Introducción a la Programación Funcional

- Funciones de orden superior, Clausuras

Clausura

- En Informática, una clausura es una función evaluada en un entorno que contiene una o más variables dependientes de otro entorno. Cuando es llamada, la función puede acceder a estas variables. El uso explícito de clausuras se asocia con la programación funcional y con lenguajes como el ML y el Lisp. Construcciones como los objetos en otros lenguajes pueden también modelarse con clausuras.

Introducción a la Programación Funcional

- En la programación funcional, las funciones son el elemento más importante. Y se puede pasar por parámetro a otras funciones.
- Muchos lenguajes imperativos copiaron esta característica, por ser muy útil y ayudar a la reutilización de código. Veamos ejemplos:

Introducción a la Programación Funcional

- Javascript

```
var a = function (i) { alert(i) }
a("hola");
var a2 = function (arreglo, fx) {
for(i in arreglo) fx(arreglo[i])
}
a2("hola",a)
```

Introducción a la Programación Funcional

Python

```
>>> def greeter():
...   print("Hello")
...
>>> #An implementation of a repeat function
>>> def repeat(fn, times):
...   for i in range(times):
...     fn()
...
>>> repeat(greeter, 3)
Hello
Hello
Hello
```

Introducción a la Programación Funcional

Java 8

```
// Print Desc
System.out.println("=== Sorted Desc SurName ===");

Collections.sort(personList, (p1,  p2) ->
p2.getSurName().compareTo(p1.getSurName()));

for(Person p:personList){
    p.printName();
}
```

Scala

```
def buscar(lista: List[Int], com:(Int, Int) => Boolean): Int =
  if (lista.tail.isEmpty) lista.head
  else if (com(lista.head, buscar(lista.tail, com))) lista.head
  else buscar(lista.tail, com)

  def max(lista: List[Int]) : Int = buscar(lista, (a:Int, b:Int) =>  (a > b) )

  def min(lista: List[Int]) : Int = buscar(lista, (a:Int, b:Int) =>  (a < b) )

  val lista = List(1,2,3,4,5,6)

  min(lista)                                  //> res0: Int = 1
  max(lista)                                   //> res1: Int = 6
```

Funciones anónimas

- Son funciones temporales que se utilizan en el momento y no tiene sentido referenciarlas con un nombre.

  En el ejemplo anterior la función que devuelve el mayor de dos valores o el menor de dos valores, tiene un uso efímero:

  (a:Int, b:Int) => (a < b)

  (a:Int, b:Int) => (a > b)

Funciones Anónimas

- En javascript es ampliamente utilizado, dado que este lenguaje se utiliza para manejo de eventos o comunicación asíncrona
- Veamos un ejemplo de una comunicación simple con un servidor:

```
$.get( "ajax/test.html", function( data ) {
  alert( data );
});
```

Funciones Anónimas

- En el ejemplo la función se ejecuta si la comunicación tiene éxito, dado que esta función no se utiliza en otro lugar, no es necesario declararla.

Currying

- Como hemos dicho una función puede ser pasada por parámetros pero también devuelta como resultado de una función.
- Currificar es la técnica inventada por Moses Schönfinkel y Gottlob Frege que consiste en transformar una función que utiliza múltiples argumentos (o más específicamente una n-tupla como argumento) en una función que utiliza un único argumento.

Currying

- Oficialmente cada función de Haskell solo puede tomar un parámetro. Todas las funciones que aceptan más de un parámetro hacen uso de currying.

max 4 5

es igual que llamar

(max 4) 5

(max 4) es una función que devuelve verdadero si el numero pasado por parámetro es mayor a 4 y falso de lo contrario.

Ejemplos:

- Por ejemplo en Haskell se puede definir una función double de la siguiente manera:

```
mult :: Int -> Int -> Int

mult x y = x * y


double = mult 2
```

Ejemplos en Scala:

```
object CurryTest extends Application {

  def filter(xs: List[Int], p: Int => Boolean): List[Int] =???

  def modN(n: Int)(x: Int) = ((x % n) == 0)

  val nums = List(1, 2, 3, 4, 5, 6, 7, 8)
  println(filter(nums, modN(2)))
  println(filter(nums, modN(3)))
}
```