1 Scripting

1.1 Introduction

Scripting languages have been around for a long time. Many applications have some sort of language to help users work with the application programmatically.

Many scripting languages are interpreted. Some are both compiled and interpreted. Most are multi-platform.

Generally easier to use than a compiled language (C/C++).

Many are written in C!

1.2 Some Scripting Languages

- Shell (sh, csh, ksh, bash, DOS batch files)
- awk
- Tcl/Tk (Jacl)
- Perl
- Python (Jython) Parrot
- Ruby (JRuby)
- AppleScript (Apple)
- AutoLisp (AutoCAD)
- Javascript (Internet browsers)
- VBScript (Windows)
- Many more (ASP, PHP) ...

A number of scripting languages interact with each other, e.g., Perl/Tk, Python/Tk, Ruby/Tk. Some scripting languages interact with major languages: SchemeTk, CamelBones (Perl/Cocoa), RubyCocoa.

Most have considerable support for regular expressions.

Closer look at Tcl, Perl, Python, and Ruby. All support modular construction (modules), procedures (subprograms/functions), decision making constructs, and looping constructs.

1.2.1 Tcl/Tk

Version 8.4.2
Very popular because of graphical support
UC Berkeley → Sun → Scriptics

```
puts "hello, world"

set greeting "hello"
set addressee "world"
puts "$greeting, $addressee"

button .b -text "Push Me" -command {tk_messageBox -message  "hello, world"}
pack .b

Here is the factorial procedure:
proc fac {x} {
    if {$x < 0} {
        error "Invalid argument $x: must be a positive integer"
    } elseif {$x <= 1} {
        return 1
    } else {
        return [expr {$x * [fac [expr {$x-1}]]}]
    }
}
```

1.2.2 Perl

Version 5.8
Text processing, regular expressions
Lots of cryptic symbols ($, @, #)
Support for arrays, tables, hashes
Bioinformatics (Lisp and Python).

```
print "hello, world"
```

```
# print array
@array = ("red", "yellow", "green");
print "I have ", @array, " marbles.\n";
print "I have  @array marbles.\n";

I have redyellowgreen marbles.
I have red yellow green marbles.
```

$$\log_n(x) = \frac{\log_e(x)}{\log_e(n)}$$

```
sub log_base {
  my ($base, value) = @_;         args
  return log($value)/log($base);
}

$answer = log_base(10, 10_000);
print "log10(10,000) = $answer\n"

log10(10,000) = 4
```

1.2.3 Python

Version 2.2 (2.3 in beta)
Object Oriented
Support for dictionaries, lists (array), tuples
Internet support
Indentation counts! Readable Perl.
Blocks are indicated through indentation, and only through indentation. (No BEGIN/END or braces.)

```
print "Hello World"
```

```
# Print out the values from 0 to 99 inclusive.
for value in range(100):
        print value
```

```
>>> list = ['a', 'd', 'f']
>>> list[1:1] = ['b', 'c']
>>> print list
['a', 'b', 'c', 'd', 'f']
>>> list[4:4] = ['e']
>>> print list
['a', 'b', 'c', 'd', 'e', 'f']
```

```
def times(n):
    for i in range(1,13):
        print "%d x %d = %d" % (i, n, i*n)

print "Here is the 9 times table..."
times(9)
```

1.2.4 Ruby

Version 1.8
Readable Perl
Object Oriented
Regular expressions
Smalltalk-like features
Support for arrays, tables, hashes
Internet support

```
http://www.rubycentral.com/book/intro.html

puts "Hello World"
```

Object Oriented

```
"Bruce".length   5

number = Math.abs(number)       //Javacode

number = number.abs

def sayGoodnight(name)
  result = "Goodnight, " + name
  return result
end

# Time for bed...
puts sayGoodnight("John-Boy")
puts sayGoodnight("Mary-Ellen")

num = 8
7.times do
  print num.type, " ", num, "\n"
  num *= num
end

Fixnum 8
Fixnum 64
Fixnum 4096
Fixnum 16777216
Bignum 281474976710656
Bignum 79228162514264337593543950336
Bignum 6277101735386680763835789423207666416102355444464034512896
```

```
3.upto(6){ |i|  print i }
('a'..'e').each{ |char| print char }
```

```
abcde
```