TEMA IV

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"Un sueño no se hace realidad por arte de magia, necesita sudor, determinación y trabajo duro"

Colin Powell

1. PROGRAMACIÓN FUNCIONAL

El paradigma funcional se empezó a desarrollar por el matemático John McCarthy en 1956, para programar los primeros proyectos de inteligencia artificial sobre un computador IBM 704 durante su desarrollo este crea el lenguaje de programación lisp en 1958.

La programación funcional nos presenta una nueva forma de programar de manera declarativa, se parecerá más a escribir y su premisa principal es que el código sea más intuitivo que nunca.

Se utiliza en particular en el desarrollo de aplicaciones técnicas y matemáticas. Inteligencia Artificial (IA) Compiladores y analizadores

Todas las funciones son puras: Dado un mismo input siempre devolvemos el mismo output. Todos los valores son inmutables: Lo único que podemos hacer es generar nuevos valores. No hay bucles: La iteración se realiza usando recursividad.

A pesar de que Python no es un lenguaje puramente funcional, nos ofrece algunas primitivas propias de lenguajes funcionales, como map, filter y reduce. Todas ellas ofrecen una alternativa al uso de bucles para resolver ciertos problemas.

Funciones lambda

Las funciones lambda nos sirven para crear pequeñas funciones anónimas, de una sola línea sobre la marcha. Una función anónima, como su nombre indica es una función sin nombre, que podemos ejecutar sin definirla con la palabra reservada def.

```
>>> cuadrado = lambda x: x**2
>>> cuadrado(2)
```

Las funciones lambda no tienen nombre, python crea una referencia a un objeto función.
La sintaxis

```
lambda parámetros: expresión
```

Podemos tener varios parámetros, pero sólo una expresión.

Una función lambda puede ser parámetro de otra función

```
lista=[(3,2),(4,1),(8,4)]
lista.sort(key=lambda x: x[1])
print(lista)
```

Estas funciones se utilizan mucho con las siguientes funciones que estudiaremos: map, filter, reduce

Función map

map(función,secuencia): Ejecuta la función enviada por parámetro sobre cada uno de los elementos de la secuencia.

Ejemplo: dada una lista de números generar otra con sus cuadrados

```
lista=[1,4,2,6]
cuadrados=list(map(lambda x: x*x,lista))
print(cuadrados)
```

Este mismo ejercicio a la manera tradicional

```
lista=[1,4,2,6]
def cuadrado(x):
    return x*x
cuadrados=[]
for e in lista:
    cuadrados.append(cuadrado(e))
print(cuadrados)
```

Al utilizar la función map con una función lambda además de escribir un código más breve nos ahorramos los bucles

Función filter

filter(función,secuencia): Devuelve una secuencia con los elementos de la secuencia pasada por parámetro que devuelvan True al aplicarle la función pasada también como parámetro.

```
lista=[3,4,1,2,2,9]
listaPares=list(filter(lambda x: x%2==0,lista))
print(listaPares)
```

Función reduce

reduce(función,secuencia): Devuelve un único valor que es el resultado de aplicar la función a los elementos de la secuencia. Se utiliza principalmente para llevar a cabo un cálculo acumulativo sobre una lista de valores y devolver el resultado. En este caso, la función que se pasa como primer parámetro recibe dos argumentos. Se encuentra en el módulo funtools

```
from functools import reduce
lista=[6,7,5,9]
suma=reduce(lambda x,y: x+y, lista)
print(suma)
```

List comprehension

Nos proporciona una alternativa para la creación de listas. Es parecida a la función map, pero aplica una expresión en lugar de una función. La estructura de las list comprehensions es la siguiente:

```
[ expresion(i) for i in list if condición ]
```

Entre corchetes definimos una expresión seguida de un bucle for al que opcionalmente le pueden seguir otros bucles for y/o una condición.
El resultado siempre es una lista.

```
>>> [x ** 3 for x in [1,2,3,4,5]]
[1, 8, 27, 64, 125]

>>> [x for x in range(10) if x % 2 == 0]
[0, 2, 4, 6, 8]

>>> [x + y for x in [1,2,3] for y in [4,5,6]]
[5, 6, 7, 6, 7, 8, 7, 8, 9]
```

El código anterior es similar al siguiente:

```
nueva_lista = []
for i in list:
if condición:
nueva_lista.append(expresion(i))
```

Ejemplos:

1. Listar los ficheros python del directorio actual que comienzan

```
import os
ficheros_python = [f for f in os.listdir('.') if f.endswith('.py') ]
print(ficheros_python)
```

2. Número, doble y cuadrado de los números del 0 al 9

```
num_doble_cuadrado = [(num, num*2, num**2) for num in range(10)]
print(num_doble_cuadrado)
```