МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**Дальневосточный федеральный университет**

▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬

ООП 010503.65 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

**Сценарные языки программирования**

**РЕФЕРАТ ПО ДИСЦИПЛИНЕ**

Современные информационные технологии

Выполнил студент гр. c8504

А.B. Войцеховский

г. Владивосток

2012

Оглавление

[1. Введение 4](#_Toc343586566)

[2. Скрипт и программа 5](#_Toc343586567)

[3. Сходство и отличие сценарных языков и императивных 8](#_Toc343586568)

[4. Классификация сценарных языков 11](#_Toc343586569)

[4.1. Командно-сценарные языки 11](#_Toc343586570)

[4.2. Прикладные сценарные языки 13](#_Toc343586571)

[4.3. Языки разметки 14](#_Toc343586572)

[4.4. Универсальные языки 15](#_Toc343586573)

[5. Язык пакетной обработки 17](#_Toc343586574)

[6. Скриптовая сторона Python 19](#_Toc343586575)

[7. Язык Lua 26](#_Toc343586576)

[7.1. Преимущества 26](#_Toc343586577)

[7.2. Недостатки 28](#_Toc343586578)

[8. Язык XML 29](#_Toc343586579)

[8.1. Зачем нам нужен XML 29](#_Toc343586580)

[8.2. Отображение HTML 29](#_Toc343586581)

[8.3. Пример XML-документа 30](#_Toc343586582)

[8.4. Теги, элементы и атрибуты 30](#_Toc343586583)

[8.5. Как XML изменяет Web 31](#_Toc343586584)

[9. VBA и его применение 33](#_Toc343586585)

[10. Примеры кода некоторых распространённых языков 34](#_Toc343586586)

[10.1. Lua 34](#_Toc343586597)

[10.2. VBA 34](#_Toc343586598)

[10.3. XML 34](#_Toc343586599)

[11. Области использования 37](#_Toc343586600)

[11.1. Forth 37](#_Toc343586601)

[11.2. Lua 37](#_Toc343586602)

[11.3. JavaScript 37](#_Toc343586603)

[11.4. Perl 37](#_Toc343586604)

[11.5. PHP 38](#_Toc343586605)

[11.6. Python 38](#_Toc343586606)

[11.7. Ruby 38](#_Toc343586607)

[11.8. VBScript 39](#_Toc343586608)

[12. Список литературы 40](#_Toc343586609)

[13. Список вопросов 41](#_Toc343586610)

# Введение

Сценарный язык или скриптовый язык (англ. scripting language, в русской литературе принято название «язык сценариев») — высокоуровневый язык программирования для написания сценариев — кратких описаний выполняемых системой действий. Это язык программирования, разработанный для записи "сценариев", последовательности операций, которые пользователь может выполнять на компьютере. Разница между программами и сценариями довольно размыта. Сценарий — это программа, имеющая дело с готовыми программными компонентами [1].

Сценарии обычно интерпретируются, а не компилируются, хотя сценарные языки программирования один за другим обзаводятся JIT-компиляторам.

JIT-компиляция - Just-in-time compilation (компиляция «на лету») – технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код непосредственно во время работы программы. Таким образом достигается высокая скорость выполнения по сравнению с интерпретируемым байт-кодом (сравнимая с компилируемыми языками) за счёт увеличения потребления памяти (для хранения результатов компиляции) и затрат времени на компиляцию [15].

Скрипты могут повторить практически все операции, которые пользователь выполняет на компьютере:

* Сконфигурировать систему;
* Сконфигурировать профиль пользователя;
* Установить или удалить программу;
* Собрать статистику работы компьютера или пользователя;
* Сделать резервные копии файлов;
* Другие операции.

Характерные примеры сценарных языков программирования: VBScript, PowerScript, LotusScript, JavaScript.

# Скрипт и программа

Главная разница между скриптом и программой — в наличии у программы обширнейшей оболочки, не связанной «содержимым» программы. В зависимости от платформы, это могут быть страницы руководства, поддержка нескольких языков, наличие функционала по установке/удалению, исполнение соглашений об интерфейсе (командной строки, или иных средств взаимодействия), интерфейсы в общем реестре и т.д. Программа должна уметь работать в любой документированной среде, предусматривать различные ситуации (лучше всего с этим у программ под Unix, которые используют ./configure для определения, собственно, где они, что можно, а что нельзя на этой (очередной) платформе).

Скрипт же, в строго обратном смысле: он предназначен для решения конкретной проблемы «здесь и сейчас». Никто не ожидает от скрипта, который отсылает статистику, способности делать это одновременно на solaris'е, freeBSD и windows embedded standard с cygwin'ом на борту.

По математико-программистким представлениям, между скриптами администрирования и программами нет разницы. Они работают по одинаковым принципам, вообще говоря, выполняют почти одно и то же.

Разница между скриптом и программой — административная.

Практически любая программа имеет в себе ТРИ важные составляющие:

1. Нетривиальный алгоритм.
2. Техподдержку, наработанные лучшие практики использования, типовые схемы внедрения и готовые конфигурации.
3. Правильную интеграцию в рабочую среду в любой разрешённой (документированной) конфигурации.

Подробнее об этих составляющих:

1) Алгоритм. У любой программы есть во-первых некая идея (что, собственно, делает программа), во-вторых — обвязка. Обычно идея заключается в выполнении каких-то действий по какому-то алгоритму. Эта идея почти всегда нетривиальна. В фактическом коде это может быть меньше, чем чтение xml, но при этом именно рабочий алгоритм — суть программы. Он может быть или «обрабатывающим данные» (вроде SQL'я), или математическим (вроде md5sum), или работающим с конкретными особенностями конкретной железки (формата файла) — но он всегда требует высокой квалификации в предметной области для адекватного понимания принципов работы.

Скрипт не должен содержать в себе алгоритма в терминах «предметной области». У скрипта нет предметной области, скрипт — обвязка вокруг программ, которые уже работают с предметными областями.

2) Любая программа должна обладать известным поведением. Математики предлагают описывать поведение программы в всеобъемлющих терминах; практика же говорит, что обычно кроме алгоритма программа ещё содержит баги и фичи, которые влияют на её поведение, к которым надо адаптироваться. Адаптироваться к ним куда проще, когда есть некоторая практика использования программы.

«Данная программа плохо себя ведёт в условиях симметричной нагрузки на аплоад/даунлоад, лучше запустить две копии, каждая из которых будет работать в свою сторону симметрично» — понимание \_ПОЧЕМУ\_ потребует титанических усилий, проще принять это как данность. Чем сложнее алгоритм, тем больше жизни нужно потратить на его исследование, адаптацию и глубокое изучение. На всё жизни не хватит, значит, проще принять как данное и сконцентрироваться на важном.

Скрипт же, обратно, должен быть кристально понятен каждому, кто его посмотрит (с поправками на знание скриптового языка). Никаких (if every in self.\_\_datarange\_\_ is not in any map(\_\_systable\_\_.lang, \_\_localtable\_\_.map, lambda (a,b):[a in b or b in a for every \_\_sys\_\_.pair(a,b)])) raise "Missed i18n constitution".

3) Скрипт решает задачу здесь и сейчас. Программа решает задачу ТАМ И ВСЕГДА (с поправкой на опыт эксплуатации из п.2). Когда вы пишите скрипт, вы делаете так, чтобы оно работало в вашей системе. Оно не годится для свободного использования в других системах (хотя может быть ЛЕГКО (см п.1) адаптировано). Программа должна быть адаптируема к куче вариантов применения, реализация этой адаптации в скрипте приводит к потере его простоты и превращению его, собственно, в программу. Кроме того, но знание как правильно писать программу не эквивалентно написанию правильного алгоритма. Вы можете написать потрясающую библиотеку, но если вы не сможете запустить её на машине, у которой понедельник первый день недели (или второй — кому как повезёт), то грош цена вашей библиотеке. Необходимость думать об этом — это уже написание программ — скрипту такое допустимо (хотя и не желательно).

Ну и ещё важное отличие скриптов от программ. Программы (в форме библиотек) могут «наслаиваться» друг на друга. Этой программе нужен libYYY, которая использует libZZZ и libAAA, при этом libAAA использует libZZZ и libc. Это нормально.

Скрипты же не должны зависеть друг от друга. Ситуация, когда скрипт зависит от сервисов другого скрипта, который зависит от третьего — ненормальная.

Что же вообще должен делать хороший скрипт? Сращивать несколько программ в конкретную систему. Можно считать программы за детали конструктора. А сам конструктор — за скрипт. Если у вас в конструкторе нет квадратной пластинки с дырками по краям, то это проблема нехватки деталей. Вы можете попытаться сделать квадратную пластину из пары прямоугольных, но не следует делать её и сотни длинных полосок.

Бывает так, что скрипты перерождаются в программы. Внезапно в скрипте появляется некая логика (алгоритм), которая становится нетривиальна (и полезна). В этот момент нужно поймать это — и не полениться потратить в три раза больше времени, но сделать её программой. Обеспечить её «мясом», которое отличает программу от скрипта. Добавить сотню проверок условий, заменить все константы на конфигурируемые переменные, приготовить её для работы в «непривычных» условиях [9].

# Сходство и отличие сценарных языков и императивных

Развитием событийно управляемой концепции объектно-ориентированного подхода стало появление в 90-х годах целого класса языков программирования, которые получили название языков сценариев или скриптов. В рамках данного подхода программа представляет собой совокупность возможных сценариев обработки данных, выбор которых инициируется наступлением того или иного события (щелчок по кнопке мыши, попадание курсора в определенную позицию, изменение атрибутов того или иного объекта, переполнение буфера памяти и т.д.). События могут инициироваться как операционной системой, так и пользователем.

Основные достоинства языков данного класса унаследованы от объектно-ориентированных языков программирования. Это интуитивная ясность описаний, близость к предметной области, высокая степень абстракции, хорошая переносимость.

Широкие возможности повторного использования кода также унаследованы сценарными языками от объектно-ориентированных предков.

Естественно, что вместе с достоинствами объектно-ориентированного подхода языки сценариев унаследовали и ряд недостатков. К последним, прежде всего, относятся сложность тестирования и верификации программ и возможности возникновения в ходе эксплуатации множественных побочных эффектов, проявляющихся за счет сложной природы взаимодействия объектов и среды, представленной интерфейсами с большим количеством одновременно работающих пользователей программного обеспечения, операционной системой и внешними источниками данных.

В отношении сценарных языков уже сформировались ложные стереотипы. В частности, это касается таких критериев оценки, как компиляция/интерпретация кода, система типов, быстродействие, требования к памяти, надежность. Склонность к поддержке интерпретации, а не компиляции кода считается едва ли не первым признаком сценарных языков. Интерпретаторы проще в исполнении, нежели компиляторы, и к тому же покрывают более широкий спектр языков. Однако те же традиционные языки Лисп, Снобол, Пролог, Форт и даже «пограничный» Бейсик чаще всего имеют реализации в виде интерпретаторов. Тогда как среди сценарных языков, хоть и нечасто, но можно встретить компиляторы. Например, на платформе Microsoft .NET реализованы компиляторы Perl и Python, порождающие промежуточный MSIL-код, исполняющая (с динамической компиляцией) в рамках среды Common Language Runtime. В настоящее время все чаще используют смешанные схемы, когда код частично компилируется, частично интерпретируется (это свойственно сценарным языкам в индустрии компьютерных игр).

Принято считать, что сценарные языки либо имеют слабую типизацию, либо вообще бестиповые. Это справедливо для части языков, но далеко не для всех. Более того, можно привести примеры бестиповых языков, не являющихся сценарными, взять хотя бы тот же BCPL, прародитель языка Си.

Недостаточная надежность сценарных языков — тоже из разряда заблуждений. Так, яркий представитель сценарных языков 1990-х гг. — язык Python обладает средствами обработки исключений, построенными по образу и подобию аналогичного механизма в языке Modula-3. А ведь именно из него были заимствованы решения структурной обработки исключений (SEH), внедренные корпорацией Microsoft сначала в Си и Си++, а затем и в среду CLR (Common Language Runtime) платформы .NET.

Главная характерная черта для сценарных языков — динамическая природа, нередко позволяющая трактовать данные как программный код (и наоборот), а также простота освоения тех средств, которые буквально тут же дают видимый результат.

Взаимосвязь между программными и сценарными языками можно проследить и на модели Ершова. В соответствии с ней выделяются три взаимодополняющих друг друга вида программирования:

* синтезирующее (формирование программных фрагментов/компонентов);
* сборочное (сборка программы из готовых фрагментов/компонентов);
* конкретизирующее (адаптация многопараметрической программы к особым условиям ее применения).

Как нетрудно заметить, традиционные языки в значительной степени доминируют в синтезирующем программировании, тогда как сценарные — в конкретизирующем. Их соприкосновение происходит в сборочном программировании, которое становится пограничной зоной двух разных языковых миров. До сих пор остается непонятным, является ли программирование по своей природе индуктивным или дедуктивным процессом. Это еще одна точка различия программных и сценарных языков. Первые исповедуют скорее индуктивный подход, тогда как вторые — дедуктивный.

Как известно, многочисленные попытки создать единый всеобъемлющий язык программирования закончились неудачно. Но это касалось именно языка, а не языковой среды и не лингвистической системы. Здесь важно отметить, что А.П. Ершов был автором не менее интересной идеи — лексикона программирования, который он понимал как «лингвистическую систему с фразовой структурой, содержащую в себе формальную нотацию для выражения всех общезначимых конструкций, употребляемых при формулировании условий задачи, при синтезе и преобразовании программ». Сценарные языки на их нынешнем этапе развития, пожалуй, ближе всего к этой идее.

Что касается формы сценарных языков (их синтаксиса), то немалое влияние на нее оказал и продолжает оказывать язык Си. Заимствование содержательных идей сценарными языками ведется по широкому фронту:

* языки функционального программирования (Лисп, Scheme);
* языки обработки строк (Snobol, Icon);
* объектно-ориентированные языки (Smalltalk, Java, Eiffel, C#);
* языки управления заданиями, командные языки (csh, Rexx);
* языки управления средой (Tcl, VBA);
* языки разметки/макрообработки (SGML, TeX);
* языки моделирования дискретных систем (GPSS, SIMSCRIPT, SLAM II).

Так же для проявление воздействия программных (традиционных) языков на проектирование и развитие сценарных языков можно выделить три направления:

1. использование ООП (практически любой новый сценарный язык поддерживает объектную модель);
2. использование идей старых языков (Лисп, Снобол; языки функционального программирования и языки обработки строк по своей динамической природе наиболее близки сценарным);
3. появление языка Java (ряд сценарных языков наиболее тесно интегрируется именно с Java) [2].

# Классификация сценарных языков

Существуют разные подходы к классификации языков программирования. Все они в той или иной мере упрощают реальную картину и охватывают лишь отдельные характеристики языков. Сложность классификации понятна: 50 лет эволюции языков программирования привели к тому, что взаимопроникновение концепций языков, которые используют различные модели и парадигмы, достигло едва ли не своего апогея. Почти каждый новый язык представляет собой «гремучую смесь» разных концепций и механизмов. Одной из наиболее примечательных является классификация моделей языков, предложенная Дж. Бэкусом в 1977 г. В соответствии с ней выделяются три категории языков:

1. Простые операционные модели (языки, основанные на конечных автоматах, машине Тьюринга);
2. Аппликативные модели (языки на основе лямбда-исчисления Чёрча, системы комбинаторов Карри, чистого Лиспа);
3. Модели фон Неймана (традиционные языки программирования).

С точки зрения такой классификации сценарные языки ближе всего к категории B. Если составить несколько таблиц, куда будут сгруппированы наиболее значимые и известные языки, которые по тем или иным причинам можно назвать сценарными, то получится четыре класса таких языков [2]:

1. командно-сценарные;
2. прикладные сценарные;
3. языки разметки;
4. универсальные сценарные.

## Командно-сценарные языки

Появились ещё в 1960-х годах для управления заданиями в операционных системах, когда возникла острая потребность в языках, обеспечивающих управление программами, иначе говоря, языках управления заданиями. Из языков того времени наиболее известен JCL для OS/360. В этот класс входят языки пакетной обработки (batch language) и языки командных оболочек, например, sh, csh для Unix. Эти языки чаще всего используются в пакетном режиме обработки [1].

* JCL
* sh
  + bash
  + csh
  + ksh
* Pilot[7]
* REXX
* AppleScript
* COMMAND.COM и cmd.exe
* Visual Basic Script

Например, язык AppleScript операционной системы MacOS имеет редактор Script Editor, который позволяет записывать действия по мере их выполнения пользователем в системе в файл сценария (текстовый файл) и оформлять в виде исполняемой программы. Такой подход позволяет составлять простейшие сценарии непрограммирующим пользователям [1].

Pilot поддерживает всего два типа данных (строки и числа) и имеет крайне ограниченный набор команд (TYPE, ACCEPT, MATCH, JUMP, USE, COMPUTE, END, YES, NO) [2].

К командно-сценарному классу относятся многочисленные интерпретаторы команд CLI (command language interpreter), так называемые языки пакетной обработки (batch language) и языки для построения системных командных оболочек (яркий пример — sh, csh и их вариации для UNIX). Как правило, такие языки ориентируются не на интерактивный, а на пакетный режим обработки, когда участие человека на этапе выполнения сведено к нулю и все работает в непрерывном потоке. Эти языки не только могли непосредственно взаимодействовать с соответствующей операционной системой, но и снабжались средствами грамматического разбора программ и трансформации данных. С их помощью можно было создавать различные программные фильтры, используемые, в частности, в конвейере (pipe) системы UNIX. В число известных языков такого типа входят Awk, впервые появившийся в AT&T UNIX Version 7, а теперь ставший частью стандарта POSIX Command Language and Utilities. К ключевым особенностям языка, нашедшим впоследствии широкое применение в среде универсальных сценарных языков, можно отнести механизм регулярных выражений, без которого разбор текста производится не очень эффективно [2].

Языки этого класса ориентировались также на обработку системных событий, генерирование текста и высокоуровневый доступ к базам данных. Здесь стоит отметить язык RPG (Report Program Generator). Он до сего времени успешно применяется для создания отчетов из корпоративных БД, работающих преимущественно на мэйнфреймах (RPG/400 для компьютеров IBM AS/400).

Наиболее активно из языков этого класса в наши дни используется Rexx, созданный в исследовательских лабораториях IBM. По набору средств он мало чем отличается от универсальных сценарных языков, однако выполнен в виде классического блочно-структурированного процедурного языка и предназначен преимущественно для интеграции и расширения функциональности приложений [2].

## Прикладные сценарные языки

Сценарные языки этого типа начали появляться в 1980-е годы, когда на промышленных персональных компьютерах стало возможным интерактивное общение с ОС. В клиент-серверной архитектуре такие языки работали в клиентской части программного обеспечения.

* AutoLISP
* Emacs Lisp
* ERM
* Game Maker Language
* MQL4 script
* UnrealScript
* VBA
* LotusScript

К этой категории можно отнести JavaScript и его диалекты (JScript, ECMAScript) [1].

Прикладные сценарные языки зарождались в 1980-е гг., в эпоху появления промышленных ПК, когда на первый план стали выходить задачи интерактивного общения с ОС, а также доступа к данным электронных таблиц и БД. Отличительная особенность сценарных языков данного класса — ориентация на клиентскую часть ПО.

Использование объектной модели в языках данного класса уже становится нормой, а не исключением. Их еще нельзя назвать полноправными языками ООП, однако они в значительной мере стараются воспользоваться удобствами объектного подхода.

Среди прикладных сценарных языков резко выделяется Visual Basic, в том числе и такая его разновидность для офисного программирования, как VBA (Visual Basic for Applications). Visual Basic — это тот самый «пограничный» язык, который скорее относится к сценарным, чем к традиционным. Он во многом задал тон такому классу языков, как прикладные сценарные. Более того, работа с пользовательским интерфейсом и встраивание программных компонентов (VBX, OCX, ActiveX) стали едва ли не визитной карточкой данного языка. Стоит заметить, что в ходе эволюции VBA поглотил другие специфические языки, в частности Word Basic и Excel Macro Language, взяв на себя их задачи. Под его влиянием были созданы такие языки, как VBScript (особый диалект языка Visual Basic, ориентированный на создание OLE-компонентов и на работу в рамках браузеров) и LotusScript (своего рода аналог языков VBA и CorelScript, предназначенный для решения задач автоматизации офиса в рамках Lotus Notes).

Несколько особо в этом ряду стоит JavaScript, ставший стандартом де-факто в Web-программировании при реализации клиентской части. Его прототипом был язык LiveScript, являвшийся частью серверного продукта LiveWire компании Netscape и первоначально встроенный в Netscape Navigator 2.0. После появления языка Java корпорации Sun Microsystems он начал играть роль самодостаточной надстройки над этим языком, и его название сменилось на JavaScript. Диалекты этого языка — JScript корпорации Microsoft и ECMAScript (стандарт ECMA-262).

Из новичков в данном классе языков упомянем экспериментальный язык Pnuts, основная идея которого — дать в рамках сценариев наиболее полный доступ к Java API. Его можно использовать для самых разных задач, но прежде всего для оперирования компонентным ПО [2].

## Языки разметки

Языки разметки, или тегированные языки, стоят несколько поодаль от магистральной линии развития сценарных языков. Им ближе всего по своей природе системы макрообработки (всевозможные макропроцессоры), столь популярные в 1960—1970-е гг. Их главная отличительная черта — встраивание специального кода (в виде обособленных «команд» — тегов) непосредственно в обычные тексты. Им родственны такие языки, как Postscript и RTF (чаще воспринимаемый просто как особый формат представления документов). Теги стали использоваться для самых разных целей: для отделения структуры информации от ее содержания, для вкрапления команд форматирования и даже для задания динамического поведения встроенных в документ интерактивных объектов.

* GML
* TeX
* SGML
* XML

Идея отделения структуры информации от содержания возникла давно. Но, пожалуй, первым осознанным решением стал запуск проекта GenCode. В сентябре 1967 г. Уильям Танниклифф, председатель Комитета по композиционным решениям ассоциации GCA (Graphic Communications Association), предложил провести четкую границу между содержанием и форматом представления информации. Руководитель GCA Норман Шарпф принял решение приступить к реализации соответствующего проекта GenCode.

Наиболее значительными достижениями в области языков разметки стали TeX, HTML и XML. Язык TeX (1979) Дональда Кнута на три года опередил Postscript, созданный Джоном Уорноком и др. в компании Adobe и предназначенный для точного описания внешней формы документов с композицией произвольной сложности. В отличие от низкоуровневого Postscript, язык TeX ориентировался на работу пользователей, не имеющих навыков программирования. Наиболее широко этот язык стал применяться в научной среде, где предъявляются самые высокие требования к качеству построения формул сложной структуры.

Язык HTML, с появлением которого понятие «гипертекст» стало простым и обыденным, создавался на базе SGML путем максимального упрощения его структуры и свойств.

Метаязык XML, созданный во многом с подачи Джона Босака (Sun Microsystems), руководителя рабочей группы SGML ERB, также основывался на SGML, но для него характерно куда более бережное отношение к идеям своего предка, да и выполняет он совсем иную роль, нежели HTML. Это своего рода язык транспортирования и промежуточного хранения данных при обмене ими между разнородными и распределенными системами. На его основе можно проводить сколь угодно сложные преобразования документов и текстовой информации, а главное, в унифицированном виде хранить данные реляционно-иерархической структуры, в том числе по настройкам и программированию компонентов. Это, правда, не мешает использовать его и не совсем по назначению — для задания динамики поведения всевозможных объектов. В 2001 г. появилась ревизия HTML, которая получила название XHTML, где были учтены требования XML [2].

## Универсальные языки

Этот тип сценарных языков наиболее известен, особенно в применении к программированию веб-приложений. Языки этого типа стали возникать с 1990-х годов.

* Tcl (Tool command language)
* Lua
* Perl
* PHP
* Python
* REBOL
* Ruby

Следует заметить, что многие языки этой категории имеют более широкое применения, чем в качестве просто языков сценариев [1].

# Язык пакетной обработки

Пакетные или batch файлы – это обычные текстовые файлы, содержащие наборы команд интерпретатора и имеющие расширение bat или cmd (cmd работают только в ОС семейства NT). Редактировать такие файлы можно при помощи блокнота или любого другого текстового редактора [5].

В основном bat-файлы применяется для копирования и удаления файлов, составления отчетов, запуска других программ и скриптов. Наиболее часто используемые команды:

* копирование файлов и каталогов: copy, xcopy;
* перемещение файлов: move;
* удаление файлов и каталогов: del, rd;
* создание каталогов: md;
* перемещение по файловой системе: cd;
* информация о содержимом файлов и каталогов: type, dir;
* запуск оболочки и выход из нее: cmd, exit;
* вызов внешних приложений, циклы и условия: call, for, goto, if, set;
* дополнительные команды: at, net, set, cls.

Чтобы из команд получился сценарий, нужно построчно написать список команд в текстовый файл и сохранить этот файл с расширением "bat".

Пакетные файлы полезны для автоматического запуска приложений. Основная область применения — автоматизация наиболее рутинных операций, которые регулярно приходится совершать пользователю компьютера: например, копирование, перемещение, переименование, удаление файлов; работа с папками; архивация; создание резервных копий баз данных и т. п.

Пакетные файлы поддерживают операторы if, goto и for (в системах семейства Windows NT его возможности существенно расширены), что позволяет обрабатывать результаты выполнения предыдущих команд или приложений и в зависимости от этого выполнять дальше тот или иной блок команд (как правило, в случае удачного завершения приложение возвращает 0 в переменной errorlevel; в случае неудачного — 1 или большее значение).

Пакетные файлы могут содержать как внутренние команды, обрабатываемые непосредственно COMMAND.COM или CMD.EXE, так и обращения к внешним утилитам, существующим в виде отдельных программ (файлов .EXE либо любых других исполнимых модулей).

Пример пакетного файла, вычисляющего выражения [1]:

@echo off

:begin

Cls

Title Калькулятор

Color 71

Echo Введите уравнение:

Set /P exp=

Set /A result=%exp%

Title Вычислено

Echo Ваше уравнение: %exp%

Echo Решение: %result%

Pause>nul

goto begin

1. Выключение эха (вывода выполняющихся строк на экран).
2. Метка, создающая цикличность программы.
3. Очистка экрана.
4. Изменение заголовка окна Windows на строку «Калькулятор».
5. Изменение цвета шрифта и фона (тёмно-синий на светло-сером).
6. Вывод строки «Введите уравнение».
7. Создание переменной exp для хранения ввода пользователя.
8. Вычисление результата выражения и помещение его в переменную result.
9. Изменение заголовка окна на «Вычислено».
10. Вывод строки «Ваше уравнение» и значения переменной exp.
11. Вывод строки «Решение» и значения переменной result.
12. Остановка выполнения кода до нажатия любой клавиши.
13. Переход на метку begin, выполнение кода начнётся со следующей после метки команды.

# Скриптовая сторона Python

Классический путь создания приложений для WWW - написание CGI-сценариев (иногда говорят - скриптов). CGI (Common Gateway Interface, общий шлюзовой интерфейс) - это стандарт, регламентирующий взаимодействие сервера с внешними приложениями. В случае с WWW, web-сервер может направить запрос на генерацию страницы по определенному сценарию. Этот сценарий, получив на вход данные от web-сервера (тот, в свою очередь, мог получить их от пользователя), генерирует готовый объект (изображение, аудиоданные, таблицу стилей и т.п.).

При вызове сценария Web-сервер передает ему информацию через стандартный ввод, переменные окружения и, для ISINDEX, через аргументы командной строки (они доступны через sys.argv).

Два основных метода передачи данных из заполненной в браузере формы Web-серверу (и CGI-сценарию) - GET и POST. В зависимости от метода данные передаются по-разному. В первом случае они кодируются и помещаются прямо в URL, например: http://host/cgi-bin/a.cgi?a=1&b=3. Сценарий получает их в переменной окружения с именем QUERY\_STRING. В случае метода POST они передаются на стандартный ввод. Для корректной работы сценарии помещаются в предназначенный для этого каталог на web-сервере (обычно он называется cgi-bin) или, если это разрешено конфигурацией сервера, в любом месте среди документов HTML. Сценарий должен иметь признак исполняемости. В системе Unix его можно установить с помощью команды chmod a+x.

Следующий простейший сценарий выводит значения из словаря os.environ и позволяет увидеть, что же было ему передано:

#!/usr/bin/python

import os

print """Content-Type: text/plain

%s""" % os.environ

С помощью него можно увидеть установленные Web-сервером переменные окружения.

Выдаваемый CGI-сценарием web-серверу файл содержит заголовочную часть, в которой указаны поля с мета-информацией (тип содержимого, время последнего обновления документа, кодировка и т.п.) [13, с. 115].

Основные переменные окружения, достаточные для создания сценариев:

QUERY\_STRING

Строка запроса.

REMOTE\_ADDR

IP-адрес клиента.

REMOTE\_USER

Имя клиента (если он был идентифицирован).

SCRIPT\_NAME

Имя сценария.

SCRIPT\_FILENAME

Имя файла со сценарием.

SERVER\_NAME

Имя сервера.

HTTP\_USER\_AGENT

Название броузера клиента.

REQUEST\_URI

Строка запроса (URI).

HTTP\_USER\_AGENT

Имя сервера.

HTTP\_ACCEPT\_LANGUAGE

Желательный язык документа.

Вот что может содержать словарь os.environ в CGI-сценарии:

{

'DOCUMENT\_ROOT': '/var/www/html',

'SERVER\_ADDR': '127.0.0.1',

'SERVER\_PORT': '80',

'GATEWAY\_INTERFACE': 'CGI/1.1',

'HTTP\_ACCEPT\_LANGUAGE': 'en-us, en;q=0.50',

'REMOTE\_ADDR': '127.0.0.1',

'SERVER\_NAME': 'rnd.onego.ru',

'HTTP\_CONNECTION': 'close',

'HTTP\_USER\_AGENT': 'Mozilla/5.0 (X11; U; Linux i586; en-US;

rv:1.0.1) Gecko/20021003',

'HTTP\_ACCEPT\_CHARSET': 'ISO-8859-1, utf-8;q=0.66, \*;q=0.66',

'HTTP\_ACCEPT': 'text/xml,application/xml,application/xhtml+xml,

text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,

image/gif;q=0.2,text/css,\*/\*;q=0.1',

'REQUEST\_URI': '/cgi-bin/test.py?a=1',

'PATH': '/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin',

'QUERY\_STRING': 'a=1&b=2',

'SCRIPT\_FILENAME': '/var/www/cgi-bin/test.py',

'HTTP\_KEEP\_ALIVE': '300',

'HTTP\_HOST': 'localhost',

'REQUEST\_METHOD': 'GET',

'SERVER\_SIGNATURE': 'Apache/1.3.23 Server at rnd.onego.ru Port 80',

'SCRIPT\_NAME': '/cgi-bin/test.py',

'SERVER\_ADMIN': 'root@localhost',

'SERVER\_SOFTWARE': 'Apache/1.3.23 (Unix) (Red-Hat/Linux)

mod\_python/2.7.8 Python/1.5.2 PHP/4.1.2',

'SERVER\_PROTOCOL': 'HTTP/1.0',

'REMOTE\_PORT': '39251'

}

Следующий CGI-сценарий выдает черный квадрат (в нем используется модуль Image для обработки изображений) [13, с. 116]:

#!/usr/bin/python

import sys

print """Content-Type: image/jpeg

"""

import Image

i = Image.new("RGB", (10,10))

i.im.draw\_rectangle((0,0,10,10), 1)

i.save(sys.stdout, "jpeg")

Модуль cgi

В Python имеется поддержка CGI в виде модуля cgi. Следующий пример показывает некоторые из его возможностей [13, с. 117]:

#!/usr/bin/python

# -\*- coding: cp1251 -\*-

import cgi, os

# анализ запроса

f = cgi.FieldStorage()

if f.has\_key("a"):

a = f["a"].value

else:

a = "0"

# обработка запроса

b = str(int(a)+1)

mytext = open(os.environ["SCRIPT\_FILENAME"]).read()

mytext\_html = cgi.escape(mytext)

# формирование ответа

print """Content-Type: text/html

<html><head><title> Решение примера : %(b)s = %(a)s + 1</title></head>

<body>

%(b)s

<table width="80%%"><tr><td>

<form action="me.cgi" method="GET">

<input type="text" name="a" value="0" size="6">

<input type="submit" name="b" value=" Обработать ">

</form></td></tr></table>

<pre>

%(mytext\_html)s

</pre>

</body></html>""" % vars()

В этом примере к заданному в форме числу прибавляется 1. Кроме того, выводится исходный код самого сценария. Следует заметить, что для экранирования символов >, <, & использована функция cgi.escape(). Для формирования Web-страницы применена операция форматирования. В качестве словаря для выполнения подстановок использован словарь vars() со всеми локальными переменными. Знаки процента пришлось удвоить, чтобы они не интерпретировались командой форматирования. Стоит обратить внимание на то, как получено значение от пользователя. Объект FieldStorage "почти" словарь, с тем исключением, что для получения обычного значения нужно дополнительно посмотреть атрибут value. Дело в том, что в сценарий могут передаваться не только текстовые значения, но и файлы, а также множественные значения с одним и тем же именем.

В примере выше проверка на допустимость произведена при вызове функции int(): если было бы задано нечисловое значение, сценарий аварийно завершился, а пользователь увидел Internal Server Error.

После анализа входных данных можно выделить фазу их обработки. В этой части CGI-сценария вычисляются переменные для дальнейшего вывода. Здесь необходимо учитывать не только значения переданных переменных, но и факт их присутствия или отсутствия, так как это тоже может влиять на логику сценария.

И, наконец, фаза вывода готового объекта (текста, HTML-документа, изображения, мультимедиа-объекта и т.п.). Проще всего заранее подготовить шаблон страницы (или ее крупных частей), а потом просто заполнить содержимым из переменных. В приведенных примерах имена появлялись в строке запроса только один раз. Некоторые формы порождают несколько значений для одного имени. Получить все значения можно с помощью метода getlist():

lst = form.getlist("fld")

Список lst будет содержать столько значений, сколько полей с именем fld получено из web-формы (он может быть и пустым, если ни одно поле с заданным именем не было заполнено).

В некоторых случаях необходимо передать на сервер файлы (сделать upload). Следующий пример и комментарий к нему помогут разобраться с этой задачей [13, с. 118]:

#!/usr/bin/env python

import cgi

form = cgi.FieldStorage()

file\_contents = ""

if form.has\_key("filename"):

fileitem = form["filename"]

if fileitem.file:

file\_contents = """<P> Содержимое переданного файла :

<PRE>%s</PRE>""" % fileitem.file.read()

print """Content-Type: text/html

<HTML><HEAD><TITLE> Загрузка файла </TITLE></HEAD>

<BODY><H1> Загрузка файла </H1>

<P><FORM ENCTYPE="multipart/form-data"

ACTION="getfile.cgi" METHOD="POST">

<br> Файл : <INPUT TYPE="file" NAME="filename">

<br><INPUT TYPE="submit" NAME="button" VALUE=" Передать файл ">

</FORM>

%s

</BODY></HTML>""" % file\_contents

1. В начале следует рассмотреть web-форму, которая приведена в конце сценария: именно она будет выводиться пользователю при обращении по CGI-сценарию. Форма имеет поле типа file, которое в web-броузере представляется полоской ввода и кнопкой "Browse". Нажимая на кнопку "Browse", пользователь выбирает файл, доступный в ОС на его компьютере. После этого он может нажать кнопку "Передать файл" для передачи файла на сервер.

Для отладки CGI-сценария можно использовать модуль cgitb. При возникновении ошибки этот модуль выдаст красочную HTML-страницу с указанием места возбуждения исключения.

В начале отлаживаемого сценария нужно поставить

import cgitb

cgitb.enable(1)

Или, если не нужно показывать ошибки в браузере:

import cgitb

cgitb.enable(0, logdir="/tmp")

Только необходимо помнить, что следует убрать эти строки, когда сценарий будет отлажен, так как он выдает кусочки кода сценария. Это может быть использовано злоумышленниками, с тем чтобы найти уязвимости в CGI-сценарии или подсмотреть пароли (если таковые присутствуют в сценарии) [13, с. 119].

# Язык Lua

Lua - интерпретируемый язык программирования, разработанный подразделением Tecgraf Католического университета Рио-де-Жанейро (Computer Graphics Technology Group of Pontifical Catholic University of Rio de Janeiro in Brazil). Является свободно распространяемым, с открытыми исходными текстами на языке Си [16].

По возможностям, идеологии и реализации язык ближе всего к JavaScript, однако Lua отличается более мощными и гораздо более гибкими конструкциями, спроектирован с целью «не плодить сущности сверх необходимого». Хотя Lua не содержит понятия класса и объекта в явном виде, механизмы объектно-ориентированного программирования с поддержкой прототипов (включая множественное наследование) реализуются с использованием метатаблиц, которые также позволяют перезагрузку операций и т. п. [16]

Синтаксис Lua можно описать как нечто среднее между Си (JavaScript) и Паскалем. Подобно таким скриптовым языкам, как Python и Icon, допускает многократные присваивания. В простейшем случае это позволяет писать выражения вида:

x, y = y, x

Программа «Hello, world» в простейшем случае выглядит так:

print "Hello, world!"

В настоящее время используется в различных проектах, где требуется встроить достаточно быстрый и нетрудный в освоении скриптовый язык программирования - например, в разработке игр, где LUA часто используется между игровым «движком» и данными для написания сценариев поведения/взаимодействия объектов [6].

## Преимущества

Простота синтаксиса видна с первого взгляда:

table\_1 = {} --создание пустой таблицы

table\_2 = table\_1 -- таблицы всегда передаются по ссылке

table\_2[length] = 99

print( table\_2["length"] ) -- выводится значение 99

print( table\_1["length"] ) -- выводится значение 99

Если в качестве ключей использовать строки, то таблицы можно применять, как структуры. В Lua для этого даже предусмотрен соответствующий синтаксис:

box = { length = 88, width = 55 }

print( box["width"] ) -- выводится значение 55

print( box.width ) -- тот же результат, но обращение, "как к структуре"

Массивы организовать ещё проще. По умолчанию в следующей форме записи:

week = { "Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Вс" }

перечисляемые строки являются значениями, которым автоматически присваиваются целочисленные ключи, начиная с 1.

Давняя мечта программистов – обмен значениями двух переменных без участия третьей:

first, second = second, first

Дело в том, что во время операций присваивания в первую очередь вычисляются выражения правой части в порядке их записи, а затем производится последовательное присваивание всем переменным из левой части. Так что, в действительности, при обмене значениями здесь задействованы даже две временных переменных, просто этот механизм скрыт от пользователя Lua [12].

Как и многие интерпретируемые языки программирования, реализация LUA имеет отдельно компилятор с исходного языка в исполняемый байт-код и виртуальную машину для исполнения сгенерированного байт-кода. Необходимо отметить компактность байт-кода: в генерируемом коде используется всего 35 команд. Причем байт-код - это не команды стековой машины, а команды некоего виртуального процессора с несколькими регистрами, что повышает эффективность исполнения. В стандартной виртуальной машине LUA используется распределение памяти со сборкой мусора (аналогично Java).

Для задач критичных по времени имеется jit-компилятор LUA LuaJIT. LUA использует единый строковой пул, что позволяет снизить расходы памяти на хранение строк [16].

Язык Lua является контекстно-свободным, однако по причинам производительности для его исполнения используется JIT-компиляция, а не потоковая интерпретация. Таким образом достигается высокая скорость выполнения (сравнимая с компилируемыми языками) за счёт увеличения потребления памяти (для хранения результатов компиляции) и затрат времени на компиляцию. JIT базируется на двух более ранних идеях, касающихся среды исполнения: компиляции байт-кода и динамической компиляции [6].

Lua имеет достаточно эффективную реализацию. Независимые бенчмарки показывают Lua как один из быстрейших языков в области скриптовых интерпретируемых языков [11].

Lua разработан как переносимый язык. Со слов авторов, «Когда мы говорим о переносимости, мы не говорим о запуске Lua под Windows и Unix. Мы говорим о запуске Lua на всех платформах, о которых мы когда-либо слышали: PlayStation, XBox, Mac OS-9 and OS X, BeOS, QUALCOMM Brew, MS-DOS, IBM mainframes, RISC OS, Symbian OS, PalmOS, ARM процессорах, Rabbit процессорах, плюс, конечно, все разновидности Unix и Windows [11].»

## Недостатки

Среди недостатков языка можно назвать полное отсутствие контроля типов, что приводит к неприятным ошибкам, а также вынуждает проверять возможность приведения типов вручную в управляющем коде. Любая переменная может менять свой тип в процессе исполнения программы, что еще более усложняет типизацию. Язык применяет очень неудобные правила областей видимости переменных. Переменная считается доступной и сохраняет свое значение в любой точке функции с ее первого упоминания (что является объявлением), вне зависимости от вложенности секций. Одним из основных недостатков языка можно назвать отсутствие встроенной поддержки интерфейса ООП, что приходится решать сложным использованием метатаблиц. Кроме того, передача параметров в функции управляющей программы и из них осуществляется через стек LUA, что происходит без проверки количества переданных параметров и, разумеется, типа. Нет возможности хранить в LUA-контексте переменные пользовательских типов, что приводит к необходимости программирования специальных менеджеров, связывающих метатаблицы LUA, указатель на объект в памяти, хранимый в виде числа в LUA-контексте и сами объекты, учитывая ручной контроль жизни этих объектов и ручной вызов деструкторов. Кроме того, отсутствует возможность управления внутренними ресурсами LUA из управляющей программы (например, очистка частей скрипта и пулов переменных). В языке нет пространств имен и каких бы то ни было их аналогов [6].

# Язык XML

XML или Extensible Markup Language (Расширяемый Язык Разметки), является языком разметки, который вы можете использовать для создания ваших собственных тегов. Он был создан в World Wide Web Consortium (W3C) для преодоления ограничений языка HTML, Hypertext Markup Language (Гипертекстовый Язык Разметки), который является основой всех Web-страниц. Как и HTML, XML базируется на SGML - Standard Generalized Markup Language (Стандартный Обобщенный Язык Разметки). Хотя SGML десятилетиями использовался в издательском деле, он представляется сложным, что отпугивает многих людей, которые могли бы его использовать (SGML также расшифровывается как "Sounds great, maybe later" - "Звучит великолепно, может быть, позже"). XML был разработан с прицелом на Web [17].

## Зачем нам нужен XML

HTML - наиболее успешный язык разметки всех времен. Вы можете просмотреть простейшие теги HTML практически на любом устройстве от PDA до мейнфрейма и вы можете даже преобразовать разметку HTML в голос и в другие форматы при помощи соответствующих инструментов. При таком успехе HTML, почему же W3C создал XML? Чтобы ответить на этот вопрос, взглянем на такой документ [7]:

<p><b>Mrs. Mary McGoon</b>

<br>

1401 Main Street

<br>

Anytown, NC 34829</p>

Беда HTML состоит в том, что он был разработан с прицелом на человека. Даже не просматривая приведенный выше HTML-документ в браузере можно понять, что это чей-то почтовый адрес. (В частности, это почтовый адрес в Соединенных Штатах.)

Люди понимают значение и назначение большинства документов. Машина, к сожалению, сделать этого не может. Теги в этом документе говорят браузеру, как отображать информацию, но теги не говорят браузеру, что это за информация.

## Отображение HTML

Чтобы отобразить HTML, браузер просто следует инструкциям в HTML-документе. Тег параграфа (<p>) сообщает браузеру, что нужно отобразить новую строку, обычно, с пропуском строки перед ней, а два тега разрыва (<br>) сообщают браузеру, что нужно перейти на новую строку без пропусков между строками. Браузер прекрасно форматирует документ, но машина все же не знает, что это адрес [7].

## Пример XML-документа

Теперь рассмотрим пример XML-документа. В XML можно назначить некоторое значение тегам в документе. Что еще более важно, эту информацию также легко обработать для машины. Можно выделить почтовый код из этого документа просто найдя содержимое, обрамленное тегами <postal-code> и </postal-code>, называемое элементом <postal-code> [7].

<address>

<name>

<title>Mrs.</title>

<first-name>

Mary

</first-name>

<last-name>

McGoon

</last-name>

</name>

<street>

1401 Main Street

</street>

<city>Anytown</city>

<state>NC</state>

<postal-code>

34829

</postal-code>

</address>

## Теги, элементы и атрибуты

Есть три общих термина, используемых для описания частей XML-документа: теги, элементы и атрибуты. Вот пример документа, иллюстрирующего эти термины [7]:

<address>

<name>

<title>Mrs.</title>

<first-name>

Mary

</first-name>

<last-name>

McGoon

</last-name>

</name>

<street>

1401 Main Street

</street>

<city state="NC">Anytown</city>

<postal-code>

34829

</postal-code>

</address>

**Тег** - это текст между левой угловой скобкой (<) и правой угловой скобкой (>). Есть начальные теги (такие, как <name>) и конечные теги (такие, как </name>)

**Элементом** является начальный тег, конечный тег и все, что есть между ними. В примере выше элемент <name> содержит два дочерних элемента: <title>, <first-name> и <last-name>.

**Атрибут** - это пара имя-значение внутри начального тега элемента. В данном примере state является атрибутом элемента <city>; в предыдущем примере <state> был элементом.

## Как XML изменяет Web

Теперь, когда видно, как разработчики могут использовать XML, чтобы создавать документы с данными, описывающими сами себя, посмотрим как люди используют эти документы, чтобы усовершенствовать Web. Вот несколько ключевых направлений [7]:

* XML упрощает обмен данными. Поскольку разные организации (или даже разные части одной организации) редко используют единый стандартизированный набор инструментов, для приложений соединение может требовать некоторого объема работы. При использовании XML каждая группа создает свою утилиту, которая преобразует ее внутренние форматы данных в XML и наоборот. Лучше всего, если производители программного обеспечения уже обеспечили инструменты для преобразования из записей базы данных (или каталогов LDAP, или заказов на покупки и т.д.) в и из XML.
* XML создает возможность изящного кода. Поскольку XML-документы могут быть структурированы для идентификации каждой важной части информации (а также и отношений между частями), возможно написать код, который может обрабатывать эти XML-документы без участия человека. То, что производители программного обеспечения вкладывают большие объемы времени и денег в построение средств разработки XML, означает, что написание такого кода - относительно простой процесс.
* XML создает возможность "умного" поиска. Хотя поисковые машины с годами неуклонно совершенствуются, все еще трудно получить результаты поиска без ошибок. Если вы ищете страницы HTML для чего-то, под названием "Chip", вы можете найти страницы про шоколадные плитки (chocolate chips), компьютерные чипы (computer chips), древесно-стружечные плиты (wood chips) и много других бесполезных образцов. Поиск XML-документов по элементам <first-name>, которые содержат текст Chip, даст вам значительно лучший набор результатов.

# VBA и его применение

Visual Basic for Applications (VBA, Visual Basic для приложений) — немного упрощённая реализация языка программирования Visual Basic, встроенная в линейку продуктов Microsoft Office (включая версии для Mac OS), а также во многие другие программные пакеты, такие как AutoCAD, SolidWorks, CorelDRAW, WordPerfect и ESRI ArcGIS [8].

К достоинствам языка можно отнести сравнительную лёгкость освоения, благодаря которой приложения могут создавать даже пользователи, не программирующие профессионально. К особенностям VBA можно отнести выполнение скрипта именно в среде офисных приложений.

Недостатком являются проблемы с обратной совместимостью разных версий. Эти проблемы в основном связаны только с тем, что код программы обращается к функциональным возможностям, появившимся в новой версии программного продукта, которые отсутствуют в старой. Также к недостаткам часто относят и слишком высокую открытость кода для случайного изменения, тем не менее, многие программные продукты (например, Microsoft Office и IBM Lotus Symphony) позволяют пользователю использовать шифрование исходного кода и установку пароля на его просмотр.

Часто VBA используется для записи макросов приложений Microsoft Office Excel, Word.

Для этого используются объекты текущего приложения. Например, в Excel существует иерархия объектов, в корне которой стоит Application. Через его свойства можно вести обращение к открытым рабочим книгам (Workbook), а через их свойства обращаться к листам (Worksheet) и т.д.

Пример скрытия строк, содержащих символ последовательность "->":

For Each ro In Range("B5:B" & ActiveCell.SpecialCells(xlCellTypeLastCell).Row)

If InStr(ro.Value, "->") > 0 Then

ro.EntireRow.Hidden = True

End If

Next ro

# Примеры кода некоторых распространённых языков



## Lua

В играх S.T.A.L.K.E.R используется Lua (скрипт смены группировки) [16]:

function select\_actor\_community(actor, npc)

db.actor:set\_character\_community("dolg", 0, 0) -- сменит группировку игрока на "долг"

end

## VBA

В первой книге есть готовая таблица в которой в столбце "A" введен номер, а в столбце "B" Фамилия. Во второй книге в столбце "A" набирается номер, а в столбце "B" вставляется Фамилия взятая из Книги 1, соответствующая введенному номеру.

Dim I As Long, sh As Worksheet

' Функция вызывается из книги в которой она расположена и проверяет данные на листе "Список"

Set sh = ThisWorkbook.Sheets("Список")

With sh

For I = 1 To 6000

If Val(.Cells(I, 1).Value) = Val(Number) Then

Список = CStr(.Cells(I, 2).Value)

' Если данные найдены, то прерывание цикла

Exit For

End If

Next I

End With

## XML

XML-код для *Средства миграции пользовательской среды 3.0*, который осуществляет перенос папки «Мои видеозаписи» для всех пользователей (если эта папка имеется на исходном компьютере) [18].

Проверяет, что папка «Мои видеозаписи» существует на исходном компьютере:

<condition>MigXmlHelper.DoesObjectExist("File","%CSIDL\_MYVIDEO%")</condition>

Отфильтровывает ярлыки в папке «Мои видеозаписи», которые не удается сопоставить на конечном компьютере. Это не влияет на файлы, которые не являются ярлыками. Например, если на исходном компьютере в папке «Мои видеозаписи» имеется ярлык, указывающий на C:\Folder1, то этот ярлык будет перенесен, только если расположение C:\Folder1 существует на конечном компьютере. Однако все остальные файлы (например, MP3-файлы) будут перенесены без фильтрации:

<include filter='MigXmlHelper.IgnoreIrrelevantLinks()'>

Перенос папки «Мои видеозаписи» для всех пользователей:

<pattern type="File">%CSIDL\_MYVIDEO%\\* [\*]</pattern>

<?xml version="1.0" encoding="UTF-8"?>

<migration urlid="http://www.microsoft.com/migration/1.0/migxmlext/CustomFile">

<component type="Documents" context="User">

<displayName>My Video</displayName>

<role role="Data">

<detects>

<detect>

<condition>MigXmlHelper.DoesObjectExist("File","%CSIDL\_MYVIDEO%")</condition>

</detect>

</detects>

<rules>

<include filter='MigXmlHelper.IgnoreIrrelevantLinks()'>

<objectSet>

<pattern type="File">%CSIDL\_MYVIDEO%\\* [\*]</pattern>

</objectSet>

</include>

</rules>

</role>

</component>

</migration>

# Области использования

## Forth

Применение: компактные и быстрые программы, управление оборудованием, робототехника, обработка образов, искусственный интеллект, научные исследования [10].

## Lua

Применение: разработка игр, интерфейсов, вспомогательных инструментов прикладных программ, расширяющих их возможности [10].

В настоящее время используется в различных проектах, где требуется встроить достаточно быстрый и нетрудный в освоении скриптовый язык программирования — например, в разработке игр, где Lua часто используется между игровым «движком» и данными для написания сценариев поведения/взаимодействия объектов [16].

Так, например, Adobe Photoshop Lightroom с помощью Lua управляет пользовательским интерфейсом, а астрономическое приложение Celestia можно дополнять модулями расширения в виде Lua-скриптов [12].

HTTP-сервер Apache использует Lua при обработке запросов (начиная с версии 2.3, модуль mod\_lua включён в ядро сервера) [12].

Знаменитый сетевой сканер nmap применяет Lua как основу для собственного языка скриптов nse [12].

В последних версиях текстового редактора SciTE, работающего на нескольких платформах, Lua принят в качестве языка создания модулей расширения [12].

Различные скрипты и модули поддержки на языке Lua можно использовать и для мультимедийного плейера VLC [12].

## JavaScript

Применение: создание сценариев поведения браузера, встраиваемых в интернет-страницы и выполняемых на стороне клиента, написание активных серверных интернет-страниц ASP и JSP, исполняемых веб-сервером, CGI-приложением или расширением веб-сервера [10].

## Perl

Применение: системное администрирование, веб-приложения, исполняемые на стороне сервера, приложения для обработки текстовых данных.

Идеология языка - каждая задача должна иметь несколько решений. Синтаксис предусматривает минимум требований по оформлению кода, типизации данных и методов. Программисту предоставляется полная свобода, что позволяет очень быстро писать код, в котором трудно разбираться даже автору и тем более трудно, если не невозможно, поддерживать другим программистам. По этой же причине на Perl трудно вести большие совместные проекты.

Применение регулярных выражений для обработки текста многократно ускоряет разработку программ в сравнении с языками С и Pascal. Пара строк на Perl заменяет сотни строк кодирования алгоритмов на С и Pascal [10].

## PHP

Применение: разработка сайтов, веб-приложений, исполняемых на стороне сервера.

С использованием PHP разработано множество приложений, которые широко используются на различных сайтах, форумах и блогах [10].

* Drupal, Typo3, Mambo, Joomla, PHP-Nuke, Wordpress — системы управления содержимым (CMS);
* Magento, osCommerce — системы для Интернет-коммерции;
* Coppermine, Gallery Project, 4images — галереи изображений;
* MediaWiki, DokuWiki — вики-движки;
* phpBB, SMF, vBulletin, Invision Power Board — форумные движки;
* phpMyAdmin, phpPgAdmin, Adminer — утилиты администрирования СУБД;
* eyeOS — удалённая операционная система, основанная на принципе Desktop Operating System.

## Python

Интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической семантикой. Встроенные высокоуровневые структуры данных в сочетании с динамическими типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (RAD, Rapid Application Development). Кроме того, его можно использовать в качестве сценарного языка для связи программных компонентов [10].

Применение: разработка веб-приложений, расширений для приложений, прототипов программ.

## Ruby

Применение: прикладные программы, расширение возможностей программ [10].

Ruby используется в NASA, NOAA (национальная администрация по океану и атмосфере), Motorola и других крупных организациях. Следующие программы используют Ruby как скриптовый язык для расширения возможностей программы или написаны на нём (частично или полностью) [14].

* RPG Maker (RPG Maker XP) — RGSS (Ruby Game Scripting System)
* Amarok
* SketchUp
* Inkscape — скрипты для обработки векторных изображений
* Metasploit
* Puppet — система управления конфигурациями
* Redmine — багтрекер
* XChat
* Для KOffice разрабатывается Kross, механизм для поддержки скриптов, который включает Ruby.
* WATIR (англ. Web Application Testing in Ruby) — свободное средство для автоматического тестирования веб-приложений в браузере.

## VBScript

Применение: создание вспомогательных приложений для программ, написание активных серверных интернет-страниц ASP, исполняемых веб-сервером, CGI-приложением или расширением веб-сервера [10].

# Список литературы

1. Сценарный язык — Википедия – Режим доступа: <http://ru.wikipedia.org/wiki/Сценарный_язык>
2. Природа и эволюция сценарных языков - № 11, 2001 | Мир ПК | Издательство «Открытые системы» - Режим доступа: <http://www.osp.ru/pcworld/2001/11/162500/>
3. System Engineering - Сценарии для администрирования (Часть 1) – Режим доступа: <http://www.sysengineering.ru/Administration/ScriptsForAdministration01.aspx>
4. Сузи, С. Р. Язык программирования Python / А. Р. Сузи – 206. с.
5. bat-файлы - WindowsFAQ.ru – Режим доступа: <http://www.windowsfaq.ru/content/category/3/7/45>
6. 1.1. Основные аспекты языка Lua - Допустить к защите в гак 2009 г. Научный Добровольская Н. Ю. Дипломная работа... – Режим доступа: <http://rudocs.exdat.com/docs/index-338360.html?page=2>
7. Введение в XML – Режим доступа: <http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/iipXML/xmlintro.html>
8. Visual Basic for Applications — Википедия – Режим доступа: <http://ru.wikipedia.org/wiki/Visual_Basic_for_Applications>
9. Теория правильных скриптов / Хабрахабр – Режим доступа: <http://habrahabr.ru/post/92590/>
10. Технологии XXI век – Режим доступа: <http://www.specperevod.ru/information/programma/script.php>
11. Ierusalimschy, R., Programming in Lua / R. Ierusalimschy – PUC-Rio, Brazil – 328 p.
12. Для чего нужен Lua: Часть 3. Сила в единстве ­­– Режим доступа: <http://www.ibm.com/developerworks/ru/library/l-lua_3/index.html?S_TACT=105AGX99&S_CMP=GR01>
13. Сузи, C.Р. Сценарные языки: Python. – «Мир ПК», 9, 2001. – 206 с.
14. Ruby — Википедия – Режим доступа: http://ru.wikipedia.org/wiki/Ruby
15. JIT-компиляция — Википедия – Режим доступа: <http://ru.wikipedia.org/wiki/JIT-компиляция>
16. Lua — Википедия – Режим доступа: <http://ru.wikipedia.org/wiki/Lua>
17. XML — Википедия – Режим доступа: <http://ru.wikipedia.org/wiki/XML>
18. Примеры XML – Режим доступа: <http://technet.microsoft.com/ru-ru/library/cc765979(v=ws.10).aspx>

# Список вопросов

1. Назначение сценарных языков
2. Сходство сценарных и императивных языков
3. Различие сценарных и императивных языков
4. Классификация сценарных языков. Отличительные черты классов
5. Командно-сценарные языки
6. Прикладные сценарные языки
7. Языки разметки
8. Универсальные языки
9. Пакетные файлы, назначение
10. Python, его скриптовые черты
11. Язык Lua. преимущества
12. Язык Lua. недостатки
13. Назначение языка xml
14. Xml в HTML
15. Как xml помогает сетевым технологиям
16. Основные единицы xml-кода
17. Пример использования тегов xml
18. VBA, его роль в офисных приложениях
19. Области использования скриптовых языков
20. Отличие скрипта от программы