JNTU SL LAB Manual□❤✌✓

Computer science (Malla Reddy Group of Institutions)

SCRIPTING LANGUAGE
LAB MANUAL

B.Tech IIIYR IISEM

OBJECTIVES:

- To Understand the concepts of scripting languages for developing web-based projects
- To understand the applications the of Ruby, TCL, Perl scripting languages
- To teach the students basics of SCRIPTING LANGUAGES programs and its execution.
- To teach the student, to develop scripting languages programs.

Recommended System/Software Requirements:

- Intel based desktop PC with minimum of 2.6GHZ or faster processor with at least 256 MB RAM and 40GB free disk space.
- Operating system: Flavor of any LINUX/WINDOWS.
- Software: Ruby, TCL, Perl.

INTRODUCTION TO SCRIPTING LANGUAGES

Scripts and programs

Scripting is the action of typing scripts using a scripting language, distinguishing neatly between programs, which are written in conventional programming language such as C,C++,java, and scripts, which are written using a different kind of language.

We could reasonably argue that the use of scripting languages is just another kind of programming. Scripting languages are used for is qualitatively different from conventional programming languages like C++ and Ada address the problem of developing large applications from the ground up, employing a team of professional programmers, starting from well-defined specifications, and meeting specified performance constraints.

Scripting languages, on other hand, address different problems:

- Building applications from ‘off the shelf’ components
- Controlling applications that have a programmable interface
- Writing programs where speed of development is more important than run-time efficiency.

The most important difference is that scripting languages incorporate features that enhance the productivity of the user in one way or another, making them accessible to people who would not normally describe themselves as programmers, their primary employment being in some other capacity. Scripting languages make programmers of us all, to some extent.

Origin of scripting

The use of the word ‘script’ in a computing context dates back to the early 1970s,when the originators of the UNIX operating system create the term ‘shell script’ for sequence of commands that were to be read from a file and follow in sequence as if they had been typed in at the keyword. e.g. an ‘AWKscript’, a ‘perl script’ etc.. the name ‘script ‘ being used for a text file that was intended to be executed directly rather than being compiled to a different form of file prior to execution.

Other early occurrences of the term ‘script’ can be found. For example, in a DOS based system, use of a dial-up connection to a remote system required a communication package that used proprietary language to write scripts to automate the sequence of operations required to establish a connection to a remote system. Note that if we regard a scripts as a sequence of commands to control an application or a device, a configuration file such as a UNIX ‘make file’ could be regard as a script. However, scripts only become interesting when they have the added value that comes from using programming concepts such as loops and branches.

1. GUIDELINES TO STUDENTS

1. Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.

2. Students are instructed to come to lab in formal dresses only.

3. Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.

4. Students are required to carry their observation book and lab records with completed exercises while entering the lab.

5. Lab records need to be submitted every Experiment.

6. Students are not supposed to use pen drives in the lab.

7. Students need to maintain social distance.

2. LAB OBJECTIVE

- To introduce Scripting languages compiler and eclipse platform.
- To Understand the concepts of scripting languages for developing web-based projects
- To understand the applications the of Ruby, TCL, Perl scripting languages

3. LAB OUTCOME

- Able to use Scripting languages compiler and platform to write and execute scripting languages program.
- Understand and Apply Object oriented features and Scripting languages concepts.
- Ability to understand the differences between Scripting languages and programming languages
- Able to gain some fluency programming in Ruby, Perl, TCL

4. INTRODUCTION ABOUT LAB

There are 30 systems (HP) installed in this Lab. Their configurations are as follows:

Processor	: Pentium(R) Dual-Core CPU
RAM	: 4 GB
Hard Disk	: 320 GB
Mouse	: Optical Mouse

5. List of experiments as per the university curriculum

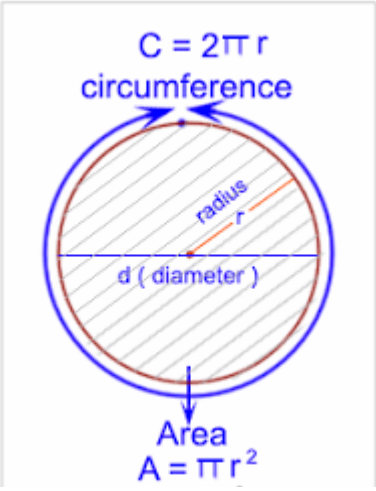
1. Write a Ruby script to create a new string which is n copies of a given string where n is a nonnegative integer
Ruby Code:

```
def multiple_string(str, n)
   return str*n
end
print multiple_string('a', 1),"\n"
print multiple_string('a', 2),"\n"
print multiple_string('a', 3),"\n"
print multiple_string('a', 4),"\n"
print multiple_string('a', 5),"\n"
```

Output:

```
a
aa
aaa
aaaa
aaaaa
```

2. Write a Ruby script which accept the radius of a circle from the user and compute the parameter and area.



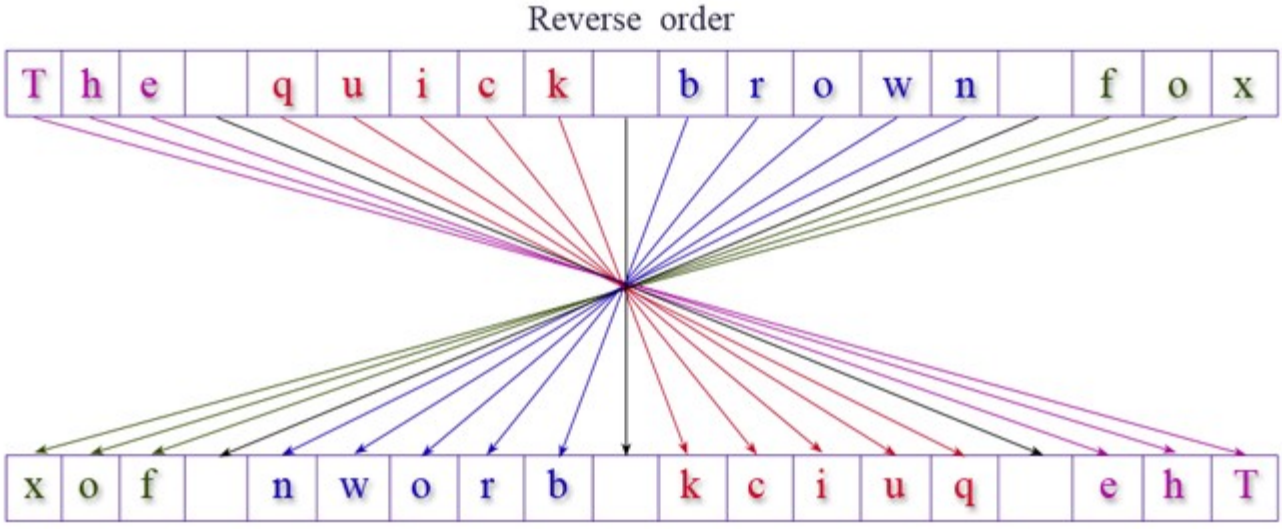
Ruby Code:

```
radius = 5.0
perimeter = 0.0
area = 0.0
print "Input the radius of the circle: "
radius = gets.to_f
perimeter = 2 * 3.141592653 * radius
area = 3.141592653 * radius * radius
puts "The perimeter is #{perimeter}."
puts "The area is #{area}."
```

Output:

Input the radius of the circle: The perimeter is 31.41592653.
The area is 78.539816325.

3. Write a Ruby script which accept the user's first and last name and print them in reverse order with a space between them



Ruby Code:

```
puts "Input your first name: "
fname = gets.chomp
puts "Input your last name: "
lname = gets.chomp
puts "Hello #{lname} #{fname}"
```

Output:

```
Input your first name:
Input your last name:
Hello Lanoie Gary
```

4. Write a Ruby script to accept a filename from the user print the extension of that

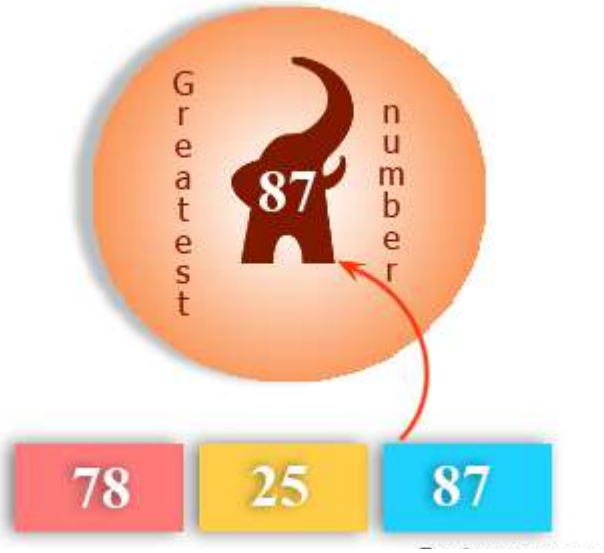
Ruby Code:

```
file = "/user/system/test.rb"
# file name
fbname = File.basename file
puts "File name: "+fbname
# basename
bname = File.basename file,".rb"
puts "Base name: "+bname
# file extention
ffextn = File.extname  file
puts "Extention: "+ffextn
# path name
path_name= File.dirname  file
puts "Path name: "+path_name
```

Output:

```
File name: test.rb
Base name: test
Extention: .rb
Path name: /user/system
```

5. Write a Ruby script to find the greatest of three numbers



Ruby Code:

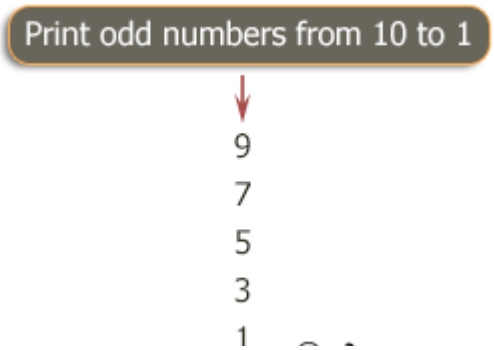
```
x,y,z = 2,5,4
if x >= y and x >= z
    puts "x = #{x} is greatest."
elsif y >= z and y >= x
    puts "y = #{y} is greatest."
else
    puts "z = #{z} is greatest."
end
```

Copy

Output:

y = 5 is greatest.

6. Write a Ruby script to print odd numbers from 10 to 1



Ruby Code:

```
puts "Odd numbers between 9 to 1: "
9.step 1, -2 do |x|
     puts "#{x}"
 end
```

Output:

```
Odd numbers between 9 to 1:
9
7
5
3
1
```

7. Write a Ruby scirpt to check two integers and return true if one of them is 20 otherwise return their sum

Ruby Code:

```
def makes20(x,y)
   return x == 20 || y == 20 || x + y == 20
end

print makes20(20, 10),"\n"
print makes20(40, 10),"\n"
print makes20(15, 20)
```

Copy

Output:

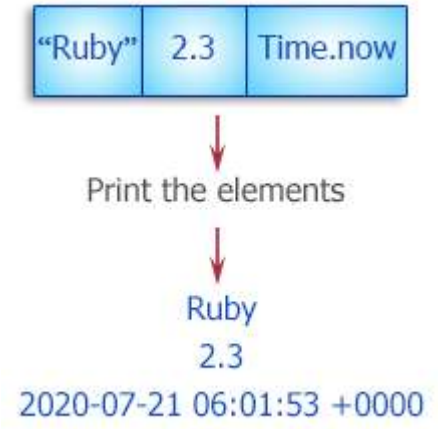
```
true
false
true
```

8. Write a Ruby script to check two temperatures and return true if one is less than 0 and the other is greater than 100

```
def temp(temp1, temp2)
   return ( temp1 < 0 && temp2 > 100 ) || ( temp1 > 100 && temp2 < 0 );
end
print temp(110, -1),"\n"
print temp(-1, 110),"\n"
print temp(2, 120)
```

Output:
true
true
false

9. Write a Ruby script to print the elements of a given array



Sample array : ["Ruby", 2.3, Time.now]

Ruby Code:

```
array1 = ["Ruby", 2.3, Time.now]
for array_element in array1
        puts array_element
end
```

Copy

Output:

```
Ruby
2.3
2017-12-28 06:01:53 +0000
```

10. Write a Ruby program to retrieve the total marks where subject name and marks of a student stored in a hash
Sample subject and marks : Literature -74, Science – 89, Math-91

Ruby Code:

```
student_marks = Hash.new 0
student_marks['Literature'] = 74
student_marks['Science'] = 89
student_marks['Math'] = 91
total_marks = 0
student_marks.each {|key,value|
        total_marks +=value
    }
puts "Total Marks: "+total_marks.to_s
```

Copy

Output:

Total Marks: 254

11. Write a TCL script to find the factorial of a number

```
set counter 4
 set factorial 1
 while {$counter > 0} {
   set factorial [expr $factorial * $counter]
   incr counter -1
 }
 puts $factorial
```

Factorial 10
=> 3628800

12. Write a TCL script that multiplies the numbers from 1 to 10

```
set n 10
 for {set i 1} {$i <= $n} {incr i} {
    puts $i
 }
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

13. Write a TCL script for Sorting a list using a comparison function
The syntax for sorting a list is given below −
lsort listname
An example for sorting a list is given below −

```
set var {orange blue red green}
set var [lsort $var]
puts $var
```

output
blue green orange red

14. Write a TCL script to

(i)create a list
(ii)append elements to the list
(iii)Traverse the list
(iv)Concatenate the list

Creating a List
Some examples are given below −

```
set colorList1 {red green blue}
set colorList2 [list red green blue]
set colorList3 [split "red_green_blue" ]
puts $colorList1
puts $colorList2
puts $colorList3
```

result −

```
red green blue
red green blue
red green blue
```

Appending Item to a List
Some examples are given below −

```
set var orange
append var " " "blue"
puts $var
```

result −

```
orange blue
```

Traversing lists

```
foreach item {1 2 3 4 5 6 7 8 9} {
   puts $item
}
```

result −

```
1
2
3
4
5
```

6
7
8
9

Concatenate the list

```
set i [concat {a b c} {1 2 3}]
puts $i
```

result −

```
a b c 1 2 3
```

15. Write a TCL script to comparing the file modified times.

```
proc newer { fp fp2 } {
if ![file exists $fp] {
    puts "file exsist"
} else {
    # Assume file1 exists
    expr [file mtime $fp] > [file mtime $fp2]
    puts "file modification times compared."
}
}
newer file1.txt file2.txt
```

16. Write a TCL script to Copy a file and translate to native format.

```
proc File_Copy {src dest} {
        if [file isdirectory $src] {
                file mkdir $dest
                foreach f [glob -nocomplain [file join $src *]] {
                        File_Copy $f [file join $dest [file tail $f]]
                }
                return
        }
        if [file isdirectory $dest] {
                set dest [file join $dest [file tail $src]]
   }
        set in [open $src]
        set out [open $dest w]
        puts -nonewline $out [read $in]
        close $out ; close $in
}
```

17. a) Write a Perl script to find the largest number among three numbers.

Vi great.pl

```
#!/usr/bin/perl
print "enter a value";
$a=<stdin>;
print "enter b value";
$b=<stdin>;
print "enter c value";
$c=<stdin>;
if($a > $b) //if compares string use gt ,lt,le,ge
{
        if($a> $c)
        {
                print " $a is largest number\n";
        }
        else
        {
                print " $c is largest number\n";
        }
}
elsif($b >$c)
{
     print " $b is largest number";
}
else
{
    print " $c is largest nnumber";
}
```

OUT PUT:

Perl great.pl

```
Enter a value 4
Enter b value 6
Enter c value 5
6 is largest number
```

b) Write a Perl script to print the multiplication tables from 1-10 using subroutines.

Program:

```
for($i=1;$i<=12;$i++)
{
   $a[$i]=$i;

}

for($i=1;$i<=12;$i++)
{
   for($j=1;$j<=12;$j++)
   {
      print(($a[$j]*$a[$i]),"    ");
   }
   print "\n\n";
}
```

Output:

```
   1   2   3   4   5   6   7   8   9  10  11  12
   2   4   6   8  10  12  14  16  18  20  22  24
   3   6   9  12  15  18  21  24  27  30  33  36
   4   8  12  16  20  24  28  32  36  40  44  48
   5  10  15  20  25  30  35  40  45  50  55  60
   6  12  18  24  30  36  42  48  54  60  66  72
   7  14  21  28  35  42  49  56  63  70  77  84
   8  16  24  32  40  48  56  64  72  80  88  96
   9  18  27  36  45  54  63  72  81  90  99 108
  10  20  30  40  50  60  70  80  90 100 110 120
  11  22  33  44  55  66  77  88  99 110 121 132
  12  24  36  48  60  72  84  96 108 120 132 144
```

18. Write a Perl program to implement the following list of manipulating functions a)Shift b)Unshift c)Push
Perl provides various inbuilt functions to add and remove the elements in an array.

Function	Description
push	Inserts values of the list at the end of an array
pop	Removes the last value of an array
shift	Shifts all the values of an array on its left
unshift	Adds the list element to the front of an array

push function

This function inserts the values given in the list at an end of an array. Multiple values can be inserted separated by comma. This function increases the size of an array. It returns number of elements in new array.

Syntax: *push(Array, list)*
Example:

- Perl

```
#!/usr/bin/perl

# Initializing the array
@x = ('Java', 'C', 'C++');

# Print the Initial array
print "Original array: @x \n";

# Pushing multiple values in the array
```

```
push(@x, 'Python', 'Perl');

# Printing the array
print "Updated array: @x";
```

Output:

```
Original array: Java C C++
Updated array: Java C C++ Python Perl
```

pop function

This function is used to remove the last element of the array. After executing the pop function, size of the array is decremented by one element. This function returns undef if list is empty otherwise returns the last element of the array.

Syntax: *pop(Array)*
Example:

- Perl

```
#!/usr/bin/perl

# Initializing the array
@x = ('Java', 'C', 'C++');

# Print the Initial array
print "Original array: @x \n";

# Prints the value returned by pop
print "Value returned by pop: ", pop(@x);
```

```
# Prints the array after pop operation
print "\nUpdated array: @x";
```

Output:

```
Original array: Java C C++
Value returned by pop: C++
Updated array: Java C
```

shift function

This function returns the first value in an array, removing it and shifting the elements of the array list to the left by one. Shift operation removes the value like pop but is taken from the start of the array instead of the end as in pop. This function returns undef if the array is empty otherwise returns first element of the array.

Syntax: *shift(Array)*
Example:

- Perl

```
#!/usr/bin/perl

# Initializing the array
@x = ('Java', 'C', 'C++');

# Print the Initial array
print "Original array: @x \n";

# Prints the value returned
# by shift function
print "Value returned by shift: ",
                    shift(@x);
```

```
# Array after shift operation
print "\nUpdated array: @x";
```

Output:

```
Original array: Java C C++
Value returned by shift :Java
Updated array: C C++
```

unshift function

This function places the given list of elements at the beginning of an array. Thereby shifting all the values in an array by right. Multiple values can be unshift using this operation. This function returns the number of new elements in an array.

Syntax: *unshift(Array, List)*
Example:

- Perl

```
#!/usr/bin/perl

# Initializing the array
@x = ('Java', 'C', 'C++');

# Print the Initial array
print "Original array: @x \n";

# Prints the number of elements
# returned by unshift
print "No of elements returned by unshift: ",
                unshift(@x, 'PHP', 'JSP');

# Array after unshift operation
```

```
print "\nUpdated array: @x";
```

Output:

```
Original array: Java C C++
No of elements returned by unshift :5
Updated array: PHP JSP Java C C++
```

19. a) Write a Perl script to substitute a word, with another word in a string.
Substitution Operator or ‘s’ operator in Perl is used to substitute a text of the string with some pattern specified by the user.
Syntax: *s/text/pattern*
Returns: *0 on failure and number of substitutions on success*
Example 1:

```
#!/usr/bin/perl -w

# String in which text
# is to be replaced
$string = "GeeksforGeeks";

# Use of s operator to replace
# text with pattern
$string =~ s/for/to/;

# Printing the updated string
print "$string\n";
```

Output:

```
GeekstoGeeks
```

b) Write a Perl script to validate IP address and email address.

```
#!/usr/bin/perl

print("Enter the IP Address you would like to validate - ");
my $ip = <STDIN>;

if($ip =~ /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/)
{
	$ip = $1;
}
chomp($ip);
if($ip =~ m/^(\d\d?\d?)\.(\d\d?\d?)\.(\d\d?\d?)\.(\d\d?\d?)$/)
{
	print("\nIP address found - $ip\n");
	if($1 <= 255 && $2 <= 255 && $3 <= 255 && $4 <= 255)
	{
		print("Each octet of an IP address is ",
			"within the range - $1.$2.$3.$4\n");
		print("\n-> $ip IP address accepted!\n");
	}
	else
	{
		print("Octet(s) out of range. ",
			"Valid number range between 0-255\n");
	}
}
else
{
```

```
        print("IP Address $ip is not in a valid format\n");
}
```

output

```
Enter the IP Address you would like to validate - 148.125.1.1

IP address found - 148.125.1.1
Each octet of an IP address is within the range - 148.125.1.1

-> 148.125.1.1 IP address accepted!
```

20. Write a Perl script to print the file in reverse order using command line arguments

```
@lines = <>;
print reverse @lines;
```

output

1
2
3
4
5

Reverse

5
4
3
2
1