Universidad de La Laguna
E.T.S. Ingeniería Informática
Área: Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación
Práctica de laboratorio **#3**

1. Iniciar una sesión de trabajo en GNU-Linux.

2. Abra una terminal.

3. Muestre el árbol de directorios de su HOME.(`tree`)

4. Sitúese en el **directorio** de la asignatura "Lenguajes y Paradigmas de Programación" esto es en el directorio *LPP*. (`cd LPP`)

5. Muestre el contenido del directorio actual. (`ls -la`)

6. Cree un nuevo directorio denominado *prct03* (`mkdir prct03`). Este será el **directorio de trabajo** durante la realización de esta práctica.

7. Sitúese en el directorio *prct03*. (`cd prct03`)

8. Ponga el directorio *prct03* bajo el control de versiones, es decir, cree un repositorio *git*. (`git init`)

9. Cree un fichero de texto, `respuestas.txt`, en el que responda las preguntas de esta práctica de laboratorio. (`touch respuestas.txt`)

10. Añada todo el contenido del directorio *prct03* al *índice del repositorio git*. (`git add .`)

11. Confirme los cambios del índice en el repositorio `git` local.
    (`git commit -m "Creado el fichero de respuestas"`)

12. Cree un repositorio en *GitHub* con el nombre `prct03`.

13. Muestre los repositorios remotos. (`git remote -v`)

14. Cree un repositorio remoto con nombre corto *origin*.
    (`git remote add origin git@github.com:aluXXXXXXXX/prct03.git`)

15. Empuje los cambios en el repositorio remoto denominado *origin*.
    (`git push -u origin master`)

16. Muestre la versión del intérprete de Ruby disponible. (`ruby -v`)

17. A diferencia de los lenguajes compilados, existen dos formas de ejecutar Ruby. Se pueden crear ficheros y ejecutarlos con el intérprete como se hizo en la práctica de laboratorio anterior, y también, se puede introducir el código de forma interactiva. Ejecute el Ruby interactivo (*Interactive Ruby*) (`irb`)

18. ¿Cuál es el resultado de la siguiente operación?

```
2.0.0-p247 :001 > puts "Hello world"
 ???
=> ???
2.0.0-p247 :002 >
```

19. Salga de la sesión interactiva. Para ello escriba el caracter de final del fichero del sistema operativo. (`Ctrl + D`)

20. Escriba el código fuente Ruby que hace que se muestre por la consola la frase `"Hola Mundo"`, almacénelo en el fichero `helloworld.rb` y ejecútelo con el intérprete interactivo. (`irb helloworld.rb`)

21. ¿Cuáles son las diferencias entre la ejecución del programa `"Hola Mundo"` con el intérprete de Ruby (`ruby helloWorld.rb`) y el intérprete interactivo del ejercicio anterior? Escriba la respuesta en fichero creado en el ejercicio 9.

22. Añada todo el contenido del directorio *prct03* al *índice del repositorio git.* (`git add .`)

23. Confirme los cambios del índice en el repositorio `git` local.
    (`git commit -m "Creado el programa Hola Mundo en Ruby"`)

24. Cree un fichero `byebyeWorld.rb` que contenga el siguiente código Ruby:

```
#byebyeWorld.rb
require 'pry'

#define a method
def byebye() puts "bye bye world!!!" end

#set x to 10
x = 10

#start a PEPL session
binding.pry

#program resumes here (after pry session)
puts "program resumes here. Value of x is: #{x}."
```

25. Ejecute el programa `byebyeWorld.rb` para que se lance PRY. (`ruby byebyeWorld.rb`)

26. ¿Cuál es el resultado de cada una de las siguientes operaciones?

```
[1] pry(main)> puts x
???
=> ???
[2] pry(main)> def hello
[2] pry(main)*   puts "Hello world"
[2] pry(main)* end
=> ???
```

```
[3] pry(main)> hello
???
=> ???
[4] pry(main)> byebye
???
=> ???
[5] pry(main)> x = "changed"
=> ???
[6] pry(main)> exit
program resumes here. Value of x is: ???
```

27. ¿Cuáles son las diferencias entre la ejecución del programa `"Hola Mundo"` con el intérprete interactivo y con PRY? Escriba la respuesta en fichero creado en el ejercicio 9.

28. Añada todo el contenido del directorio *prct03* al *índice del repositorio git.* (`git add .`)

29. Confirme los cambios del índice en el repositorio `git` local.

    (`git commit -m "Creado el programa byebyeWorld.rb en Ruby para usar con PRY"`)

30. Muestre el estado del repositorio `git` local, esto es, confirme que el fichero con las respuestas a las preguntas 21 y 27 está correctamente almacenado. (`git status`)

31. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

32. Empuje los cambios en el repositorio remoto denominado *origin.* (`git push -u origin master`)

33. Escriba la dirección del repositorio que ha creado en GitHub en la tarea habilitada en el campus virtual.

    Para realizar los siguientes ejercicios utilice el intérprete interactivo *irb* o PRY. Copie **manualmente** los comandos. NO CORTE Y PEGUE desde el fichero PDF, esto puede introducir carácteres extraños e invisibles dependiendo de la codificación que se esté utilizando.

34. ¿Qué diferencia hay entre `"\t\n"` y `'\t\n'`?

35. ¿Cómo funciona `%q`? ¿Qué es `%q{hello world\n}`? ¿Qué es `%q{'a' 'b' 'c'}`?

36. ¿Cómo funciona `%Q`? ¿Qué es `%Q{hello world\n}`? ¿Qué es `%Q{"a" "b" "c"}`?

37. ¿Qué queda en `c`?

```
>> a = 4
=> 4
>> b = 2
=> 2
>> c = <<HERE
0:0:0" --#{a}--
0:0:0" --#{b}--
0:0:0" HERE
```

38. ¿Qué queda en `c`?

```
0:0:0> a = 4
=> 4
0:0:0> b =2
=> 2
0:0:0> c = <<'HERE'
0:0:0' --#{a}--
0:0:0' --#{b}--
0:0:0' HERE
```

39. `s = "hello"`. ¿Cuál es el valor de las siguientes expresiones?
    - `s[0,2]`
    - `s[-1,1]`
    - `s[0,10]`

40. ¿Qué queda en `g`?

```
>> g = "hello"
=> "hello"
>> g << " world"
```

41. ¿Qué queda en `e`?

```
>> e = '.'*3
```

42. ¿Cuál es el resultado?

```
>> a = 1
=> 1
>> "#{a=a+1} "*3
```

43. ¿Qué es esto? `%w[this is a test]`

44. ¿Qué es esto? `%w[\t \n]`

45. ¿Qué es esto? `%W[\t \n]`

46. ¿Qué contiene `nils`? `nils = Array.new(3)`

47. ¿Qué contiene `zeros`? `zeros = Array.new(3, 0)`

48. ¿Qué queda en `b`?

```
>> x = [[1,2],[3,4]]
=> [[1, 2], [3, 4]]
>> b = Array.new(x)
```

49. ¿Qué queda en `c`?

```
>> c = Array.new(3) { |i| 2*i }
```

50. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = ('a'..'e').to_a
=> ["a", "b", "c", "d", "e"]
>> a[1,1]
=>  ???
>> a[-2,2]
=>  ???
>> a[0..2]
=>  ???
>> a[0...1]
=>  ???
>> a[-2..-1]
=>  ???
```

51. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a
=> ["a", "b", "c", "d", "e"]
>> a[0,2] = %w{A B}
=> ["A", "B"]
>> a
=>  ???
>> a[2..5] = %w{C D E}
=> ["C", "D", "E"]
>> a
=>  ???
>> a[0,0] = [1,2,3]
=> [1, 2, 3]
>> a
=>  ???
>> a[0,2] = []
=> []
>> a
=>  ???
>> a[-1,1] = [ 'Z' ]
=> ["Z"]
>> a
=>  ???
>> a[-2,2] = nil
=> nil
>> a
=>  ???
```

52. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = (1...4).to_a
=> ???
>> a = a + [4, 5]
=> ???
>> a += [[6, 7, 8]]
=> ???
>> a = a + 9
=> ???
```

53. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> x = %w{a b c b a}
=> ???
>> x = x - %w{b c d}
=> ???
```

54. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> z = [0]*8
=> ???
```

55. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = []
=> []
>> a << 1
=> ???
>> a << 2 << 3
=> ???
>> a << [4, 5, 6]
=> ???
>> a.concat [7, 8]
=> ???
```

56. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = [1, 1, 2, 2, 3, 3, 4]
=> [1, 1, 2, 2, 3, 3, 4]
>> b = [5, 5, 4, 4, 3, 3, 2]
=> [5, 5, 4, 4, 3, 3, 2]
>> c = a | b
=> ???
>> d = b | a
=> ???
>> e = a & b
=> ???
>> f = b & a
=> ???
```

57. ¿Qué resultados dan las siguientes operaciones?

```
>> a = 1..10
=> 1..10
>> a.class
=> Range
>> a.to_a
=> ???
>> b = 1...10
=> 1....10
>> b.to_a
=> ???
>> b.include? 10
=> ???
>> b.include? 8
=> ???
>> b.step(2) {|x| print "#{x} " }
=> ???
>> 1..3.to_a
=> ???
```

58. ¿Qué resultados dan las siguientes operaciones?

```
>> r = 0...100
=> 0....100
>> r.member? 50
=> ???
>> r.include? 99.9
=> ???
>> r.member? 99.9
=> ???
```

59. ¿Qué resultados dan las siguientes operaciones?

```
>> true.class
=> ???
>> false.class
=> ???
>> puts "hello" if 0
=> ???
>> puts "hello" if nil
=> ???
>> puts "hello" if ""
=> ???
```

60. ¿Qué resultados dan las siguientes operaciones?

```
>> x = :sym
=> :sym
>> x.class
```

```
=> ???
>> x == 'sym'
=> ???
>> x == :sym
=> ???
>> z = :'a long symbol'
=> :"a long symbol"
>> z.class
=> ???
>> x == 'sym'.to_sym
=> ???
>> x.to_s == 'sym'
=> ???
```

61. ¿Qué resultados se dan?

```
>> s = "Ruby"
=> "Ruby"
>> t = s
=> ???
>> t[-1] = ""
=> ???
>> print s
 ???
>> t = "Java"
=> ???
>> print s, t
 ???
```

62. ¿Cuál es el resultado?

```
>> "%d %s" % [3, "rubies"]
=> ???
```

63. ¿Cuáles son los resultados?

```
>> x, y = 4, 5
=> ???
>> z = x > y ? x : y
=> ???
>> x,y,z = [1,2,3]
=> ???
```

64. ¿Qué resultados dan las siguientes operaciones?

```
>> x = { :a => 1, :b => 2 }
=> {:b=>2, :a=>1}
>> x.keys
=> ???
>> x.values
=> ???
>> x[:c] = 3
=> 3
>> x
=> ???
>> x.delete('a')
=> nil
>> x
=> ???
>> x.delete(:a)
=> 1
>> x
=> ???
>> x = { :a => 1, :b => 2, :c => 4 }
=> {:b=>2, :c=>4, :a=>1}
```

```
>> x.delete_if { |k,v| v % 2 == 0 }
=> ???
>> x
=> ???
```

65. ¿Qué hace la siguiente sentencia? `counts = Hash.new(0)` ¿Qué diferencia hay con la asignación `counts = {}`?

66. ¿Qué retorna esta expresión regular? `'hello world, hello LPP'.scan /\w+/`

67. Cierre la sesión.