Innovative Features of Scripting Languages

Introduction

- Traditional programming languages are intended for constructing self contained applications
  - Programs that accept input,manipulate them and produce appropriate o/p
- But real world programs need the coordination of multiple prgrams

Example

- Payroll system for an institution
  - Process timely reported data from card readers,scanned paper forms and manual entry
  - Execute hundreds of queries(Full day,Half day,DL)
  - Enforce hundreds of legal and institutional rules
  - Create paper trail for record keeping, auditing and tax prepparation
  - Print paychecks
  - Communicate with servers around the world
- These tasks may involve lots of seperately executable programs which needs to be coordinated.

- It is possible to write coordination programs with traditional programming languages like java,c etc
- Conventional languages focusses on efficiency,maintainability,portability, and static detection of error
- Their type systems are built around H/W level concepts
- Scripting languages focusses on flexibility, rapid development, local customisation and dynamic checking
- Their type systems are built around tables,patterns,lists and files

What is a Scripting Language

- Have 2 principle sets of ancestors
- 1st set are command interpreters or shells
- Eg; IBMs JCL,MS-DOS command interpreter,Unix sh and csh shell families
- 2nd set has tools for text processing and report generation
- Eg: IBMs RPG,Unix's sed and awk
- From these evolved Rexx and Perl

- Perl most widely used general purpose scripting language
- Other purpose scripting languages are TCL,Python,Ruby,VBScript and AppleScript
- Perl and PHP are widely adopted for server-side web scripting
- For scripting on client computer, all major browsers use Javascript

Common Characteristics

1. *Both batch and interactive use*
   - Languages like Perl read the entire source code and produce output
   - Languages like Rexx, Python, Tcl,Guile, and Ruby compiler interprets their input line by line

2 Economy of expression

- ◦ Some languages like perl make heavy use of punctuation and very short identifiers
- ◦ Others like Rexx,Tcl tend to be more “Englishlike

Example in java

```
class Hello {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
```

- in Perl, Python, or Ruby it is simply

  ```
  print "Hello, world!\n"
  ```

3. Lack of declarations; simple scoping rules

- Most scripting languages dispense with declarations
- In Perl, everything is global by default
- In PHP,Tcl, everything is local by default
  - Globals must be explicitly imported
- Python uses the rule:
  - Any variable that is assignes a value is local to the block where the assignment appers

4. *Flexible dynamic typing*
    - Due to lack of declarations most scripting languages are dynamically typed
5. *Easy access to system facilities*
    - Perl provides over 100 built in commands for OS functions
    - They are easier to use than the corresponding functions in C

6. *Sophisticated pattern-matching and string manipulation*

- scripting languages tend to have extraordinarily rich facilities for pattern matching, search, and string manipulation

7. High-level data types.

- Sets,bags ,dictionaries,lists and tuples are built into the syntax and semantics of scripting languages

Names and Scopes

- Most scripting languages dont require variables to be declared
- Few languages like Perl and Javascipt permit optional declarations
- Perl can be run in a mode that requires declarations
  - Use strict 'vars'
- Most scripting languages use dynamic typing

Nesting & Scoping Conventions

- Scheme ,Python ,Javascript and R provides nested subroutines with static scoping
- Tcl provides subroutines with dynamic scoping
- Named subroutines donot nest in PHP or Ruby but they do in Perl
- Perl,Ruby,Scheme,Python,Javascript and R provide first class anonymous local subroutines
- In Perl nested block are statically scopped
- In Ruby nested blocks are part of named scope where they appear
- Perl,Ruby,Python,Scheme,Javascrpt and R provide unlimited extend for variables captured in closures

Scope of an Undeclared Variable

- In languages with static scope without variable declaration
- If we access a variable 'x'
- Is x local, global or intermediary?

- There are different approaches taken by different languages

- In Perl all variables are global unless explicitly declared

- In PHP variables are local unless explicitly imported

- Ruby uses prefix characters on names to distinguish between scopes
  - foo is a local variable;
  - $foo is a global variable;
  - @foo is an instance variable

Scope in Tcl

- Employs dynamic scoping
- Variables are not accessed automatically
- They must be explicitly asked by programmers
- 'upvar' and 'uplevel' commands are used for this
- 'upvar' command access a variable in specified frame and gives it a new name
- 'uplevel' command provides a nested Tcl script
- This script is executed in the context of specified frame using call by name mode

```
proc bar { } {
    upvar i j ;# j is local name for caller’s i
    puts "$j"
    uplevel 2 { puts [expr $a + $b] }
    # execute ’puts’ two scopes up the dynamic chain
     }
proc foo { i } {
    bar
      }
set a 1; set b 2; foo 5   prints--- 5 and 3
```

Scoping in Perl

- Undeclared variables are global by default
- Variables declared with **local** operator are dynamically scoped
- Variables declared with '**my**' operator are statically scoped

String & Pattern Matching

POSIX REGULAR EXPRESSIONS

- A regix is typically delimited by a pair of forward slash, in the form of /../
- The leading ^ and trailing $ are known as position anchors, which match the beginning and ending of the input string, respectively
- The [...] encloses a list of characters, and matches any character in the list
- \d is a metacharacter that matches any digit, which is identical to [0-9]
- | represents the OR operator

- \. matches the .character
- \w+ matches one or more word characters
- The \ is known as the escape code, which restore the original literal meaning of its following character
- The @ matches itself
- The \s matches white space
- The \S+ matches anything that is not matched by \s, i.e , non white space.

Basic operations in POSIX REs

- /ab(cd|ef)g*/ matches abcd, abcdg, abefg, abefgg, abcdggg, etc.
- + indicates zero or one repetitions, * indicates zero or more repetitions ,
- {*n*} indicates exactly *n* repetitions,
- {*n*,} indicates at least *n* repetitions, and {*n*,*m*} indicates *n*–*m* repetitions

- /a(bc)*/ matches a, abc, abcbc, abcbcbc, etc.
- /a(bc)?/ matches a or abc
- /a(bc)+/ matches abc, abcbc, abcbcbc, etc.
- /a(bc){3}/ matches abcbcbc only
- /a(bc){2,}/ matches abcbc, abcbcbc, etc.
- /a(bc){1,3}/ matches abc, abcbc, and abcbcbc (only)

- 2 zero length assertions
- ^ matches at the beginning of target
- $ matches at the end of the target

- **Example**
- /abe/ will match abe,abet,babe,label
- /^abe/ will match only abe,abet
- /abe$/ will match only abe and babe
- /^abe$/ will match only abe

Character classes

- /b[aeiou]d/ matches bad, bed, bid, bod, and bud
- a dot (.) matches any character other than a newline
- The expression /b.d/, matches not only bad, bbd, bcd, but also b:d, b7d, etc
- A caret (ˆ) at the beginning of a character class indicates negation
- The class expression matches anything other than the characters inside
- /b[ˆaq]d/ matches anything matched by /b.d/ except for bad and bqd
- To match a literal backslash, use two of them
- /a\\b/ matches a\b

Perl Extensions

- The built-in =˜ operator is used to test for matching

- $foo = "albatross";
- if ($foo =˜ /ba.*s+/) ... # true
- if ($foo =˜ /ˆba.*s+/) ... # false (no match at start of string)

- The string to be matched against can also be left unspecified, in which case Perl

  uses the pseudovariable $_ by default:

- $_ = "albatross";
- if (/ba.*s+/) ... # true
- if (/^ba.*s+/) ... # false

- The !~ returns true when a pattern does not match
- if ("albatross" !~ /^ba.*s+/) ... # true

Substitution

- the binary “mixfix” operator s/// replaces whatever lies between the first and second slashes with whatever lies between the second and the third

- $foo = "albatross";
- $foo =˜ s/lbat/c/;   # "across"

Modifiers and Escape Sequences

- Both matches and substitutions can be modified by adding one or more characters after the closing delimiter

- A trailing i, for example, makes the match case-insensitive:

- $foo = "Albatross";
- if ($foo =˜ /ˆal/i) ... # true

- A trailing g on a substitution replaces *all* occurrences of the regular expression:
- $foo = "albatross";
- $foo =˜ s/[aeiou]/-/g;   # "-lb-tr-ss"

Greedy and Minimal Matches

- rule for matching in REs is sometimes called “left-most longest”:
- In the string abcbcbcde
- the pattern /(bc)+/ can match in six different ways:
- abcbcbcde
- abcbcbcde
- abcbcbcde
- abcbcbcde
- abcbcbcde
- abcbcbcde

- The third of these is “left-most longest,” also known as greedy
- First “left-most shortest” or minimal match

Variable Interpolation

- Interpolation means "Introducing or inserting something"
- It is the name given to replacing a variable with the value of that variable
- Any string that is built with double quotes will be interpolated
- Any avariable that appears within the string will be replaced with the value of that variable
- Example
- my $apples=4
- print "I have $apples apples";

- will print
- I have 4 apples

- Any dollar sign that does not immediately proceed a vertical bar, closing parenthesis, or end of string is assumed to introduce the name of a Perl variable

- $prefix = ...
- $suffix = ...
- if ($foo =˜ /ˆ$prefix.*$suffix$/) ...

Variable *Capture*

- Every parenthesized fragment of a Perl RE is said to capture the text that it matches.
- The captured strings may be referenced as \1, \2, and so on
- Captured string can be used later in RE-> backreference

- $text="Joe Smith";
- if($text)=˜s/^([a-z]+)/i){
- print "Hello $1";
- }
- will print
- Hello Joe
- print"$text";
- will print   i Smith

- $str="abc";
- $str=~/(((**a**) (**b**)) (**c**))/;
- **$1 – abc**
- **$2 – ab**
- **$3-a**
- **$4 - b**
- **$5 - c**

Data Types

- No declarations for variables in scripting languages
- So it performs run time checks to ensure if values are used properly
- Scheme,Python,Ruby performs strict checking
- Explicit conversion needed for one type to another

- Ruby
- a = "4"
- print a + 3, "\n"
- we get the following message at run time:
- “In ‘+’: failed to convert Fixnum into String (TypeError).”

- Perl ,Rexx, Tcl - programmers should check for the errors they care about
- $a[3] = "1”; # (array @a was previously undefined)
- print $a[3] + $a[4], "\n";
- $a[4] is uninitialized -> value undef.
- undef evaluates to 0.
- 1+0=1
- 1 is converted to string and printed

- Ruby
- a = [ ]   # empty array assignment
- a[3] = "1"
- a[3] is a string, but other elements of a are nil.
- If we want concatenation we must say
- print a[3] + String(a[4]), "\n"
- If we want addition, we must say
- print Integer(a[3]) + Integer(a[4]), "\n"

- Perl uses value model of variables
- Scheme,Python and Ruby use reference model of variables
- PHP and Javascript use value model for variables of prmitive type and reference model for variables of object type

NumericTypes

- JavaScript -> numbers are double precision FP
- Tcl-> numbers are strings
- Converted to int or float as needed
- PHP -> supports integers and double precision FP
- Perl, Ruby -> Supports integers ,double precision FP, arbitrary precision integers(long int)
- Python supports bignums, complex number
- Scheme -> all above formats + rationals
- Ruby -> classes for Fixnum, Bignum, Float

Composite Types

- Perl has array and hash inherited from 'awk' language
- Uses prefix characters on variable names

- $foo is a scalar
- @foo is an array;
- %foo is a hash;
- &foo is a subroutine;
- foo is a filehandle or an I/O format

- Arrays are indexed using square brackets
- Start with 0
- @colors = ("red", "green", blue"); # initializer syntax
- print $colors[1]; # green

- Hashes are indexed using curly braces and character string names:

- %complements = ("red" => "cyan", "green" => "magenta", "blue" => "yellow");
- print $complements{"blue"}; # yellow

- Python and Ruby uses square brackets for indexing arrays & hashes
- Ruby
- colors = ["red", "green", "blue"]
- complements = {"red" => "cyan", "green" => "magenta", "blue" => "yellow"}
- print colors[2], complements["blue"]

- Python uses : in place of =>

Set Operations in Python

- Python provides tuples and sets
- Tuple -> immutable list
- crimson = (0xdc, 0x14, 0x3c) # R,G,B components
- Provide multiway assignment
- a, b = b, a # swap

- Sets -> indicate if elements are present or not
- X = set([’a’, ’b’, ’c’, ’d’]) # set constructor
- Y = set([’c’, ’d’, ’e’, ’f’]) # takes array as parameter
- U = X | Y # ([’a’, ’b’, ’c’, ’d’, ’e’, ’f’])
- I = X & Y # ([’c’, ’d’])
- D = X - Y # ([’a’, ’b’])
- O = X ˆ Y # ([’a’, ’b’, ’e’, ’f’])
- ’c’ in I # True

Context

- Type compatibility determines which type can be used in which context
- C
- double d = 3;
- the 3 occurs in a context that expects a floating-point number.
- The C compiler coerces the 3 to make it a double instead of an int.

- Perl extends the notion of context to drive decisions
- made at run time.
- Assignment operator (=) provides a scalar or list context to its right-hand side based on the type of its left-hand side
- This type is always known at compile time
- Prefix character ($) ->implying a scalar context
- (@) or (%) -> it is a list

- $time = gmtime();
- Return the time as a character string, "Sun Aug 17 15:10:32 2008".
- @time_arry = gmtime();
- Returns an 8-element array indicating seconds, minutes, hours, day of month, month of year (39, 09, 21, 15, 2, 105, 2, 73)

Object Orientation

- Perl 5 has object-oriented features
- PHP and Javascript -> object-oriented + imperative features
- Python and Ruby -> only object oriented features
- Perl-> value model for variables
  - Objects are accessed via pointers
- PHP,javascript->Primitive type,composite type
- python and Ruby->reference model

Perl 5

- Object support in Perl 5 provides 2 main things:
- (1) a blessing : Associates a reference with a package,
- (2) special syntax for method calls that automatically passes an object reference or package name as the initial
- argument to a function

Example

```
{ package Integer;
sub new {
    my $class = shift; # probably "Integer"
    my $self = {}; # reference to new hash
    bless($self, $class); # points to reference of Integer class
    $self->{val} = (shift || 0);
    return $self;
}
sub set {
    my $self = shift;
    $self->{val} = shift;
}
sub get {
    my $self = shift;
    return $self->{val}; } }
```

- $c1 = Integer->new(2) # Integer::new("Integer", 2)
- $c2 = new Integer(3); # alternative syntax
- $c3 = new Integer; # no initial value specified

- Integer->new and new Integer are same as
- Integer::new("Integer",2)

- print $c1->get, " ", $c2->get, " ", $c3->get, " ", "\n";

- $c1->set(4); $c2->set(5); $c3->set(6);

- print $c1->get, " ", $c2->get, " ", $c3->get, " ", "\n";

- will print
- 2 3 0
- 4 5 6

Inheritance In Perl

- Inheritance in Perl is obtained by means of the @ISA array

```
{ package Tally;
@ISA = ("Integer");
sub inc {
my $self = shift;
$self->{val}++;
}
}
...
$t1 = new Tally(3);
$t1->inc;
$t1->inc;
print $t1->get, "\n"; # prints 5
```

PHP and JavaScript

- PHP4 provided a variety of object-oriented features
  - Interfaces
  - Abstract methods
  - Classes
  - Final Methods
  - Static & constant members
- Javascript provides features like
  - Inheritance
  - Dynamic method dispatch

Example in Javascript

```
function Integer(n) {
this.val = n || 0; // use 0 if n is missing (undefined)
}
function Integer_set(n) {
this.val = n;
}
function Integer_get() {
return this.val;
}
Integer.prototype.set = Integer_set;
Integer.prototype.get = Integer_get;
```

- c2 = new Integer(3);
- c3 = new Integer;

- document.write(c2.get() + "&nbsp;&nbsp;" + c3.get() + "<BR>");
- c2.set(4); c3.set(5);
- document.write(c2.get() + "&nbsp;&nbsp;" + c3.get() + "<BR>");

- This code will print
- 3 0
- 4 5

- We can override methods and fields on an object by object basis:
- c2.set = new Function("n", "this.val = n * n;");

- c2.set(3); c3.set(4); // these call different methods!
- document.write(c2.get() + "&nbsp;&nbsp;" + c3.get() + "<BR>");

- this code will print
- 9 4

Inheritance In Javascript

```
function Tally(n) {
this.base(n); // call to base constructor
}
function Tally_inc() {
this.val++;
}
Tally.prototype = new Integer; // inherit methods
Tally.prototype.base = Integer; // Tallys base class is Integer
Tally.prototype.inc = Tally_inc; // new method
...
t1 = new Tally(3);
t1.inc(); t1.inc();
document.write(t1.get() + "<br>");
```

This code will print a 5.

Python and Ruby

- Constructor in Python
- init
- Constructor in Ruby
- initialize
- To create a new object in Python
- My_object = My_class.(args).
- In Ruby
- my_object = My_class.new(args).
- New fields are added
- my_object.new_field = value