# Programación funcional con Python (III)

## Recursividad

La recursividad es un concepto muy importante en la programación en general y en programación funcional en particular, ya que en este paradigma no se cuenta con estructuras de repetición (ej. `for, while`)

La idea de recursividad se trata de descomponer un problema en varios problemas sencillos del mismo tipo, e implementar esos problemas sencillos como funciones que se llaman a sí mismas.

Un ejemplo típico: la función factorial.

```
5! = 5 . 4 . 3 . 2 . 1 = 120
```

Observando este desarrollo, podemos observar que:

```
5! = 5 . 4!
4! = 3! . 2
3! = 2! . 1


1! = 1
```

Podríamos escribir: `n! = n . (n - 1)!`

que en Python podemos escribirlo:

```
def factorial(x):
    return x * factorial(x-1)

print(factorial(5))
```

Este código genera `RecursionError: maximum recursion depth exceeded`

¿Por qué? Porque no dijimos cuándo terminar (y no lo hace al menos hasta que Python detecta que se excede la cantidad de llamadas recursivas)

Cuando usamos recursión es muy importante proveer un **caso base** que actúe como **condición de salida**.

En este caso, vemos que 1! = 1 por lo que podemos fijar el caso base como 1 para 1!

```
def factorial(x):
      if x == 1:
            return 1
      else:
            return x * factorial(x-1)

print(factorial(5))
```

```
factorial(5)   ---------------------------------------->

 devuelve 5 * factorial(4)

              devuelve 4  *  factorial(3)

                               devuelve 3 * factorial(2)

                                                devuelve 2 * factorial(1)

                                                                devuelve 1

                                                          2   *    1

                                              3  *  2   *  1

                           4  *  3  *  2 * 1

               5 * 4 * 3 * 2 * 1

     <-----------------------------------------------------
```

La recursión puede ser indirecta: una función puede llamar a una segunda función que llama a la primera y así sucesivamente… (Puede ocurrir entre más de dos funciones)

```
# recursion indirecta
def is_even(x):
  if x == 0:
    return True
  else:
    return is_odd(x-1)

def is_odd(x):
  return not is_even(x)

print(is_odd(17))
```

```
print(is_even(23))
```

## Algunos ejemplos

- Sumatoria de los números en una lista.

  `lista[p] + lista[p-1] + lista[p-2] + ... + lista[0] + 0`

  - caso base: si no hay elementos en la lista, devuelvo 0

```
# sumatoria de los números en una lista
def sumatoria(c):
      if len(c) == 0:
            return 0
      else:
            return c[0] + sumatoria(c[1:])

lista = [3, 5, 1, 2]
print(sumatoria(lista))
```

- Suma dos números

  `ej. si a = 7, b = 4`

  `7 + 4 = 6 + 5 = 5 + 6 = 4 + 7 = 3 + 8 = 2 + 9 = 1 + 10 = 0 + 11`

  - caso base: `a == 0`

```
# suma dos números
def add(a,b):
    if a == 0:
       return b
    else:
       return add(a-1, b+1)

print(add(7, 4))# suma dos números

def add(a,b):

    if a == 0:

       return b

    else:
```

```
        return add(a-1, b+1)


print(add(7, 4))
```

- Cuenta la cantidad de una letra c en un string

```
# cuenta la cantidad de una letra
# en un string
def cuenta_a(s, c):
      if len(s) == 0:
            return 0
      elif s[0] == c:
            return 1 + cuenta_a(s[1:], c)
      else:
            return 0 + cuenta_a(s[1:], c)

print(cuenta_a("aparato", 'a'))
print(cuenta_a("experto", "a"))
print(cuenta_a("experto", "o"))
```