Universidad de La Laguna. Escuela Técnica Superior de Ingeniería Informática
Tercero del Grado de Informática
LENGUAJES Y PARADIGMAS DE PROGRAMACION. TERCERA PARTE
5 páginas

Nombre: _____ Alu: _____

1. Sea `x` una variable que contiene un objeto de la clase `Class`. ¿Por que otro nombre se conoce a los métodos singleton del objeto `x`?

2. ¿En que clase se alojan los métodos singleton de `x`?

3. ¿Que diferencia hay entre la eigenclass de un objeto de la clase `Class` y la de un objeto ordinario?

4. ¿Que queda en `z`?

```
z = class << String; self end
```

5. Sea `M` un módulo y `C` una clase. Que ocurre si ejecutamos `C.extend(M)`? ¿Donde acaban los métodos de instancia de `M`?

6. Cuando se llama `o.m` al método `m` con receptor el objeto `o` ¿En que clase busca Ruby primero por `m`?

7. ¿Cuanto vale `self` en las líneas 2 (dentro y fuera del método), 3 (dentro y fuera del bloque), 4 (dentro y fuera del `class_eval`) y 6 (cuando el código generado a partir de la cadena es finalmente evaluado)?

```
1  class Module
2    def my_attr_reader(*syms)
3      syms.each { |s|
4        class_eval %{
5                      def #{s}
6                        @#{s}
7                      end
8                    }
9      }
10   end
11 end
```

8. El método `my_attr_reader` en el ejercicio 7, ¿es de instancia? ¿de clase? ¿o ambas cosas? ¿de que tipo es el método generado en las líneas 5 - 6? Justifique su respuesta

9. ¿Que hace el método `extend` cuando es llamado en un objeto? ¿Que argumentos recibe?

10. Dado el código:

```
def multiplier(n)
  lambda do |*arr|
            arr.collect { |i| i*n }
          end
end
```

    a) ¿Que retorna `multiplier`?

    b) Ponga un ejemplo de uso del objeto retornado por `multiplier`

11. ¿Que diferencias hay entre `instance_eval`, `class_eval` y `eval`?

12. Dado el programa:

```ruby
v1 = 1

class Tutu
  v2 = 2
  puts "%2d" %(__LINE__.to_s)+' '+local_variables.inspect

  def my_method
    v3 = 3
    puts "%2d" %__LINE__.to_s+' '+local_variables.inspect
  end

  puts "%2d" %__LINE__.to_s+' '+local_variables.inspect
end

obj = Tutu.new
obj.my_method
obj.my_method
puts "%2d" %__LINE__.to_s+' '+local_variables.inspect
```

¿Cuál es la salida de este programa?

13. ¿En que clase se define `define_method`? ¿Es un método de instancia o de clase? ¿Que visibilidad tiene `define_method`? (público, privado, protegido) ¿En que clase queda definida el método creado por `define_method`? (Justifique brevemente sus respuestas)

14. Suponga que quiere crear dos métodos `counter` e `inc` dentro del cuerpo de un método de instancia `define_methods`:

```ruby
def define_methods
  shared = 5

  define_method :counter do
    shared
  end

  define_method :inc do |x|
    shared += x
  end
end

define_methods
```

Este programa cuando se intenta ejecutar `defined_methods` produce un error.

a) ¿Que clase de error? (explique su respuesta)

b) ¿Cómo puede arreglarse? (explique su respuesta)

15. *a*) ¿Es posible definir en Ruby un método cuyo nombre no sea un identificador, esto es, no case con la expresión regular `/^[a-z_]+\w*$/i`?

b) ¿Cómo puedo hacer para invocar a dicho método?

16. Escriba un método `compose` que reciba como argumentos dos lambdas y retorne la composición de las mismas

17. Leyendo el siguiente fragmento de código en el que se llama repetidas veces a `element`:

```ruby
class HTMLForm < XMLGrammar
  element :button, :name => OPT, :value => OPT,
                   :type => "submit", :disabled => OPT
  element :br
end
```

Justifique brevemente sus respuestas.

- ¿Sabría decir cuantos argumentos espera `element`?,

- ¿De que clase son?
- ¿Existen argumentos opcionales?
- ¿El método `element` es de clase o de instancia?

18. ¿Como puedo incorporar un método definido en una cadena `x = 'def tutu; "tutu"; end` como método de clase, en una cierta clase `A`?

19. ¿Cómo puedo distinguir si un `Proc` es un `proc` o una `lambda`?

20. En la llamada `tutu(a, b, &c)` ¿A quien hace referencia `c`? ¿Puede ser `tutu` llamada con dos argumentos? ¿A que clase pertenece `c` dentro del cuerpo del método `tutu`?

21. Sea `p` un objeto `Proc` y supongamos que tengo un método `f` que espera además de algunos argumentos `a,b` que le siga un bloque al cual llama para realizar su tarea. Podría llamar a `f` usando `p` así:

    ```
    z = f(a, b) { |x| p.call(x) }
    ```

    ¿Cómo puedo escribir una versión mas abreviada del código anterior que haga que Ruby *convierta* el `Proc` en un bloque?

22. Describa el algoritmo de búsqueda de `Ruby` para el método de clase `my_method()`:

    ```
    class MyClass
      def self.my_method; 'my_method()'; end
    end
    class MySubClass < MyClass
    end
    MySubClass.my_method()
    ```

23. Describa el algoritmo de búsqueda de `Ruby` para el método `my_method()`:

    ```
    module M
      def my_method; 'M#my_method()'; end
    end
    class C
      extend M
    end
    class D < C ; end
    D.my_method()
    ```

24. ¿Cuál es la salida? ¿Por qué?

    ```
    > def a_method(a = 10, b = 20, c = 100, *d)
    >   return a,b,c,d
    > end
    => nil
    > a_method(1,2)
    => ?
    >
    ```

25. ¿Cómo se puede encontrar una gema en la maquina local? y ¿en remoto? ¿ y en ambos?

26. ¿Qué comando permite actualizar a la última versión una gema ?

27. ¿Qué comando permite saber donde está instalada una gema?

28. ¿Qué estructura de directorios genera la orden `bundle gem ejemplo`?

29. ¿Qué objetivos tiene el `Rakefile` que se genera por defecto?

30. ¿En qué directorio hay que colocar los ficheros con los códigos fuente `Ruby` de la gema?

31. ¿Qué convenios se siguen con los nombres de gema, nombres de ficheros y nombres de clase cuando se desarrolla una gema en Ruby?

32. Describa que comentarios aprecerán en la documentación generada por `RDoc`:

```ruby
# Esta es una clase que no hace nada
class AClass
  # Esta es la documentacion de este metodo
  def a_method_1
  end

  def a_method_2 #:nodoc:
  end
end
```

33. ¿Para qué se utiliza la opción `--all` de la herramienta `RDoc`?

34. En la práctica de las expresiones regulares la idea era asociar con cada uno de los operadores de las expresiones regulares un método que retornaba una lambda. Esa lambda recibe como entrada la subcadena no procesada y retorna `false` a menos que un prefijo de la cadena case con la expresión regular. En otro caso retoran el sufijo no procesado. Escriba el método asociado con el operador or: `r | q`

35. Escriba el método asociado con el operador de concatenación: `rq`

36. Escriba el método asociado con la cadena vacía ϵ

37. Escriba el método que recibe como entrada una `String` y retorna la lambda asociada que reconoce las cadenas que contienen un prefijo de diche `String`

38. ¿Donde espera un programador de cultura Ruby que esta visitando nuestro proyecto `my-gem` en GitHub encontrar definida una constante cuyo nombre es `MyGem::VERSION`? ¿En que directorio y fichero del proyecto?

39. ¿Que se debe poner en el fichero de librería principal `lib/my-gem.rb`?

40. ¿Que se debe poner en el directorio `lib/my_gem/`?

41. ¿En que directorio y fichero del proyecto debo esperar que se encuentre la clase `MyGem::Widgets::FooBar`?

42. ¿Que hace este comando? ¿Qué gema hay que instalar para que funcione?

```
gem yank tic-tac-toe -v0.3.4
```

43. En clase vimos como diseñar un DSL que llamamos `xripper` para procesar un documento XML. El DSL dispone de métodos como `on_path`, `action`, `before`, `after`, `run_path_actions` y `run`. Este es un ejemplo de uso del DSL:

```ruby
on_path '/document/title' do |t|
  puts "Title: "+t.text
end
on_path '/document/author' do |a|
  a.text = "J.R.R. Tolkien"
  puts "Author: "+a.text
end
action { puts "Chapters: " }
on_path '/document/chapter/title' do |ct|
  puts "  "+ct.text
end
```

El método `initialize` de la clase `Xmlripper` que implanta el DSL hace posible la construcción de las estructuras de datos necesarias para la ejecución del *programa xripper*. Complete el código que falta en la definición de `initialize`. Recuerde: para preservar el orden de las acciones el atributo `@path_action` es un array de pares `[path, action]`:

```
require 'rexml/document'
class XmlRipper

  def initialize( path = nil, ___________ ) # block
    @before_action = _________________ # by default, do nothing
    @path_action   = _________________ # Array containing the actions
    @after_action  = _________________ # by default, do nothing

    if path then    # Read the xripper script from "path"
      _________________( File.read( ___ ) )
    elsif block_given? # Execute the block in the apropriate context
      _________________( ___________ )
    end
  end
  ...
end
```

44. Complete la definición del método `on_path` que indica que una cierta acción debe ser ejecutada cada vez que un XPath case:

```
def on_path(path, &block)
  @path_action ___ ___________
end
```

45. Complete la definición del método `before` que describe la acción a ejecutar antes del procesado del documento:

```
def before( _____ )
  @before_action = ________
end
```

46. Complete la definición del método `run_path_actions` que ejecuta las acciones asociadas sobre cada elemento del documento que casa con el XPath correspondiente:

```
def run_path_actions(doc)
  @________.each do |pa|
    _____, _____ = pa
    REXML::XPath.each(doc, _____) do |element| # An XMLPath
      _____[element]
    end
  end
end
```

47. Complete la definición del método `run` que indica como procesar un script escrito en el DSL XRipper:

```
def run(file_name)
  File.open(file_name) do |f|
    doc = REXML::Document.new(f)
    @___________[doc]              # Execute "before" action
      _____________-actions(doc)  # Execute all pairs
    @___________[doc]
  end
end
```