Tercer parcial Lenguajes y Paradigmas de Programación
Curso 2017-18

Nombre:_____

Normas importantes

- **Los profesores no contestarán ninguna pregunta durante la realización del examen, exceptuando aquellas que estén relacionadas con algún posible error en el enunciado de alguna pregunta.**
- **Todos los ejercicios tienen una puntuación de 0,5 puntos**
- **La duración del examen es de 2 horas.**

Ejercicio 1

Suponiendo el siguiente enumerado, define dos variables que contengan dos valores del enumerado, uno del caso 'a' y otro del caso 'b'.

```
enum A {
    case a(Bool, String, Int)
    case b(Int)
}

let _____________________________________

let _____________________________________
```

Ejercicio 2

Supongamos el siguiente código de Scheme. ¿Qué aparece en pantalla?

```
let x = [(10, -2), (3, 40), (-5, 26), (22, -3), (10, 8)]
let a = x.map{$0.0 + $0.1}
print (a.reduce(a[0]){if $0 < $1 {return $0} else {return $1}})
```

Respuesta:

Ejercicio 3

Dada la siguiente definición del tipo enumerado que representa un árbol genérico:

```
indirect enum Arbol<T> {
    case nodo(T, [Arbol<T>])
    case hoja(T)
}
```

Escribe una sentencia en donde se declare e inicialice la variable denominada `arbolString` y se le asigne el siguiente árbol

```
    "hola"
     /   \
  "que"  "tal"
```

Respuesta:

Ejercicio 4

Dada la misma definición del tipo enumerado del ejercicio anterior, define la función genérica `esHoja(arbol:)` que recibe un árbol genérico y devuelve `true` si es un árbol hoja y `false` en caso contrario

Ejercicio 5

Indica qué muestra por pantalla la función print:

```
let posiciones = [(2,1), (3,4), (2,4), (8,4), (4,3)]

print(posiciones.filter{$0.0 % $0.1 == 0}.map{($0.1, $0.0)}.reduce(0){$0 +
$1.0})
```

Respuesta:

Ejercicio 6

Supongamos la función foo que recibe un entero y devuelve un entero opcional:

```
func foo(x: Int) -> _____________ {
   ...
}
```

Completa la definición de la función, indicado el tipo devuelto. Y escribe el código en Swift que llame a la función `foo` con el parámetro 10 y que imprima el doble del resultado devuelto. Si el valor devuelto es `nil` se debe imprimir `"Sin resultado"`.

Ejercicio 7

Dado el siguiente código en Swift:

```
enum Dispositivo {
    case iPad, iPhone, AppleTV, AppleWatch
    func lanzamiento() -> String {
       switch self {
       case .AppleTV: return "\(self) salió en 2006"
       case .iPhone: return "\(self) salió en 2007"
       case .iPad: return "\(self) salió en 2010"
       case .AppleWatch: return "\(self) salió en 2014"
       }
    }
}

print(_____________________________________________________)
```

Queremos imprimir por pantalla "`iPhone salió en 2007`" utilizando el código anterior. En el caso en que creas que es posible completar la sentencia print, escribe cuál sería. En el caso en que creas que se genera un error, explícalo.

Ejercicio 8

Dado el siguiente código en Swift:

```
var x = 1

func foo(_ b: Int) -> ((Int) -> Int, (Int) -> Int) {
    var x = 1 + b

    func bar1(_ a: Int) -> Int{
        x = x + a
        return x
    }

    func bar2(_ a: Int) -> Int {
        x = x + a + 1
        return x
    }
    return (bar1, bar2)
}
```

Rellena los huecos con lo que se imprime por pantalla:

```
let f = foo(x)
f.0(x)
print(f.1(x)) → ________
let g = foo(x+2)
print(g.0(x)) → ________
```

Ejercicio 9

Dado el siguiente código en Swift, ¿qué se imprime por pantalla?

```
func foo() -> (Int) -> Int {
    var x = 10

    func bar(_ a: Int) -> Int {
        x = a + x
        return x
    }
    return bar
}

var x = 1
let numeros = [Int](0...5)
let z = foo()
print(numeros.map(z))
```

Respuesta:

Ejercicio 10

¿Con qué argumentos hay que llamar a la función `aplica` en el ejemplo para que imprima [[4, 5, 6], [4, 5], [1, 2, 3]]?

```
func paraPares(_ par: Int) -> [Int] {
    let n = par / 2
    return [n,n+1]
}
func paraImpares(_ impar: Int) -> [Int] {
    let n = (impar+1) / 2
    return [n-1,n,n+1]
}
func aplica(_ numeros: [Int] ,_ funciones: [(Int)->[Int]]) -> [[Int]] {
      return numeros.map(){funciones[$0%2]($0)}
}

// Ejemplo:
print(aplica(_____________________________________________________))
// Imprime: [[4, 5, 6], [4, 5], [1, 2, 3]]
```

Ejercicio 11

Rellena los huecos de la función `componer` para que el ejemplo imprima:

```
hgf(x)=2x^2+5 para x=5.0 es 55.0
```

```
func componer(funciones:_____________________) -> ____________________ {
    func compuesta(_ x: Double) -> Double {
        return funciones.reduce(x){$1($0)}
    }
    _____________________
}

//Ejemplo:
func f(_ x: Double) -> Double {return x*x}
func g(_ x: Double) -> Double {return 2*x}
func h(_ x: Double) -> Double {return x+5}
let hgf = componer(funciones: [f,g,h])
print("hgf(x)=2x^2+5 para x=5.0 es \(hgf(5.0))")
// Imprime:
// hgf(x)=2x^2+5 para x=5.0 es 55.0
```

Ejercicio 12

Completa las líneas de código indicadas para que el resultado de la última llamada imprima el total de pasos incrementados por todas las instancias de `Pasos`.

```
class Pasos {
    var contador = 0 {
        willSet {

            _____________________________
        }
    }

    _____________________________

    func inc(pasos: Int) {
        if (pasos > 0) {
            contador += pasos
        }
    }

    var totalPasos: Int {

         _____________________________________
    }
}

var p1 = Pasos()
var p2 = Pasos()
p1.inc(pasos: 10)
p1.inc(pasos: 5)
p2.inc(pasos: 5)
print (p1.totalPasos)
// Imprime 20
```

Ejercicio 13

Completa el código de la estructura `MiStruct` para que compile correctamente:

```
protocol A {
    var a: String {get}
    func foo(a: String) -> String?
}

protocol B {
    mutating func bar()
}

struct MiStruct: A, B {
      // Completa el código









}
```

Ejercicio 14

Completa el return del código que aparece en la estructura `Vector2D` para que la función devuelva el vector2D resultado de multiplicar un escalar por el propio vector2D, es decir, de multiplicar el escalar por cada componente del propio vector.

```
struct Vector2D {
   var x = 0.0, y = 0.0
   func producto(escalar k: Double) -> Vector2D {

          return ____________________________________________
   }
}
```

Ejercicio 15

Indica qué muestra por pantalla la función print:

```
func foo(palabra pal: String) -> String {
    let lon = pal.count
    if lon < 2 {
        return pal
    }
    else {
        let start = pal.startIndex
        let end =  pal.index(start, offsetBy: lon/2)
        return String(pal[end])
    }
}


class MisPalabras {
    var guardadas: [String] = []
    func guarda(palabra: String) {
        guardadas.append(palabra)
    }

    var x : [String] {
        get {
            return guardadas.map(foo)
        }
    }
}


let palabras = MisPalabras()
palabras.guarda(palabra: "sal")
palabras.guarda(palabra: "limon")
print(palabras.x)
```

Respuesta:

Ejercicio 16

Completa el código de la siguiente extensión utilizando propiedades calculadas, para que el ejemplo funcione correctamente.

```
extension Int {
      // Completar el código









}

let a = 3.sum
print("a: \(a)")  // a: 13
let b = 100.res
print("b: \(b)")  // b: 90
let c = 30.mul
print("c: \(c)")  // c: 300
let d = 24.div
print("d: \(d)")  // d: 12
let op = 30.sum + 10.res
print("op: \(op)") // op: 40
```

Ejercicio 17

Completa la siguiente implementación para que las pruebas funcionen correctamente:

```
enum CodigoBarras : Equatable{
    case upc(Int, Int)
    case qrCode(Int)


    ____________________________________________________________________ {
        switch (c1, c2) {
           case let (.upc(codeA1, codeB1), .upc(codeA2, codeB2)):

               return ____________________________________

           case let (.qrCode(code1), .qrCode(code2)):

               return ____________________________________

           default:
               return false
       }
    }
}

print(CodigoBarras.qrCode(11) == CodigoBarras.qrCode(11)) // true
print(CodigoBarras.upc(1234, 1234) == CodigoBarras.upc(2222, 1111)) //
false
```

Ejercicio 18

Dado el siguiente código con errores en Swift:

```
protocol Base {
    func saluda() -> String {
        return "soy base"
    }
}

class Foo: Base {
    func saluda() -> String {
       return "soy foo"
    }
}

// Las siguientes líneas no debes cambiarlas

var arr = [Foo(), Base()]
for elem in arr {
   print(elem.saluda())
}
// Imprime
//    "soy base soy foo"
//    "soy base"
```

Corrige las definiciones de `Base` y `Foo` en el código de arriba (tachando lo incorrecto y sustituyéndolo por el código correcto) para que el compilador no lance ningún error y la salida por pantalla sea exactamente la indicada. En el método `saluda` de la clase Foo no puedes escribir la cadena `"soy base"`.

Ejercicio 19

Dado el siguiente código Swift, completa el bucle para que se imprima lo indicado:

```
protocol P {
    func saluda() -> String
}

class A: P {
    func saluda() -> String {
        return "Soy de la clase A"
    }
    func foo() -> Int {
        return 0
    }
}

class B: P {
    func saluda() -> String {
        return "Soy de la clase B"
    }
    func bar() -> Int {
        return 100
    }
}

let instancias: [P] = [A(), B()]

for x in instancias {
    // Completa el código









}
// Imprime:
// Soy de la clase A
// 0
// Hola soy de la clase B
// 100
```

Ejercicio 20

Define una extensión para el tipo `Array` sólo cuando sus elementos sean numéricos (puedes usar el protocolo `Numeric` definido en Swift), que implemente un método que sume todos sus elementos.

Ejemplo:

```
let a = [Int](0...5)
print(a.sum())  // 15

let b : [String] = ["h", "d", "e"]
print(b.sum())  // Error
```

Respuesta: