текущего сеанса).
; Это пример того, как не надо делать. ***
```

Вы можете защитить все встроенные Rebol слова, используя **"protect-system"**. Это выдаст предупреждение с ошибкой и отвергнет любые попытки переопределить служебные слова Rebol. Вы все еще должны быть осторожны, чтобы случайно не переопределить слова, которые вы создали. Если это произошло, то надо сделать рестарт интерпретатора Rebol.

Вот действительно важная концепция: В Rebol, вы можете использовать отдельные слова, чтобы представить полные блоки действий (то есть, совокупности сгруппированных служебных слов). Фактически, блоки подобно этой форме составляют программы на языке Rebol. Вот пример нескольких служебных слов, сгруппированных в блок (заключены в скобки), и новое служебное слово:

```
some-actions: [
    alert "Первое действие."
    print "Второе действие."
    write %/c/anotheraction.txt "Третье действие."
```

```
]
```

Вышеупомянутый код создал своего рода суперглагол, который обращается к нескольким действиям. Вы можете исполнить действия, содержавшиеся в любом блоке, используя служебное слово **"do"** . Чтобы получить все действия в вышеупомянутом блоке кода, наберите:

```
do some-actions
```

Вы можете также включить слово **"does"** в определение слова - будет делать действия, содержавшиеся в блоке, исполняя автоматически каждый раз, когда новое слово используется в Rebol:

```
more-actions: does [
    alert "4"
    alert "5"
    alert "6"
]
```

Фактически, включением **"does"**, вы только что создавали новую функцию (или **"подпрограмма"**), которая может использоваться подобно любой другой встроенной в Rebol! Вы можете теперь общаться с интерпретатором, используя это слово. После того, как вы ввели код выше, пробуйте набрать **"more-actions"** в Rebol интерпретатор:

```
more-actions
```

Вот пример действия, чтобы очистить командную строку в Rebol интерпретаторе.

```
cls: does [prin "^(1B)[J"]
```

Родной способ очистки командной строки интерпретатора: набрать **"prin" ^ (1B) [J"**. Это неудобно делать каждый раз. Вместо этого, мы можем поручить слову **"cls"** исполнять действие - точно так же как в **Basic**. Теперь только нанаберите:

```
cls
```

и экран чист - это намного проще.

Вот небольшая программа, которая создает новое слово, **"send-email"**. Это создаст простой текстовый интерфейс для пользователей, чтобы послать электронную почту:

```
send-email: does [
    email-address: to-email request-text/title/default
        "Введите адрес email:" "user@webpage.com"
        ; вышеупомянутая строка создает новое переменное слово "email-address"
        ; "email-address" назначено значение
        ; текстовый ввод, используя встроенный слово "request-text"
        ; "title" и "default"  настраивают
        ; отображение информации в текстовом поле.
    message: request-text
        ; строка выше создает новое переменное слово "message"
        ; и назначает на некоторый требуемый текст
    send email-address message
        ; строка выше посылает сообщение
```

```
        ; на адрес электронной почты, данный ранее
    alert "Ваше сообщение послано."
]

send-email ; сделайте подпрограмму выше
send-email ; сделайте это снова, чтобы послать другое сообщение кому - то еще
```

Вышеупомянутый процесс ОЧЕНЬ важен. Это основа того, как вы будете исполнять более сложные действия в Rebol. Блоки действий и блоки данных формируют основание программы. В Rebol вы только группируете биты данных вместе в блоки, назначаете имя и обращаются к нему. Вы также группируете функции вместе в блоки, чтобы выполнить действия, используя данные, назначаете имя, чтобы через него обратиться к тем действиям. Вы можете даже комбинировать законченные группы данных и функций в блоки, которым могут быть назначены уникальные идентификаторы слова, которые исполняют законченные программные задачи для вас. Это позволяет вам создавать ваш собственный уникальный язык в любой программе, которую вы пишите, используя слова, которые вы определяете. Слова не встраиваются в Rebol, но им могут быть назначены соответствующее действие и значения данных, чтобы сделать предложение полностью функциональной частью кода, который понимает компьютер. Создание своих слов называют, это ваш диалект в Rebol, и это - одна из особенностей Rebol, отличающего его от других языков. Другие языки сосредотачиваются на различных способах группировать и управлять функциями и переменными. Как правило, другие языки используют более сложные способы. Например, блоки, которые содержат и полностью скрытые функции, и переменные, называют **"объектами"**.

10. Несколько способов создания функций в Rebol

Есть несколько встроенных слов в языке Rebol , которые позволяют вам создавать более сложные служебные слова. Чтобы создавать простые функции, вы можете использовать команду **"does"** , как описано выше. Но некоторые функции более сложны, чем это. Они работают с разными данными. Например, следующая простая функция отображает квадратный корень 4.:

```
sqr-four:  does [print square-root 4]
```

Слово **"sqr-four"** теперь назначено на действие **"вычислить квадратный корень 4"** (слово **"print square-root 4"** встраивают в Rebol). После ввода вышеупомянутой строки в Rebol интерпретаторе, напечатайте:

```
sqr-four
```

Получим 2.

Предположим, что мы хотим сделать кое-что, используя другие числа кроме 4... Например, нужно создать функцию, которая добавляет 4 к некоторому другому числу, и затем вычисляет квадратный корень той суммы? Имя переменной надо тогда **"передать-passed"** к функции. Встроенное слово **"func"** используется, чтобы создать функции, к которым изменяемые переменные можно передать. Синтаксис для слова **"func"** ожидает, что это сопровождается двумя блоками кода. Первый блок содержит названия переменных, которые будут переданы. Второй блок содержит действия, которые будут предприняты. Вот как это выглядит:

```
func [имена переменных, которые будут переданы] [
    действия, которые будут предприняты к тем переменными
]
```

В следующей строке, создана функция, в которой вызывают **"anumber"**. Функции присвоено имя **"sqr-var"**:

```
sqr-var: func [anumber] [print square-root (4 + anumber)]
```

Теперь можно использовать слово **"sqr-var"**, при этом Rebol интерпретатор знает, что сделать с назначенными данными. Пробуйте следующий код:

```
sqr-var 12 ; печатает "4", квадратный корень 12+4 (16)
sqr-var 96 ; печатает "10", квадратный корень 96+4 (100)
```

Вычисление квадратного корня 4+somenumber не очень интересно, но это помогло иллюстрировать один из самых важных методов, используемых во всех современных языках программирования. Процесс принятия параметров переменных для функций - фундаментальная часть всего современного программирования. Это возможно единственный, самый общий элемент в современных языках, и понимание этого, является необходимым.

Следующая строка создает простую функцию для вывода изображения:

```
display: func [filename] [view layout [image load filename]]
```

Это принимает имя файла изображения как переданный параметр **(%somefile.jpg, %somefile.gif, %somefile.png, или %somefile.bmp)**, и затем создает графический интерфейс пользователя, чтобы отобразить изображение. Этот набор действий назначен на слово **"display"**:

```
image1: to-file request-file/title trim {
    Выберите изображение на  жестком диске:} ""
; получает имя файла изображения от пользователя

display image1
; отображает вышеупомянутое изображение, используя новое служебное слово

display http://rebol.com/view/bay.jpg
; отображает изображение в вышеупомянутом url

display %/c/bay.jpg
; отображает изображение, которое было сохранено на
; жестком диске в примере ранее
```

Вот пример, который спрашивает у пользователя 2 URL сайта, и затем открывает их в отдельных окнах браузера:

```
openwebsite: func [nameofwebsite] [browse nameofwebsite]
; Строка выше создает новую функцию, которая передает url
; к встроенному в Rebol слову "browse", чтобы открыть переданный
; сайт в заданном по умолчанию браузере пользователя. Также
; назначает новое служебное слово "openwebsite" на тот набор
; действий.


website1: request-text/title "Введите URL сайта:"
; Строка выше назначает новое переменное слово "website1" для текста
; возвращенного встроенной функцией "request-text".

website2: request-text/title "Введите URL другого сайта:"
; Получает еще некоторый текст и назначает новое переменное слово "website2"
; к этому.

openwebsite website1
```

```
; Строка выше использует новое "openwebsite" определенное служебное слово
; , и пути к "website1".

openwebsite website2
; Использует функцию openwebsite снова, на сей раз передавая
; переменную website2.
```

В этом примере, слово **"openwebsite"** назначено на новое функциональное определение. **"Website1"** и **"website2"** - метки для переменных. Ниже пример, в котором назначаем единственное слово на весь полный процесс:

```
display-website: does [
    openwebsite: func [nameofwebsite] [browse nameofwebsite]
    website: request-text/title "Введите URL сайта:"
    openwebsite website
]
```

Теперь вы можете использовать слово **"display-website"** в ваших программах, чтобы выполнить весь блок кода.

```
display-website
```

Привыкание к вышеупомянутому синтаксису и мышлению является необходимым. Помните, иметь дело со всеми типами данных - главная работа, которую вы делаете как программист. Принятие переменных данных к функциям, которые вы создаете - главный способ, которым вы выполняете действия с данными во всей вашей программе. В Rebol легко иметь дело с данными, используя встроенную поддержку общих типов данных. Понимание того, как вводить, управлять, и выводить эти данные - ваша главная цель. Использование функций и переменных как описано выше - фундаментальная часть обучения.

11. Условные операторы

Давая компьютеру, разнообразие действий, основанных на разнообразии ожидаемых условий, является фундаментальной методикой программирования, используемой во всех языках.

Математические операторы типа **= <> <>** (**равный**, **менее - чем**, **"больше чем"**, **не - равный**) часто используются, чтобы выполнить операции сравнения. Наберите следующий код:

```
if now/time > 12:00 [alert "После полудня."]
; now/time - измененная, или "усовершенствованная" встроенная
; функция "now", которая возвращает только текущее время.
```

Вот более сложный пример:

```
daily-calories: to-integer request-text/title {Сколько калорий вы употребили
          сегодня?}
; получает некоторую информацию от пользователя и назначает переменное слово
; "daily-calories". Встроенная функция "to-integer"
; проверяет, что Rebol интерпретирует ту информацию как число.
; " {} " символы, окружающие текст заголовка работают
; так же как кавычки, но позволяет охватить
; несколько строк. Очень важный.

if daily-calories > 2500 [alert "Сегодня не ешьте."]
```

Встроенное в Rebol слово **"either"** выбирает выполнить или нет между двумя блоками функций, основанных на условной оценке. Его синтаксис:

```
either {condition} [
    выполнить блок, если условие true
][
    выполнить блок, если условие false
]
```

Вот пример:

```
either now/time > 8:00am [alert "Пришло время вставать!"] [
    alert "Вы может спать дальше."]
```

Вот разновидность вышеупомянутого примера:

```
wake-up: to-time request-text/title "Во сколько вы хотите проснуться?"

either now/time > wake-up [alert "Пришло время вставать!"] [
    alert {Вы может спать дальше.}]
```

Встроенное Rebol слово **"switch"** выбирает выполнить или нет между многочисленными функциями, основанными на условных оценках. Его синтаксис:

```
switch/default {main value} [
{value 1} [выполнить блок если значение 1 = main value]
{value 2} [выполнить блок если значение 2 = main value]
{value 3} [выполнить блок если значение 3 = main value]
; и.т.д...
] [выполнить по умолчанию блок кода, если ни одно из значений не соответствует]
```

Вы можете сравнить много значений с основным значением, и выполнить блок кода для каждого значения соответственно. Вот - пример:

```
favorite-day:  request-text/title "Каков ваш любимый день недели?"

switch/default favorite-day [
    "Понедельник" [alert "Понедельник день тяжелый..."]
    "Вторник" [alert "Вторник тоже не очень..."]
    "Среда" [alert "В среду разгар работы!"]
    "Четверг" [alert "Выходной не за горами..."]
    "Пятница" [alert "Можно начинать расслабляться!"]
    "Суббота" [alert "Кураж во весь рост!"]
    "Воскресенье" [alert "Болит голова!"]
] [alert "Вы не выбрали день!"]
```

Оператор **"Switch"** используется часто, потому что программы требуют многочисленные оценки выбора результата.

Rebol включает богатый набор слов и функциональных структур, которые помогают вам оценивать условия во всех типах ситуаций, и со всеми типами данных. Понимание, как использовать их – одно из главных направлений в изучении языка.

12. Циклы

Программы часто проверяют условия и выполняют функции неоднократно. Фактически, в большинстве больших приложений компьютер часто исполняет циклы для повторной работы, какой либо функции. Например, в программе напоминания, приложение должно непрерывно проверять время и дату, чтобы видеть, нужно ли пользователю напомнить о чем-то в настоящее время. В других типах программ, компьютер, возможно, нуждается в просмотре условий через совокупности данных, или неоднократно запрашивает и отвечает пользовательскому запросу. Чтобы обрабатывать такие ситуации, **"loop"** структуры систематически повторяют действия в программе.

Встроенное слово **"forever"** создает простой цикл. Его синтаксис:

```
forever [блок действий для повторения]
```

Следующий код создает простой таймер, который предупреждает пользователя, когда проходит одна минута. Использует цикл, чтобы непрерывно проверять время.

```
alarm-time: now/time + 60
; назначим переменную на время 60 секунд с этого времени
forever [if now/time = alarm-time [alert "прошло 60 секунд" break]]
```

Заметьте слово **"break"** в примере выше. Этот оператор используется для остановки цикла, как только отобразится сообщение, иначе цикл будет работать бесконечно.

Вот - более диалоговая версия. Сердце программы - цикл:

```
event-name: request-text/title "О чем вам напомнит?"
; запрашивает напоминание от пользователя
seconds: to-integer request-text/title trim {
    Через какое время напомнить?}
; запрос на количество времени ожидания
alert join "сейчас " [
    now/time ", будьте готовы в " seconds " seconds."
]
; отобразить сообщение
alarm-time: now/time + seconds
; установите сигнальное время
forever [
    if now/time = alarm-time [
        alert join " сейчас " [
            alarm-time ", и " seconds
            " прошли секунды.  Это время для: " event-name
            ]
    break
    ]
]
; цикл непрерывно сравнивает сигнальное время набора с
; текущим временем, затем отображает предупреждение, когда оно соответствует.
```

Заметьте **"join"** слово, используемое в нескольких строках выше. Формат синтаксиса:

```
join { данные} [блок  данных]
```

Используя этот формат, соединяем вместе переменные, текст, блоки и другие данные, чтобы они могли быть вместе использованы, отображены, и задействованы, чтобы формировать единую часть данных. Создается особый блок данных, собирающий все

индивидуальные элементы в одну часть данных. Синтаксис:

```
rejoin [item1 item2 item3 ...]
```

Теперь, назад к циклам. Есть простой цикл, который отображает и обновляет текущее время в GUI:

```
view layout [
    timer: field
    button "ПУСК" [
        forever [
            set-face timer now/time
            wait 1
        ]
    ]
]
```

Вышеупомянутое GUI содержит два элемента: текстовое поле, которому назначена переменная метка **"timer"**, и кнопка со словом **" ПУСК "**. Блок действия для кнопки содержит цикл, выполняет 2 действия в цикле (пока пользователь не закроет GUI): встроенное слово **"set-face"** показывает текущее время в поле **"timer"**, программа ждет 1 секунду, и затем повтор.

Подобно большинству языков, Rebol включает разнообразие функций и с программным управлением структур, которые позволяют вам образовывать циклы через блоки данных и исполнять операции, использующие последовательно измененные значения. Обычное название оператора цикла на многих языках - **"for"**. Это позволяет вам определять стартовое значение, значение окончания, возрастающее значение, и имя переменной, чтобы держать текущее значение, так, чтобы вы могли образовать циклы через последовательное изменение значений управляемым способом. Вот основной синтаксис для оператора цикла **"for"**:

for {переменное слово, содержащее значение} {запускающий значение} {заканчивающий значение} {возрастающее значение} [блок кода для исполнения, который может использовать текущее переменное значение]

Здесь простые примеры. Наберите их, чтобы увидеть, как они работают:

```
for counter 1 10 1 [print counter]
; starts on 1 and counts to 10 by increments of 1

for counter 10 1 -1 [print counter]
; starts on 10 and counts backwards to 1 by increments of -1

for counter 10 100 10 [print counter]
; starts on 10 and counts to 100 by increments of 10

for counter 1 5 .5 [print counter]
; starts on 1 and counts to 5 by increments of .5

for timer 8:00 9:00 0:05 [print timer]
; starts at 8:00am and counts to 9:00am by increments of 5 seconds

for dimes $0.00 $1.00 $0.10 [print dimes]
; starts at 0 cents and counts to 1 dollar by increments of a dime

for date 1-dec-2005 25-jan-2006 8 [print date]
; starts at December 12, 2005 and counts to January 25, 2006
; and by increments of 8 days

for alphabet #"a" #"z" 1 [print alphabet]
```

```
; starts at the character a and counts to z by increments of 1 letter
```

Обратите внимание, что Rebol может легко перебрать различные типы данных, может автоматически увеличивать дату, время, и т.д. На других языках, это сделать сложнее.

Обратите внимание на использование слова **"prin"** в последнем примере. Он работает как **"print"**, автоматически не вставляет символ возврата каретки (то есть, он печатает каждый символ вывода последовательно рядом с предыдущим).

Вот примера цикла **"for"** , который отображает первые 5 имен файла в текущей папке на вашем жестком диске:

```
files: read %.
; получает список текущего каталога,
; и назначает тот блок имен файла к переменным "files"

for count 1 5 1 compose [print files/(count)]
; печать начал с 1-ым элементом в блоке,
; и подсчитывает к 5-ому элементу.
```

В примере выше, **"files/1"** - синтаксис, представляющий первый элемент в списке файла, **"files/2"** представляет второе, и так далее. Обратите внимание на слово **"compose"** , используемое для цикла.

"Foreach" - другая полезная структура выполнения цикла. Он позволяет вам легко перебирать совокупность данных. Его синтаксис, выглядит следующим образом:

foreach {имя переменной, ссылающееся на каждый последовательный элемент в данном блоке} [данный блок] [блок функций, которые будут выполнены на каждый элемент в данном блоке, используя имя переменной, чтобы сослаться на каждый элемент последовательно]

Пример ниже печатает имя каждого файла в текущем каталоге на вашем жестком диске:

```
folder: read %.
; получает список текущего каталога,
; и назначает тот блок имен файла к переменной "folder"


foreach file folder [print file]
; перебирает каждое имя файла, содержавшееся в блоке "folder"
; и печать каждый последовательно.
```

Следующая строка читает и печатает каждое последовательное сообщение в почтовом блоке пользователя:

```
foreach mail read pop://user:pass@website.com [print mail]
```

"While" другая полезная структура выполнения цикла, используемая в большинстве языков программирования. Он неоднократно выполняет условную оценку, и затем выполняет блок программы, пока условие истинно. Синтаксис:

```
while [условие] [
    блок функций, которые будут выполнены, пока условие истинно
]
```

Вот простой пример:

```
x: 1  ; создайте значение
while [x <= 5] [
    alert to-string x
    x: x + 1
]
```

Читаем код **"x первоначально равен 1. Пока x меньше чем или равен 5, отобразить значение x, и прибавить 1 к значению x"** (то есть, программа отображает количество от 1 до 5).

ОТМЕТЬТЕ: В Rebol, код **"x: x + 1"** прибавляет 1 к текущему значению x. Одно из обычно используемых выражений во всем программировании, найденном во всех видах ситуаций выполнения цикла. Обратите внимание также на слово **"to-string"**. Он конвертирует значение числа **"x"** к тексту **("string")**. Это требуется, потому что слово **"alert"** в Rebol отображает только типы данных строки.

Вот некоторые дополнительные примеры цикла **"while"**:

```
while [not request "Закончить программу сейчас?"] [
    alert "Нажмите ДА, чтобы закончить программу."
]
```

В вышеупомянутом примере **"not"** полностью изменяет значение данных, полученных от пользователя (то есть, yes становится no).

```
alert "Пожалуйста выберите дату" while [request-date <> now/date] [
    alert join "Пожалуйста выберите дату" [now/date]]

while [request-pass <> ["секретный" "пароль"]] [
    alert "Имя пользователя и пароль являются 'секретными'"]
```

Пример ниже использует несколько циклов, чтобы напомнить пользователю, когда покормить кошку, каждые 6 часов между 8am и 8pm. Используется цикл, чтобы увеличить время, для напоминания, цикл с условием продолжения, чтобы непрерывно сравнивать прошедшее временя с текущим временем, и цикл, чтобы делать то же самое каждый день, непрерывно.

```
    for timer 8:00am 8:00pm 6:00 [
        while [now/time <= timer] [wait .5]
        ; ничего не делайте, пока не время
        ; кормить кошку. "Ждут.5" действий дают
        ; пользователю шанс закрыть программу.
        [alert join "Сейчас" now/time ".  Пришло время кормить."]
        ; если цикл с условием продолжения вышел, пришло время
        ; кормить кошку. После приведения в готовность, возвратитесь и ждите
        ; следующего кормления.
    ]
]
```

13. Работа с более сложными примерами

Большинство программ более длинно, чем примеры, которые показаны в этом учебнике. Ниже пример более сложной программы. Он позволяет пользователю использовать Web-камеру, для отображения видео у себя. Он отображает видео изображения в GUI, у которого есть несколько кнопок, используемых, чтобы управлять размером,

местоположением, и действием на экране. Всякий раз, когда вы сохраняете программу Rebol в текстовом файле, код должен начинаться со следующего текста:

```
REBOL []
```

Этот текст говорит интерпретатору Rebol, что файл содержит программу Rebol. Он должен быть включен в начале любой сохраненной программы Rebol. Вы должны включить дополнительную документацию о программе, например, заголовок и информацию о версии в квадратных скобках, но она не обязательна.

Примечание: этот пример включает многие слова Rebol и методики, которые еще не были обсуждены в учебнике. Цель примера состоит в том, чтобы просто предоставить более длинный текстовый пример, который может быть вырезан, вставлен, и сохранен в файл. Не беспокойтесь, если пока не все понятно.

```
Rebol [Title: "Средство просмотра Webcam"]

; try http://www.webcam-index.com для некоторых ссылок webcam.

temp-url: "http://type-some-website-here.com"
while [true]  [
    webcam-url: to-url request-text/title/default trim {
        Введите URL:} temp-url
    either attempt [webcam: load webcam-url]
        [break]
        [either request [trim {
            Этот webcam  в настоящий момент не доступен.} trim {
            Повторить} "Выйти"]
            [temp-url: to-string webcam-url]
            [quit]
    ]
]
resize-screen: func [size] [
    webcam/size: to-pair size
    window/size: (to-pair size) + 40x72
    show window
]
window: layout [
    across
    btn "Стоп" [webcam/rate: none show webcam]
    btn "Старт" [
        webcam/rate: 0
        webcam/image: load webcam-url
        show webcam
    ]
    rotary "320x240" "640x480" "160x120" [
        resize-screen to-pair value
    ]
    btn"Выход" [quit] return
    webcam: image load webcam-url  320x240
    with [
        rate: 0
        feel/engage: func [face action event][
            switch action [
            time [face/image: load webcam-url show face]
            ]
        ]
    ]
]
view center-face window
```

Используйте программу, например, **XpackerX** http://www.marmaladefoo.com, чтобы упаковать и распространить программу. **XpackerX** позволяет вам упаковать интерпретатор Rebol и webcam.r в отдельный исполняемый файл. Это позволяет вам создавать отдельные

Windows-программы в одном файле.

ВАЖНО: Чтобы выключить используемое по умолчанию значение, которое непрерывно спрашивает право доступа к чтению и записи на жесткий диск, напечатайте **"secure none"** в интерпретаторе Rebol, и затем выполните программу с **"do {filename}r"**. Выполнение **"C:\rebol.exe -s {filename}"** делает то же самое. **"-S"** запускает интерпретатор Rebol без любых включенных средств защиты, заставляя его вести себя как типичная Windows-программа.

Использование **XpackerX** особенно полезно (не только для Rebol, но и для других интерпретируемых языков). Он избавят ваших пользователей от необходимости загружать интерпретатор Rebol. Упаковывая интерпретатор Rebol, сценарий, и файлы поддержки данных в отдельную выполнимую программу, работает как компилятор Rebol, который создает обычные Windows-программы. Чтобы это сделать, вы должны создать файл **"template.xml"**:

```
<?xml version="1.0"?>
<xpackerdefinition>
    <general>
        <!--shown in taskbar -->
        <appname>your_program_name</appname>
        <exepath>your_program_name.exe</exepath>
        <showextractioninfo>false</showextractioninfo>
        <!-- <iconpath>c:\icon.ico</iconpath> -->
    </general>
    <files>
        <file>
            <source>your_rebol_script.r</source>
            <destination>script.r</destination>
        </file>
        <file>
            <source>C:\Program Files\rebol\view\Rebol.exe</source>
            <destination>rebol.exe</destination>
        </file>
        <!--put any other data files here -->
    </files>
    <!-- $FINDEXE, $TMPRUN, $WINDIR, $PROGRAMDIR, $WINSYSDIR -->
    <onrun>$TMPRUN\rebol.exe -si $TMPRUN\script.r</onrun>
</xpackerdefinition>
```

Загрузите бесплатную программу **XpackerX**, измените вышеупомянутую матрицу так, чтобы она содержала имя файла, которые вы дали своему сценарию, файлам поддержки, и правильному пути к интерпретатору Rebol. Выполните **XpackerX**, и он создаст красиво упакованный.exe файл, который не требует никакой инсталляции

14. Встраивание двоичных данных

Вы будете часто использовать изображения и другие двоичные данные в ваших программах Rebol. Например, простые игры часто используют графические и звуковые файлы как часть их интерфейса. Вы можете читать их с жесткого диска, вы можете загрузить их из Интернета, вы можете читать их из любого другого локального или сетевого носителя данных, и т.д. Возможная альтернатива должна сделать файл, который включает вашу программу и все ее вспомогательные файлы. Установка таких пакетов может быть сложной для пользователей, различные операционные системы используют различные типы сжатия (почтовый индекс, tar, и т.д. - и **XpackerX** работает только в Windows). Чтобы устранить эту проблему, Rebol предоставляет метод, чтобы закодировать и включить внешние файлы в ваши программы. Чтобы видеть, как он работает, используйте код ниже:

```
Rebol [Title: "Конвертирование в двоичные данные"]

system/options/binary-base: 64
file: to-file request-file/only
data: read/binary file
```

```
editor data
```

Программа позволит вам выбирать файл, читать его, и затем отображать двоичное представление его данных во встроенном редакторе. Вы можете скопировать эти данные и вставить непосредственно в вашу программу, назначить имя переменной, и использовать эти данные, как будто они читаются прямо с жесткого диска или другого носителя данных.

Вот пример. Загрузите изображение ниже, и затем сохранить образ на ваш жесткий диск. Или, вы можете загрузить его, используя Rebol, как демонстрируется ранее в учебнике:

http://musiclessonz.com/test.png

Как только вы загрузили файл, используйте программу выше, чтобы выбрать и конвертировать его в двоичные данные. Вы должны получить следующую распечатку:

```
64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAAU1JREFUeJztlzEOgzAQBHkaT7s2ryZUZoYRz4t
e9xsSzTjEXIktqP3trsPcPPo7z36e4/+3qO/9y76t/qjn3766V/oj4jBb86nUyZP
lM7kidKZPFE6kydq/Pjxq/nSElGv3qv50vj/o59++hNQM6Z93+P3zqefAw12Fyqh
v/ToX+4Pt0ubiNKZPFE6Ux5q/O/436lkh6affvrpp38ZRT/99Ov6+f4tPPqX+8Ps
/meidCZPlM7kidKZPFE6kydKZ/JE6UyeKJ3JE6UzeaJ0Jk+UzuSJ0pk8UTMmvn8L
j/7l/nC7tIkonekLdXm9dafSmeinn376D/rpp5/+vv1GqBkT37+FR/9yf7hd2kSU
zuSJ0pk8UTqTJ0pn8kTpTJ4onckTpTN5onQmT5TO5InSmTxROpMnasbE92/h0b/Q
//jR33v09x79vUd/73XvfwNmVzlr+eOLmgAAAABJRU5ErkJggg==
}
```

Теперь используем это:

```
picture: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAAU1JREFUeJztlzEOgzAQBHkaT7s2ryZUZoYRz4t
e9xsSzTjEXIktqP3trsPcPPo7z36e4/+3qO/9y76t/qjn3766V/oj4jBb86nUyZP
lM7kidKZPFE6kydq/Pjxq/nSElGv3qv50vj/o59++hNQM6Z93+P3zqefAw12Fyqh
v/ToX+4Pt0ubiNKZPFE6Ux5q/O/436lkh6affvrpp38ZRT/99Ov6+f4tPPqX+8Ps
/meidCZPlM7kidKZPFE6kydKZ/JE6UyeKJ3JE6UzeaJ0Jk+UzuSJ0pk8UTMmvn8L
j/7l/nC7tIkonekLdXm9dafSmeinn376D/rpp5/+vv1GqBkT37+FR/9yf7hd2kSU
zuSJ0pk8UTqTJ0pn8kTpTJ4onckTpTN5onQmT5TO5InSmTxROpMnasbE92/h0b/Q
//jR33v09x79vUd/73XvfwNmVzlr+eOLmgAAAABJRU5ErkJggg==
}
```

Вы получите ответ ниже, что изображение определенно и создано:

```
make image! [85x100 #{
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF...}
```

Здесь эта картинка находится в GUI:

```
view layout [image picture]
```

Пример ниже отображает фотографию моей собаки, и затем сохраняет ее на жесткий диск как изображение.png:

```
Rebol []
dog: load 64#{
```

```
iVBORw0KGgoAAAANSUhEUgAAAFkAAACNBAMAAAAuisulAAAAMFBMVEUeEw6kh3OM
TRZDSE/RxbWFal1KLiCtkoakZSXW0cm9mnlCNzq5no5oSDTg29B0UkzBjRe5AAAA
CXBIWXMAAC4jAAAuIwF4pT92AAAQ80lEQVR4nG1Yf2gjZ3r+kiW6gjHJZHdOjuLU
W0GMKRgShG7/UIVqqu2eUtd3c9pvpe6uGJ32dnqca+zhIoa6glubczCBZVsQhGkX
9UR369zgEKeqXeic0G6xERHTxLGxWFCH+WehGIKIVx2viTJ8fd9v5M2m9JN/CPx8
zzzv8/74PpkIQTUYVDUtqMFP+L22uKitaaXFktaEZZXgyzpdD0hQEFRcz+CA10pr
HIurVLKa36KpAHhhAA8CeBHIm8BcwgXA5+APCGM0GBSCQU4ML5VvaDWfLav5HDez
40dAHhSQHaHwDNUw1pB5EXiR3hqIR/S+I3FupA4iWDPWjRYXUlovIfe3SuRcp+6q
Ag92QK5ebVmGhgFqKMQq8e8Sj5LadkMQnoFBh321ZTSbmtZcszBIHuuAW5L6pv1U
OPUFvhy3FbpnlLQScnPDTzc8IKLssc4Bxsmlgw67fdlo7WjWwMJBmPgFuj3W7zuc
G/wGx4PC3Kix/THsazZ9fNPHct2s0fC2JBW5ueuC91/r1nr6Ok++LwQf40cpyZ1+
o3+Anmg898VZuzX6+f4FID/VUnoWpTvm1Fmn4SdIQ+3Uvh/6OAJoXljIPjAQ0EfU
Yw22JQHcLys1/cX2vfuRaPM73AMlDvX6rNH5Arm54X9z45OL60YkgnnC7OAOyy8W
rCrZ6bP2pm94MFiMzb40tb6ejmSwvJrWoMx5/h8Qm8m2Z3v7AleuqbFI9BNrvfU6
CA8OytaXA/zALdO+Y7MnEk+QGolsmKMP32n9MvI2VkGz+XwP8eww16XyFz64Vntr
46+qllE0o6h7kTNDXvmOBeLYDnPlrHsTuIuRWiQSe/13Rsu4b9Z4mH4dwo41rtur
1z2XUgrCg2kAR8yEZRmGEakFfVMAvjbQskBYpwNSKLOF4NlYDNAbG9eNe8Z6j5uy
1iwtDroIFkRZ7zgMinz2SOXgmPlV6GurZfVqaAqKBkfWfO4HyN1waNGc/SJdQyGR
WnT0nXVjZz92QfNdATHgJP6GOjmGOmGsvvUJp45EzpilQ4bxT/COTxcYRc9sB91e
w5EZ65i1Wo3DzUDwfWM9B++CPhy0NK0BNwMowPtbphnx14ZlhCDIiJ97BA/qC7gp
ZbQGVWiasVP0N62fhK5BvCCczyJtbTCIsKqY3IMfpmnWfPRbLy++aqQBHeHDhXNj
dSG37bq5PmOJ3iz+HVeMXDDW/9NHa0E/obxaAG0yythmRO6btdqplO+1WuhJLePP
3NKpKwukwUSXNVw3fYoF0pes9c/xDS8soF9r+llaIHVJ7rMelKFZi32Ltjj6Wyl+
Gz0gW7LEWJ+5aTNyamHsTMvaQSU8nX7hLvpKzHlXhqpywO9TdCQw3RK5sAt+Yy/y
SJvNBTJXTx9BycZMns0aN5K0jCo+I1JrDsjXBp50zCO3waRrpjkLyA0SGDJjeahw
4IZnZXA0ojF8PC+Qrc4BTBR6La+Y9Y6iBAhRlI3R9RbnPk0nwuGwA0+2nIzsCa8T
Zc5htn1ntZ0PkOuW1ePwqCZwX4KLqAmibNejMpNyRHl0kpvIZulKm5DrrVAPpNQi
Mc0/6PwF3HN1lx24VwJvrtyZfHN1ZSmXy5PZg2Cac0cuqKr6HNo052SXuf0AXZm0
DyRh5D1QQsjNaxEeZkLjZ5fa9NGz7brnMHYlYNu2Yp9kaX4WwOTM6zVeZNHTowuT
Ckpmbc/NOcdKwW4rc7BlN5+/sZW/+X2QXat98uw8Rzxwzx04jiQrCkADbaXd2c0r
b3uBr38D9teikn9iYKAqcs/Z0Glf5PKAPgarbaVOhjobc6F/DkCy7AZVtec9cRwn
7tGOEphdOQ7Y7XZ7bjewFfja+CigbHU6diPDrxgaRqkuEJexHGWe8uIcvf0izOb2
7DXSrhvGq6TWUeYU+6B5yq1ybteTXY8MP6K5+YM7nq2wgKJUhY/JDa/TNh0m+Acj
KoJ54sA8dj1l+OAk+3O6cscLOCQ/N1qtEuKmZ2EAXxXUb7k3mStRdqwMe4p4kHVp
LnCg5OcMwSCk1rZPbHbET9ES8oMST6Ii27CH7Xw2m7NdOe/eHi9cNQRCAqbNnBO3
iuTWwG9ZEsV+3j4/pLgrVwIb9UcrdMVeCoXyhEQajic7SG1ZeCuCOqFUlNJ52w7k
Xa89fMyosJLNHhnGLqTfTMA5kwENpekm545RWRyTQVCeOCd5olBRGJHuvG8Yt0DK
0GbcZZhPranyKGXqjhXN/LyzG2B38oF5ikfhyteC+GMoxcANV04/5XcoPKVharqC
IPZNxesoNtTVAaIl+Q9CxitDCiFD0Wjubf92CTeABdKjY1RuNAAKy3OoJIgivfKL
6vu/ma1vkPwNNx4N8pslryrXFWGCs858p922nQMqHwhS9vgFQ/j+nA09RzbNTd4+
uGGBpNkBhSONzbdnoSps1v49OJW/ekEQXp1LIPpMrIG6+aULMu/yF2NbirJVNxXy
klSq5l+uVj+aW/EAPWSagsZ1Yy7rDegzWN5u3lQATDZDhkjmq9WReXqb95wZ9G9c
qKSD0AYMzo2AsgEtGThaN64E5sCZE5rNc3JB4zqC2isEkC6GyfL5PAGLX75utXaH
hkNidUWiu0huZhCMTbFA4p0j6qMJyQ8FyOaodT/fCRjgupRF9NbQU1DB75avELNB
JUT3kWaIBDJB60rgB0QQQmA85x7KwC1Hww2ge1Pk3MeIJRsBQRXzDiGhkGFU5TZn
gHnl3/+BG64b4KDHcEJtkTMSjQ/Bc65WBUPEkQjr3wS8cMEHjFfglILbPZX37a0N
CJ+c6e0rL8FzlkBKNeuhKeR7eBHFO/ePCcfKcr2zZZKhfv5GQlEOfkACV6uGIdA7
HP0yVwKefkVkF8+dXqfeCfRqdEOJBEwxT4alqiBUBZmjAxAxXgCFXSJx6r7Xr2+B
8UptP3AgbhDIzki1KmZ94XMuflYQgEWGKF0ZzqiOCSe+UtuapWMwkK8u3Qd2bjie
LYUJLOU8UUUq0V6stm/Os4ZMbhzPS1cIca6i4QJ9j6NhHClK+OQNQoIihfLvwYgE
NCU34pvCFTIEHWQAN7x9hoZFoC8pFTTITwyV5JVEgr5F5kXBGAGwX4UAz8MMViBk
0JGWKd5SYg3mbQ0n6NhG4EAQ0RI4V3xwWwF4ANAQZcaV5V4PTvp6vD7rlu6+VcuE
AAsbBF6zhHi37XYehMBE7kmy25Oj4GS93utcmOmmQfV94b4hQs+NbSD6Xwp79hOo
aJ7LNHzD6pkwnK/PdC+iH6ERcWlFWro3xS0cHx8P77W5EkGuJVxG0zSWkMXRbndK
DVn3YceH4pI4c/gWpmc8XCjMLecJEyFMvM4Uo4m06UjnKodfautWtWqIwtKSNJNc
APRwGLgLYIpMpSgk8whsT9N+jb7b7X6plqwd4cNQVUD03wL6xUKhoIyHzxNIvACF
Ar+CxUS67pa63cfC9dJU6EOQPiLM6H+GFoZhjY8DtyWqwR24VMNhJLu16Llut/tQ
spJCNWQsSdKM/g+oBIUUgJsW5SKI6blPNfFX0VjjOqDLglUxQqGqNCLO6P844AYx
MLTpAZXSImUylIucTvxRBeCHle7D6vvVEUF8V/8TRO+hhePnibPP5KD4UDzrRqlI
0+6FmUqle9g9bIHnwP0aV0ImUXc4T+Jxu+dCsRQxUFFMRD/rVgBcnjJC1RGRfqr/
KaILBZ/bi8MLGwhshNMtkv4MmLuHqQ9CSyNLAk3pf0wCdeXFcAGUg+59247HqewW
0/RteHmvge6KXvlvFZplSdL1B6TO2oFCGNjPk307ntjfx09IqgvJ76W/Qe7uzAdB
NbskZnT9lQbNHiun3HGPxSOuy+QLIgrq/bsO3F39cTIp0qVRXU9TaeWJUsAoz5Oe
E9+3HYqlSCVJjCbcFIR5qFf0aQG4U2UpKGa9AmAL44T0mQPUULVQsbBBZrnfgoVd
/bA8nR0RR1NlzaqKe1AkhfPgDe31bAduEWkqylBfCSZf7lYqlcOkNSVJwrs6oK0R
8GScNxEV7YTrwT1CALsBzORvdG5heQo67VO9DB8sxfEAthl4InmOg40DlQsvFkn0
6QwKP9SnBENI6eXtbeuNPWiy4QJoGYvXMD6QMQaFEmGA/uywWyl39UtqFdC6Nf0w
ayt5ZS8MnlxzDvADI5VgaNFcohGtb547hCB1valWReC2th/a7YKNhXKenN0/QLAY
1EQ3Gq9H3PpPL6PfUClBzn1p+mN7r4DtANyZiAfRjWVEYSftxGPM7bxwWUfZoEQV
dF1PNnN7T07R4lg8CjkMCsUii8ccx2vfRAvLlXIzpIqpsp7c8Zw9ZYB2pbRXlKWH
```

```
gnY2kbYTCda2r0M36F9OXwq2qqlyqnyxPdk+5abS5z13LLMjFYu93j4eQt5lqCvI
/KWSVQW/k9NtZbyw56N3opmEqz5UqVBMRPYbUC7HoAT06h+UrNanKPwaTvvwMqKb
NCP3mk2VBqWeF09AUnOXu4AB7net5l/rern8I6Uwh+Dl8+TSr9xfskwJjiza68eZ
K9DcX0CUun5417KsUbDw7lSuDdDl5fAtUiqmpQj+O0zIsHgcL57Hv9/FgtXvvmtZ
5yA9+vTYJFIj90MBToeSoDWFo17f9grHk23I/GESfLa2LSulJ/WyHA5PIDugNTUK
52xRUDNyxLEVZa89A5kE7qSV3C6BJ3ryYyCemMAowei0WBRLmaDI4jzFNlZg6hBq
dXu7qZeB/T+4f6hkVINTjcINXlJZHDqqEJ7AGLGszm1PN5PJZDn5owmf+xYoCWpH
VJAeSiKzMWWFX1TAk5ReuRvcti5D5svTF4EX4beAu6ip0O1FVWb2OPb2z2CyHZb1
u1S2Wj8rg/7kFHLDBsi8RoPQNEeqKHvIXZj8NZQgDIhL0vH2pd+VpVS5PBVe5XGe
J0UYZ+JZMbMDByyUfGF8kncxUDqxjJiZCkRTyallsA/YoWI1KmagYvGAxdFYmPwt
2FeZeVyOJ9i/Tv/5T7WMPrU8gUKWbxFVE0WQjTcxbxye94h+CgZWKjNlmrn0P2Ur
un12+yKCl1F3UwvCbHgqwPUU/AtPrkATH6KSSvIQivuzi9Jr537IDVwNI7cGrUNV
l8sOhyXxUxiCjzGZKTD93sxdIfnRKq8qjFKVZEiP6LoehBhWQgJwdzm2rM+k7pUr
enIEaSbCgM6oIESG+6bsYX4nJOGzFFY3wqG4p/VUahpV+35ndvBe9XTMZbwYlJs0
iY2DdTUD7JfK5fIPl1HGMvqtqYIruxmRHo/z0lHe3AbsDNDCK1X+u1Qq+SrECHAf
rUbhwClCjNB8sObvpRB4yHsz9fcpzPsyZwclF7Qig9GmHg8/CfvrXjKVgnRDhaTK
j7/cTmZ5blZXJ1Zvkb+Ez0aOTD/3z3Jcl5Plx2Wdy4E99+5jk61OLMPXLfISzTHK
pF1A+yMm/IdY02Xkr1SSZQrQMIpeXV49T/JwqaL0GpzOJz54+c0kwvXyYQp+ffjo
0SpmErATE7fI+TdyTKZfwei/fSrl1wjXU8lU8p3VlTso2H8B966D/0zEq9kjnjD8
sZ3k6537q7DeXD1d4ODkPNySj/EEguzA/gl8rJF8x2hVVzjmwwF2eRWiHHfuMPpj
fvJjfiFly/yPSwPQ0tQzbnAwPM/Y0TEq+fkjnHX8Ac/D7z0DI/dN+Dx/uw3wR8sY
+nJ4efU769zIc9zjc8xzH9mT7w2DCCDGLc+tidX3q8/egxLwJDu5fPJkNcwL0/cK
14Bz6TvcDnNyJ5MYHCZs9XkdE/9H1C1ye555P11G43ybTpn/v3XrfwFzPoFQpggI
EQAAAABJRU5ErkJggg==
}

view layout [image dog]
save/png %dog.png dog
```

Код выше может быть сокращен, используя встроенное слово **"compress"**. В Rebol встроенное сжатие, и распаковка работают только с текстовыми строками. Таким образом, программа, предоставленная ранее **(Конвертирование в двоичные данные)** должна быть откорректирована следующим образом:

```
REBOL [Title: "Конвертирование в двоичные данные"]

system/options/binary-base: 64
file: to-file request-file/only
if not file [quit]
uncompressed: read/binary file

compressed: compress to-string uncompressed
; код выше преобразовывает двоичные данные
; в текстовую строку, и затем сжимает ее.

editor compressed
alert rejoin ["Файл готов"]
```

Чтобы использовать сжатую версию данных выше, вы должны будете полностью изменить текстово - двоичное преобразование после декомпрессации. Чтобы сделать это, используйте следующий код:

```
to-binary decompress {compressed data}
```

Используя переменные выше, следующие две строки программы отображают то же самое изображение:

```
view layout [загрузить распакованное изображение]
view layout [загрузить двоичное изображение для распаковки]
```

Вот законченный пример, демонстрирующий различие размера между исходником и сжатыми

данными:

```
; распакованное внедренное изображение:

image-uncompressed: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAP8AAAEsCAIAAACDt/KoAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAB9FJREFUeJzt3bGRHEkQBMETDaKd5v8UmlmzIaoz
MGtZ4QLUIBtBLXM//0lb/bz9D5BeY/3ay/q1l/VrL+vXXtavvaxfe1m/9rJ+7WX9
2sv6tZf1ay/r117Wr72sX3tZv/ayfu1l/drL+rWX9Wsv69de1q+9rF97Wb/2sn7t
Zf3ay/q1l/VrL+vXXmD9P1ICmCh4WkoAEwVPSwlgouBpKQFMFDwtJYCJgqelBDBR
8LSUACYKnpYSwETB01ICmCh4WkoAEwVPSwlgouBpKQFMFDwtJYCJgqelBDBR8LSU
ACYKnv7w+1f8W+fyP307B+a8MxA8Xf92Dsyx/jnjIPQPvD/xixn8G4wjqH/g/Qnr
r42jfuD9CeuvjaN+4P0J66+No37g/Qnrr42jfuD9CeuvjaN+4P0J66+No37g/Qnr
r42jfuD9CeuvjaN+4P0J66+No37g/Qnrr42jfuD9CeuvjaN+4P0J66+No37g/Qnr
r42jfuD9CeuvjaN+4P0J66+No37g/Qnrr42jfuD9CeuvjaN+4P0J66+No37g/Qnr
r42jfuD9CeuvjaN+4P0J66+No37g/Qnrr42jfuD9CeuvjaN+4P0J66+No37g/Qnr
r42jfuD9CeuvjaN+4P0J66+No37g/Qnrr42jfuD9CeuvjaN+IHi6/u0cmGP9c8ZB
6B8Inq5/Owfm9Nf/56/ftHP54T8szoFBDwPBRMHT3/F28W8dDgyyfuTt4t86HBjU
X780ACYKnpYSwETB01ICmCh4WkoAEwVPSwlgouBpKQFMFDz94fy8Ff/Ww4908W8d
Dgx6ZyB4uv7tHJhj/XPGQegfCJ6ufzsH5lj/nHEQ+geCp+vfzoE51j9nHIT+geDp
+rdzYI71zxkHoX8geLr+7RyYY/1zxkHoHwiern87B+ZY/5xxEPoHgqfr386BOdY/
ZxyE/oHg6fq3c2CO9c8ZB6F/IHi6/u0cmGP9c8ZB6B8Inq5/OwfmWP+ccRD6B4Kn
69/OgTnWP2cchP6B4On6t3NgjvXPGQehfyB4uv7tHJhj/XPGQegfCJ6ufzsH5lj/
nHEQ+geCp+vfzoE51j9nHIT+geDp+rdzYI71zxkHoX8geLr+7RyYY/1zxkHoHwie
rn87B+b01//K37mPf+twYNDDQDBR8PR3vF38W4cDg6wfebv4tw4HBvXXLw2AiYKn
pQQwUfC0lAAmCp6WEsBEwdNSApgoeFpKABMFT384P2/Fv/XwI138W4cDg94ZCJ6u
fzsH5lj/nHEQ+geCp+vfzoE51j9nHIT+geDp+rdzYI71zxkHoX8geLr+7RyYY/1z
xkHoHwiern87B+ZY/5xxEPoHgqfr386BOdY/ZxyE/oHg6fq3c2CO9c8ZB6F/IHi6
/u0cmGP9c8ZB6B8Inq5/OwfmWP+ccRD6B4Kn69/OgTnWP2cchP6B4On6t3NgjvXP
GQehfyB4uv7tHJhj/XPGQegfCJ6ufzsH5lj/nHEQ+geCp+vfzoE51j9nHIT+geDp
+rdzYI71zxkHoX8geLr+7RyYY/1zxkHoHwiern87B+ZY/5xxEPoHgqfr386BOf31
v/J37uPfOhwY9DAQTBQ8/R1vF//W4cAg60feLv6tw4FB/fVLA2Ci4GkpAUwUPC0l
gImCp6UEMFHwtJQAJgqelhLARMHTH87PW/FvPfxIF//W4cCgdwaCp+vfzoE51j9n
HIT+geDp+rdzYI71zxkHoX8geLr+7RyYY/1zxkHoHwiern87B+ZY/5xxEPoHgqfr
386BOdY/ZxyE/oHg6fq3c2CO9c8ZB6F/IHi6/u0cmGP9c8ZB6B8Inq5/OwfmWP+c
cRD6B4Kn69/OgTnWP2cchP6B4On6t3NgjvXPGQehfyB4uv7tHJhj/XPGQegfCJ6u
fzsH5lj/nHEQ+geCp+vfzoE51j9nHIT+geDp+rdzYI71zxkHoX8geLr+7RyYY/1z
xkHoHwiern87B+ZY/5xxEPoHgqfr386BOdY/ZxyE/oHg6fq3c2BOf/2v/J37+LcO
BwY9DAQTBU9/x9vFv3U4MMj6kbeLf+twYFB//dIAmCh4WkoAEwVPSwlgouBpKQFM
FDwtJYCJgqelBDBR8PSH8/NW/FsPP9LFv3U4MOidgeDp+rdzYI71zxkHoX8geLr+
7RyYY/1zxkHoHwiern87B+ZY/5xxEPoHgqfr386BOdY/ZxyE/oHg6fq3c2CO9c8Z
B6F/IHi6/u0cmGP9c8ZB6B8Inq5/OwfmWP+ccRD6B4Kn69/OgTnWP2cchP6B4On6
t3NgjvXPGQehfyB4uv7tHJhj/XPGQegfCJ6ufzsH5lj/nHEQ+geCp+vfzoE51j9n
HIT+geDp+rdzYI71zxkHoX8geLr+7RyYY/1zxkHoHwiern87B+ZY/5xxEPoHgqfr
386BOdY/ZxyE/oHg6fq3c2CO9c8ZB6F/IHi6/u0cmNNf/yt/5z7+rcOBQQ8DwUTB
09/xdvFvHQ4Msn7k7eLfOhwY1F+/NAAmCp6WEsBEwdNSApgoeFpKABMFT0sJYKLg
aSkBTJQ7LX0569de1q+9rF97Wb/2sn7tZf3ay/q1l/VrL+vXXtavvaxfe1m/9rJ+
7WX92sv6tZf1ay/r117Wr72sX3tZv/ayfu1l/drL+rWX9Wsv69de1q+9rF97Wb/2
sn7t9T/igalLLsvMjgAAAABJRU5ErkJggg==
}

; Вот сжатая версия тех же данных:

image-compressed: load to-binary decompress 64#{
eJzrDPBz5+WS4mJgYOD19HAJAtL/GRgYdTiYgKzm7Z9WACnhEteIkuD8tJLyxKJU
hiBXJ38f/bDM1PL+m2IVDAzsFz1dHEMq5ry9u3GijKcAy0Fh3kVzn/0XmRW5WXGV
sUF25EOmKwrSjrrF9v89o//u+cs/IS75763Tv7ZO/5qt//p63LX1e9fEV0fu/7ap
7m0qZRIJf+2DmGZoVER5MQiz+ntzJix6kKnJ6CNio6va0Nm0fCmLQeCHLVMY1Ljm
TRM64HLwMpGK/334Hf4n+vkn+1pr9md7jAVsYv+X8Z3Z+M/yscIX/j32H7sl/0j3
KK+of/CX8/X63sV1w51WqNj1763MjOS/xcccX8hzzFtXDwyXL9f/P19/f0vxz4f2
OucaHfmZDwID+P7Hso/5snw8m+qevH1030pG4kr8fhNC4f/34Z89ov+vHe4vAeut
SsdqX8T/OYUCv9iblr++f67R8pp9ukzLv8YHL39tL07o+3pekn1h/dDVBgzLU/d3
9te/Lki4cNgBmA6/lO+J/RPdzty8Rr5y94/tfOxsX6/r8xJK0/UW9vlH93/9oAzR
e09yKIUBVbT9/br/U/m7x6CU98VAAJS2ZPPF/197eEDhtfs9vX9rDzc6/v3qzUyo
nJA/dz76Y77tHw+w3gXlbEMpDKihza/+7/o/c3+DU54tDwsobR2/fXR/qYXBiV8T
t3eDEmpA/d9LDASK0y/tnz+H/Ynmt78E1vti7lAKA6pouxz/X7v+uR045ZFdRE6x
1q21pG7NiSzx1f5R40pvvdNn+oB1P4Onq5/LOqeEJgCemFy1KQgAAA==
}
```

```
view layout [не сжатые данные]
view layout [сжатые данные]
```

Когда вы используете сжатие как это, при загрузке убедитесь, что включали слово **"to-binary decompress"**.

Вы можете преобразовать любой тип файла в сжатые текстовые данные. Изображения, звуки, видео, даже все выполнимые программы могут быть включены в ваш исходный код!

15. Модульное программирование и повторное использование кода

Многократное использование существующего кода является основным, если вы хотите стать продуктивным программистом. Создание функций - основной способ осуществить многократное использование кода. Как только вы создаете новое служебное слово, чтобы достигнуть нужного действия, вы можете скопировать его определение и использовать это много раз в ваших программах. Это избавляет вас от необходимости повторно изобретать каждый раз функции и блоки, когда они необходимы. Также уменьшается вероятность появления ошибок - старый код, не будет содержать ошибки, если это было уже проверено в работе.

Вы можете импортировать существующие модули кода в вашу программу. Существующий код может содержать функциональные и переменные определения, новые программируемые структуры, и даже завершить программы любой длины и уровня сложности. Как только слова и определения были импортированы в вашу программу, вы можете использовать включенные функции и переменные, как будто они - встроенные слова языка. Вам даже не обязательно знать, как они были созданы. Наберите следующий пример кода, и сохраните его к **"C:\play_sound.r"**.

```
Rebol [title: "play-sound"]

play-sound: func [sound-file] [
    wait 0
    ring: load sound-file
    sound-port: open sound://
    insert sound-port ring
    wait sound-port
    close sound-port
]
```

Код выше создает новое служебное слово **"play-sound"**, который принимает переданное переменное имя файла **"sound-file"** (**файл должен быть ".wav"**), и запускает звук через компьютерные динамики. Теперь, всякий раз, когда хотите запустить звук, вы можете включить код в свою программу:

```
do %/c/play_sound.r
```

И используйте функцию **"play-sound"** точно так же как любое другое встроенное служебное слово (синтаксис - **"play-sound {sound-file}"**):

```
play-sound %/C/WINDOWS/Media/chimes.wav
```

Чтобы подключить этот модуль в новую программу, надо ввести сроку **"%/c/play_sound.r"**. После этого определено слово **"play-sound"**, и его можно использовать на протяжении всей программы:

```
alert "Вот звук:"
play-sound %/C/1.wav
```

```
alert "И вот другой звук:"
play-sound %/C/2.wav

alert "Теперь попробуйте выбрать.wav файл на вашем жестком диске:"
play-sound to-file request-file/file %/C/1.wav
```

Чтобы снова использовать модули, необходимо сначала изучить язык, так, чтобы вы могли понять код, написанный другими разработчиками. Вы, возможно, должны приспособить и расширить код, написанный другими, так, чтобы он соответствовал вашим потребностям более точно. В большинстве случаев, однако, вы можете только узнать, как использовать слова в импортированном модуле, или диалекте, выполнить **"do {filename}"** команду.

Вебсайт http://www.dobeash.com/it/ предлагает несколько свободных модулей расширения для языка Rebol. Например, модуль **"RebGUI"**, расширяет встроенный в Rebol уже сильный синтаксис GUI, встроенный в Rebol **("view layout...")**. Используя RebGUI, можно легко создать графические элементы, которые не возможны в Rebol. Чтобы использовать RebGUI, загрузите файлы с вебсайта выше, и распакуйте на диске C:\. . Вы можете подключить, это любым из средств, описанных в предыдущей главе **("do %/c/rebgui_example.r", и т.д.):**

```
REBOL []

do %\c\rebgui.r

display "Сетка" [
    table #WH options [
        "День" left .5 "Время" left .3 "Имя" left .3
    ] data [
        Monday 9:00 "Вася" Tuesday 9:30 "Маша" Wednesday 10:00 "Миша"
    ]
]

do-events
```

Код выше использует некоторые новые команды, которые не являются встроенной частью языка Rebol. Функции и переменные **"display", "table", "#WH", "options"**, и **"data"** определены в файле rebgui.r, и они добавляют новые функциональные возможности к языку Rebol. Используя те слова, как определено в rebgui.r, вышеупомянутый код показывает данный блок данных **[Monday 9:00 "Вася Tuesday" 9:30 "Маша Wednesday" 10:00 "Миша"]** в GUI, где можно автоматически сортировать, нажимая на заголовки в колонках GUI. Этот тип интерфейса - общее требование в современных программах, которые показывают списки данных. RebGUI содержит широкую коллекцию дополнительных функций, которые полезны в построении сложного GUI. Чтобы использовать их, вы должны сначала понять основной синтаксис Rebol, и затем узнать, как использовать языковые расширения RebGUI. Как только вы освоите это, вы можете просто включить файл **"rebgui.r"** в свои программы, и использовать эти языковые расширения, как будто они - часть языка. Чтобы их использовать, вы должны сделать импорт rebgui.r.

Вот некоторые сайты, содержащие свободные модули, которые могут быть вам полезными:

http://www.hmkdesign.dk/rebol/list-view/list-view.r - Возможно это, самое полезное дополнение к Rebol GUI.

http://www.dobeash.com/it/rebdb/ - модуль базы данных, который позволяет вам легко хранить и организовывать большое количество данных, используя язык данных "SQL". Есть также модуль спеллчекера, который может быть включен в ваши программы.

http://www.rebol.org/cgi-bin/cgiwrap/rebol/view-script.r?script=rebzip.r - модуль для сжатия и распаковки zip файлов.

http://www.colellachiara.com/soft/Misc/pdf-maker.r - диалект для создания pdf

файлов.

http://softinnov.org/rebol/mysql.shtml - модуль, чтобы управлять mysql базами данных в пределах Rebol. Очень полезный для веб-программирования.

http://www.rebol.org/cgi-bin/cgiwrap/rebol/view-script.r?script=menu-system.r - диалект, чтобы создать все типы меню GUI в Rebol. Более поздняя глава посвящена этой теме.

http://softinnov.org/rebol/uniserve.shtml - структура, чтобы помочь построить сетевые приложения клиент-сервер.

http://www.rebol.net/demos/BF02D682713522AA/i-rebot.r
http://www.rebol.net/demos/BF02D682713522AA/objective.r
http://www.rebol.net/demos/BF02D682713522AA/histogram.r - эти примеры содержат трехмерный машинный модуль, написанный полностью на чистом Rebol. Модуль позволяет вам легко добавлять и управлять трехмерными графическими объектами в ваших приложениях Rebol.

http://web.archive.org/web/20030411094732/www3.sympatico.ca/gavin.mckenzie/ - библиотека анализатора (парсер) Rebol XML.

http://box.lebeda.ws/~hmm/rswf/ - диалект, чтобы создать flash (SWF) файлы непосредственно из скриптов Rebol.

http://www.rebol.net/docs/makedoc.html - конвертирование текстовых файлов в отформатированные файлы HTML.

http://www.rebol.org - вся библиотека Rebol - полная многих дополнительных модулей и полезных кодовых фрагментов. Первое место, где нужно искать необходимый исходный текст Rebol.

Используйте внешние программы как **"modules"**:

Помните, что данные часто являются общими между существующими программами. Встроенное слово Rebol **"Call "** позволяет вам управлять другими программами на вашем компьютере. Используя **"Call"**, вы можете выполнить все команды **"shell"**, включенных в операционную систему вашего компьютера (то есть, DOS и команды Unix). Пример ниже открывает, блокнот Windows, чтобы отредактировать **"rebgui_example.r"**:

```
call "notepad.exe c:\rebgui_example.r"
```

Этот следующий пример открывает программу Paint, чтобы отредактировать изображение, которое мы загружали ранее:

```
call "mspaint.exe c:\bay.jpg"
```

Вот пример, который включает выполнимую программу в код, развертывает, и пишет программу накопителю, и затем управляет этим с функцией запроса:

```
program: load to-binary decompress 64#{
eJztF11sU2X03K4VqJsrkZJp6OzchhFJsx8qDB9od1fHdIO6ds7AgJX2jttyey/p
vWUjJuNnmNhMibzwaCSLi+EBE1ziGIkBGh0BSYTwwAMme9Dk4kgkgSiKcj3nu7es
QrKFhMUQOcn5+c7fd875+vXe27FJAg4AbIiGAQwWIwZMEbqTcmODN5xRdmRi6aoy
Z83YogngLlaNtV+s6kV7q9KelHeu9LYqQTXt7e/v97UqLcLuqKJIvriShnAIoJ0r
gXvPn+StlDAF5dyzHLwAdlw4TZ1Mm7oQvWDu7jKLslsxBc4KQ30bb9bMHF3F/D5j
MFAHEIbHD+cwb88s9riSEIjvK7EKogZs//bxAvQmYlqM5JsOUwHPWFgEAYDTvqTp
eYdy1Fn5Sh/O96h9nLrrDcD4IpQm7UOkWL/nt6MlqMvxrkl+GVWS7xqWalzDzqGz
9rbyD5ehpmnl+ezt3M/RSPe7Q9/ajeh5+9Ztm3vKh9xoM7SaimLUR18C2JKf+Kg2
APoJwzDOuiAF+hHU/pHXryObdLyP+y2kEhx7UaLfo0gq/RJa60/n88Ndrpz7FmqG
```

```
u5bk3L8zwdWXc0+jdOYXkn4lnYfW++/qOPLyDz7BfH3jTXVnplx949inhPvnSgw/
8RSIHM7P8PdSUYtxlxSkONE+o/u7EkNElMbpcuRKUhTjmLH/iHbDQQ7DHqL77zbh
oQxeRa9duBQHkRj+HnIdr7y/e178AvmmnHt5VQAmaNo59/EZ8QSJAY7EURJvMu2x
KipYj2CaEToYve2eYYiwl4rWY6jN8RWF5XtsuWSyhO7aJG8XXQFkNdWYIqIHK8nH
8FOSFJMoteEfZfQEo1SNCPCW2/BTjWK1uXkp9dDDegjrDqpkAUtiJhNp4ma3qUrx
MG6dqkyFMQ2ExQmaxgU2c/07D2ZJsCz3Q68Xh76Cvac2pZwi8jCO8rIZd4jielmc
uHxmsEMe1vMBZJf0YY8Pda95yH5p+tWrI86XMZbTE5a1gVlXFKyryeowp0Cy4Wf+
hdSrWGp26N008hW4XnS6/OBS7MnUVHoK0osoTV+22qF56c95qKdtZBzB66J/imSc
/Rmsg/KDdHFbA9O3RrZWByD/qPf1KTCwze3y2KCbn9vnP4ExoItiwr11zvncqq6+
oXGV//XVa5qCzXxL6M3ZfBfMZyFPBvywgD3FGDjLnGVl83o4T+HJAZ/PFxWTqrcj
GxerHljRqyL9sWXxqU2/nkHki1H4HDkvJeM7vZooeLdnNU2R10K34G1XdgveTmE7
vmv7fNDcFY1u3ABpNa5J6rZd9MouqGpjw6z1GLXn6vDxV/s9o1cYvcroNUanGP2J
UZ3RG4zeZPQ2o3cY/YtRqCdqZ3Qho6WMuhitYHQZ0pr6mRr21Zvv03VFuuMoX0Gd
VqT7BlupKFoXw8eo/8yynUR+HvEa4g3EPxEXYuwSxOWIaxADiGHEBKKGeADxCOIx
a1wXkE81zH/ut0OdG0LtjQ2+hCSBzLUKWoeSyErC+pickIQgfAmhgaSG319xPEvo
ioQ6Ld9D0CL04ddZQuknaxA4W1hRtXeySa0DXWM7BHjDFhHkhLUKYs2cJTcrA0H4
mmtXYgk+m1GVTBBOsVVbXJGDsNTWKexIqpqQ4aWYqgbps4LPCDFNMPcLYXQpldrC
g0bcVHcKcQ220DqyB4PTHYKWScZVgCGsw/LBEgHWsjYLZR2zRTMxWZUwfaFwOAot
SXVXTIuLM9V/ZeuSMw/UxW/s4KOF6W2GNjmp8Uo6rci8ImsZRVLxG+1hZWhgrlv6
/4F/ABcSIgQAEAAA
}
write/binary %program.exe program
call %program.exe
```

16. Ключевые характеристики Rebol

Список ниже показывает некоторые ключевые особенности языка Rebol. Знание, как использовать эти элементы, составляет фундаментальное понимание того, как Rebol работает:

1. Для начала, в Rebol есть сотни встроенных служебных слов, которые выполняют общие задачи. Как и в других языках, служебные слова типично сопровождаются переданными параметрами. Чтобы достигнуть цели, функции устроены так чтобы, один за другим закончить данную задачу. Выражения идут слева направо, отступы строк и пробелы не требуются, но могут быть вставлены, чтобы сделать код читаемым. Текст после точки с запятой и перед новой строкой рассматривают как комментарий. Только знание предопределенных функций в языке и способность использовать их в полной мере позволит вам работать с Rebol.
2. Много общих типов данных понятны интерпретатору Rebol. Числа, текстовые строки, деньги, время, кортежи, URL, двоичные данные изображений, звуков, и т.д. К ресурсам сети и интернет-протоколам (http документы, справочники программы передачи файлов, электронной почты, dns услуги, и т.д.) можно легко получить доступ. Данные любого типа могут быть написаны и прочитаны с фактически любого связанного устройства или ресурса. Помните, что символ процента **("%")** используется, чтобы обратиться к файлам.
3. И код управления и данные сохранены в **"блоках"**, заключенные скобки **("[]")**. Данные и код, содержавшийся в блоках, отделены пробелом. Функции, содержавшиеся в блоках, могут быть оценены (их выполненные действия), с помощью слова **"do"**. Новые служебные слова могут быть созданы, используя **"func"**. Слово **"func"** позволяет вам передавать свои собственные указанные параметры к недавно определенным служебным словам. Вы можете использовать модуль кода, содержавшегося в текстовом файле, пока он содержит минимальный заголовок **"rebol []"**. Данные, содержавшиеся в блоках, могут иметь любого типа, и блоки можно автоматически рассматривать как списки данных, названных **"series"**. Последовательностью можно управлять, используя встроенные функции, которые позволяют поиск, сортировку, и иную организацию данных собранных в блоках.
4. Синтаксис **"view layout [block]"** используется, чтобы создать базовые GUI. Добавьте слова к блоку, чтобы создать графические элементы: **"button"**, **"field"**, **"text-list"**, и т.д. Добавьте цвет, положение, интервал, и другие слова аспекта после каждого слова элемента. Добавьте блоки действия для создания активного элемента. Обратитесь к пунктам в расположении GUI, используя обработки пути (то есть, **"face/offset"** обращается к положению элемента). Эти простые руководства могут использоваться, чтобы создать хороший GUI.
5. Любым данным или программному коду, можно назначить слова как имена. Имена назначаются на значения, переменные, функции, выражения, и блоки любого типа, используя двоеточия **(":")**. После этого назначенные переменные слова могут использоваться, чтобы представить все функции и данные, содержавшиеся в данном

выражении, блоке, и т.д. Поместите двоеточие в конце слова, и после это передавайте все следующие действия или данные. Это является базовой частью языковой структуры Rebol.

17.Завершенные Rebol программы для исследований.

Следующие программы используют множество методов, о которых говорилось выше. Они демонстрируют, как части кода могут быть соединены, чтобы создать законченные приложения, и раскрывают, как могут быть собраны основные стандартные блоки языка в работающую программу. Изучение этих примеров - возможно, самая ценная часть этого обучающего материала.

Загружаемый пакет всех демонстрационных программ доступен в: http://musiclessonz.com/rebol_tutorial_examples.zip. Файл содержит скриншот, отделенный исходный текст, и "exe" файлы каждого примера. XpackerX XML файлы, также включены.

```
mystring: trim {Это строка}
```

Тоже самое:

```
mystring: "Это строка"
```

"Join" использовался, чтобы присоединить слово к длинным разделам текста:

```
join "Всё " [
    "равно "
    "линия "
    "текста."
]
```

тоже самое:

```
"Всё равно линия текста."
```

Блоки могут быть написаны с переносом строк:

```
myblock: [Это
блок
]
```

тоже самое:

```
myblock: [Это тоже блок]
```

Чтобы выполнить примеры программ из этого раздела, вы можете использовать любой из методов, описанных ранее: наберите код в Rebol интерпретатор, сохраните программу как текстовый файл и выполните его, используя **"do {filename}"**, или упакуйте **XpackerX**, и т.д.

17.1 Маленький почтовый клиент.

Первый пример - почтовый клиент, который может использоваться, чтобы читать и

посылать сообщения. Код полностью прокомментирован, чтобы обеспечить объяснения каждой строки, и как каждый элемент работает:

```
Rebol [Title: "Маленький почтовый клиент"]
; (каждая программа требует минимального заголовка)

view layout [
        ; строка выше создает GUI
    h1 "Отправить почту:"
        ; Эта строка добавляет текстовую метку к GUI
    address: field "recipient@website.com"
        ; Эта линия создает текстовое поле для ввода, содержащее
        ; заданный по умолчанию адрес "recipient@website.com". Это назначает
        ; переменное слово "address" к тексту, введенному здесь.
    subject: field "Тема"
        ; другое текстовое поле для ввода темы письма
    body: area "Здесь ваше послание"
        ; Это создает большую, многострочную текстовую область ввода для
        ; основного текста послания.
     btn "send" [
            ; Создана кнопка для отправки писем.
        send/subject to-email address/text body/text subject/text
            ; Использует встроенное слово "send", чтобы послать
            ; электронную почту.
            ; посылает функцию, с ее "/subject" на обработку
            ; принимая три параметра. Это передает
            ; текст, содержавшийся в области, показано выше(названный
            ; "address/text" "body/text" и
            ; "subject/text"). Встроенная функция "to-email"
            ; гарантирует, что текст адреса рассматривают как
            ; данные электронной почты.
        alert "Сообщение отправленно."
            ; приводит пользователя в готовность, когда предыдущая строка закончена.
    ]
    h1 "Читать почту:"
        ; Другая текстовая метка
    mailbox: field "pop://user:pass@website.com"
        ; Другое текстовое поле ввода. e-mail пользователя
        ; информация введена здесь.
    btn "Читать" [
            ; Дополнительная кнопка, на сей раз с действующим
            ; блоком, который читает сообщения от указанного почтового ящика.
        editor read to-url mailbox/text
            ; Встроенная функция "to-url" гарантирует что
            ; текст в поле почтового ящика обработан как url.
            ; Информационные наполнения почтового ящика читаются и отображены
            ; во встроенном редакторе Rebol.
    ]
]
```

Вот тот же самый код, без комментариев - это очень просто:

```
Rebol [Title: "Маленький почтовый клиент"]

view layout [
    h1 "Отправить почту:"
    address: field "recipient@website.com"
    subject: field "Тема"
    body: area "Здесь ваше послание"
    btn "send" [
                send/subject to-email address/text body/text subject/text

        alert "Сообщение отправлено."
                ]
    h1 "Читать почту:"
            mailbox: field "pop://user:pass@website.com"
```

```
    btn "Читать" [
        editor read to-url mailbox/text
    ]
]
```

17.2 FTP чат

Второй пример - простое приложение чата, которое позволяет пользователям обмениваться сообщениями через Интернет. Это включает защищённый паролем доступ для администраторов, чтобы стереть информационные наполнения чата. Это также позволяет пользователям приостанавливать деятельность, и запрашивать продолжения через имя пользователя и пароль **["secret" "password"]**. Чат создан для, динамического создания, чтения, и сохранения текстовых файлы через **FTP** (чтобы использовать программу, вы должны будете обратиться к **ftp-серверу**: адрес программы передачи файлов, имя пользователя, и пароль).

```
Rebol [title: "FTP чат"]  ; необходимый заголовок

webserver: to-url request-text/title/default trim {
    Адрес веб-сервера:} {ftp://user:pass@website.com/chat.txt}
; Имя пользователя программы передачи файлов, пароль, домен, и
; имя файла должны быть введены в форму.

name: request-text/title "Введите имя:"
cls: does [prin "^(1B)[J"]
; "cls" теперь очищает экран (объясненный ранее ).

write/append webserver join now [
    ": " name " вошел." newline
]
; Строка выше пишет некоторый текст на web-server.
; "/append", добавляет к существующему файлу
; на web-server  (в противоположность стиранию).
; Используя "join", записывает на web-server - объединенное
; значение, {имя пользователя}, некоторый статический текст,
; дату и время.
forever [
    current-chat: read webserver
    ; читает сообщения, которые в настоящее время находятся на web-server,
    ; и назначает переменной слово "current-chat"
    cls ; очищает экран, используя слово, определенное выше
    print join "-------------------------------------------------" [
        newline { Вы вошли как: } name newline
        {Type "участок памяти" выбрать место для дискуссий.} newline
        {Type "блокировка" приостановить или блокировать ваш чат.} newline
        {Type "выход" закончить ваш чат.} newline
        {Type "очистить" удалить текущий чат.} newline
        {Press [ENTER] периодически обновлять дисплей.} newline
        "-------------------------------------------------" newline]
    ; отображает приветствие и некоторые команды

    print join " Текущий текст в чате: " [webserver newline]
    print current-chat

    sent-message: copy join name [
        " Вам говорят: " entered-text: ask " Вы говорите:  "
    ]
    switch/default entered-text [
        "выход"  [break]
                ; если пользователь выбрал "выход",
                ; остановит цикл с выходом из прогаммы
        "очистить" [
            if/else request-pass = ["пароль"] [
                write webserver ""] [alert trim {
                Вы должны ввести пароль администратора
                чтобы очистить участок памяти!}
```

```
            ]
        ]
                ; если пользователь выбрал "очистить", для стирания
                ; текущего текста в чате
                ; спросит пароль администратора.

        "участок памяти"  [
            write/append webserver join now [
                ": " name " вышел." newline]
            webserver: to-url request-text/title/default {Новый адрес веб-сервера:}
to-string webserver
            write/append webserver join now [
                ": " name " вошел." newline
            ]
        ]
        "блокировка" [
            alert trim {Программа сделает паузу в течение 5 секунд.
                Чтобы продолжить, необходимы правильные имя и пароль.}
            pause-time: now/time + 5
                        forever [if now/time = pause-time [
                ; ждем 5 секунд
                while [request-pass <> ["пароль"]] [
                    alert "Пароль неверный - доступ запрещен!"
                    ]
                break
                ]
            ]
            ; выйдет из программы после 5 секунд.
        ]
    ] [if entered-text <> "" [
            write/append webserver join sent-message [newline]]]
]
; при выходе из цикла "forever" из цикла выходят, сделаем следующее:
cls print "Досвидания!"
write/append webserver join now [
    ": " name " чат закрыт." newline]
ждите 1
```

Объем программы выполняется в пределах цикла **"forever"**, и использует условный оператор **"switch"**, чтобы решить, как ответить на пользовательский ввод. Это - классическая структура, которая может быть откорректирована, чтобы соответствовать множеству обобщенных ситуаций, в которых компьютер неоднократно ждет и отвечает на пользовательские запросы.

17.3 Использование цикла для обработки данных

Одно из самых важных приложений структур цикла должно пройти через списки данных. Шагая через элементы в блоке, циклы могут использоваться, чтобы обработать и выполнить действия для каждого элемента данных. Эта методика используется во всех типах программирования, и это - краеугольный камень способа, которым программисты пользуются в работе с таблицами данных. Поскольку много программ работает со списками данных, вы будете очень часто сталкиваться с ситуациями, которые требуют использования циклов. Пример ниже демонстрирует несколько путей, которыми вы будете пользоваться.

```
; Во-первых, маленькая пользовательская база данных определена. Это организовано
; в блочную конструкцию: "users" блок содержит 5
; блоков, для которых каждый содержит 5 элементов информации
; каждого пользователя. Пропуски в анкете представлены с пустыми котировками.
; Отметьте, что каждый блок разбит на две строки, так, чтобы
; они поместились на эту страницу (они не должны быть разбиты ):

users: [
    ["Саша" "Петя" "Москва"
        "Лубянка" "555-1234"]
    ["Паша" "Толик" "Новгород"
        "Заводская" "555-2345"]
```

```
    ["Маша" "Наташа" "Омск"
        "Набережная" "555-3456"]
    ["Гриша" "Миша" "Томск"
        "Советская" ""]
    ["Ира" "Соня" ""
        "" "555-5678"]
]

;  Эта программа не имеет GUI. Основа этого, консольная программа.

draw-line: does [loop 65 [prin "-"] print ""]

; Это не самый эффективный способ, ниже пример лучше.
;       a-line: copy []
;       loop 65 [append a-line "-"]
;       ; remove the spaces and turn it
;       ; into a string of characters:
;       a-line: trim to-string a-line
;       ; now you can print "a-line"
;       ; anywhere you need it:
;       print a-line
;
; Неэффективный код выше остается в этом примере чтобы
; показать, разницу работы с эффективным кодом.

; Следующая - маленькая функция, которая распечатывает все данные
; в базе данных. Это использует foreach цикл, чтобы циклически пройти
; каждый блок пользовательских данных, и затем печатает
; отображение каждого элемента в блоке.
print-all: does [
    foreach user users [
        draw-line
        print rejoin ["Пользователь:     " user /1 " " user /2]
        draw-line
        print rejoin ["Адресс:  " user /3 "  " user /4]
        print rejoin ["Телефон:    " user/5]
        print newline
    ]
]

; Следующий код непрерывно использует цикл forever,
; запрашивает выбор у пользователя. Использует несколько
; циклов foreach, чтобы выбрать информацию из базы данных, и
; условную структуру "switch", чтобы решить, как ответить
; на запрос пользователя:

forever [

    prin "^(1B)[J"  ; этот код очищает экран.
    print "Вот текущие пользователи в базе данных:^/"
    ; "^ /" в конце строки для перехода на новую строку.
    draw-line

    ; Теперь выводит список имен пользователей. foreach цикл
    ; используется, чтобы получить первое и последнее имя каждого
    ; пользователя в базе данных.

    foreach user users [prin rejoin [user/1 " " user/2 "  "]]
    print "" draw-line

    ; Теперь спросите у пользователя выбор:

    print "Введите имя пользователя ниже.^/"
    print "Выберите 'all' для законченного листинга базы данных."
    print "Нажмите [Enter] для выхода.^/"
    answer: ask { О каком человеке нужна информация? }
    print newline

    ; Теперь решите, что сделать с ответом пользователя:
```

```
    switch/default answer [
        ; Если они выбрали "all", выполните "print-all"
        ; функция определила ранее:

        "all" [print-all]

        ; Если нажата только [Enter] (""), выводит
        ; прощальное сообщение, и заканчивает программу.
        ; "ask" используется, чтобы отобразить сообщение, вместо
        ; "print". Это позволяет программе ждать пока пользователь
        ; не нажмет клавишу прежде, чем закончить программу:

        "" [ask "До свидания!  Нажмите [Enter] для выхода." quit]
        ; Если ни один из вариантов выше не был выбран,
        ; ниже выполняется блок значения по умолчанию
        ; (последняя часть структуры выключателя):

        ][

        ; Этот раздел начинается, с создания переменной "flag",
        ; которая используется, чтобы проследить существует ли
        ; пользователь в базе данных.
        ; Слово "found" показывает что имя пользователя еще не было найдено:

        found: false

        ; Затем, цикл foreach цикл шагает через каждый
        ; пользовательский блок в базе данных:

        foreach user users [

            ; Если введенное имя пользователя найдено в базе данных (или
            ; первое или фамилия), информация этого пользователя выводится
            ; на экран.

            if find rejoin [user/1 " " user/2] answer [
                draw-line
                print rejoin ["Пользователь:     " user/1 " " user/2]
                draw-line
                print rejoin ["Адресс:  " user/3 " " user/4]
                print rejoin ["Телефон:    " user/5]
                print newline
                found: true
            ]
        ]

        ; Если переменная "found"  - все еще ложь после выполнения
        ; цикла через всю пользовательскую базу данных, и имя
        ; пользователя не было найдено в базе данных.
        ; Выведем сообщение на экран:

        if found <> true [
            print "Данных нет!^/"]
    ]
    ask "Нажмите [ENTER] для выхода"
]
```

17.4 Эффекты для изображений

Следующее приложение создает GUI, загружает и отображает изображение из Интернета, позволяет применять эффекты к нему, и сохранять на жесткий диск.

```
REBOL [Title: ""]

effect-types: ["Invert" "Grayscale" "Emboss" "Blur" "Sharpen"
               "Flip 1x1" "Rotate 90" "Tint 83" "Contrast 66"
               "Luma 150" "None"]
```

```
; создает список эффектов, которые встроены в Rebol, и
; назначены переменному слову "effect-types" в блоке

do %/c/play_sound.r
; Строка выше импортирует простую функцию "play-sound"
; созданную ранее. Чтобы программа работала
; правильно файл play_sound.r  должен находится на C:\

image-url: to-url request-text/title/default {
    Введите url откуда использовать изображение:} trim {
    http://rebol.com/view/demos/palms.jpg}
; спрашивает у пользователя расположение изображения (со
; значением по умолчанию местоположения), и назначает это всё на слово "new-image"

gui: [
; Следующий код отображает меню программы, используя
; "choice" графический фрагмент кнопки (тип выбора меню
; встроенный к Rebol). Кнопка составляет 160 пикселей , и помещена в верхний,
крайний ; левый угол в GUI (0x0) используя встроенное слово "at".

    across
    ; горизонтально выравнивает все следующие графические
    ; фрагменты GUI.
    at 20x2 choice 160 tan trim {
        Сохранить изображение} "Показать сохраненное изображение" "Скачать новое"
trim {
            -------------} "Выход" [
        if value = "Сохранить изображение" [
            filename: to-file request-file/title/file/save trim {
                Сохранить как:} "Сохранить" %/c/effectedimage.png
           ; просит как сохранить файл,
           ; по умолчанию "c:\effectedimage.png"
            save/png filename to-image picture
            ; сохраняет на жесткий диск
        ]
        if value = "Показать сохраненное изображение" [
            view-filename: to-file request-file/title/file trim {
                Сохранить файл:} "Сохранить" filename
            view/new center-face layout [image load view-filename]
            ; читает выбранное изображение с жесткого диска
            ; и отображает это в новом окне GUI
        ]
        if value = "Скачать новое изображение" [
            new-image: load to-url request-text/title/default trim {
                Введите url нового изображения} trim {
                http://www.rebol.com/view/bay.jpg}
            ; просит местоположение нового изображения,
            ; и назначает это на слово "new-image"

            picture/image: new-image
            ; заменяет старое изображение новым
            show picture ; обновляет экран GUI
        ]
        if value = "-------------" [] ; ничего не делает
        if value = "Exit" [
            play-sound %/c/windows/media/tada.wav
            quit ; выход из программы
        ]
    ]

    choice tan "Информация" "О программе"
        [alert "Эффекты изображения - Copyright 2005, Nick Antonaccio"]
    ; простой блок "О программе"

    below
    ; вертикально выравнивает фрагменты виджета
    ; противоположность "across"

    space 5
    ; некоторые фрагменты виджета
```

```
    pad 2
    ; помещает 2 пикселя пустого пространства перед виджетом
    box 550x1 white
    ; чертит линию 550 пикселей шириной, 1 пиксель высотой
    ; (декоративный разделитель)
    pad 10
    ; помещает некоторое пространство между виджетами
    vh1 " Двойной щелчок на каждом эффекте в списке справа:"
    ; большой заголовок текста для GUI
    return
    ; усовершенствования к следующей строке в GUI
    across

    picture: image load image-url
    ; вводит изображение в начале программы,
    ; и дает ему метку

    text-list data effect-types [
        current-effect: to-string value
        picture/effect: to-block form current-effect
        show picture
    ]

    ; Код выше создает виджет текстового списка и назначает блок
    ; действий к нему, которые выполняются всякий раз, когда
    ; пользователь нажимает на список. Блок действий выровнен
    ; и каждое действие помещено в отдельную строку для легкости
    ; прочтения кода.
    ; Первая строка назначает слову "current-effect" значение
    ; которое пользователь выбрал от списка. Вторая строка
    ; применяет этот эффект к изображению.
    ; Третья строка отображает недавно обработанное изображение.
]

view/options center-face layout gui [no-title]
```

17.5 Игра мозаика

Вот простой пример, который осуществляет GUI версию классической мозаичной игры в 8 строках кода. Поле содержит 15 подвижных частей и 1 пустое пространство.

```
Rebol [Title: "Мозаика"]

gui: [

    ; Определите основные параметры схемы размещения. "origin 0x0"
    ; запускает схему размещения в верхнем левом углу GUI.
    ; "across" располагает рядом друг с другом виджеты:

    origin 0x0 space 0x0 across

    ; Слово "style" ниже позволяет вам переопределять
    ; появление и характеристики любого действия  встроенного в
    ; виджет. Раздел ниже создает недавно определенный стиль
    ; кнопки "piece", с блоком действий, которые меняют позицию
    ; текущей кнопки. Эти действия работают всякий раз, когда одна
    ; из кнопок нажата.

    style piece button 60x60 [

        ; Строка ниже выясняет, нажата кнопка смежено с пустым
        ; пространством или нет. "offset" содержит позицию данного
        ; графический фрагмент. Слово "face" используется, чтобы
        ; обратиться к нажимаемому в настоящее время фрагменту.

        if not find [0x60 60x0 0x-60 -60x0]
            face/offset - empty/offset [exit]
```

```
        temp: face/offset

        face/offset: empty/offset

        empty/offset: temp
    ]

    ; Строки ниже располагает кнопки стиля "piece" на
    ; экран GUI.

    piece "1"   piece "2"   piece "3"   piece "4" return
    piece "5"   piece "6"   piece "7"   piece "8" return
    piece "9"   piece "10"  piece "11"  piece "12" return
    piece "13"  piece "14"  piece "15"
    empty: piece 200.200.200 edge [size: 0]
]

; Отображает целый блок GUI:

view center-face layout gui
```

17.6 Гитарные аккорды

Пятый пример - программа, которая создает, сохраняет, и печатает коллекции гитарных аккордов. Это демонстрирует полезный пример, и методику манипуляции GUI:

```
Rebol [Title: "Гитарные аккорды"]

; загрузим внедренные изображения:

fretboard: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAACXBIWXMAAAsTAAAL
EwEAmpwYAAAA2UlEQVR4nO3YQQqDQBAF0XTIwXtuNjfrLITs0rowGqbqbRWxEEL+
RFU9wJ53v8DN7Gezn81+NvvZXv3liLjmPX6n/4NL//72s9l/QGbWd5m53dbc8/kR
uv5RJ/QvzH42+9nsZ7OfzX62nfOPzZzzyNUxxh8+qhfVHo94/rM49y+b/Wz2s9nP
Zj+b/WzuX/cvmfuXzX42+9nsZ7OfzX4296/7l8z9y2Y/m/1s9rPZz2Y/m/vX/Uvm
/mWzn81+NvvZ7Gezn8396/4l2/n+y6N/f/vZ7Gezn81+tjenRWXD3TC8nAAAAABJ
RU5ErkJggg==
}

barimage: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAEoAAAAFCAIAAABtvO2fAAAACXBIWXMAAAsTAAAL
EwEAmpwYAAAAHElEQVR4nGNsaGhgGL6AaaAdQFsw6r2hDIa59wCf/AGKgzU3RwAA
AABJRU5ErkJggg==
}

dot: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAAoAAAAKCAIAAAACUFjqAAAACXBIWXMAAAsTAAAL
EwEAmpwYAAAAFElEQVR4nGNsaGhgwA2Y8MiNYGkA22EBlPG3fjQAAAAASUVORK5C
YII=
}

; Дизайн Gui:

; Подпрограмма ниже была скопирована с
; http://rebol.com/how-to/feel.html

movestyle: [
    engage: func [face action event] [
        if action = 'down [
            face/data: event/offset
            remove find face/parent-face/pane face
            append face/parent-face/pane face
        ]
        if find [over away] action [
            face/offset: face/offset + event/offset - face/data
```

```
        ]
        show face
    ]
]

gui: [
    backdrop white
        ; делает белый фон GUI
    currentfretboard: image fretboard 255x300
    currentbar: image barimage 240x15 feel movestyle
    text "КОМАНДЫ:" underline
    text "Перетащите точки и другие графические фрагменты на поле."
    across
    text "Измените размеры поля:"
    tab
    rotary "255x300" "170x200" "85x100" [
        currentfretboard/size: to-pair value show currentfretboard
        switch value [
            "255x300" [currentbar/size: 240x15 show currentbar]
            "170x200" [currentbar/size: 160x10 show currentbar]
            "85x100" [currentbar/size: 80x5 show currentbar]
        ]
    ]

    return
    button "Сохранить" [
        filename: to-file request-file/save/file "1.png"
        save/png filename to-image currentfretboard
    ]

    tab

    button "Печать" [
        filelist:
            sort request-file/title "Выберите изображение, чтобы напечатать:"
        html: copy "<html><body>"
        foreach file filelist [
            append html rejoin [
                {<img src="file:///} to-local-file file {">}
            ]
        ]
        append html [</body></html>]
        write %chords.html trim/auto html
        browse %chords.html
    ]
]

loop 50 [append gui [at 275x50 image dot 30x30 feel movestyle]]
loop 50 [append gui [at 275x100 image dot 20x20 feel movestyle]]
loop 50 [append gui [at 275x140 image dot 10x10 feel movestyle]]

loop 6 [append gui [at 273x165 text "X" bold feel movestyle]]
loop 6 [append gui [at 273x185 text "O" bold feel movestyle]]

view layout gui
```

17.7 База данных

Этот пример - простая программа базы данных с GUI. Сохраняет в память, ищет, изменяет, организовывает и отображает данные. Этот тип приложения распространен в любой ситуации, где нужная информация должна быть организована. Хотя этот пример обрабатывает клиентскую информацию списка, методики, которые здесь демонстрируются, могут быть непосредственно применены к другим типам приложений - материальные ценности, списки клиентов и продавцов, отчеты платежной ведомости и налога, мультимедийные коллекции, альбомы, фотографии, и т.д. Эта программа использует **listview** модуль, найденный в http://www.hmkdesign.dk/rebol/list-view/list-view.r, который сжат и находится в пределах скрипта. **listview** модуль делает всю основную работу отображения, сортировки, фильтрации, изменения, и управления данными, со

знакомым пользовательским интерфейсом. Документация доступна в http://www.hmkdesign.dk/rebol/list-view/list-view.html. Изучение использования **listview** является полезным, если вы намереваетесь сделать программный продукт, который использует информационное управление.

Программа ниже создает заданную по умолчанию базу данных. Есть дополнительная функция, чтобы препятствовать, программе закрываться случайно.

```
Rebol [title: "Моя база данных"]

evt-close: func [face event] [
    either event/type = 'close [
        inform layout [
            across
            Button "Сохранить" [
                ; при нажатии кнопки, резервные данные сохраняются
                ; в автоматически созданном файле:
                backup-file: to-file rejoin ["backup_" now/date]
                write backup-file read %database.db
                save %database.db theview/data quit
            ]
            Button "Не сохранять" [quit]
            Button "Отмена" [hide-popup]
        ] none ] [
        event
    ]
]
insert-event-func :evt-close

; Код ниже - модуль list-view.r в сжатом формате.

do decompress #{
789CCD3D6B73E3C871DFF52BE6369592F67629905A9F73A6B2A7B2EFECD8E557
2AE738A962215510311421810400D8092B8A9FC97FCD474F7BCBA07038ADA3B3B
E6D56909CCABA7BBA75FD333FCB75FFEE28FBF538B33A5FE54F5B59EAB377FFE
CD77EA77BFF9FE4F933FFFE697FFA17E552CF51B28FD558585FF58575D3F79AC
F4D3650B2F7FBEEFD74D3B578B37BFD6DBB67A50BFAF1E0A5DABDFB6504D6F3B
BD7D9343B56F9BDDA1ADEED63DF47E359D7EF55EC1DF9FAA89FAF5EF7FABBED3

5D75B7C521BE6D75D1EB728EA55F4D665793AB9FC1DBEFE01DBDFAE9643A9B5C
CDE0D59F75DB55CD76AEA697D3CBABAFE1CDEFAA250E3757FF0D0F4AFDE2FBEF
D4C5D3D3D365B383D7CDBE5DEACBA6BDCB6A53ADCB6EBB72621F2E77EBDD5B6A
F5EF9D5645AF0E505F354F5B059378B88492FF81FFFF75DFEE1A1AE05FF456B7
45AD76E68D428C2042D453D5AFD5A6D81ED40AE6B16F75A7564DABF650A7DA2A
C0EA2576F487A6F760FE695D411D40AC827F8BC7A2AA8BDB1A619853F1BAEF77
F32CC379AC370F25E1E9B27CC85A7DDBD499A74426690218D39B4615DB52F5BA
EB9705CCF787765E428FAEF36609DDB59A7509B3DB140FBA6C965738E30DE0F0
F3A7D03FF79FD972DD6FEACF6B3A013AF44D7BF8FCC17D0F1608A4F4AFCDAB39
2D2EA5BED740F71FDC717E969F9D75FDA1AE3EE96C53C02A6BA97F6AB0AA740D
CBE7177FFC4FC38C66E00EAAC25279BE9AD2A32EEFE01109465FD5C294CF9E67
4AAF567A098BF4BCBAD58FB08C974D8DABFBEA27B0CACCFF3975317C4FAF57CD
161A6F9BADA6C75DD1167620FCAA164F6DB1BB99AB555177DA74B4AA8BBB0E10
4480ABBEB8BDD5A56A352C9EAD6AB693FD76D52CF71D30D86EDFDB165AD7B6D3
65FF3CE9F5739FE9B2EAD56D050C6F26ACA08BB22D9E60A8FD7609BD830853C5
B257B05E735745A96AA58ABA560B605FAA714ED3EAD46DDD2C1F6EE85D665EE5
2AB452AC60AE76D5F281D7343DFDF337808B65514FF0C9B7CCCFE4BFA5EE09DB
1E614A358FBA152FF4F6AE4072451301EA6C7B3695FA71AEEE748F2B713537F0
00CAA1CE24FEAECED9836FDF01BB2CD7D4359F69895290BF008080AF80E5F45F
F6456D70A4904DBFCCC284D562BFAD51DA1375F2A8036413A069B66D26DDBA79
A22EF2A886E9710980027A9AD5AAD3D05733A117664C4240668AA2C6BE57F69E
0F80388E60024ED836FDC44E8A8DFEF2E083D999BE6C276B40834145D7172D54
4EBE9EF321635CA841577A5B8A16F3410BF56AA4A5D12611C7BF3FE843CC18B0
0889E06C2CCF93C478C05DA0B463C41BA0B0BF8FEA1FDEFCD7E4CD00A382E506
70C3B2D7CFC089B8FC138CBFD5C349E1A78695D5AF6F469AC420E05250AB0C06
A9BAF584A69A980DA068A42148B663AD4E40B9141E3962D5C93F75DEE97A7566
CA480FE0DB97D48011D62448F1AB9FB2D50633608802D6BF91460A358EB6DF0B
6043B0BDCE6BBDEA9D2AB8AD8BE5C35900F015B29F8404BDE95B90726801BA8A
F4B62DB64102EE8AAA9D29FC7BA5B21A458E6AD50EDE5C0539B39BC1C8801F56
D7955C214CCF891298C2128C54B5707846CB0D2ACE32EC1BFECC184F60D901CB
```

AEB0EC0ACB2A3024617DF7601201404537C1CEA18383A79B6A197A56FBBAB654
62EA72ABD9235374F8D50F6F5404D307CD63EB50B10245404A4EFD78AA019628
8EC019BB7ECC108AC952FB59505F7E56AC2E6958D7A06D9EEC0AC065411A3B51
543FF22500E353E78139A2D579ADBE47D1D5374DADFA6A37B27AAEDD572B4D08
AEF056D124091898172C0754E1386C59F4C5B5AA9BED1DC8DD667FB70E6D72D1
DEB49D23136BFE7E25D06DBB9F87CE79DD6BB009EAE26015231FC9F58EEC11DB
11A336422FF9023919987355FEC8EC615029308116D404451108F5B0922D4780
83B440003FAA733230ECF7A2063F039E39F1A5FDB4906C01E3A04C4517E48678
6C5D959AC46CCC3F25C377C43DB05AE67EBD9EA64B18ED784F82B194F80029B2
6E5DAD62AD4BA800C0417E832538D934A5464C6CF6755FA991F7B852621BC154
45C64A16401FBA9C80A4DE6FA4EA19DA66D0C0CA5B12876047EF61C2D63C8C20
1AEA3A03F86258A01C8A13255285C31868DA7878BBC41CC6E76B3E40D3C1FB7C
685779741E8518A47B18EEE8349C49319842B2769A8992A0C6CF4EC1396265B3
F0F52AAE6B55DE297523E22FE2719D92EC1A5075963586988BA8899527F48E34
E66BAA5F0DAAC7101DD3D2BC0E703294B6465727C8C1D5B7C302D3E202AE6C7E
1892ECB845878A0298B2AB6EABBAEA0FA0AB9461695528E43F80B05096B76F01
BFBAEFABED9D0279026F9BB6ACB6A0F8B0D527DD36D09769F5D4ECC1695E178F
9A6A3E633B57E9BDEAF6E8D07560F0FD945AD8EEA3468774A30FCF5368D46378
CA34D075B5B15040ABAD06371DF1EA56E135859B6E8B4EDB89811EEF804F6A5D
1FC09167B3E8A82621B8734380BAC3316E01E5A0C43072A61E4D84EFF2F2122A
B560FA57184E6BF5535B010CCD8AC020610CCF6D417643DBEC016DB1A9FF1253
0FC58BF5EA8792084898CD86CD9BBA9C5844CC5F12C5B1F21F679A11971C3FC6
2D77961559FC43AB8ABF068B2A9EB6D3B6A11F7C33D2575C34E82F02DD982FE9
6234188CC2DFEFD4A2450F83D011A935F20CFBAC6CF6A0DE27CB1A293270DA53
96C6005B29854CE4307D53D77C76290FCE2162A40914BD20A89C99B42A85D992
1BF50FF81AD82D273A82F41759095C3BE32E2146A7DC1B442E9EAB5FFDFCDB5F
7277705D8238873F93DB7E6BBEAC2A30DFFDD3B269B7DA96D65D4F7F262B0C2F
764B68A94BEEBB18A7F1C314BD4B034659B5FD01FD38E8D41AC5F89A50C71A8E
0423A7CF53E759CE7E32BDB4FFC7F14933F96B0A21BC574D59BEB7D2E73D88A2
E5C31D08846D6986A5E01C389B2C64A9AEAEE0DF0FF0FFD554CDBE867FA730CA
D7533EA20877960D88A08535F3705E18B76C3BEB037760A417E1C1B4EC76C512
24F9C4F600D2AE2D6D0D2A47D181C47322C60AA9C42B60C06AA9DDD3B2814712
B27327D1B02B29AF4C90883FAF7551A2DFE69E3D19B0F15355F66B78E9FEDD3D
FB57A21E460BE09DFD2714451EA12FE80F3BCD2B3A478EFBF8655BDCF1E755D5
33A601866BEA1ABA2778E6C8E293B536FB49369A0153F2D0AE2786E9A69C04C4
533C9EE011ED31081A0698CF2B8181562013C03D300EB66B99C38F9C402B9A55
5C4FCC3C78BD60557828E815DAD748DAA25B824B087FB60DBEB735820D8E753A
985DADADED8D7641B06B0DB083568E1179479E0CBE5F7AB0668A2D78B4BEA7AF
87EB1B57406818E60012C01802AE529899E5830AC39FBD9E60012337185DDB3E
3069C096779E85C6F55F3CA64D1D74970DC5AB12D06B14B7571782DB802ACDD3
0D0301390CB51217713D182E456BF8DF8AACEED0F57A93B9A2CC44417EFC685A
B1240368821EBE67946AB323BC19B7DFB8B6CDED3DE0FC8BDC3976A6A67E3635
8D00B390D1046CFDC00F2BD68C948799C06351EFB5FD3E658BC58DDE7DCA9DE8
EE3EA9FD0EA071513EC26D2C3C0DC2A109AD0858CCADCE1764368096C9D9F085
E5A028463457914982032455B30B8E0F43B7731545652DB1A901ED019168BE71
B063549F74387ED19B1DA837921F14EE777B466635C05B9884D54EDBE2B1BA33
162A228BA4F699957250DBF0AE208D0B789B31409881890B4BA5459F60E1D97D
C6E2210497F351C3CA083ECB2CC4210DF2199D77AD7E044FE74E7350E8253C67
DD43B523CA077E60CD8C394B3CC06AD21F1B7B22D670ABE1D3DC4D2E5426BEC9
171EC2E45CFC4C8FE286194CE8F3CE5C733581A1830565D5368260FC4EE674BA
165B7D87DE4E34E724FEB6A0EFFE8E118181021B9948CCF39DC40C6B6807409F
F8E638AE60C94B039B0CA15091DBB1B2E2098460E025CDE0117AC4FCEC8834CE
CFA4B35F5A8C0CC085503C6C62FEFDA9241A92274D9131B40A1C48CF00DDF63E
E84A3133E79378960739B64293F9668C18E8AE582CF1419CA524442522E5C637
74C2D020011110C942B22C3986AC85967AF7E68D159F5444115CBB3B445B1F45
DF23CA7153268496785CA971E6B9EB5CA24720250CE2E722DA5AE3CDC7C5C40B
D0D467D7185AA76885896C204B6816F2C0221E1B6A566AB9060B887044639890
09E86B1B13DAB8584EB7D3CB6A552DCF184060FE16308752AF0A30032795AAD4
93B23C073A1BDEA9BB8961B8208E1273F17112CBC481A45110DDF536F76C0C64
DA156D87012BC7ED92F7597E83EFDF75A3BE5153B138AE0943B4606DD8E7B6EA
3168B5D9C36AB8B586EFAEAE2815E9F6A0BE668DCD8C9D0B65FE1502C8E3C95A
74A6EF2FD48587E79DFA1AD44778CE32F5F5DBB7AC9327675B9ACF7EDB5771C0
C186172DFE9F4419347F8241645CE909D0E0464C8A3FC4CB1AD882B2AF688582
58DB217291BD040676BAE8D5BDA74E620FC42DAE0D2C422ACFE6F7A2821FCB32
5E0992B332CB371ECF8FB8DD6F025117B80B43CBD070C05D36C772B76D5A61ED
28A4042D2AF5D1E16C9065E11C38CFE7D76A59EBA28546CB624F296FEA767FF7
45DC8E057A39D72B3961117A11159964B6EBC46B22F9C28A3C63E1C29C6F982F
61340FD9E4B45A2B34B0D9AE2D17246CFD71A33320049D7A87F42A22339FEE22
57D508AC8AC61F59F9CCED65A30201B878E5110BDEC2D77F990F39AC088EDF61
B01C99CD2B68EEDD767833B40622D43DF72D98F699A1309F8EF74D69E311FC6B

0E08D62033C180F121AD8C7DAD0CEC608DE22EAEAD3EA82B8FEF0017D79CC0DD
DA7182D0CDE958C3D0B33E3781015BEAF9CAC9F5A6D5C572AD7618D1B3992DB0
AE761939A038771B0004D599B9E01A59330EA1C695F3DD4596965D18C69F264E
76695D47795918440CF56E071577454DFC30693BE5AED4FAF061DAF9822FC7C0
AB2BD1E7C0D4A0CDFE68044669320DA6ECC549B60AB7DB22B2707EADB6A8E131
C6C22C9484C0712D6C7A99CB9E5B10FF2C5B698EF513670C58F73C9A550A7801
35233B6754D7AB60D5CFB0DC1203B0488065A5AECE07A66A6C63FB67261E1306
32DFB5065A9F47B2292E268BE27829C622BC07C2B229B8B91D7AA8A5B01C1AD6
BE2AF11ABA6C5D0D56C74C0DCA2ED0B18542E3BB7917E62DD4F60F9CF1FD4BF5
CF293770C0E7173878D4390BFFA4C761E2D7B063C213B125B19C5BB61925EBDE
B1E132EBBD8761596D13B2B5A222D40061163A584CD9EAC7C964EA8255655378
EB40A7D5C40260D65F212DD56AE061FD888934D269B983395525F33E141E0950
6BD5AACCA833B1EC2CD020F25ACA44B6CF6BDC47F05E86CC51764ACB7A23CE2D
C128412E144988B1D9B622BE27CC1EAB149652693371298D060B028B88A9055A
F630F7A706B733845AE425C6BB37BEC052A62E79E57E6EF7334838F019216ECD
EEA009AA50DA6E46CE13A2296B8B27836DCC1DED5C90A562C94D481943211548
83EDA1692457AA32BDCCC7177F3ED45B0407C70560DFBA15946492D35E3BBC34
FBCBC45A3030BCC817FC4D6A870FE506470771B3C2A473B4C9B7C5C687992AB5
020F9E6121E82E2C988B7C2CA9908D491645336C151C49088B39EDB954E5B303
E0251F0864E0C0CF21B51C99E88620B89EB9A5684CBE2A9B2FCDA2476A8477C3
DDEDDAD98DACC947DE8B410646FC4FDA755DFC75A7E7E7C418213929371356EF
B55389498BCD5934ABB2769E5B81F9629874189628EEF839A6C4EFE873F4FA4E
B75F2894015FE4E935FB43576BE40E792916BDE7ABD487851856088B769E045D
550A0C5A0CE14C6E949BDE88D783C539E1E0F8663D0E47C2D4DA39E0B3B6434B
874D84362CC54C705B1C98ADB8055F6B7F4B4E8E15B62673890285F495B58141
61C8A26D8B0328A7AAAF800005A6440D5ECD22638D32BF5CF6D884F1F8C0EB34
C30B0C3798028E683610D87E600D5E415738BC7F31A317720551CBC0EBAC7168
C63C2E33865732DC9F357FF7DBEA2F6006D02AF26143E7DC1A86054C7B12F91C
3026EBB0EA541A089B625BEDF6F56093E7FA69ADB78ACE7850ECAE51C56EA781
7188B2A815BAF783F01625B35486951FB6CDD3F6CC420EFA85897F162EC06966
A5D63B91836AB7A4C82B256EA56F4E397F2114987094A9DE95BACACD300C7B16
0B0E0AEC273FB222A8DC34C4D08C33508C056769E31345507C7187D26E72922D
C2359F053A48152F70722660F84B436C332D2F78CCAC5E237D52BA32B299B266
5B9BE88430503B3C1185275C089F0BB40878166FBC8FC71640E5E271F0850F51
F4CE584889AB53064D1C476384907E2292C3BE49C84A41282BD88241CC082199
96A68A35CC04F19B353CDDE4F095E5DE882BECAE38AD2561240A52A38F4CE61B
2333C321D9B7AFA5D57124F9436DCC4B37091461DF44B96DEE0446B949309CA8
C0AD014F4C8BCD084B5F07B1834F82E79C7F4CFD10883E7DF57D96A6776ADE33
BF0181F1FD0B7243003E60CA48A362266BA57C01E60847E39106626E73F58A45
3304C546390C3C0696FF5DF00A160EB93D6525E91A8B1202D1F0FCE974E99B5D
208EEA37BBD74BC401950C70B188526C41052B07C70FA20EC6B72A61C41A0B3C
1897041561973F55800E99A1E2332D7E54996769312EF3E88899957C562187DC
1C2FAEB1D28D211A530586299DE3EF2403F504FC69867E993F5FCDA77C4E824F
47E660D394D3E7426C6ADA8277EA4D2A3735675AC5873C38BF28E3418B7E5C7B
1749A0FAA1813D8B229A381B5676CDCE819C50DB198B2F56CD8758E5CE92B7DD
68B7E36FBC729927E5162FB70703778948053382O9E6E15249AC6CBEB6DD58AC
7020D85919559FCDFD1439432797A544BAD3DDE9840E11CF922E1057C3C1BDB3
21AA45225665F3F386F915CC8BE3397C30A0B75D9CCD5236A28AD844BC568F55
B707C8305B2D913CE113C8BD7EB65C8A94B2AC22AD2F80CDC714831B50BAD3E1
D9811F2ABF007320034793D21BD155C90E6FD597917AF2F5CD56A2A5CEB8AB18
9DAD10994700076E56A177679289F91BE33E1063924CB299D150946012B266A0
35E74F244A2C6C16836C5FE34F4B2843BFE65E0A02C2C12642FCA307C04238CD
4891A6CEE3B15302C41F6DB44103494D966EECBC94B2A76305FCB02CDBE37D91
3A7E5222E627D57B847D778C34428A3A1727829920F5960C3757CD5405C924B0
3C1CE597875DB9D52772C1CFAC4752E1166AE9F6932830ADB4DFD190F9DE5671
82392C93E473B99C43028CF062995D938A4A0524CACCFEA32EA5B29D9BDAB380
B854969E9066C22774E17E178E4FE47F380F8345FB7DB0AC6F2656F5A8325F9C
FF61BFB9D56D8EAF29C45FE6A22711715C48F8C73D4F8FD95826C4E717068723
5C0F03B2C930B71F405662684874408E45FC7214027FC0A772795282BD727429
C4F18263F2C5C34B67278008E608050D7E82AA3A37572A78E7909F1C89B33B2C
EF8D4AE3646249A8C73D4F3B10CF32E1D62C71E1EA98DC7701EE792296B452C9
A18CC8853629EEB25FAE69DF05A551A7F63B9EB04467A0CC150B8538CB6A8E25
893CB5E8FA0CFC981B1B245AF3D0353B50951AC31E02E015B99149679FAEA697
F67F26C5DB08B2D13354B3A367A8D8112A09F628C429AC9849B83C92085B5702
5D43608E4D6AB53201EB10DCC413294F32B8ACB7E6100521109F9ED6783E74D7
D487BB66AB3E3C7FA5FEE979F613359B3D7F95AB555DA50C49B29EC879227E08
D728C834109F2ECBF2A1C82202E0D5479EB1131FABA75A0EEFCE619269480381
8C394994D9C5D99E58CD62062FEC8A7788E8A8D08BCDA691CC16FB3B1235DCED
C1E38011910677C888C32CEEC80B6344220FB3A246EE38210618BDD522CA2147
9D1BAE8148DE0171C269DAE86590586B963B45F45F0FB3A7D6E9E4293619BBD3

662E0A434C0A3E1AA49033C0792DC76E897A2E5B5F1CAA4A9E759747CC44963F
26C9B3D291E636E530D4CB1751C32102F00384B2E96A6EB91C3922363AF65866
9CFCA421102B3D7D2FD4E088821837003C02DDF124C3735CA06045C1431A4021
298EAEE3CF1A1F45C6622000C63016BF19EE47BB23E47FDDC5F312C5C74D5099
5278ECCE99780A8610A0EFB2E8CC71546F98DA673ED2F71F643A8AA9D9E4C853
3301D31355F20C88ADB248CC43E87D0B86BBFB6DECEE9FA3CA81DB0EFCD4F867
9810A1F9E75A129C978E98177F3BCD732A070EB8E35416F416BA4F0570A6BA5F
7B3380CE7EC7088A586F516F2F2DB6E309C8EE33B62CFEDF79946E327895C53F
9B82FD320D0C66E360AFE0A7976E6C93A41F879DB22729A849BD6058C408D373
7740DE2DC4936BD22D0E27A2C344E258524874619D2352946206D07B3FCC8769
B4BD65E03D2E6573BE89EE13F03EEB2ACA9D32EEB0BF92C0BF0F171760F254A2
D0A4BE7EA344BEEAF0AE04AC928CD971E20CA0727045E35A4F3631CA29033306
1043B9EB0D4400402C9191D06ADF4C4ABDAC36C06E33958DD44A98FCEC0208B2
3CA2595EAB6F8B7A89592FDA5CE3B36B9B9D6EEB43387333081E5852456E9585
CE15632C510A430F0914CD150B4985065FAA0B8A4C3FE359DF656BC2D4CF6F8F
2EAC231435DC98202815847B30CC55BFF0658C6AF69A8C74D4262294B95380DF
33C0AF2BCA8FB0AFA5D00849F9ED1E434AFA38893BE06F73A42FAE00A9249629
513A28798BDBB96D9039C3C4E4725B2AC83B3B82F4CDCC18F13D6AAE2B41BDB8
F400A545C977885C06B5BD5AC35D28E086CD17C2D4C5E1FDFE4614008B2383D3
7CE1A77AC8D5D9708E073F9DCC9D29E4B790940DB27F8D4A0A99071CB34D666E
88C753769EA3D59B45FE4699FFDE29275C07F3F2B340D9CE261140C0847D363F
D9A078AE00A4F367F9D64CC46FC8FABE9E5F85D735CBE12FAB478CDE8BA5C8DA
BD65180AB75BF20E180CD978378E73176BC72C74BD0B5394160FF3D4E4049A24
C2317ACB639F830D82F9405E244F1B470B3B0A05B3B80D5E0410401B179C815F
0601D3D4360648A58AFAEE3E01F6B8F81C9182A94E16AC5D1215DDA744A4173A
0B44C6432701F10B77330F7B3761B5638327488D49DDD85D9CBA41A3059EC174
EA76E2EC4DD8A484118A43B3EFB31E2F09F2E72EECA52E74EB187D93A7EB1B7F
B9D2867ED801CCF5B6E93ABA4148AB29D83BA5BA70FC3475170B6587B75CC6BC
669F132F27F27288A784C76D4018B9D19EA5552C2E2E6C02E8297B9CEE8AF613
E04C812B6CC7AE557A1B351733879A5359211D3B245630141F1CEAF68047F09F
0740DD70B68700DCC8607593E0EC24E5DDD551C35DED3CA1B96D7A496C100CE8
CC7B4BDC21BE6284FE26DC3E4E3684B938CC98110BFA472C549A5DD43BBA71FC
22226134A4AFDECB532BC5FD8E81B8A82C360C643023664EB33C5DEFC6297039
391866AB31AEB7A42D6912F56C5FC6A43A27F66ED59038CF0941C96E28129917
D7E682CB0298A279C0AD6AB44F2F2F2F55853BFF7A0336F11A89D753B3ED3970
5B4919327D43BC62B1115C9F607132859B2538289260FCDE24016130D68F5E12
42671812DBBDB2B615D78B5C1C334EB88F99BBE8C8E510ECC51E073FABCE12C5
EC3484E1ED4EE281898867F5A6FC24DEC849445256F66A6D33F4E0F021BD662F
A354D944C8E9874FEA07D969C61EB6C684E5D0F064EF1032470F112D9E7F317D
66ED7028BCE77006915BE6369BC25FD5FD6AF56674A6DBAC948A8D25D9BD4279
71C96D35725AFB845FA69189444E8C599E993ECFD2DE5A226BDB8C274639418B
B2081D3F283E32197F4011955F42FBE6128A1CAF47F7911017FFD62C91468F0B
EC452E486DE4452C5CBC0B228ED80E73F030C9F0661E3CC79642328B9C4B562B
F5C55D96B1D2C45B356FF8A1DA4EE3B593FEB64B71F7A5B703F91D98D28CBB56
DD7EB7031292D4A72546DCCFC0323CCE226546B2D958D9D25DD5A11E14B8E094
CE4363A121B8B037ED90187FB257148338DF14DBE24E2BF30B01F67783FCE599
78B930ABBBA4DF1FB387663BB56A9B0D9DD1F189BAAD06389650B969F15663BA
D5A7061FC2A30E80999FE4118CA47C053E3672F646258E69870D032C7A27CE88
F35F352247801F8B26D78588CD6FB35E3A7D11AD5B42F05C3DC8BB0F5E6FF98C
5AB9C376E3762E7422564454780F168C1328713BE62D5F18A4E0C1FA2FA1C0A5
324E822D1F3576D9825E8724F6706C9D207EEF990A20240EB7B42C6E0D0FBF53
F7CE1B7A0790A406F04702FD28C221B3275B1D77A1F598DC3FBDF7A6D4C0FA1F
D47E69F7EF5E0A9141FBA389E1AC9A4D114F148D5DDD6C1B0E5978ACA64D74FD
F88A3628553EAA9897CCE7B4DD539BB23E3EAF0430C7414D6C8D8674F4D12EED
4C928D6D8EFCAB710F7DFE60AAA4CE469F363A8DF31934B5908FD2153FE96D79
FCF09BF1E4EF0354AF8021DDFFA91CC54E2AFCBDD2ECF55CFD778C7190A2FCCA
6E7B4D8379F0E64896A92B3CF91F374F8A6D160F39A60A53E2DB0B5CFB0554D8
F4F2672F4869BC77AD2EEE68F01BB09C283534D13968F987C1F505665493E87C
0AC2EF333A6D3156683610EF33F6E357DE547A318DD97CA2E3E02966F309E90F
6A48CFD13EC7BA1B76319ADC556DC0D0BCB1F31C51BF6E83DB7A4A36AF31C9A1
836447FEE9613D74B4F3767171EF3C2833B2D9AD51C0947114CF7D08507561AA
A72B9D2A901C51937958496CDF67C90C8DF039F2334A7CCC6491BB5A3E59C842
5847BAB8E8FD5D4DF16F28DE7BF9E3EDB5D9576FD1C41E41B45BDDE11ECCD181
D3F61A4D96FD289E885F0C71028BABC1DB9B306376873F37190FACF026E40FEA
CDE5E5E59B14100B391CFBB13E51EDA8CC89B236AC91CD2FED272BD7DADBEA5D
D499F77F702D8B3B9D7C9B68574FB91F8CE5E1597E5AC216B3E89559E3E9081E
17E52C5469F60747829532403A7AC82058FDEC22753301EB52F1CB9AA36E5E19
92F1B93FE4F2BA5CBB74842699DB4BC4A02C908BE46F05BF9C5A2E458BDB90F3
FB08C6F989B6120C1B5F444C11F8D89D3B21A12D31342697AD128ECE77708259
95915E83E18053BE8882F212744A328B21E730F0C4F1D9689C2A0505CCBAAEEC

```
A55A3CCF7CC10B6437831D0409ACA1E345FAE7CC1C835AE039ACA3BC1D4F182D
45BA50BAEAFD36722EB614019F21203A9029B266483B381C9954AB6BF3FBF12E
B7342D1106EE7733769BDDF806D6F1A5E6CE738CAC37897C374498E2EB174E1E
4FC9F49998D25D7DD8213D8A5ADBCBFEE4DE3A0703E4E387A86151DEEFBB7E98
BF11FF480BDA1FE2AA2096397218141EC766480A8DED26F6D3994E86452AF214
E41DC926FD31456044F590E6324279D795E49DA459586CC12D04AD547438E1A8
707848067FE8C8FEEF8ECA446D2EA6CF138C91316E01A6E22C10DB8D17B3E7E9
EB1AE0B190573598BC6E88B1132DFEABE334BF5DE95E881A2E62166568BC2EC3
5D1E3C58477B73E62F9A16738CDBDA63ABE6ADB928D59FEB8661CDB924CBFA89
7B1DF88657E2AE3FEA810F3308FCBB94499F3EC92BF39B56E2B38FB875257E31
96D241F19761DCC656986F7E969FFD1F598CCF767B840000
}

; Следующая условная оценка выясняет какие файл базы данных
; существуют. В противном случае создает файл с пустым блоком

if not exists? %database.db [write %database.db {[][]}]

; Теперь хранящиеся данные назначаются в переменное слово:

database: load %database.db

view center-face gui: layout [
    h4 { Чтобы ввести данные, дважды щелкните любую строку, и набирайте
непосредственно в колонках.}
    theview: list-view 800x200 with [
        data-columns: [Студент Учитель День Время Телефон
            Родители Возраст Платежи Переупорядочить Примечания]
        data: copy database
        tri-state-sort: false
        editable?: true
    ]
    across
    button "+ строка" [theview/insert-row]
    button "- строка" [
        if (to-string request-list "Правильно?"
                [да нет]) = "да" [
            theview/remove-row
        ]
    ]
    button "фильтр" [
        filter-text: request-text/title trim {
            Фильтр (оставьте пробел, чтобы обновить все данные):}
        if filter-text <> none [
            theview/filter-string: filter-text
            theview/update
        ]
    ]
    button "сохранить db" [
        backup-file: to-file rejoin ["backup_" now/date]
        write backup-file read %database.db
        save %database.db theview/data
    ]
]
```

Ниже упрощенная версия программы “Моя база данных” без встроенного **list-view.r**:

```
Rebol []


if not exists? %list-view.r [write %list-view.r read
    http://www.hmkdesign.dk/rebol/list-view/list-view.r
]
do %list-view.r
if not exists? %database.db [write %database.db {[][]}]
database: load %database.db
```

```
view center-face gui: layout [
    theview: list-view 800x200 with [
        data-columns: [Студент Учитель День Время Телефон
            Родители Возраст Платежи Переупорядочить Примечания]
        data: copy database
        tri-state-sort: false
        editable?: true
    ]
    across
    button "+ строка " [theview/insert-row]
    button "- строка " [theview/remove-row]
    button " фильтр" [
        filter-text: request-text/title trim {
            Фильтр (оставьте пробел, чтобы обновить все данные):}
        if filter-text <> none [
            theview/filter-string: filter-text
            theview/update
        ]
    ]
    button " сохранить db " [save %database.db theview/data]
]
```

Особенности манипуляции данных в примере выше допускаются импортированным модулем **list-view**. Это делает создание и управление базами данных, в Rebol достаточно легкой процедурой. Используем схему выше как основу, чтобы создавать графический интерфейс для любого типа Rebol приложения базы данных. Это отличный инструмент, чтобы начать разрабатывать программы, и работать с данными в любой сфере, где это необходимо.

Для получения дополнительной информации о том, как работать со списками данных, используя инструментальные средства Rebol, см. страницы ниже:

http://compkarori.com/vanilla/display/VID+list+style

http://compkarori.com/vanilla/display/list+examples

Ссылки ниже для примеров того, как организовать, и управляли данными, используя чистый синтаксис Rebol, и некоторые сторонние модули:

http://www.rebol.net/cookbook/recipes/0012.html

http://www.dobeash.com/it/rebdb/

http://softinnov.org/rebol/mysql.shtml

Есть ссылки к инструментальным средствам базы данных на **rebDB** странице. Понимание и способность использовать таких инструментальных средств - важная часть создания современных приложений всех типов.

17.8 Одноранговый сетевой менеджер

Этот пример позволяет двум пользователям соединяться непосредственно через порт сети TCP/IP, для обмена сообщениями. В отличие от чата FTP выше, этот пример не требует, чтобы сервер хранил или передавал данные. Текст посылают непосредственно между двумя компьютерами, через установленное сетевое подключение (в Интернете или локальной сети). Приложение может действовать как клиент или как сервер, в зависимости от выбора пользователя. Чтобы работать должным образом, у машины сервера должны быть выставленный адрес IP или открытый порт маршрутизатора и межсетевой защиты. Клиентская машина может быть расположена позади маршрутизатора или межсетевой защиты, без любых отправленных входящих портов. Операция программы может быть продемонстрирована на единственном компьютере. Для команд, см. документацию справки, включенную в код.

http://www.rebol.net/cookbook/recipes/0028.html См. для детализированного объяснения того, как Rebol открывается, соединяется, и отправляет данные через порты TCP/IP. Для информации о передаче двоичных файлов и данных непосредственно через

одноранговое подключение, см. http://www.rebol.net/cookbook/recipes/0058.html.

Основная информация о том, как конфигурировать маршрутизаторы и сетевые порты для использования в этом примере, см. http://portforward.com/. Этот тип конфигурации вне возможностей этого обучающего материала, но требуется для каждого вида однорангового ("p2p") сетевого приложения. Совместное использование файлов, многопользовательские сетевые игры, совместное использование веб-камеры, мгновенная передача сообщений, обслуживание сети, и т.д. Для еще более фундаментального объяснения о том, как сети работают, и на информацию о том, как конфигурировать типичную сетевую установку в MS Windows, см. http://com-pute.com/FreeTutorials/Other/NetworkBasics.html.

```
REBOL [Title: "Одноранговый сетевой менеджер"]

connected: false
; Переменная используется, чтобы узнать действительно ли
; две машины уже соединились.

insert-event-func closedown: func [face event] [
    either event/type = 'close [
        if connected [
            insert port trim {
                ************************************************
                ПОВТОРИТЕ СОЕДИНЕНИЕ.
                ПЕРЕЗАПУСТИТЕ ПРИЛОЖЕНИЕ,
                УДАЛЕННЫЙ ПОЛЬЗОВАТЕЛЬ ОТСОЕДЕНИЛСЯ.
                ************************************************
            }
            close port
            if mode/text = "Режим сервера" [close listen]
        ]
        quit
    ] [event]
]

view/new center-face gui: layout [
    across
    at 5x2  ; этот код устанавливает следующие элементы в GUI

    ; Текст ниже появляется как опция меню в верхнем левом углу GUI.
    ; Когда это нажато, текст, содержавшийся в области "display", сохраняется
    ; как файл.

    text bold "Сохранить чат" [
        filename: to-file request-file/title/file/save trim {
            Сохранить как:} "Сохранить" %/c/chat.txt
        write filename display/text
    ]

    ; Текст ниже - другая опция меню. Отображает IP адрес пользователя.
    ; Обращается к серверу, чтобы найти внешний адрес (whatismyip.com).
    ; Команда "parse" извлекает IP адрес из страницы.

    text bold "Поиск IP" [
        parse read http://whatismyip.com [
            thru <title> copy my-ip to </title>]
        parse my-ip [
            thru "-" copy stripped-ip to end]
        alert to-string rejoin [
            "Внешний: " trim/all stripped-ip "  "
            "Внутренний: " read join dns:// read dns://
        ]
    ]
    ; Текст ниже - третья опция меню. Отображает текст справки.

    text bold "Помощь" [
        alert {
                Введите IP адрес и число порта в полях.
                Чтобы использовать свой компьютер как сервер,
                нажмите кнопку "Режим сервера".
```

```
                Вы должны иметь выставленный IP адрес или
                открытый порт чтобы принимать входящие.
                Выберите "Режим клиента" если вы
                соединяетесь с чьим либо сервером.
                Щелкните "Соеденить", чтобы войти в чат.
                Чтобы проверить приложение на одной машине,
                откройте два приложения, оставьте набор IP
                в "localhost" для обоих. Заставьте одно
                приложение выполняться как
                сервер, а другое как клиент, затем попробуйте соединиться.
        }
    ]
    return

    ; Используемые ниже графические элементы, служат для ввода информации
    ; подключения.

    lab1: h3 "IP Адресс:"  IP: field "localhost" 102
    lab2: h3 "Порт:" portspec: field "9083" 50
    mode: rotary 120 "Режим клиента" "Режим сервера" [
        either value = "Режим клиента" [
            show lab1 show IP
        ][
            hide lab1 hide IP
        ]
    ]


    cnnct: button red "Соеденить" [
        hide cnnct
        either mode/text = "Режим клиента" [
            if error? try [
                port: open/direct/lines/no-wait to-url rejoin [
                    "tcp://" IP/text ":" portspec/text]
            ][alert "Сервер не отвечает." return]
        ][
            if error? try [
                listen: open/direct/lines/no-wait to-url rejoin [
                    "tcp://:" portspec/text]
                wait listen
                port: first listen
            ][alert "Сервер активен." return]
        ]

        focus entry
        connected: true

        forever [
            wait port
            foreach msg any [copy port []] [
                display/text: rejoin [
                    ">>>  "msg newline display/text]
            ]
            show display
        ]
    ]

    return  display: area "" 537x500
    return  entry: field 428  ; числа - размеры пикселей

    button "Отправить" [
        if connected [
            insert port entry/text focus entry
            display/text: rejoin [
                "<<<  " entry/text newline display/text]
            show display
        ]
    ]
]
```

```
show gui do-events  ; требуется потому что "/new" используется выше
```

Чтобы использовать этот пример, пользователь должен знать IP адрес сервера.

17.9 Пример r3D

Этот последний пример - простая демонстрационная версия того, как использовать r3D модуль **Andrew Hoadley's**, чтобы использовать трехмерную графику в ваших приложениях. Для более детализированных и интересных примеров кода, см. примеры в **http://www.rebol.net/demos/download.html**.

```
REBOL [
    Title: "Пример r3D"
    Notes: {Упрощенный вариант:
        http://www.rebol.net/demos/BF02D682713522AA/i-rebot.r }
]

; Этот раздел загружает сжатую r3D библиотеку:

do to-string decompress 64#{
eJzdPGtT28iWn+Nf0cOXsTMYkGXznL1bBMzEtQSnjPMAipqSpTboRpa8kmwwv37P
Od0tdethOzNTu1VLJUTqPu9XP5VR/8Pwmj002NhPA37KdmL7cqfBzhfpcxTD63no
xfyFfYwcL+Ar6PnK48SPwlNm7R3sHTQeG43GGbuI5qvYf3pOWdNtsc7BweEuMzER
6jOPZ36C2MxP2DOP+WTFnmInTLm3y6Yx5yyaMvfZiZ/4Lksj5oQrNgd+gBBNUscP
/fCJOcwFbgiZPgOZJJqmL07MGTBwQo85SRK5vgMkmRe5ixkPUydFllM/4Alrps+c
7dxKpJ0W8fG4EzA/ZNinutiLDyZYpCzmSRr7LtLYBR5+6AYLDwVRAIE/8yUPJEB2
SJDsIgE1UNhdNos8f4r/ctJtvpgEfvK8yzwfiU8WKTQm2OjyELCELvtRzBIeBEjD
B9lJ5VzCXdIX+MzRrqm0FHF+eY5mpjZgqekiDoEpJxwvAsshH+D6b+6m2IYI0ygI
ohdUz41Cz0etklN03hg6nUm05KSRcHYYpSCwEAN9Mc8dLLuSZwfkn3BpNuANZnY0
lWIUIEkhBnzwAfCZRzExLWq7R0J87LPb4dX42/mozwa37PNo+HVw2b+EOL2F951d
9m0w/jj8MmYAMTq/Gd+x4RU7v7lj/zW4udxl/e+fR/3bWzYcscGnz9eDPrQNbi6u
v1wObv5gHwAPmNwMx+x68GkwBrrjIfGU1Ab9W6T3qT+6+Aiv5x8G14Px3S67Goxv
kOwV0D1nn89H48HFl+vzEfv8ZfR5eNsHCS4F5ZvBzdUIePU/9W/Ge8Ab2lj/K7yw
24/n19fE7fwL6DAiKS+Gn+9Ggz8+jtnH4fVlHxo/9EG48w/XfcENVLu4Ph982mWX
55/O/wABgQ90DIHQiCCljN8+9qkJWJ7Dn4vxYHiDylwMb8YjeN0FXUfjDPXb4LYP
OTwa3KJlrkZD4IB2BQw03g0m/fDmpi/ooNVN5wAQvn+57efSXPbPr4HaLcqgA+9R
DWn/1R+qP1BDwpSCJra99syBrHrdi1mbiUeK0SXEOQRc4E9iJ141zpCrG3OoFZCH
0ynUoxDyYDWXqQaFKZlG8YxSO2k0kLDvAYyfrk5ZA2qm9gPVkB3kf40+bNP7S506
QGWnAsByi2Jgqs9BG38JJXu6CF0mpNm5EOo4TINQJsB6BT1LB6sAZvg88kFhjy1C
H0w3jWXNcB2oUs4OEfTYg8ddf+YEvzyyHQ/LCIJg2XJCyG9ZNYo021Bfwl9TqF4p
86m2vPE4Yr/8stN4NO1GLzH3FkDsQTeSVL/WkOq91l5rAZDKPmiH/z7SIIZmJZcH
YMAao6p+LHDCqMJK3w0rZVSgXZjWCSIwDRoKAvHVTwTWXQ3WXQlrpWHd12Ddl7De
JJa0eJWRv5eto7rv6g13vz5IH1WUJq4T1JmS+tYYUfRHYmyCaYLDpg5lL2Tm960M
uo7C3VbG3ZZCjaG/F2xTMtqdBlBp0ftNMW6YO45gFsK/GwY3jKoEhKrnBO6CwiYK
5QCegGKc1ElwiIYWN0ogsYMFTwgLmpG4eGDfqQ1ARKt4gNZSqCmRdfG155JqGll6
brYF51aNKTLRZJuOW4tQJ1CFPe8Me94V7fk3DXqXGfROM+hdZtA706Cyu6R81lKp
rdLMbCtCS0PftUqOuFtjyWqrmmVVGPLeMKSeaf+AHe8zO95rdrzP7HhfZcd7Xe/7
lqmJoW0OnZEuab7RPEU3bIWgno3IhDLE254fc1qVGHa9hAka/jxMgsj9gVXsK816
bIbTAViD4CCdoWIlmwY4lxf1bzFnBewvczVtmjkrMZGnZVlOgr+6fA6rDyeGeT0P
cLAHIQS9fSAD0/pXrI9sRb/f8LeqRPRySlO2EGdasCjiiE2dr1Wd+ObGUZLM4wh8
maLInkRYbYWAFhICNczZh2jdtzJS4lEIKZ+LgSFxOjlOR8PprMOxcxxbw7GrcbSY
2Fi85BwR4hQd1BYDmBEloulPBWL6/IwWaDZ5dMZTmEgyXEHa4HNYTePqDlwqgkIk
oCLzZxr9GUTRjz+dNKdmxJX+oyD0QEHXP2aBLFx5u5jALAccV8WmqMg/FVUQUiD0
/1ZUFbTYt346xooUOj8dcUUK9t+Nv1n3ddY1om5m5WEx61QGQMyTRZDmMyl8OxUb
Pg8CEFZk3HGf2YPDJsxl3iORzYR15nMOQ4TAzJ1gjnQOew8C7Fst9htrTsRLj15c
8XJCL54Es1s5biWhjk7oUCdkHRiUuq1qArZO4MggYBkEejUEujqBY4NAxyBwqBF4
NJR61EyuBhrw4dI2fZi7cLnGg9lcM+WzOTSZPiz6tc6xZb9Kcuscu9T8upSucemZ
rOzV66/mIGKNYu2/sCRiLzgVCXDAW8LiP5+WpPwZFoGghj+HXrE6wDKFVag5dRKo
m9LUfrhsv5wqkZti6SkVAcc1asNWwVggPRHZDNrZHtReAwoyqglTuojDqqhoO3Hs
rIzYWPIYN/2SQvGfddcFCiLx120yXkBqXDYlfSYriiCxH3Vfl7Qj9dxoBlWQt4vV
i9aqUP1TY6hk1ALzKBiNAg5+ZxbrvnYleHVtE1PUWbcdcJi9wa+n9Pk/NQYqEn3a
AoWxF8ffMGI8BAiKfpIb58RqQ0rnB2iSOPsPdqAniVJY28kyY98PfdyIxSGO9rKF
RWnniHaH/RgURDFWar+c5C25LtcFBrOiNjBIRjAnF2QyXRxJuKyCVaWCLDOG9BAp
jEIlFrTn6I5MbhVMRNlnnYyDPn6c5TmNiJApSz9aJIrfRDSH/BWkDTieLmiSCQNk
gx+b++4PPW4EY0lKyo7lpz7VaH+nMhq3GD7PWIYOcVoIykxalS0zqDOz/R78PcFn
SBqoJrP9Q3w5wF+Avo+tR/iCsBYCY+sxviCwdZitUaC0wMzxp+QGf4GfIFmxLGGV
97imS/SDjoywID9Fe3t75SWjQEcPizg9Vi6S01TsNhXGyopS56NA8YcUey+UXQd0
```

```
ICltINLdgN+r67fW45+w7fjbG/A38d+Ev0k/+7HstkpnZUGQO6zRbJIL94UsYp4l
WrrUcqS19KjluNVi7QyvU8KzS3iHCq9RQOtpQEclNKmfyc8qIR6XEE/K/OwaNF1w
6Q+TX7dGUAOxkyGWXJUROiwROqkjdGRI0CshSsNUyH5UJYIhhuDQKfnLLvmrW2H3
Tik+7FJ8IF6jgGeVDG+XDNGtiCurZHe7bAeTYbcGsVO2s8nRrpG0UzazybFXQpSC
VWDaBsvDEuZJnaw2sazI9LxAJxyPl7WkZzUlWmh6XFNWZN6rwKkG6mxBoq5uyiCv
6bXX4lpb8K2tt1vwXY+7XqNehXfKPmm6kTjxSFrFoiyn3PlYXVOlrW4pDa1eKQ9h
WVqRUN1S5huohxlqsVZbdimlrIqiVOBqlXGPy7gnFVztWsxuOa0KXLu1EnfLmUW4
m+q2Va5XlbR6FaXbqqhZlUr0NlZvaRSr7DG9ZEhZDqpkMUqLgitFk2UUuJM61JOy
cyyD6VFZXmWAssAnBtOjsnDKyOVR6MQMOqsMZ9cJfGAOV1ZZ4l4JVelgWvikXuKy
w0Diqmru4d7wzA/hmfrg/ZQ1Zf1RdUGMGHmkNGUFyvo7pX7b6LdL/V2jv9vK1664
xqd7Y8vIx+tfeNYhl3OcNnASusmRCStvAfBUrcDqNx9etVL3UL3h0ASg90iNtYob
DIWVXL4EVHsOXpTKnWRzR0XbMV3+zI6pEmlpkTuWarMT3jvivaPebfFut9TKTd/X
/uvSNKrkedAlAM+2dQk6rcqCooNYCsXKhK5FsTI92zpXq5XtdOobCXoFk3YQ+zKm
BX7eBcl/L5yYt+MoSiFCmkshl3KHkEo5Q6iJhsmcce555g0KaMCbKBGbgF4ObUVO
IClDPJvjPm6wiFW+K27XyXMaNlmoq02zBay2AMR9Fmd0jqnTpFJFP+Wz7TbnBaiV
7Wo5bOP2fFNslDgClUwhWiayxdifrtglUUdEpqWyg6OJuNvk/L81WPtnDfZJ7nGZ
BlOtQHiyYmGtuRx1oywz1y6jK73yRL/STKF+4P9/Zaj3LNxkmkva5zcNI9r+slnC
KGzT/be/aR8cwtbZyJ8y2vHdwz3fzzFexFOig+AkgZPi9n/KvR2lvSONAbTVgEjn
wf+o1VHwNXYXls9OV3+65OJG5wprYn5A6wh95d0FdNcqWrAXPwjYE4zQZvfEcX8Q
IdqPz2s/E1JL/zaZ2lFvaRZQjngwLmbJY4YKuGwkxJFgn0hWj2GkPIwPBLMOxJZk
6o82WD6ozTEs1mzKSotmh3FMnsYx8ziOyOi9mntl36PJsnCOuKxliatP7C7womZW
w+bsL1waxsu/hWvD4HQ/5HsxgxyHdzxG8MDxohnvCzdG1GJo8hLFgWcGqbhBa7bB
hOrf8sZMATZcOsmt/wYxgZPuyo1xmjyK2S0+dTz8oIAH4rcQACqPEz4FPFlXITIg
s1nIOxA79iIB8u172Wse0gKIvBcp1W9k7iKZiMiQILTbDLm4pzkF7a4CNLhRgN+d
5Nh2TvkTol+QNPopiym+YPKNpMpRz/Lr3OLCsjpEhBiKGLg6mTsunnfh7d5ldq9Z
qq6oEI2vgHrKCueggiueoOC/BTS8rgUWFyAdXSjs8cNpRJHo+GHCOoyCg0IyVAd+
u8xpEawscIkYf1r4SQeICdOTe4CfJjw1eA6AspD0Qi3arqD5RtLItBECbmEs5wlk
ZIvElxdftZjWTpYyS3W8sp3ySM6Ri6NJFqWELVKu441VazvxZ7igU6lA0mcK6wml
4kkvjGcwCs+htCdRLKxVTAzs2I+5CKY8q4zwV82aqHm9msIgg2wXEMeqHxbmmnln
zg9xvhot0gCvJyZ0Bw/vEdCHNVBz0mcnlEeX9LUMkmNFQxUPvH+FVhimcgFg6LX2
cJX/qHHH8U+nOnES7jHwIQlBn/5AuCQbmSGNNnLU+D02atB+VZao6c8M1dkEYG8C
6NaOg9lyuyoftrnKIEJtzQkmxdYSswsWy7BGXtpsmc7mFv2GV3fOAvy8LclvyFTV
6YxMzR0ZqgVSmIJnjYSlrzHEB18wAyRUH7LpVR3eI4kMf2mdiolbho7dUNNyiE4l
RO4yHOQrAOxGMfzyWpbjop1EwUjUyg0taBkQnSIEGDjnn4O689PyJT3NFWXIfO4I
TcUw3qc7QGo2ZQBIjVQlfgOPtaPYy/qlv09Z2zqAH+1iJ8zXaWvjX1lMPOTQ1PNo
wHZqYTslWLsW1jZgZQ7lcatQdItmuaQu1hSDPZvrlYv1LRVrI7my0YV7X7fOs6y+
bzN7gsJ/gYWfPaQL4F49p8KFkKrjSxGSIjkiKMKQIG60CEWovq3LVRgtIy/CVaHr
LmLcGFV1NB9wxEaprPlEXYhJj6esqanxHtYPPbmnOTWqkzSAnE9pJULNItTcggSH
xVzmvw1lw33mkLK+/KBW5CbnHn1JCstbXB1RI5TMAIqJnmfYLhJHJr4mspAjA1UC
S8A8eEwwKbww/m/MMljdn2octZEAZKdu9jved8pzU+iXDbeKZ91dJoosvPRHt5Hy
Flo2Trg2cwAz0FUgX36JYHDUI8uMlULWGSGohDOlV64TkUluMD+gEyrqlR/gzLpP
YhKBEuZSeqMqJQXLEkqV7NnCX6bOb8xJ2piPtMrFVay5UWtao+AuSqcshwxjYC5C
IRUPwOVgz9po12ZeDd5LxFYFklHhdQIZbGF9rW8aXUeO1x5eXZmX7KdTz0mdTZcf
E5mR4WL2lV7h4Uq19D0oxgQm83IxIxxwFXqV2lTFnr2yGSxE3nAm1MCth8Y7yAsp
Bbjh93+xX0FG6Hj3Tiyedz5QeNMWVQprCobd+J38jrZtQfOnh0z5d0rO04y0GEyh
XS50VLut2jlqkbd3649bloKysfMniGbzoHfZUKbG0mSGX56oFmqi2D1lPclozh1h
7NzKxT0s4X/hkPz+AdBIo7bcmGNNyhXlV+JRcwpa+JmtNtHBUrclrbctaHX0Kb80
huqz9TiGCIG4qZwpzNRtWYRZ1cCsNJi3Gpg3k9nvmcMAKvddgV0dlMmwDopY6s5X
2aNGv8w4KqOuongMGYWZJ2uiYVN9vieLLLVTndRTh+7q0uYirat25R1yHFDpaNLz
6PIrPGJdh1SUAkLSsqaKTk0YqHYdteH4riJihUL5+ZoFHv474QU/9TRQyH+SkhGm
+XV8sEohZpFOhXHKFVnfYNWKa1PFZTuLEjqk/Zm9Q9oyxEovZgaO2mti1IH/P8aI
T6CO+jilCZM0hiUHOJ3OnuhTYg9mUBPczXmOXnB/AUJl4sPIG/vgR7x1HkCcnQmi
iTpcWEAQ7bFbDhx5EL3AwIrRxl+dEPdAoikSw2iSXy1IZNy4Ryl1HtpdvX56xmwi/
bcfGZDHH/2kD5vugkNrP2cuCWPy/B9AYR4sncfebtsVxnANX0jBHWzdnjQYq1Za7
fvKi9lm2AKTXvNY+0BGDZeyS5+3tqo72+p46WupvJY9iR3t9T94hA07oKEILJv4h
qIoj6QuuHl8gZtXExVCc9dihKQ68d4yWDrQcFVqOmG202NByXGg5Zl2jpQstvUJL
D/jpLcdA57DQcljAsoC/XWixM1606vsfGt6vyUFIAAA=
}

; Этот раздел устанавливает некоторые значения по умолчанию:

base-rot: 150.0  ; степень основного значения вращения по умолчанию
; трансляция камеры и просмотр значений
cameraTransx: 300.0  cameraTransy: 300.0  cameraTransz: 300.0
cameraLookatx: 0.0  cameraLookaty: 0.0  cameraLookatz: 100.0

update: does [
    world: copy []  ; обновляет "мир"
    ; рабочее поле
    base-modelWorld: r3d-scale 100.0 100.0 50.0
```

```
    base-object: reduce [ cube-model base-modelWorld navy ]
    append world reduce [ base-object ]
    ; рабочий стенд
    stand-modelWorld: r3d-compose-m4 reduce [
        r3d-scale 35.0 35.0 150.0
        r3d-rotatez base-rot
        r3d-translate 0.0 0.0 52.0
    ]
    stand-object: reduce [ cube-model stand-modelWorld red ]
    append world reduce [ stand-object ]
    ; устанавливает камеру и создает для нее преобразования
    camera: r3d-position-object
        reduce [ cameratransx cameratransy cameratransz ]
        reduce [ cameraLookatx cameralookaty cameralookatz ]
        [ 0.0 0.0 1.0 ]
    ; Получаем матрицу проекта
    Projection: r3d-perspective 250.0
    RenderTriangles: render world camera Projection 400x360
]

RenderTriangles: copy []

out: layout [
    r3d-viewport: box 400x360 black effect [draw RenderTriangles]
    across
    style lab label 55 right yellow
    style lab2 label 40 right
    vh2 "Вращение:"  rbslider: slider 60x16 [
        base-rot: (value * 300.0) update show r3d-viewport ]
    return
    lab "Позиция"
    lab2 "x" cpos_x: slider 60x16 [cameratransx:
        (value * 600 - 300.0) update show r3d-viewport]
    lab2 "y" cpos_y: slider 60x16 [cameratransy:
        (value * 600 - 300.0) update show r3d-viewport]
    lab2 "z" cpos_z: slider 60x16 [cameratransz:
        (value * 600) update show r3d-viewport]
    return
    lab "Смотреть"
    lab2 "x" clook_x: slider 60x16 [cameraLookatx:
        (value * 400 - 200.0) update show r3d-viewport]
    lab2 "y" clook_y: slider 60x16 [cameraLookaty:
        (value * 400 - 200.0) update show r3d-viewport]
    lab2 "z" clook_z: slider 60x16 [cameraLookatz:
        (value * 200 ) update show r3d-viewport]
]

update
view out
```

18. Меню

Вот простой прототип, который может быть включен в ваши программы, чтобы обеспечить дополнительные функциональные возможности меню:

```
Rebol [Title: "Пример маленького меню"]

view center-face gui: layout/size [

    at 100x100 H3 "Вы выбрали:"
    display: field

    origin 2x2 space 5x5 across
    at -200x-200 file-menu: text-list "элемент1" "элемент2" "выход" [
        switch value [
            "элемент1" [
                face/offset: -200x-200
```

```
                show file-menu
                ; ПОМЕСТИТЕ СВОЙ КОД ЗДЕСЬ:
                set-face display "Файл / элемент1"
            ]
            "элемент2" [
                face/offset: -200x-200
                show file-menu
                ; ПОМЕСТИТЕ СВОЙ КОД ЗДЕСЬ:
                set-face display "Файл / элемент2"
            ]
            "Выход" [quit]
        ]
    ]

    at 2x2
    text bold "Файл" [
        either (face/offset + 0x22) = file-menu/offset [
            file-menu/offset: -200x-200
            show file-menu
        ][
            file-menu/offset: (face/offset + 0x22)
            show file-menu
        ]
    ]


    text bold "Помощь" [
        file-menu/offset: -200x-200
        show file-menu
        ; ПОМЕСТИТЕ СВОЙ КОД ЗДЕСЬ:
        set-face display "Помощь"
    ]
] 400x300
```

Здесь версия файла с расширенными возможностями:
http://www.rebol.org/library/scripts/menu-system.r. Вот минимальный пример:

```
do http://www.rebol.org/library/scripts/menu-system.r
menu-data: [edit: item "Меню" menu [item "Элемент1" item "Элемент2"]]
reb-style: [item style action [alert item/body/text]]

view center-face layout/size [
    at 2x2 menu-bar menu menu-data menu-style reb-style
] 400x500
```

Демонстрационная версия http://www.rebol.org/library/scripts/menu-system-demo.r
отличается продвинутыми особенностями модуля. Модуль меню был сжат и внедрен в код:

```
REBOL [Title: "Пример меню"]

; Следующая строка импортирует сжатый модуль меню.

do decompress #{
789CED7DED761B3792E8EFEBA740343F24AD4D5352EC24C3331E1F59A213258E
E548B233191EDE735A64536A9B6433ECA64566BD8F71DFF7A2AAF0D9F8E82645
27D9DD602672B31B28140A8542A150285C745F9CBF62BD078CA78BECE6B62C3A
8CFD27FE847492CF567378CDF606FBECE8E0E0293BB99D6745992553D69D16E9
58653D1E8F19662DD83C2DD2F9C774F8F881FA7A910E79A97976BD28B37CCA92
E9902D8A94655356E48BF920C537D7D93499AFD8289F4F8A47EC2E2B6F593EC7
7FF345C914AC493ECC46D92001488F58324FD92C9D4FB2B24C876C36CF3F6643
FE50DE2625FF937268E3717E974D6FD8209F0E3328546858507A92961D8DE9FF
AB225BB07C24B11CE4439E7F5194BC8D65C2B1871A92EBFC237C12A45290789A
E66536481FF16C59C1C61C28C032F08056DB48F29A07E3249BA4F3C7319478D5
06B5244ABCF5C30547D38315DB165A8C5A6D821BE683C5249D9689ECDB36EFB6
9C679AB34952A6F32C1917BA67B05F01BAD922ABB1AFD30C0B43A6693249013B
```

78D68DB9CDC7439E619AEB4C9C266561B5923788E0E7F38223B262D729301D6F
5ACED2E990BF4D81BF386293BC4C19118FC3E09033CEBD16AC11CF44E42AF251
79076C23F9B298A503E0470E2003769D03274E89278B02DBA6205D7D7776C92E
CF5F5EFD7C7CD165FCF9CDC5F9BBB3D3EE297BF10BFFD86527E76F7EB938FBF6
BB2BF6DDF9ABD3EEC5253B7E7DCADFBEBEBA387BF1F6EAFCE252C1DA39BEE410
7630C3F1EB5F58F75F6F2EBA9797ECFC829DFDF8E6D51907CA6BB9387E7D75D6
BD7CC4CE5E9FBC7A7B7AF6FADB478C0362AFCFAF74035F9DFD7876C5F35F9D3F
422CDCF2ECFC25FBB17B71F21DFF79FCE2ECD5D9D52F58F1CBB3ABD750E94B5E
AB1605ECCDF1C5D5D9C9DB57C717ECCDDB8B37E7975D060D3E3DBB3C79757CF6
63F7F431C787E3C0BAEFBAAFAFD8E577C7AF5E55DAAFA09DFFFCBA7B01AD32A9
C05E7439D6C72F5E75B16A68FFE9D945F7E40A1AAA9F4E386D39C2AF1E697097
6FBA2767F0A6FBAF2E6FE6F1C52F8F04F0CBEE4F6F796EFE919D1EFF78FC2D6F
F55E53AAF17E3C797BD1FD115AC34975F9F6C5E5D5D9D5DBAB2EFBF6FCFC143B
E5B27BF1EEEC840315E9F4F8EA18ABE6653919D57B4E86F38B5F000A341049FD
88FDFC5D97BFBF00AA21198EA17D979C1C275766360E8D53C7C0513780BDEE7E
FBEAECDBEEEB932E643B07703F9F5D76F779CF9C5D42060E1BE8F8F331AFFC2D
36037AE42DEF3BDD172F6D167E841DC8CE5EB2E3D3771CD8A92CC5BBFCF24CF0
09D2E3E43B41D4C73BAC2E6175FFF5A0FFE0C1A05CB6B86459F059094673BA2C
F95CF5C0CA57DC26C3FCAE954D929BB483934A0FE69F92E19BBE98DB20E1EB0E
2B5645994EDAF90C655D9BC468EB3A29B44C0B67E9B0AF9EA86CA2CE719E0CF9
EBBFFDE783ECDD8BF38BBB831FBEBDC98F797A7DF9F6B6FBF6863FBD809FC73F
9D1CFF02FF8EBE69FFFD161E5E4CBE7F7571F0D371FBEEB47DFCE6E1CD838F49
7A051F4EFEF5E2ECE77FFDC89F0AF8FDAA7BD73D9ECCEEB0F48B2F2FBEBF7AFB
F6BBC39F7E7A7BF2DB2FDFAEDE24C9F827FEE1ECF5F70F2EBA2FDFA6AFE7C383
5FBBE777EF8FDFBE181E9F9E9E7D7F7AF64BF2EF9FDFBCFF7E74F9EFC5717E74
FD53B6BAFEE1877FBF9AFCFAF6B78B41F7F0EADFE70FDF7ED91D3D78F3F4EFA7
776F57DF9DDCFDF0EDAF372FF2C1CB7CF1F0EED5CDEDEDACFCE1DDE9FBF7DFFE
38FB707EFDD5ABF4F8CDCDC9AFBF7DDF3DBC7C77F9E4E3FBEF6F2E0E8BE3F70F
7EFEF7F7B7C7CB9BF7DF1497072F3ECE9E941F3EBE990C96CBB47BD25EFDB67A
F26A79F8F1F8F8E555FBFCB7BF1F5FBE6FBFBC3DCDAEFFF5DD3552A83B7EF9E0
EAC3E5E2A7C9C9C983FF6AD41FD8AB7697E0AF3EF1C92029789F8F16D301EB7D
4CC68B14DEA405E78B820BF0019F8956B3F479FB2E9F0F59C7CC40C581E55A45
F65BFA5C02D9B948CBC51C2649364A06E96E817918E481D9E46079C0391CBEB0
5E7EFD3E1D945FF4597B9C0F9231E5E1F82EC625AF5F8F2A78DFC1326D7854EF
D59B0E3B3CE069897F6B878F7F3C8911C0EBEE60852D441BAAF0D567E141C504
551540247585 36AFCF5F775B45324AE92B02E323349F978345B9432F590FFFF9
82CFDED3F48BBE394293E98AF512AE48521603481F08DB7750981665321DA4AD
7C2451D0D3A3ECA7AB8BB75D968DF8945F66E58A65A8CFCDD35F17D99C4FF25C
AB290AB6078893CA729715E9FEE31D054714939D29B016E57AC0375F08BC8C76
40134CF58151E9E7029AF5691799E519BB01993595F5ED025F5AF946F9623A7C
CE461957CE9050950202A3BEC04D9575A8E6658DA6090B4CD3BBD61014E0B13B
2AA66C31CDA65C911CF36E1B72A52C9D30CAFB78C7ECEB49F2219503FB6396DE
F13F43F8AF4563675DBC3061C33B04FA9A8FB90F5F704EE2150864FB9BC04C87
20E307F9389F77583A1AF16E94FFF2AA801B9AC1B1D96134A2B908071B676DBB
A7D3742C5A8143127E57F8091267DF7972277B809A28204B89038DEF7B8A4282
6F1DD135ED19D768A765CB120756E691CC299A1E000AC9CAD861B36CF0A15A18
B9E21FFFC47FDB30C1B7930157D38953ABFFD9F37D90B6D8D135A4530F1C1F44
8E6887E8A41F3905AAC4E2ED063D3F9B3EA7EF6D415FBE286AD1E3736A0CB604
C59D4B18DD4A408733799195FA253EF9CB60765E2BACBA7C991025EB6DDF18F6
86FC3619C5CABF73CABFE0A8850A1AE866350969A1273E41ADDE2CC9E65F68DC
883B2D44B8202F531A66BA093D374B47FD44BEEA1DB1C33ED5FACCCB4D95C254
CD4355F8A02F7A0EDFB30A8FD9AC00F9841CA8BEE66B47BEDE2BB3B468231D31
5BD1EE2058174AC695D9B6D01E235020DB63CC56034A0EB5FB81BACE87ABF688
ABD91D16C30AB23D866C11088A4E7108ED1A3A213C2E9A920618413617C287B4
D2243F049E2DD0240940F77C14405D8B009CD5A030387F83683A725E57014036
7F619B858385EB1A8279EAF9CEC81681658289C00A83A9CAEC695EB274322B57
CF231C43C32124AF91F72466837C32CB8BB43D4CD35960E603111B991449A1DD
4EDA4B8A1648547660A00A734F7BC55A6CAF48C19467EAE7B5441034DD676D76
B41FD509F79A82F242E93B6FFB81E90B92F543B02D6ABE9E91E915405C094C06
1FFC6A9FCB8638DB4237B626C9FC8398CB08CEBAA9022F99CFF3BB4DE11913B9
42CE525CF4744BC07BE56236AEACAC1452505AB1B5BB4EB1F3B9030D12AC8F43
3AA55E9069306C779E0CB33C32381A0C2F93045C9F48A76C0F5BBBCF1744E371
CB7C31C8E68371CABE5E7EC3BE74190E49CA7CAC1869C0E036057D617B0D8866
311B33CEA669EB2E1B96B7EC887E0C92192B7E5D80091A7FBFCF613380AB82E3
2850C8CBBEE434F96A7978C80E8F964FB59EE5FBCF5E2F6A36DE9CF5B07813DE
2B16D72ED3DD5B02477966968F5737F9342EFAA4D8355A835277C9A52EFBC679
BD0269CA3F1DD648D41858DE5521B04F3E07D887B5F2FFFE44A859DCE1F24872
590C9535D2CE3B002AD69D7B435CF364E56EC16EF38F7C714533165FFDD84C6D
2C442AF345712BE5F9E6A962A0B15EE19ACFBF60EBC227D912C043B4840F1A34
959BB62B488126B5A7E9DD3D90F7266504CA26937498719A8E57B473067866D3

```
9BFBAF2FFF4A0D53DFB163A0C0DD661552566F0B1E28ED7A31EE482130B509F3
A769369CE5B3166E617BA78E7ECFB51A89DD651800FED902B815C7120E12A891
CFC0D3A11CF26BB5BB5F5305FEBA6715768BA382659C261FD32D08575B26BD02
A81EE9BA98A27C4D87C2EE027E0583319FC65D99E5015BD130EE87677F7BAC6F
2149BB15B2BBE403E9FED854F5AEEF9F418C69D1370FDCC022CE51F90DF36E51
D5B502EAF906AA39171AF968A40C6B6A45105BE6DA393BAC9C2FFC16655F8A82
2D3A86810F467C7B964CC3B047F93C4D06B7ACC8AEC7E0C182206A146F29256B
113689292BD805C45A5E3B6D35C922C2025E5F9DA2EACD3C5FCCF466910024FA
123F468105A4910737AB1347C9B8A8EF456469B3B8BBD8673ECD4FB572FDB597
C36E6A0E89AC5F0DBD4EFEAA0C38AFE685032F19A08F131ADE6D761543729E73
0CF89CC47B89D73E2FC1D348F09E8D4D608216153C63BB1C024D794ABAD81364
3F2AC05DEA8EC876B2F6BA7DED75B24BF048E7892D0E25E6FC03C26596E0AC0E
963FDCA9296EB351793F8D073AD3943A882CBCD4CDE6AFBD8D835CE62C00BF9B
12C69D1B54817A3A6423BFB54F30D6BDE831CC8D49A80F1E88C36865196D2BB1
DD221D8F02EBA5980110F8BFA2CD78F6B1F1C7753207F8E606A0FA6D307DB198
C1C24A7D521F404F11DBB98E30B0D0B037D3B04DB4CBE5B57908AB256E37693E
22DB7A6817D46AA0264868CB524A1E2CDB501930048B516F4CC6422E216377B9
9A3A6D95790B57A935659AEF78421AE58345D19EE62D3504FC23CC1D073652D6
66604C67F9DC88621D23C28E234592095C3B6A752893E0A845C729AE0AD6ED19
43AA33C343AA54DA9CA2BCB513703DB65B1BD7ED36E982462DF66B1B1EB902E9
F7B1BAD880EB4D26751A4388369BA4ED2FE97FFF353B4963EFD25D73143672AF
EAC5B1CF1EB2BD83E521FB0F9129FB2DDD6FD4D0BA65BEF41BF9AC1845ECF790
C06B0B80783C1985CB56C23B852FD6154EE8B1156DB7E9C5051AE130990FDBE4
C165CF43E49965BB66A90553DF193BB019DCDCB72A9EA09A6DC2DB76FA98D076
FCB6F003C74EDEC7E8F5209C4CD4EAE8CF4001AF47DD9F03354C8A587F342281
A495EDED908D5466AD2DDF370D9332D9668772C1B855FE206B1A78447518380C
74D88794FF11BB9BEB57046628F043C7A3571B4EC7CEEC414892BCD4EEB6AC87
361EC2722D932835D702B5A1499512116E7BF0B00B62F01C0AC9FE5A1F035A48
B2E04479BDC8C6436BAA549F775EC037E530F98895F94D8AAA091E6B03854CE9
85300D1686FEE6B167EF24A8D2C0971DE38023CCBF958C70CC0DDFC303429AA7
4536044BFA545781C718C8295D3AAAEFC8021C2B9E7F5A66A32C9DEB221C046F
204DC9827E6611F83CCFF0E00595118674AB556DA21569287CA9EF1AB295AED3
61D63A09DFD3040858DBEF499000062E400DCB303ED8C54D3989E348BAF0BBEB
7461B0E818C04D43869B5F398057F24B0767772743F884BA55086380AF88E1B5
C9CBF55C35897228647C3994B365C7AE36ECC6A9FD297D45BC6E89A603580053
E921D9A9345FBEF7345F7A41FA8AF8D1B0BD238D6E190AE73BDF6E0B328BE1FC
EAB568B9E4D2C508C770310B57CC66B8A57A8B39FEA8AA94A047B0945B974113
6F29CB55D42A5A35B80597E1243087191CF07517D04A66920013D9B42F80CA48
87100CC9B743C784313F9745E23B40F1CB22E5F7EF5F7DB88B0F01DB66571447
96890061A3C02114AC0F5575CCAE99B4DCC3AF8F1E1F7EF5CDE3C3A75F8668D8
2C5167EAF30DC8D8F73401D0211476B83C723A58A02F5611A6AE1EB0DA2A23AD
CB30B0FCDD1D27AB7C51B66843A4CA27AFF0235959CC89D39E0FFD3317BEAD9C
2BA0E37C3859EA4E12B37FCFE4BA2215C06C84F389CFC805D4EEED0ACE619FD8
EE7582FFB45AAD3EDB8BD0D94A519356C71E51F694594DF67E6807B666784B2B
AF6BCC68540495282534 0BA5D2A24AECAED3CD640E828A94F196719DCC3E3162
11EAB47C56C2AF16A9327BC17A31B7702614C737CB9C5129FCE947D9EFE3063D
BB4B26BE608590CC2EC2ECD2B2429C4A68586A8B2F6DA1DF109978DF4905ABA6
FF20ADDB8790FC8484BEDC2DC6C8BBCD49290A0031C5E35FE4C4E4AFC7FB32BC
C50B498F2F88BC31BD898D2C485AC5219B1296DD801774BD65D2AC525075A84E
3E9C25B29B56DFDB15FB911A0FA1057C725EF56BB0F3EF746E4E9ADE2EA8F030
7FC0BF45DF40489E93FE24E4D927705BC6DFB3A4BC857F3743D95A55C0F97859
5F140A24C2834E1E47A4AB990865D6C320090DCB8876F71A66276AB0DE306F5A
81D4011B648F7F15478D10DEF318A11B1037DE4DE4FEDE000CA4BD08ACFDF8D7
3F1CFE7DBA234A4221C22379EAC5927970731340FB7EA92D04011E1103498040
FCA2408A002912EE270AAC65FB5AA2609371BD81F8587F6CAF293C7E2751103F
70584DF16EBA9F2830607986AAF9F50F87BF4D51609130200ACC4E6235B131B4
6A141007E691C8C34DF46761173484C02E2EC43E9165D718F3750A95F0FCA3CD
788245A09E894125D81F0C88EB5155DA34054137558276A5D9B44163FFB0B62A
2BAB1A807B08693F0C26D26234791ACD6DD8BC8AA93268A51436A2E0F79AAE52
751986CA908D52EE5285BEB3CDD9028C99F72112194B8336D91891B0EA358824
ECB221936C8C4858D5E64402746B26830DB9CD26E6E7E7B8F08C700F6EB01BF1
F93922DC08BF9CC0A67136F8CC5DD86820C7C7F11FDD818D86597C94DDABFBC2
BAFB2E9AE17FD7C94B458EAA99C58C2D20DFC64F8060D81E42C09FEE6504EA89
53479FC481813ABA24B3993C45D1A62867BB9345B948C6F5A76DCCA2782484CA
3714ECE6C9927C345AB3D1587FD41E0769371FB11E98BBE9F88EB5EA1B67C202
DE788CE843427526A938DE950EABAD7C379F42A15D38EE850F2BAE437FC2D35F
EC5383D2A31114C27346F000BE579F1A1C3BEAAF4316D1917229B97D298EE66A
08526C1B191B682BE4E930CAE66A05BE7122270C7099127B6DE423415BA7AAB2
C0618C4AFA4B63D72D361B3ACC8AE47A8C9C9A4EE1A9914A2AF88F803C63124A
C45864A6863B03A1E334FA496D268A09ED41ED10C20CC9F0FDA2103BB7EE16FF
```

317E65D24B496CF062DC120A273E17BEC6655ECA48A860EF5E4CD221BB5E395B
BE543EB6B9AB1998A2F5E03B30A56342A51813A8569828088F6A8F8481119210
063D010C7A0218F4C461D003C2A0471521B5C199CA5052F57734021D8D4147A3
D05138740C24ECD091F2C42B524610D0420D43F1D1213B2014D208C88300FB72
118FA7BA22D9EC435DAEDB4E8BE27E70DCAC4FA05208A4D5E373773FCE063772
0E1A1AC76C7DD39241C6C3AF96875FB9390CEA4E92A5F19B1E45FCDBDAE05966
40BDC3469DEDA262F42EA2A2198EA9905E1D2F7341F620D789B3126ADF8EF5BE
5A3E610F8D58C6F88DCB551553370226ECD6A45A5F73A87A9D3864CD48A99313
99A59AAAB155ADC2EE17F78D1E77D8475A16E0A36017412BB96BE9A138FF2408
1E72658DB2ADF382661431D28C00701D37AB44133E928AC15B924D1693160642
A68D6D2F91D48068AFCC3A0C88EEFCE32121695F0203333610B4D6DB32418B20
B63121EDC12080923829B4A2AE35280AA18C0CA4318491CB46D4921810270E92
871791C6514CCC5E4020767B2231F3C2F2537A16F5D8BDBCB6EB6B234F8E90BC
D6624E341A73DB51D015F8EDF3454500EB32347CAB2835A7BB3938CDDE7CC844
70C6CA1468026810EF18492163487399E2D168DA4B43A7682FA1359236F84BB5
157F491AE10F2D8DF0A7F60AFC932818C66831085191459AEACB8E4915DE20E2
9ECFA78468C9B1F40B38359E971DB357C2A303321A1DE69DA45446B3335D1E32
23B679B10B9141B54C907B5955F054D37406BB0027BD411A7F6B8DB2162CC1BF
32D49C9F0246610B9860F750614B8E43610B182F6C9077CD89B9D9D8AF889575
457804850AEFC7917DEA10B5716A82747C263065ADEE0A6DB837BAE3CB268557
6661359756253ACDA851FA9B12A663CB1F3545D8AA50DF95C6F2CC112DB84DF8
F843BB3C46EE5348E84205E11A899AF38EBA59045F3AD72B28BF54AF3BBCB08D
0A8BAAD13D5666F2A69499D18952A28A97ED14E321AF976C4DED126FE782771D
89E8E1D1C1F2F09B3EC37C151A19F4305D40232708ACD6ABEFE2ADF76C157DAB
9EAE62D1F355025EF08415F39DB1A242B15356CC77CE4A15734E5A550C1E9C80
557B83E1826C7ACC228ECA61B6C27456E76258D70FD3FC5A925C9CD0984BC682
6F22B6EBCA642F2AC327684E42A57DED8E53CE49E4719BA2953E9F1355DBF0D4
07C00A82385A5039C1E58FA6AEFD275622964AAFF79471A6EA7DC39EF6112E18
D4A0520BC326F1BB03516363D162DDCC91E8B12A602A22240431A8F1D89C7620
386CB0548B7DC94B0582B4064B3DDCA8941743CF590F56E12A8013BA89E3FF40
060AB6431C871C062F9E07E2E8205673349DA29B3FE5C5B7C6AD25F6B3E627E0
AFC750DA137E0FABFE5CC90C2B62C614C34AAD306262A801A69EEB478C202A9C
C5CBCC7B26850E27C568E22D42A153B8D8C9C7543F8268DADDEE3C867A493218
A4EAB455E8829F36E5124D51906E52C2B683873FE869304E93B978C6CBBEC4F3
9F2862C0DA49F3C6321DDC271CA7D5073B976959C02564181D106605DC6B28D8
5EF6387D8C752D4AFE136F3115ABACDD42DEAC8741382D789E73C8BAAE2B3561
E6C0CE2C9FDA1170C073132E7E01FF4D9A2C335ECD4D3A87F9F20D7C848262A9
974C5777B7299F2C304E92B857B59CA7691525445D4CBE045BCFA83B27F051B4
72E83650E3E7B97F06D1D0BC5E352BDADE9D8405FCD5E7FEE157657F8A17C204
E83E370882871A087BF8EB292448D5B890D3146923A192B827098F9E7C66DC3C
BE10E27596F315EBF1CA647FD15917DF1C093371E5804DB59C2D4C5C18428530
8EDAA8F35B6E71EB97405C5CF3719B262AC610142FDAF314C35011059D7A8739
D9213AE6602093431DCA728755C8B036709C7A49F501D0A818D54B043C6FE40F
3A755C96C9E03655D10680B23866D2255CC13CBD41B255224F6100CAAA7A5B49
7AF0426ECFB865FE91ABF4DDEAF885C7215EAC6B45C5AA8EDECA5D7A7EDC7E04
BCF2B98E42C4A7FD39EFDD95D8D04E87C640C66EC7FD4771D59CB8F519DF8B6B
832B446A5FA760C8623B67487A11D44BBC448D9C2EF485B5D6027B3887C3CAF0
2C258F1BD72B1941F8B10A4C7A69839C71799112D01DFC11868967D01C98D43E
71D79B6A1FDB1BA6A364312ECD4064EDC4BE10194E363B4B979F61B170BD6277
B7D900FB749E8E00EB5C459AC093DA31D909CB2CD55B05FB98CDC139C710A6CE
ACA2C40E8D4E0C96698E268AA28ADF604462F8E43EE58B08BEEA11C040F043C1
339EF9B622CB1079D74E459C6217C7106795E2C031AAA1F202486223F8DBC7B6
55841F7E70EB14ECA5EA9CE2DD9D9E3A81A39C3A89CDE0AFB74EFC60D7D916DC
14B20EF62C49AB0F4BC668D777FB4A38136CD64F84BC0B1952B58FAE133E5369
7A19B486AFEB92DB20DB56A8166A49EC36243FC96484936774ADAD12B85E8C22
F7F820116A5DD74C1B9FBC85541BC6372B2AAC9AC1C235DE1618BB0131EF80B0
AB5C91CA5FC53D88AC0EF29F20F6DCE0DA180A75B4D41801418216478AF2F8A2
E738C4097E09D1BBA6878528EFC8320DF0ECC84252A00B3D4CBC65E1B33090FE
EADC353BB7492857E18758150A6CE775FB78A7B9CCA94EAF162550584742F39B
F21E52A030E608A3209CD090732D6279364D89EF02B79EA615CD1A43AE2A65CA
7458CB4A9F82DD64552C181F407FEEA57183056DDD7AB14E07ABD8CF3C2B2D9D
462A30785DF8050F1CB05A95C68CADE1E879121B77E4B4420C23273E8D55ADB1
C1832169C0E6845DE663141D2255C4251999D15137E38BDFCF5A62868373AC25
A8F88B85AF752F538869221C62C56EF38C452474F13F97D06410B23677344A40
E80E7B07CDBA2F9D2DBB4288CE348CC2A4FE9F26FC22E4135828027A68E7D28F
26F520FD7A83A41C54EC6FF730D5DCCF50F359CC34B4655C6FA4F9CB38F3A730
CE8406825F5D263BA7A32CC7D7E0B689860608762FB48219024AEC3A783471D1
875E48F40D60358124BAD30B09BF21520D20C9F5BA1F5221B3D541F21DB278A0
7FBA920678E23A99DB3B65C6DE19EF072D91E8C212E3ABFE666BB855326D96DC
D9610FC4043EE5D3F16ABF66B6A8C2136D35833A4A9906AF6B670A264DD1FD10

9B0B50B2FB906052F0CB6A24284F8A4DA79FC738FF97C4FF4BE263FA5398E371
67138EDE0536CACA5B383685220AF707BF901400162FE1220FB87A9AA9BD3B31
E4387D8AE92E9F7126B3713A819B55866C95965FEC0444A74CBFCF0C4642627B
F398056F0BB399056F0B739A0D6FE399AD492C605F52B39EE91554B910069C3A
F0335C8FD24AEE12D3FA5C9D0E71856FCC83E6955C95C50EDE469AF88471C5BD
41DCD0D20357A8F86C83B68CB1BCDB4BBEABF81CE1D75051E997E62F4A5F8365
653C677F59339AB3674A33E7B66C48ABC4E796F71F3A1C55B7FB21BA7D367D2E
BC9104A9CCCB6A9449CE3EBCE19F5465C52D58CA35A8DDBF15A13D98CCB6300D
C61D6BC1CB3D8D43D2E1EFD5BA087DB36C3B527943A51112DC8F645FF716A64D
40ECD5212C7CCF827778A1E9AD87CBEEC58CA13285BE67784E635CB6F00B3CC0
4597E840D61A67702745A32BCEFC1472A9E47F835105FCC00D6F166A8185BC81
281DC86B84ACB6CDDB174046ADF53A3BDDFF18CC1CD9D0DA9C4A067719E7BB11
93ED2E17D8CE09DDE72C82BBEFC17A008619BAFDEF8A58DEBB4AF12CF66BEE80
B7857245CDD912CE12A6BCBE6F0B2970371B72015DAEA81FD19B213464550E83
7914A2768E6DE0DD34393E6AE13BDD02DB8A8B29DE9928DDEF7FFF24D7996A1B
DFF6949DA66BA266C20B1222D2C99E7D0A488B29FACECB2EEFE98E37F08DEDD9
6523333BEBD957555627ABE83C8477316E5B60BC03A0855EA13C0275906EE8C6
350E2C5F9040359242A48A1657BF928DA4EACD8392943482C54DADD1C9769C26
1FD76524234520CFF96C20BCC42B7B7B565EC15ACE461B36A794B765EA4B369B
2B25D834753EE915FC12BD88753E12915C78AF55BAC4B8ACBE7A9DA0B8ACB863
92F9BE8C563BC23419A8B7E810DE66294A31A1283867BA00C35DB13AA47589D7
F91F56911FFD07FCE49ADD1AE8D7F334F9D026631CB3C7B6499B8E230322C224
D63CF430F6B58F5C8F85ADC266065F1BD591E2CA66B1B7DDD1A3B8160590DF82
ECAD756D4EE4D60C4254785A021FB37C51F81B23393B5D96F384E5A07956DB67
FDC07CC2A051D50C881F9342699631D915AEC2A2A7765B43462A527204AFFABC
20F691515747F508A2F124FCE375FB412C5D8FE50A458872206BC7D39B0694A9
184861281667504F3556A121D9FD28DC2BF1AFC7090FDF6F27795607A67CC3BE
7372DCDD66E3D08226BC04B661C36CA7492A877AD3358FDD057E2CB1796E2744
D6E7C6D465D81BF02D8EE2107E7E3948EDB4FB3CD4326F8AC946187A41A982E3
F2F7912856C7F5D7171CEBCA0A2BFD2538FED70B0E7346FB33CA0D1CA75B901B
516B1EC9A6B5958BBB7932633B3FC3DFA4944A6399CFF0566E1F5F193AA150A5
632B9550CF7828197605DF40DF88E227D33D353DB7373CE442FAF642DA572F24
44EF63D815ECB6D68CF079D9203CB36F830FFEBB747D659A0C31C51A5D6F77F9
E036991BFD6E957299006E20A7FD55BC54180AB33D4EB3D7E7AFBBFBA6C30FA4
0ABB60E61EFCB557DEDAA9BA6AAA249F6C4198680F89AC0DBAD7E51C116B04E4
016D15C323B9FA56A0B82EE0A1B9CCEF696EF7496DCC2A6CD82832BD4598D19F
BF342F4DF373E1AF8B64FC9C2D3851E6B87F83BDA67FDA203CD787540D553239
63E33EC24AB07EC87C019F3758E4C7A686DF67C52F950CA76157176FBB58235C
4D3A1EAF644E5728DB56C66A7B41AB8C6C59627BAB3184D4EF7FB003B75FEB52
03B856AC222674B4C1389BC957D5E4A51DD8905B60A0B6776D4D527870E16DEA
DB20201ECC5A201E1A6DB4DB40BBE2F44ED7720B7B9E6B23FACF67AC252A42B7
091FD06DA0FE8F676CCFE900910FE08B57FB7D97F0A2664FD5B658371637D416
0EE0798825110D2308A5F34E3542932502494614 73DFF1572E811B0C938E8FC5
1B40D27B1E5670A02A7F3BBC19ECDF2091A9DB6A892C49E310A6E5F47EA43190
ABB6316A08346ECCD6B8C5ED992AA02061BC3CE3725F0C1E66A38841DE0D0020
CC1A4CA146FD7A74BC2F43B8E220D066DDEEDA11E3C2F46386191A51B01127EA
F1850E187E3A5238A62059637E2EB863FA4C0008DB011AD3A728D399FF1247BD
3050A2226CDB909C80F9F584249409CFA746AB0C850656BE8D56B7D66CB542B5
61AB71C6F4B75A11B0766DE5AB0A9295CBDCE2E65A418515D54FDA4B0DADC6E0
5BA1DC6DD9DEF54AF02D7ABC82FBF084AF1652AEFD26937C312DF982089C2C93
F9AA6E7166AB8D2AA7DC9CB71139CDC17F165B0495F225C2F42635EF48F0B8FE
0ED371993090220CBA38B4E6C36C143918F21AC2078B194C12B665CA7B3B7C2A
FB01D7CDB00E12E816B705D7FD325515D74DCCC46B0DD3CA00A98E0DD190CF3A
C22181575B83A18FE4FD4790BCF58EB955FA362D20E6A0DAEC2D66CE629B19F9
D712385BEDBFCD6415A406FDE70B26452AA4D7DD874BB0B05B9F95DC997692CC
5A1FD255DCE23401B3CC9075DF755F5FB57FE8FE127790161EA3F88F362EF14A
426206EB27174B339768367D80D86C5CB67AE4089646112342D6802F276EA9F1
7F6FF3494A7FD011155FE3135899F87F7D7454451F4F3BF5AB187AD4158B9076
F44624892F7A23D9CC50C0D3550FFEE6CA388EFE0BFFC4E68C6184F3DEE6A094
64E44DD1D362CBC4EB32EABC88847ED99D630CDF672C7E27A13499119A51BF46
A78584379C0999B795B7574DBC18F412AA38472AABAD7A5F0BA3ADE9A67DA0C8
6B3B68D093E92EE1F97BE1701EEBD8A1FFB6C3A4771B519A0528B0F7B79DFFFB
E38E9173BFA6536AACEB9084AD573B92C6B343AAF61BF39B79CDD4E05E240B2A
0C3381FE2E1FE161E76755C1567A222D06C98C8257A89EF076C4EE381D954C67
DC0F76D92E04AF2866C920555DA6D94EBB83FBF1D945A966E2D3ABF22CB2AB67
F3025204B0949D8D019BAE5131C008731D8CC58E286D6D4500A3445F07B0D85F
AB05ACE78B75003720054C3F6B612CF7976A3A4F1FEB0A323D4663174E96B65F
A56E81D887222450C07BD1432881FB4ED7DDFF6C7266EC816C3C9C8A368E88C1
AB96081C2DF39A07A2E193FAE01E988193BF7466C67F6298144C3C1AA74FDC30
ABA4473B90A7CD2CC38875B44240081886E8AC726087AD9225229D9D7C55F162

7D6C68535054D17ECCFC5530ABB8D170972BFC830FAD326FE1DCE06417AEFFB5
6622B92CB7286B7B5275C478519F6DF7CD8E9485213DD847C9B5D3EFAA3FFB17
D95225C63989CB5FFE7FD2DB50A8E1633FA07A43F27F71CE11FC795574F3849C
C52E145ADE7AE5B055E58A0FB82503BA2A72F7065DBD01996CAF8B688C73FFAD
1B58444464EBB9027F344E6E78AEDE284BC7C30AA224199947F64152D124521D
2AC2FC6E23436757E92C29DD3E8BB70617E635D1FC199DC43B7261E2D882AD1F
683FECB027EC0F4B36FF92B78245EF7B5D76B6E564D3EE36E163A383F7ABF03E
A65D1BB2CD098B1CEE3FFC6131EF29E29F0C6EB81D98B2CDEA8E08160A546A32
AB773F0212DE20D911505BF8CBAFF08BB3D73D3EB1A4785A71769B4C1663F6E5
5113D3BBE8A1A64807D1AD698DC43298C1EE91A06A35D6116648263 9C4146245
5DDA115E56095BB186155D9E35EBB5359BA39A458322D62C82680FFF70DB20AD
BBA08444E3727DFAF6ABFFB91A32DDEE63CE4CEAB3BE10890C52182651C47701
34AA57292B2DC4BAB108CAF242C6DDE04C7899552E2EAA46B6D3B72599971611
F6955D09CC651CAC37541C6A99487212B6EF696ACB09C832CEE0179AB498BEE7
C8A21F15366632156F5C4C68EA74AE68BDF1DCAE14767622E93A89C066E46321
87443240D217A79B55411514C229485F8205C1D4904D6F3A6E41F12558E3D0C0
D4AE714898060AD2D6B0AF207C89145431339C8274C99448EEC208FCCCC8EE5A
2190F12558505D04EF1494D7D8070AD2FD53BE1AE94BB046E37AE80EBBDFB4AD
E1A966DC1F9E71E7F256F04378706517D2CA2494FAE22794BCEA8B686C16945F
021446B83C7FE2AF11BE846B54E59C1AE14B888B7CA3457DC197A125EE3D0488
77E6F4C88BDECD3C196620CA0E97877092A3843BEAC71F7D07563C52E36879D4
4446980B1E70CB1C26F3611B4FD0230D7A74992E0726A77F4241A933BBD7102D
2A80912D55BE7AD2448460439B888BE051E08A74902A088E091F518F9E3E7DFC
D513FEFF3E0BE61184F7ECDEBA52C5AA50900B9FE14F00822B5E22F9D6101B91
A1D6A0365B0884590547754FDE5F7CB43CE0D37176C37B8A4CEC1FE5AF016D35
487249425D8F213634FDE5B45FE19FBE5A4E44C8660B9B3F2182B6500B238852
AA97CFB39B8CE3F1940FB74932C7E76C3A4405077C3BC00AFE1CEA8B90434BC2
CF5DDB268284695162D017F91886E1E1DF8FE03F63B830C5EC56E6DD0CE50E33
7FF4E3766B6B3828DF03A1A6DB6B77081A4CC110BD41838DD8D6F8790F9AB98F
EA7D49D1348B72354ED1CE0662C4D0A9D9304BC0B0F638A490EF9C52065AE240
D41D84553C624959CEB36BB8EFED11AE0F08C1425C2D5A0975B5F3421EE7B0A2
2F1776D4C8D0A282A3714C66277C0DCD13D52032462D89B04DB3DC88938999F8
E84853B39DF8DB68E8B1994F85074F448078D1ECC7DE9B5331278515D75ABDEA
3EC39827AE4B67B16BE48DEBD12D6A884868F44C77FE86927F21A4977BD42271
65AC89A1B9A0A92C525C9ECD042FB09EB8A5020122F97E7F33587555A703D955
DAE189626714095C1AABF5025F21A14F790BD192CC251E1F1F9C5569AC6DA5DD
90FCA67CBA8765577077EEBDDF13F3D10A09788C7859440EFDC4CAC56C0C8BFA
BDA095C24B2BBC7849018B5A46A826AEEAC8CAE3861442082E418966F57F712F
A9FDC4FFDB954B59A3FD2AFAED2746411BD7A080D27EF156455E9880AE83114E
54063ABB38BB7C22130BFF578AAEBDF0C6A774A2A4239B0487C03C13741644EC
C81991CFB4112B966F1C09EB94EFD3DA6DEEA16BC39083E23F7771F0F47D1C39
CAC64802CCB156BF88C1FAB938931063BD719E0C9BE4A706484E363E6C87A985
6127C24492A2E3AC6CC967C558DBE02BBF362A933F34A9B09F115C71B736EBED
E1EFFD90E3804B02BB5A24081DF06B280B8B46A36F2B83AFD998B3567742015D
5FAC0893D49FB16D4ABB763A7D9D16E2F2CA685E7D73AA78182BCDAA70B3BF47
C8EF836B2C10BD70954D6AFD6EC545D4FD1B4DEBB570A3B19AF51B2D9681C146
23D8F51B0D284504F8069C1024CC16B8C16D846AC8FABD1744740B3DE847D41D
83887ADC8F774B5DA0074E7CDCFCC11DA0593DCEE95B247F632DF18F50120DC3
8C97205A49F4835A4716883D1A4D0AB12BDB5C004A83EE3A6228169B0996771D
EB0D5F0CA4D3E84D66C2CE439124D416A8BDEC34EE488C5D8BE8B12E599F1E58
996209331977232A4FA6637C57A87B29B85681260CEBD2167A198DE2E0ACDAD7
5BB6D7AEC05D9A88D6683A570818722BADA51766A0FBE8F06898CF6077C9BF78
2E0DB8CBA64338A3E31CE0B309472628610EDA66DA793347D3E0755A30BA168D
F32BA7E6D4B857651D7889BAEDE0FEB8D961FF772ED2A29C670315AF08BE6A42
4E922599C8CCCB16E028880670CE9B055D00E74304DD2DAE9D4CD2619694E978
55DB66153B5C41B7050859DBE8DE61CB9487264DC453381509B35ED57E677CED
B027CB2D38C04902C9D07E925E4409A475DFE275856A277C476A4548C9B3F41D
7D9A58FC3E5C1EB0FFD020DB4BF6904F298747216CC11FBA85C790C99485C7F5
EE91AAF0F0E0DD3D92473ACBBD1DD8B8D83C697872BBA2480739C403E772EA79
456C3585A74F4F2E3B951ADA4B9F74956DC157C48B66FB3CDD6E9530223798AC
60BD0958CF5CD632C34899014FEE951EB23D4157986090279F1B04D8DF00DE91
60F050CBDAABFDCA54F2D01CE6D5E3A4FF5423B65D55B68DC106B1203ABE8821
1DB64A0BB754252E8659C7FD53AB86A87513696B1322FAC0F8C9EABB76C7A4A2
43C1695E43C02A4FEAAA98B9A3571D01F17838460D0139ABC31375EA29B6EFC0
E52FBCFDA3672F376C0B53AE6D3A3252A78A8D37A64F9F3970CD3026EB403562
8CF43D1B2CA6B03266210DC91E706106F2A88EA59836C385D64EC9763539E792
866DC0847657AF6492156C060F6620F82BFB54F83C09A560DD64B799BC2C69B3
BF5979AF280922A9C057674E99BCA1152A0B9D502C5C940A89113B451709C5D5
71AB33A03C14E1C04C02EB284A5283681FCA2F6D76E40A6882C727097680CF9E
0C91C5B0A929F42BC3548F4F63C4FAEE141B593D21F8E49FA6DACA7FAFC7ECA6

```
4621478BAFE94BA3E94BBB5B44BD2D0F13DB048903B65BD132DB18A06BBF4299
954599558532EB4A2CB3BF6D38266EBED3539F031B0392CB2BF0CE02EECED59E
6945B5E6899BDD9DDAAD498897A98BA8D64C070B6A028E12618F19A93E1B63CE
582FF2596D96CF16B3B6200A5E77888B1DB1D65D2BB937FCA8AAE46D427C15DD
13F10BE4B15CDB76D2C8CEE44B644F29577044AE3D498AD20A6D8E8BC86BB8C6
134F47603CDB4A7050FBC09BF92974620E00DA39A55BBFE39B48A7049CD70D0E
AEA2A38CA7289D3C810398A34DC8E524E4929B74ABD72CED5C828B94B830B5C4
9B3DC78BB4607BE9321D804B181BE4C354DEA5BA5B488F64BC9CCDE104E7DAE1
3E732F1AE60A1799A4F08867462629B86ED007116BD7AE5D27F053A03674B1B2
0FD57AA25E4172A277F47B0177D68E3C3C4BB645BAAC065A277C41A0991DF8C7
292DA6A274322B57CF290C72E0D25C48C39C5D83575DC7A435DB05C6F1E6C7C1
EB8FD91B084C8CBE938BEB36D8AA343658DF5A7670AF7FE5E1D7DF3C3EFCE6E0
F1E1DF0FAB1195D5360FF9C94E93091F4B3B57C96D3E497698B0211D4A0F70F2
8395F210C36290DD25EC0F6B9C9D330F5219A750552CEAEAF004060571E21000
A52BA51E1787F2FC10753F522F99DF14FAC9EFA3DE122E750041488886102A87
ABA770B19A8B25ED1A5070644FA8996D0B1F41C55075F2C0F1A05CD2ED8DA855
CB200DEEF0924790DDFCE2627B9FCA8DAE24E4D5097DD2D7FE8AFC97F05774CA
61B00BDA46923302DEEA5DE840DFD28A4BB116F063283A919EA582A353380402
4A7E18C10914894BB871F9C231127E3C7BF84E9E9FC7EFA21FFA61E34BF043C4
594665513B6F1A0972B8896FA242125D82F79CAB28ED420AC0D13DF2076EA444
5049162675350935540C590879F7541D8AB8976C5744F372A84D78F33DC15246A
C5B785CD244F19C8FADADE1350BE241DF2EBDCF11B2B6CDA6FFF89F2DB4711AA
1CF61B818AFBC9C9E44A0CDE781C8D4EBC82CD2A31195111976672397DE1BC08
F7C5BBF1E0AB69838106297C0F86998CF156CEB3E94D6CABDB4C883F283EB19D
6E33C56DB5B8F98EB3DF1A5BEF0E3E345D0AEA8B56A1CC6A0E82AB2A1D669E93
683E9C94AA639E076E52D093E2D48AF34BF0E33A11C782CA5CCBB76B6326EB22
85C08D0926C0B6DC111C32E0276144C6D28651392A44E5048DE050F2A8A73649
96BA907F0622EA516EC1546185FB26DC2F16E5CC1A8C30A7724FD3C87B9F0E73
B369425B3B8074699A26B533E7D990268B7199C9D857F0EC212D7830CBB522AE
A9AFC71F425DC03B6F96E151B40FEC30D24D38ADD00847C768526979A986446A
12016C031343DCB4005EE8D2DB056C073EE3026AEBBEF53FAD510E0E964787D5
7BF0F4EAC67714F0ABD869BF3E0BAD85CCF38336B50CEF2BE7F81B1C28350381
C88551706D66F8B619AA829AE8D5FC7F74B47C5A59D5E9A86B95806B32C9B867
06EB71D59ACD72AE56E34237C8832440A6640292AEDB38053D2734E9554C90E8
6C1D6269A31CC1FDC73FC10E968C31FE859F6F0392D746907AFC1E5842791755
FDB639BEEE38033382116E2A604DB043D6AA9EA25078B1BE2AF810E2D30B640E
37120D3C8D028C8A6B8E1001DD56D65B4CC710A0AC057352BF7A41395E6F1E97
1503AE04F011418312AC77F882EAA1906562E9175F0EC9DAC2734E1840EEBF98
D94C14D9B925884028D6A35CAF568A88D1B79C844446BC2D8CF52A2F3A54650D
2921E9827C79D5D9127521D5531852E47C58B88AB8533324C98332D21D445D6E
C0BA9084E89CA66DF2388DA38F16DB5A98B020C2CBBCD261469C6F50940CB552
4036A9F4DE8A68035152D14D70A65DD3DE6658DAEE6368A3597CAB86366578C2
6A7C8627F810303C25D3C16D2E83098526849019 89E20F218488BD498C725A39
3E67142F84168582B3E9155F124C9810A4683EDAD9091DC0AA2A2E14AA62958E
C77C983E6418D302C35A043097E1B4508AB761F5C776AFCBD880AA865A3B7C82
BE7DAC073EAB078FF9FF409D1DB7F4CF593E5EDDE453F674F935579BF89F6F96
87315F76A1277DBD3CFC3A2CABAD2580B1FDCB5AECE860D972AFD753D069588A
6D675A449B0B58010BF82508433AADC7C58E2F44AA838DF457960218EE606C89
0D4381A3E1FCC0F64C6710FA00FE457B075CABB4E8B0CF0971E889FF22536399
125D064012EB8107FDFF0F1217899678240100
}

; Ниже пример, как форматировать блок меню.
; Отметьте, что есть два меню в одном блоке.
; Меню "файл" в несколько строк.
; Меню редактирования в одной строке.
; Обратите внимание, что вы можете поместить блоки действия
; после каждого пункта меню:

menu-data: [
    file: item "Файл"
        menu [
            new:    item "Элемент 1" [print "Вы выбрали элемент 1"]
            open:   item "Элемент 2" ; icons [1.png 2.png]
            ---
            recent: item "Посмотрите здесь..."
                menu [
                    item "ВЫИГРАЙТЕ ПРИЗ!"
                    item "Попробуйте дверь, номер два"
                ]
```

```
            ---
            exit:   item <Ctrl-Q> "Выход"
        ]
    edit: item "Правка" menu [item "копировать" item "вставить"]
]

; Обратите особое внимание на блок действия в меню
; определение стиля ниже. Измените его, чтобы выполнить ваши
; собственные выбранные функции для каждого возможного выбора меню.
; Большая часть определения стиля является полностью дополнительной. Это
; зделанно так, чтобы быть похожим на родное меню Microsoft.
; пример http://www.rebol.org/library/scripts/menu-system-demo.r
; содержит еще много примеров стилей меню и опций.
; Единственная часть требуемая в примере ниже, блок действия "item style":

winxp-menu: layout-menu/style copy menu-data xp-style: [
    menu style edge [size: 1x1 color: 178.180.191 effect: none]
        color white
        spacing 2x2
        effect none
    item style
        font [name: "Tahoma" size: 11 colors: reduce [
            black black silver silver]]
        colors [none 187.183.199]
        effects none
        edge [size: 1x1 colors: reduce [none 178.180.191]
            effects: []]
        action [

            switch/default item/body/text [
                "Выход" [quit]
                "ВЫИГРАЙТЕ ПРИЗ!" [alert "Вы победили!"]
                "Попробуйте дверь, номер два" [alert "Плохой выбор:("]
            ] [print item/body/text]
        ]
]

; Вот простая функция, чтобы задействовать GUI.
; Должна быть включена всякий раз, когда модуль меню используется.

evt-close: func [face event] [
    either event/type = 'close [quit] [event]
]
insert-event-func :evt-close

; Финальный пользовательский интерфейс:

window: layout/size [

    ; код ниже показывает меню стиля winxp:

    at 2x2 app-menu: menu-bar menu menu-data menu-style xp-style

    at 150x200 btn "Кнопка меню" [
        show-menu/offset window winxp-menu
        0x1 * face/size + face/offset - 1x0
    ]
] 400x500

view center-face window
```

19. Программирование CGI

1) Оболочка: данные могут быть, введены и рассматренны непосредственно в интерпретаторе Rebol. В коротких сценариях могут использоваться слова **"ask"** и **"print"**, чтобы ввести и отобразить данные. В формате **"command line"** можно

управлять, вводом текста используя, условные операторы.

```
Rebol []

forever [
    prin "^(1B)[J"
    print "Выберите опцию:^/"
    print "1 - Время сообщения 1"
    print "2 - Время сообщения 2"
    print "3 - Время сообщения 3"
    print "4 - Выход^/"
    answer: to-integer ask "Вы выбрали?"
    if answer = 4 [ask "^/До свидания!  Нажмите любую клавишу, чтобы закончить."
quit]
    print ""
    loop answer [print "Rebol лучший!"]
    print ""
    ask "Нажмите [ENTER] чтобы продолжить"
]
```

2) GUI: в пределах графического интерфейса, текст введен и отображен в текстовых полях, областях, и списках. Процессом выполнения программы управляют, используя мыш, кнопки, списки меню, и другие графические фрагменты:

```
Rebol []

view layout [
    text "Сколько времен должно отображаться сообщение?"
    choice "1" "2" "3" [
        loop to-integer value [alert "Rebol лучший!"]
    ]
]
```

3) CGI интерфейс позволяет вводить данные через web-страницу, используя текстовые поля, области, и всплывающие поля в форме html. Данные поступают в форму, читаются и обрабатываются в соответствии с программой, которую вы создаете на вашем сервере. Возвращенные данные отображены как форматированный текст, таблицы, и другие элементы html, которые динамически созданы и выведены в соответствии с вашей программой.

```
Два варианта:

1) Html страница:

<HTML><HEAD><TITLE>Форма</TITLE></HEAD><BODY>
<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">
Сколько времен должно отображаться сообщение? <BR>
<INPUT TYPE="TEXT" NAME="times" SIZE="25">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Пуск">
</FORM>
</BODY></HTML>

2) CGI скрипт:

#!/home/youruserpath/rebol -cs
REBOL []
print "content-type: text/html^/"
print [<HTML><HEAD><TITLE>"Форма"</TITLE></HEAD><BODY>]
times: decode-cgi system/options/cgi/query-string
loop to-integer times/2 [print ["Rebol лучший!" <BR>]]
print [</BODY></HTML>]
```

Все три метода выше могут использоваться, чтобы управлять данными, в онлайн. Вы

можете, например, написать программу, которая использует оболочку или GUI, чтобы обратиться и управлять данными на сервере. Многопользовательские бизнес приложения, которые совместно используют базы данных, хорошо подходят для этого типа клиент серверного дизайна. Каждый пользователь получает копию программы, и все соединяются с централизованным архивом данных, который существует в одном местоположении в сети.

Чтобы представить данные в Интернете, вы можете загрузить интерпретатор Rebol на сервер и написать, cgi приложения в языковом синтаксисе Rebol (выбирают версию Rebol для операционной системы, выполняющейся на сервере). С CGI пользователи могут взаимодействовать полностью через знакомый интерфейс web-страницы. Это - один из самых общих типов компьютерного приложения в современном использовании. И Rebol облегчает создание этого типа программ.

Требуется знание html, если вы намереваетесь создавать интернет - приложения. Html - язык разметки, используемый, чтобы форматировать текст и GUI на web-страницах. Html не язык программирования, он не позволяет вам управлять данными.

В html элементы на web-странице включены между начальными и конечными тэгами:

```
<STARTING TAG>Некоторый элемент, который будет включен в web-страницу</ENDING TAG>
```

Плужирный текст:

```
<STRONG>полужирный текст</STRONG>
```

Чтобы создать таблицу с тремя строками данных, сделайте следующее:

```
<TABLE border=1>
<TR><TD>Первая строка</TD></TR>
<TR><TD>Вторая строка</TD></TR>
<TR><TD>Третья строка</TD></TR>
</TABLE>
```

Если хотим отобразить текущее время и дату в теле web-страницы:

```
#!/home/youruserpath/rebol -cs
REBOL []
print "content-type: text/html^/"
print [<HTML><HEAD><TITLE>"Тиры"</TITLE></HEAD><BODY>]
print ["Текущая дата и время: " now]
print [</BODY></HTML>]
```

Код выше содержит встроенный слово **"now"**, которое выводит время и дату на веб странице.

Пример ниже - короткая программа, которая читает и сортирует информационные наполнения текстового файла, названного **"users.txt"**, и отображает его в вашей web-странице:

```
#!/home/youruserpath/rebol -cs
REBOL []
print "content-type: text/html^/"
print [<HTML><HEAD><TITLE>"Титры"</TITLE></HEAD><BODY>]
if exists? %users.txt [data: sort read/lines %users.txt print data]
print [</BODY></HTML>]
```

Если вы хотите, чтобы пользовательские данные были вводимы в вашу программу CGI, вы должны создать страницу html с **"form"**. Формы Html включают текстовые поля ввода, всплывающие рамки выбора и другие графические фрагменты, которые учитывают ввод данных. Форма должна содержать единственное **"action"**, которое связывается с адресом сети вашей CGI программы. Шаблон формы ниже содержит текстовое поле ввода и действие, которое указывает на **http://yourwebserver/yourrebolscript.cgi**.

```
<HTML><HEAD><TITLE>Моя форма</TITLE></HEAD><BODY>
<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">
<INPUT TYPE="TEXT" NAME="username" SIZE="25">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Пуск">
</FORM>
</BODY></HTML>
```

Чтобы извлечь информацию, включите следующий код в свою программу CGI:

```
a-word: decode-cgi system/options/cgi/query-string
```

Вы можете использовать назначенное выше слово, чтобы обратиться ко всем посланным данным. Rebol автоматически назначает слова на ввод данных в форме html. Если вы напечатаете **"Fred Thompson"** как название в текстовом поле входа в форме выше, то Rebol декодирует это как:

```
[username: "Fred Thompson"]
```

Вы можете использовать эти данные в программе CGI следующим образом:

```
#!/home/youruserpath/rebol -cs
REBOL []
print "content-type: text/html^/"
print [<HTML><HEAD><TITLE>"Титры"</TITLE></HEAD><BODY>]
username: decode-cgi system/options/cgi/query-string
print ["Привет " second username "!"]
print [</BODY></HTML>]
```

В коментариях не нуждается.

Программа выводит простую страницу, которая отображает имена всех ее прежних посетителей, позволяет вам вводить свое имя, и затем называет себя, когда вы нажимаете, запускает процесс снова:

```
#!/home/youruserpath/rebol -cs
REBOL []
print "content-type: text/html^/"
print [<HTML><HEAD><TITLE>"Моя страница"</TITLE></HEAD><BODY>]
print ["Вот список посетителей, которые побывали здесь:" ]
print [<BR>] ; prints a carriage return
either exists? %users.txt [
    newname: decode-cgi system/options/cgi/query-string
    if newname/2 <> none [
        write/append %users.txt join "<BR>" [newname/2]
    ]
    signatures: read %users.txt
    print signatures
] [write %users.txt ""]
print [<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">]
print [<BR><HR><BR>"Введите имя:"<BR>]
```

```
print ["Имя: "<INPUT TYPE="TEXT" NAME="username" SIZE="25">]
print [<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Пуск">]
print [</FORM>]
print [</BODY></HTML>]
```

Если вы помните синтаксис предыдущих двух примеров, вы находитесь на верном пути к тому, чтобы быть способным создавать динамические приложения для Интернета с Rebol. Шаблоны с примерами, являются базой для создания CGI приложений независимо от того, какой комплекс программ вы создаете:

1. Проектируйте web-страницу с формой html, которая получает ввод от пользователя. Укажите действие формы к url вашего сценария CGI.
2. Проектируйте CGI программу, чтобы обработать ввод данных через форму. Первые три строки должны быть такие же как в шаблоне. Используйте **"decode-cgi system/options/cgi/query-string"**, чтобы автоматически назначить переменные слова на данные.

Информация по Rebol CGI программированию:

http://rebol.com/docs/cgi1.html

http://rebol.com/docs/cgi2.html

http://rebol.com/docs/cgi-bbs.html

http://www.rebol.net/cookbook/recipes/0045.html

Сайт ниже содержит бесплатную программу сервера, написанную полностью на Rebol:

http://plain.at/vpavlu/plain-dev/r80v5/r80v5.html

Это обеспечивает альтернативу программирования CGI, который позволяет вам включать код Rebol непосредственно в ваши страницы html, подобно языку программирования PHP. Используя вышеупомянутый сервер, вы можете включить код Rebol, который будет работать непосредственно в ваших web-страницах, используя следующий синтаксис:

```
<?rebol {Код rebol} ?>
```

Вот пример:

```
<HTML>
<BODY>
Текущее время: <?rebol print now/time ?>.
</BODY>
</HTML>
```

CGI интерфейс даже не требуется. Это - удобная опция.

20. Parsing или синтаксический анализ

Встроенная функция **"parse"** - важная часть языка Rebol. Используется, чтобы импортировать и преобразовать организованные участки памяти внешних данных в блочный формат, который понимает Rebol. Также обеспечивает средство разделения, поиска, сравнения, распаковки, и распределения в организованную информацию из бесформатных текстовых данных. Его способности сопоставления с образцом, подобны регулярным выражениям в языке Perl и других языках.

Основной формат для синтаксического анализа:

```
parse <data> <matching rules>
```

У синтаксического анализа есть несколько режимов использования. Самый простой режим, разбивает текст разделителями и преобразовывает эти части в блок Rebol. Чтобы сделать это, не определяйте **"none"** как правило. Общие разделители, запятые, вкладки, точки с запятой, и newlines. Вот некоторые примеры:

```
text1: "яблоко апельсин персик"
parsed-block1: parse text1 none

text2: "яблоко,апельсин,персик"
parsed-block2: parse text2 none

text3: "яблоко       апельсин                персик"
parsed-block3: parse text3 none

text4: "яблоко;апельсин;персик"
parsed-block4: parse text4 none

text5: "яблоко,апельсин персик"
parsed-block5: parse text5 none

text6: {"яблоко","апельсин","персик"}
parsed-block6: parse text6 none

text7: {
яблоко
апельсин
персик
}
parsed-block7: parse text7 none
```

Все вышеупомянутые анализируемые блоки оценивают как **["яблоко" "апельсин" "персик"]**. Это полезно, потому что возможно простое импортирование файлов данных, созданных другими программами. Вы могли, например, использовать правило **"none"** чтобы импортировать файлы CSV, созданные приложением базы данных или электронной таблицей. Данные автоматически преобразованы к блокам данных Rebol для использования в ваших сценариях.

Если надо разбить файлы, основанные на некотором символе кроме общих разделителей, вы можете определить разделитель, как правило:

```
text: "яблоко*апельсин*персик"
parsed-block: parse text "*"

text: "яблоко&апельсин&персик"
parsed-block: parse text "&"

text: "яблоко    &    апельсин&персик"
parsed-block: parse text "&"
```

Вы можете также включить смешанные множественные символы, которые будут использоваться как разделители:

```
text: "яблоко&апельсин*персик"
parsed-block: parse text "&*"

text: "яблоко&апельсин*персик"
parsed-block: parse text "*&" ; порядок не имеет значения
```

Используя режим **"splitting"** синтаксического анализа отличный способ получить отформатированные таблицы данных.

Разбивая текст переводами каретки, вы сталкиваетесь с небольшой проблемой:

```
text: { Фамилия
        Имя
        Улица
        Город, Страна }

parsed-block: parse text "^/"

; ^/ символ в Rebol для перевода каретки
```

Места включены в правило синтаксического анализа по умолчанию, таким образом, вы получаете совокупность данных, это более логично, чем ранее:

```
["Фамилия" "Имя" "Улица" "Город" "Страна"]
```

Вы можете использовать **"/all"** , чтобы устранить места из правила разделителя. Код ниже:

```
text: { Фамилия
        Имя
        Улица
        Город, Страна }

parsed-block: parse/all text "^/"
```

преобразовывает данный текст в следующий блок:

```
[" Фамилия" "      Имя" "      Улица"
    "      Город, Страна "]
```

Теперь вы можете урезать дополнительное пространство в каждой строке:

```
foreach item parsed-block [trim item]
```

и получим это:

```
["Фамилия" "Имя" "Улица" "Город, Страна"]
```

Пример выше мог быть откорректирован, чтобы переместить восстановленные строки информации из любого текстового файла в ваше приложение. Но это не может сделать весь синтаксический анализ. Вы можете также использовать это, чтобы проверить, существуют ли какие-нибудь определенные данные в пределах данного блока. Чтобы сделать это, определите правило как элемент, который вы ищете. Вот пример:

```
parse ["яблоко"] ["яблоко"]
```

```
parse ["яблоко" "апельсин"] ["яблоко" "апельсин"]
```

ВАЖНО: По умолчанию, как только синтаксический анализ сталкивается с тем, что не соответствует, все выражение оценивает как лож, ДАЖЕ ЕСЛИ данное правило найдено однин или более раз в данных. Например, следующее - ложь:

```
parse ["яблоко" "апельсин"] ["яблоко"]
```

Такое поведение только значения по умолчанию. Вы можете управлять, как синтаксический анализ должен отвечать на элементы, которые не соответствуют. Добавление слов ниже к правилу возвратит истину, если данное правило будет соответствовать данным:

1. **"any"** - правило соответствует нулю данных или больше раз
2. **"some"** - правило соответствует данным один или более раз
3. **"opt"** - правило соответствует нулю данных или одно время
4. **"one"** - правило соответствует данным точно одно время
5. **an integer** - правило соответствует данным данное число раз
6. **two integers** - правило соответствует данным, неоднократно включенным в диапазон между этими двумя целыми числами

Следующие примеры - истина:

```
parse ["яблоко" "апельсин"] [any string!]
parse ["яблоко" "апельсин"] [some string!]
parse ["яблоко" "апельсин"] [1 2 string!]
```

Вы можете создать правила, которые включают множественные опции соответствия - только отделяют выборы символом **"|"** и заключают их в скобках. Следующее истинно:

```
parse ["яблоко" "апельсин"] [any [string! | url! | number!]]
```

Вы можете включить действия, чтобы они срабатывали всякий раз, когда соответствуют правилу. Заключите действие в круглые скобки:

```
parse ["яблоко" "апельсин"] [any [string!
    (alert "Блок содержит строку.") | url! | number!]]
```

Вы можете перескочить через данные, игнорируя участки памяти, пока не добираетесь до или мимо данного условия. Слово **"to"** игнорирует данные, ПОКА условие не найдено. Слово **"thru"** игнорирует данные, пока ТОЛЬКО МИМО условия. Следующее истинно:

```
parse [234.1 $50 http://rebol.com "яблоко"] [thru string!]
```

Реальное значение сопоставления с образцом - то, что вы можете искать и извлечь данные из восстановленного текста, организованным способом. Слово **"copy"** используется, чтобы назначить переменную на соответствующие данные. Например, следующий код загружает необработанный html, игнорирует все кроме текста между тэгами:

```
parse read http://rebol.com [
    thru <title> copy parsed-text to </title> (alert parsed-text)
```

```
]
```

Следующий код расширяет пример выше, чтобы обеспечить полезную особенность показа внешнего IP адреса компьютера. Читает **http:/whatismyip.com**, анализирует текст заголовка, и затем анализирует тот текст снова, чтобы возвратить только номер IP. Сетевой адрес также отображен, используя встроенный в Rebol dns протокол:

```
parse read http://whatismyip.com [
    thru <title> copy my-ip to </title>
]
parse my-ip [
    thru "-" copy stripped-ip to end
]
alert to-string rejoin [
    "Внешний: " trim/all stripped-ip "  "
    "Внутренний: " read join dns:// read dns://
]
```

Вот полезный пример, который удаляет все комментарии из данного сценария Rebol (любая часть строки, которая начинается с точки с запятой ";"). Этот код основан на сценарии в **http://www.rebol.org/library/scripts/uncomment.r.** Сначала просит имя файла, и назначает информационные наполнения того файла к переменному слову **"code"**. Используя слова **"to"** и **"thru"** , информационное наполнение тогда анализируется для любого текста, который начинается с символа ";", и до конца строки. Блок действия использует правила синтаксического анализа, встроенную функцию **"remove/part"**, чтобы удалить не нужное. Наконец, анализируемый код, посылают встроенному редактору текста Rebol для того, чтобы рассмотреть, сохранить, и т.д.:

```
code: read to-file request-file
parse/all code [any [
    to #";" begin: thru newline ending: (
        remove/part begin ((index? ending) - (index? begin))) :begin
    ]
]
editor code
```

Чтобы узнать больше, см. следующие ссылки:

http://www.codeconscious.com/rebol/parse-tutorial.html

http://www.rebol.com/docs/core23/rebolcore-15.html

http://en.wikibooks.org/wiki/REBOL_Programming/Language_Features/Parse

http://www.rebolforces.com/zine/rzine-1-06.html#sect4.

21. Объекты Rebol

Объекты Rebol - структуры кода, которые позволяют вам формировать и копировать пакеты функций и данных вместе так, чтобы они могли быть легко скопированы, многократно использованы, и сохранены как модули. Объекты Rebol обеспечивают простой способ проектировать шаблоны многократного использования связанного кода, который может быть повторен и изменен, чтобы создать новые модули кода. Объекты также обеспечивают способ создать аналоги объектов, так, чтобы любой объектный код мог служить основанием для нового объектного дизайна. Таким образом, понятие объектной имитации может быть расширено для множественных поколений, чтобы создать изменения кода, полученного из единственного шаблона.

Объекты, прежде всего, полезны в Rebol, для создания дубликатов структур данных и копий кода. Объекты также полезны тем, что они скрывают внутренние данные и функции в пределах их структуры. Они могут гарантировать, что слова, используемые в одном

месте кода, по ошибке не используются в другом месте в пределах той же самой программы. Руководящее использование слова таким образом (скрывающиеся и выставляющие определения слова в пределах различных разделов программы) упоминается как управление **"namespace"**.

Отметьте: В отличие от других языков программирования, в которых большинство кода задумано и организовано в терминах объектных конструкций, Rebol обеспечивает маленькое подмножество инструментальных средств, которые позволяют вам просто имитировать проекты кода. Классы - фундаментальная постройка, в других объектно-ориентированных языках, но они толичаются в Rebol. Если вы знакомы с другими объектно-ориентированными языками программирования, то знайте что объектная модель **("class")** свойства в Rebol просто скопирована, а не наследована. Различие - то, что изменения в **"родительской - parent"** объектной модели прототипа в Rebol не передают ее полученным дочерним объектам, когда они созданы.

Чтобы сделать объектный проект в Rebol, используйте следующий синтаксис:

```
label: make object! [object definition]
```

Объектное определение может содержать функции, значения, или данные любого типа.

Вот пример, который определяет пустой объект учетной записи пользователя. Содержит 6 переменных слов (отметьте, что каждое слово расположено каскадом вниз, чтобы не равняться **"none"**):

```
account: make object! [
    first-name:
    last-name:
    address:
    phone:
    email-address:
    none
]
```

Определение учетной записи выше просто упаковывает эти 6 переменных в контейнер, или контекст, названный **"account"**.

Вы можете обратиться к данным и функциям в пределах объекта, используя усовершенствование **("/ path")**:

```
object/word
```

В примере выше, **"account/phone"** обращается к данным телефонного номера, содержавшимся в учетной записи. Вы можете сделать изменения элементов в объекте следующим образом:

```
object/word: data
```

В примере выше:

```
account/phone: "555-1234"
account/address: "4321 Москва Останкино, Россия 54321"
```

На многих обьектных языках примечание **"dot"** используется, чтобы обратиться к элементам, содержавшимся в объекте (то есть, **"methods"** и **"properties"**), используя

формат **"object.method"**.

Как только объект создан, вы можете рассмотреть все его информационные наполнения, используя встроенное слово **"help"**:

```
help object

; or

? object

; "?" синоним для слова "help"
```

Если пример набрали в интерпритаторе Rebol, то увидите только это:

```
? account
```

Отобразит информацию:

```
ACCOUNT is an object of value:
   first-name      none!     none
   last-name       none!     none
   address         string!   "4321 Москва Останкино, Россия 54321"
   phone           string!   "555-1234"
   email-address   none!     none
```

Как только вы создали объектный прототип, вы можете сделать новый объект основанным на оригинальном определении прототипа. Новый объект будет содержать все данные и функции, определенные в оригинальном прототипе. Чтобы создать такой имитируемый объект, используйте следующий синтаксис:

```
label: make existing-object [
    значения, которые будут изменены из оригинального определения прототипа
]
```

Например, код ниже создает метку блока **"user1"** новой учетной записи. Вы можете изменить любую часть оригинального объектного блока, переопределить переменные, и т.д., следующим образом:

```
user1: make account [
    first-name: "Джон"
    last-name: "Смит"
    address: "4321 Москва Останкино, Россия 54321"
    email-address: "john@hisdomain.com"
]
```

Обратите внимание, что переменная телефонного номера не установлена в вышеупомянутом примере. В этом случае, переменная телефонного номера сохраняет значение по умолчанию **"none"** установленного в оригинальном определении учетной записи. Это один из ключевых моментов использования объектов.

В дополнение к изменению существующих переменных в объектном определении, вы можете также расширить существующее объектное определение с новыми значениями:

```
label: make existing-object [new-values]
```

Определение ниже создает новый объект учетной записи, переопределяет все существующие переменные, и прилагает новую переменную.

```
user2: make account [
    first-name: "Сергей"
    last-name: "Михаил"
    address: "4321 Москва Останкино, Россия 54321"
    phone:  "555-1234"
    email-address: "bob@mysite.net"
    favorite-color: "красный"
]
```

"user2/favorite-color" теперь обращается к **"красный"** (**"user1/favorite-color"** все еще не определен - вы получите ошибку, если вы попытаетесь обратиться к **"user2/favorite-color")**.

Код ниже создает дубликат учетной записи **user2**, с только измененным названием:

```
user2a: make user2 [
    first-name: "Павел"
    email-address: "paul@mysite.net"
]
```

"? user2a" покажет следующую информацию:

```
USER2A is an object of value:
   first-name      string!   "Павел"
   last-name       string!   "Михаил"
   address         string!   "4321 Москва Останкино, Россия 54321"
   phone           string!   "555-1234"
   email-address   string!   "paul@mysite.net"
   favorite-color  string!   "красный"
```

Вы можете создать любое число параметров пользователя, основанных на шаблоне **"account"**. Любое поле, которое не заполнено, сохранит значение по умолчанию **"none"**, установленного в определении шаблона.

С вышеупомянутыми пользовательскими определенными объектами вы можете выполнить сравнения или любые другие операции, которые ссылаются на включенные данные, такие как:

```
if user1/address = user2/address [
    print "Оба пользователя соседи."
]
```

Вы можете также включить функции в своем объектном определении:

```
account: make object! [
    first-name:
    last-name:
    address:
    phone:
    none
```

```
    email-address: does [
        return to-email rejoin [
            first-name "_" last-name "@website.com"
        ]
    ]
    display: does [
        print ""
        print rejoin ["Имя:      " first-name " " last-name]
        print rejoin ["Адрес:   " address]
        print rejoin ["Телефон:     " phone]
        print rejoin ["Email:    " email-address]
        print ""
    ]
]

user1: make account []

user2: make account [
    first-name: "Сергей"
    last-name: "Михаил"
    phone:  "555-4321"
]

user3: make account [
    first-name: "Боб"
    last-name: "Джонс"
    address: "4321 Москва Останкино, Россия 54321"
    phone:  "555-1234"
    email-address: "bob@mysite.net"
]

; распечатать все данные, содержавшиеся в каждом объекте:

user1/display user2/display user3/display
```

Важно видеть здесь, что оригинальное определение объекта учетной записи включает функцию **"display"**. Прототип учетной записи также определяет все переменные, чтобы не быть **"none"** по умолчанию. Каждый новый объектный экземпляр класса **(user1, user2, user3)** содержит весь тот код по умолчанию. В результате объект **user1**, например, является полностью функциональным объектом даже при-том, что никакие переменные или функции не были определены во время его создания. Все, что требуется, чтобы создать объект, использовать **"user1: make account []"**.

Также важно распознать, что в каждом пользовательском экземпляре класса, переменные, показанные функцией **display**, являются временными данными к приведенному примеру. Например, когда вызывают функцию **user3/display**, она печатает переменные, содержавшиеся в объекте **user3**. Ни одна из переменных, содержавшихся в **user1** или **user2**, не перепутанна, даже при-том, что они упомянуты тем же самым словом. Это одно из приемуществ объектов, группа переменных и функций формируется в легко повторяемые объекты со своим содержанием. Переменные, содержавшиеся в любом из объектов, могут быть обновлены в любое время. Отметьте, что переменная **"email-address"** первоначально назначена на функцию, которая определяет адрес электронной почты значения по умолчанию, **"first-name_last-name@website.com"**. Вы можете отменить это определение, как пользователь, просто обратиться к строке адреса электронной почты. Как только вы сделали это, функция адреса электронной почты больше не существует.

Следующий пример показывает пользу обьектов. Это маленькая игра, в которой множественные символьные объекты созданы из дублированного шаблона. Каждый объект содержит некоторый код и данные, которые позволяют пользователю перемещать несколько воображаемых символов. Основанная на пользовательском выборе, текущая позиция символа обновляется. Созданы пять экземпляров класса символов, каждый с различным начальным свойством позиции. Игрок должен найти скрытый приз и заставить один из символов двигаться в его местоположение. После каждого пользовательского перемещения выяснена позиция символьного объекта, если это соответствует позиции скрытого приза:

```
Rebol []
; Создайте случайную координату в который
```

```
; будет найден скрытый приз:

hidden-prize: random 15x15

; Вот основное "character" определение прототипа.
; Это содержит переменную, которая держит символ
; текущей позиции. Это также содержит функцию которая
; получает выбор движения от пользователя, затем обновляет
; текущую позицию символа, и сообщает,
; если найден скрытый приз .

character: make object! [
    position: 0x0
    move: does [
        direction: ask "Верх, Низ, Лево, Право:  "
        ; меняет позицию объекта, основанную на выборе пользователя
        switch/default direction [
            "верх" [position: position + -1x0]
            "низ" [position: position + 1x0]
            "лево" [position: position + 0x-1]
            "право" [position: position + 0x1]
        ] [print newline print "не верное указание!"]
        ; отобразит сообщение, если найден скрытый приз:
        if position = hidden-prize [
            print newline
            print " Вы нашли скрытый приз. ВЫ ПОБЕДИТЕЛЬ!"
            print newline
            halt
        ]
        print rejoin [
            newline
            " Вы переместили символ " movement " " direction
            ".  Символ" movement "теперь"
            hidden-prize - position
            " далеко от скрытого приза. "
            newline
        ]
    ]
]

; Теперь созданы объекты на пять символов, каждый с
; различной начальной позицией:

character1: make character[]
character2: make character[position: 3x3]
character3: make character[position: 6x6]
character4: make character[position: 9x9]
character5: make character[position: 12x12]
; Пользователь получает 20 возможностей. Во время каждого шанса,
; выполняется функция выбранного символа и
; отображается текущая позиция всех символов.

loop 20 [
    prin "^(1B)[J"
    movement: ask " Какой символ вы хотите переместить (1-5)? "
    if find ["1" "2" "3" "4" "5"] movement [
        do rejoin ["character" movement "/move"]
        print rejoin [
            newline
            " Позиция каждого символа сейчас:  "
            newline newline
            "СИМВОЛ ОДИН:   " character1/position newline
            "СИМВОЛ ДВА:   " character2/position newline
            "СИМВОЛ ТРИ: " character3/position newline
            "СИМВОЛ ЧЕТЫРЕ:  " character4/position newline
            "СИМВОЛ ПЯТЬ:  " character5/position
        ]
        ask "^/ Нажмите клавишу [Enter], чтобы продолжить."
    ]
```

```
]
```

Другое использование объектов должно избежать столкновений пространства имен, которые происходят в результате использования того же самого слова в больше чем одном контексте. Поскольку вы пишете больше кода, вы можете создать библиотеку функций для общих задач. Если вы используете то же самое слово в двух функциях, чтобы представить отдельные действия, они могут вызвать проблему, если вы импортируете и используете функции в той же самой программе. Если это случается, одна функция может неумышленно переопределить слово, которое вы используете в другой функции, и заставляете ее выступать неожиданным способом. Простое решение такого беспорядка пространства имен состоит в том, чтобы обернуть ваши библиотечные функции в объекте, и выставить служебные слова, которые вы хотите в своей программе. Таким образом, вы можете назвать функцию в контексте названия объекта, и не волноваться о столкновениях пространства имен. Например, в коде ниже, первая функция **"bank"** и **"var"** переменная записаны поверх, когда те слова переопределены во второй раз:

```
Rebol []
var: 1234.56
bank: does [
    print ""
    print rejoin ["Ваш счет в банке:  $" var]
    print ""
]

var: "КЕДР"
bank: does [
    print ""
    print rejoin [
        "Ваш любимый банк:  " var]
    print ""
]

bank
```

Оценивая **"bank"** в конце кода **"Ваш любимый банк: КЕДР"**. Нет способа обратиться к коду счета в банке в этом пункте, потому что слова были полностью переопределены. Если бы такое переопределение слов было сделано не умышленно в каком-нибудь коде, оно, вероятно, вызвало бы ошибку. Вы можете избежать проблем, просто обертывая вышеупомянутый код в отдельные объекты:

```
Rebol []
show-money: make object! [
    var: 1234.56
    bank: does [
        print ""
        print rejoin ["Ваш счет в банке:  $" var]
        print ""
    ]

]

show-place: make object! [
    var: "КЕДР"
    bank: does [
        print ""
        print rejoin [
            "Ваш любимый банк:  " var]
        print ""
    ]
]

show-money/bank
show-place/bank
```

Функция **"bank"** может использовать переменную **"var"** тем способом, которым должным образом связано с его контекстом. Обертывание кода в объектах обеспечивает простой метод организации, группирования, и поддержания переменных и функций, которые не должны быть изменены другими частями программы.

Чтобы просмотреть, как вышеупомянутые объекты могли быть имитированы и расширены, объекты ниже создают GUI, которые изменяют информационные наполнения переменных и используют функции, содержавшиеся в вышеупомянутых объектах. Добавте код ниже к выше показанному:

```
deposit: make show-money [
    view layout [
        button "Депозит $" [
            var: var + 10
            bank
        ]
    ]
]

travel: make show-place [
    view layout [
        new-favorite: field 300 trim {
            Наберите название банка, и нажмите [Enter]} [
            var: value
            bank
        ]
    ]
]
```

Как полученный объект, у объекта **"deposit"** автоматически есть доступ ко всем данным и функциям, содержавшимся в объекте показа денег, и объект **"travel"** получает полный доступ к информационным наполнениям объекта. В каждом новом объекте блок действия графического фрагмента GUI изменяет переменную **"var"** и выполняет действие **"bank"**, наследованное от объектов, созданных ранее.

Поскольку вы исследуете Rebol, вы найдете, что большая часть языка основана на объектных конструкциях. Работа с графическими символами часто требует обширной манипуляции с объектами. Название самого языка означает **"Relative Expression Based Object Language"**. Этим сказанно, объектно-ориентированные понятия не столь важны в Rebol, как в других языках. Суть Rebol состоит в том, чтобы организовать данные и функции, используя блоки, и обратиться к блокам со словами. Факт, что любой блок может автоматически быть обработан как порядковая прогрессия и управляем использующимися встроенными особенностями манипуляции списка, является фундаментальным понятием идеи Rebol. Другое отличие, которое существенно отделяет Rebol от других объектно-ориентированных языков, то, что языковые диалекты могут быть основаны на определениях слова, обеспечивая естественный интерфейс данным и функциям. Объекты просто обеспечивают дополнительную возможность формировать, копировать код и расширить встроенные объектные конструкции, которые уже существуют в языке.

Для получения дополнительной информации о том, как использовать объекты Rebol , см.:

http://rebol.com/docs/core23/rebolcore-10.html

http://en.wikibooks.org/wiki/REBOL_Programming/Language_Features/Objects

http://en.wikipedia.org/wiki/Prototype-based_programming

22. Общие ошибки

Ниже показанны решения общих ошибок, с которыми вы столкнетесь, экспериментируя с Rebol:

1) **"** Syntax Error: Script is missing a REBOL header**"** - Всякий раз, когда вы

создаете сценарий, он должен содержать минимальный необходимый заголовок наверху кода. Включайте следующий текст в начале сценария:

```
Rebol []
```

2) **"** Syntax Error: Missing] at end-of-script"** - Вы получите эту ошибку, если не поместите закрывающую квадратную скобку в конце блока. Вы будете видеть подобную ошибку для открытых строк. Код ниже даст вам ошибку, потому что отсутствует **"]"** в конце блока:

```
fruits: ["апельсин" "яблоко" "груша" "виноград"
print fruits
```

ниже правильно:

```
fruits: ["апельсин" "яблоко" "груша" "виноград"]
print fruits
```

Выравнивание блоков помогает находить и устранять эти виды ошибок.

3) **"** Script Error: request expected str argument of type: string block object none"** - Этот тип ошибки происходит, когда вы пытаетесь передать неправильный тип значения к функции. Код ниже даст вам ошибку, потому что Rebol автоматически интерпретирует переменную вебсайта как url, и функция **"alert"** требует значения строки:

```
website: http://rebol.com
alert website
```

Код ниже решает проблему, преобразовывая значение url в строку прежде, чем передать это к функции **"alert"**:

```
website: to-string http://rebol.com
alert website
```

4) **"** Script Error: word has no value"** - Проверки правописания выявится этот тип ошибки. Вы можете столкнуться с этим в любое время, если пытаетесь использовать слово, которое не определено или зарезервированно в интерпретаторе Rebol, или вами, в предыдущем коде:

```
wrod: "Hello world"
print word
```

5) **ВАЖНО**: Вот причуда Rebol, которая не выявляет ошибку, но может вызвать запутывающие результаты, особенно если вы знакомы с другими языками:

```
unexpected: [
    empty-variable: ""
    append empty-variable "*"
    print empty-variable
]
```

```
do unexpected
do unexpected
do unexpected
```

Строка:

```
empty-variable: ""
```

повторно не инициализирует переменную к пустому статусу. Вместо этого каждый раз, когда блок выполнен, **"empty-variable"** содержит предыдущее значение. Чтобы задержать переменную, как предназначено, используют слово **"copy"** следующим образом:

```
unexpected: [
    empty-variable: copy ""
    append empty-variable "*"
    print empty-variable
]

do unexpected
do unexpected
do unexpected
```

6) **Load/Save, Read/Write, Mold, Reform, etc** - другой пункт ошибок, с которым вы можете столкнуться в Rebol. Они имеют отношение к различным словам, которые читают, пишут, и форматируют данные. Сохраняя данные в файл на жестком диске, вы можете использовать **"save"** или **"write"**. **"Save"** используется, чтобы хранить данные в формате, более пригодном для использования в Rebol. **"Write"** сохраняет данные как есть. **"Load"** и **"read"** сопоставимые отношения. **"Load"** читает данные пути ввода. **"Read"** открывает данные в формате, как есть, байт в байт. Для получения дополнительной информации, см. следующие словарные статьи Rebol:

http://rebol.com/docs/words/wload.html

http://rebol.com/docs/words/wsave.html

http://rebol.com/docs/words/wread.html

http://rebol.com/docs/words/wwrite.html

Другие встроенные слова, такие как **"mold"** и **"reform"** помогают вам работать с текстом такими способами, которые являются более удобочитаемыми интерпретатором Rebol. Для полезного объяснения, см. **http://www.rebol.net/cookbook/recipes/0015.html**.

7) Порядок очередности - выражения Rebol ***всегда*** оцениваются слева направо, независимо от вовлеченных операций. Если вы хотите, чтобы определенные математические операторы были оценены первыми, они должны или быть включены в круглую скобку или помещены в начале выражения. Например:

```
2 + 4 * 6
```

то же самое как:

```
(2 + 4) * 6   ; сначала вычисленна  левая часть

== 6 * 6
```

```
== 36
```

Это противоречит другим знакомым правилам оценки. В других языках, например, умножение обычно обрабатывается перед суммированием. Так, то же самое выражение:

```
2 + 4 * 6
```

обработан как:

```
2 + (4 * 6) ; сначала выполнен оператор умножения
== 2 + 24

== 26
```

В Rebol порядок старшинства с лева на право является естественным. Признаки конца строки даже не требуются, как в других языках (точки с запятой в С-образных), смотрите следующий пример:

```
save %text.txt read http://rebol.com print sort read %text.txt
```

может быть написан более интуитивно как это:

```
save %text.txt (read http://rebol.com)
print (sort (read %text.txt))
```

Этот сценарий оценен следующим образом: функция **"save"** принимает два параметра - сначала имя файла, и данные, которые будут записаны. Те данные читаются из **http://rebol.com**. Затем функция **"print"** берет один параметр. Данные - из **text.txt** файла.

22.1 Ловушка ошибок

Есть несколько простых способов препятствовать, программе терпеть крах, когда происходит ошибка. Слова **"error?"** и **"try"** вместе, обеспечивают проверку и обработку ожидаемых ошибочных ситуаций. Например, если интернет-подключение не будет доступно, то код ниже потерпит крах:

```
html: read http://rebol.com
```

Откорректированный код ниже обработает ошибку более изящно:

```
if error? try [html: read http://rebol.com] [
    alert "Недоступен."
]
```

Слово **"attempt"** альтернатива подпрограммы **"error? try"**. Это возвращает оцененные информационные наполнения данного блока, если это успешно. Иначе возвращает **"none"**:

```
if not attempt [html: read http://rebol.com] [
    alert "Недоступен."
```

```
]
```

Дополнительный материал: http://www.rebol.com/docs/core23/rebolcore-17.html.

23. Проектирование программ

Попытка заняться программированием на пустом месте требует не только понимания того, как языковые компоненты работают и как они соединены, но и думать в терминах организации тех компонентов рабочей программы из вашего предполагаемого проекта. Пока вы не сделали это много раз, может быть трудно, сформировать программу из мысли и необработанного кода. Этот текст предназначен, чтобы дать немного понимания того, как вы можете приблизиться к организации ваших мыслей и написать код, чтобы удовлетворить ваши определенные потребности для любой ситуации.

Отличный способ начать писать код проекта состоит в том, чтобы продумать блок-схему деятельности и выписать все необходимые требования кодирования в схеме дизайна, состоящей из естественного языка. Используем разговорный английский псевдокод, для детализации кода, который вы напишете. Описать в общих чертах, как программа будет работать, выделить структуру программы, привести к псевдокоду в пределах той структуры, далее к конечному детализированному коду. Как последний шаг, отладить код, добавить или изменить функциональные возможности.

Удачи Вам!

Попытка перевода: Andrix, Andrix7@yandex.ru.