

# **Práctica 1**

## **-**

# **Posta en producción segura**

# Índice

1. Sucesión de fibonacci .....	3
2. Test unitario de la función Fibonacci .....	5

# 1. Sucesión de fibonacci

Para a realización de unha función que calcule a sucesión de Fibonacci dado un número creamos o script fibo.py, no interior do script declaramos unha función chamada “Fibonacci(n)” que toma un parámetro ‘n’ que determinará o número de elementos da sucesión.

```
en_produccion_segura > Practica1-PPS > Fibonacci >
def Fibonacci(n):
```

A cotinuación declaramos unha variable ‘count’ que contará as veces que se executa un bucle while aumentando o seu valor en 1 ata que sexa maior ou igual ao valor de ‘n’.

```
def Fibonacci(n):
    #Inicializamos a variable 'count'
    count = 0
```

No paso posterior inicializamos dúas variables ‘a’ e ‘b’ que serán os dous primeiros elementos da sucesión de fibonacci que serán 0 e 1.

```
#Se inicializan as variables 'a' e 'b' cos 2 primeiros valores da sucesión fibonacci.
a = 0
b = 1
```

Para continuar declaramos unha lista chamada ‘fibonacci’ no que se irán gardando en orde os diferentes elementos da sucesión de fibonacci, está lista será o resultado que devolva a nosa función.

```
#Inicializamos un array no que se gardará o resultado da sucesión.
fibonacci=[]
```

Antes de declarar o bucle do que falamos anteriormente declaramos un condicional if a modo de excepción, que se encargará de distinguir si o ‘n’ é menor ou igual a 0. Si este é o caso a función devolverá unha cadea de texto no que se indica que ‘n’ debe ser maior que 0 para que a sucesión fibonacci devolva máis de 0 elementos, en caso contrario comezará a executarse un bucle tantas veces como ‘n’ (mentres ‘count’ sexa menor a ‘n’) realizando as seguintes acción en cada iteración; a variable ‘a’ gardarase dentro da lista ‘fibonacci’, unha variabe ‘result’ gardará o resultado da suma das variables ‘a’ + ‘b’ de forma que na primeira iteración será igual a 1, a variable ‘a’ muta o seu valor ao valor de ‘b’, b muta o seu valor ao valor de result de forma que a vaia gardando e añadiendo á lista os valores da suma dos dous anteriores dende 0 ata o elemento dado e para rematar aumentamoso valor de ‘count’ en 1 e fora do bucle, unha vez terminou todas as iteracións devolvemos a lista ‘fibonacci’ no que se foron gardando todos os valores de ‘a’ completando a sucesión.

```

if(n<=0):
    return("Ingrese un numero mayor a 0")
else:
    #Mientras a conta sea inferior ao número introducido volvera a executarse a suma máis a reasignación das variables 'a=b' e 'b=result(a+b)'.
    while count<n:
        fibonacci.append(a)
        result=a+b
        a=b
        b=result
        #Incrementamos a conta en 1.
        count+=1
    return(fibonacci)

```

Así quedaría o script completo:

```

def Fibonacci(n):
    #Inicializamos a variable 'count' para contar as veces que se rep
    count = 0
    #Se inicializan as variables 'a' e 'b' cos 2 primeiros valores da
    a = 0
    b = 1
    #Inicializamos un array no que se gardará o resultado da sucesión
    fibonacci=[]
    #Si o número introducido é negativo non se pode realizar a sucesi
    if(n<=0):
        return("Ingrese un numero mayor a 0")
    else:
        #Mientras a conta sea inferior ao número introducido volvera a exe
        while count<n:
            fibonacci.append(a)
            result=a+b
            a=b
            b=result
            #Incrementamos a conta en 1.
            count+=1
        return(fibonacci)

num=int(input("Introduce un numero: "))
print(Fibonacci(num))

```

## 2. Test unitario de la función Fibonacci

Comenzamos creando o script test.py e importando a biblioteca de 'unittest' que importa funciones e clases útiles para a realización de tests de código, tamén importamos o módulo a función 'Fibonacci' desde o módulo creado anteriormente 'fibonacci'.

```
import unittest
from fibonacci import Fibonacci
```

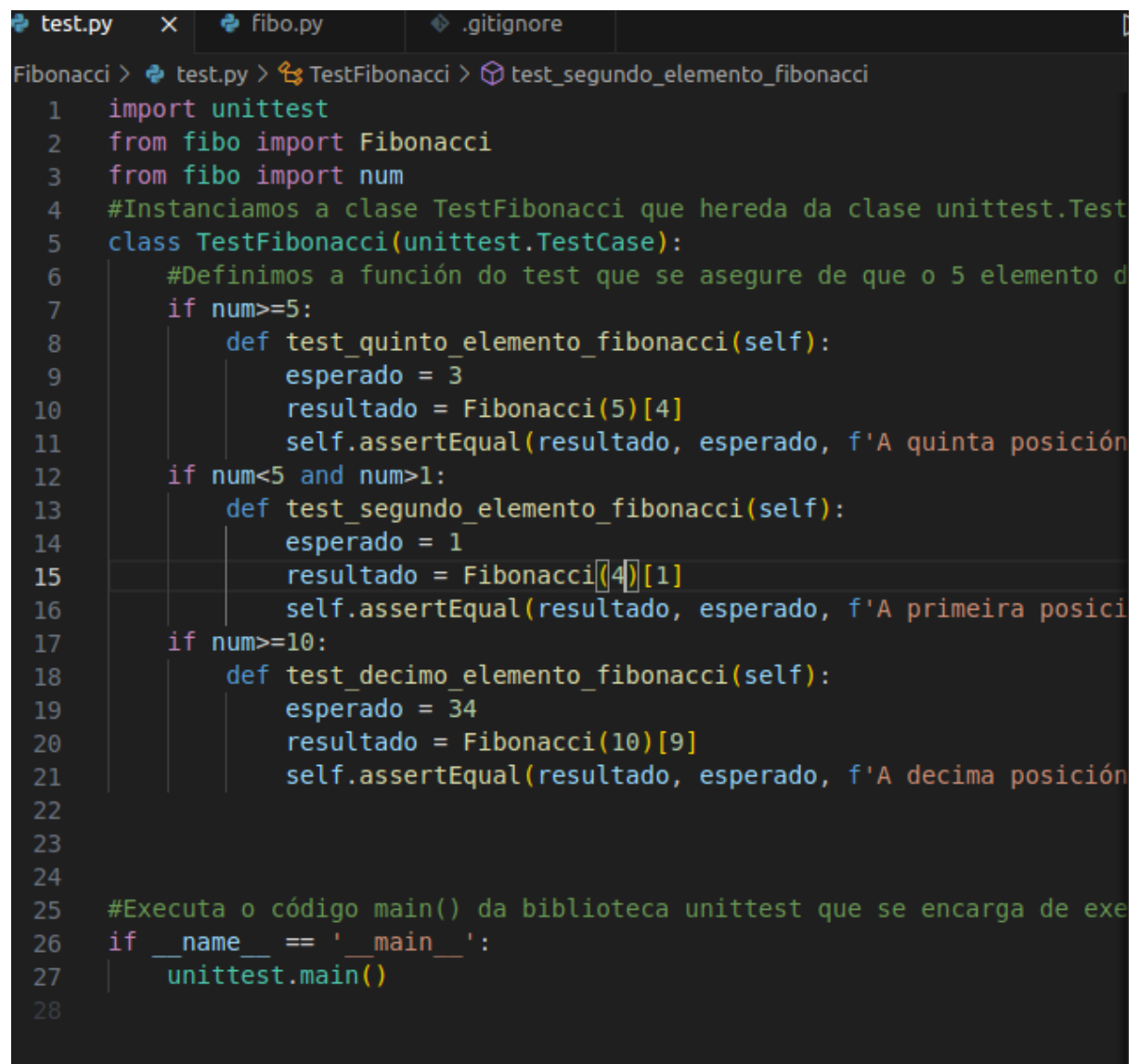
Continuamos creando unha clase cun nome descriptivo 'TestFibonacci' que herda da clase 'TestCase' da biblioteca 'unittest'. Dentro de esta clase declaramos unha función que debe comezar pola palabra 'test'. y se le pasa como parámetro la palabra reservada 'self' que llama a una instancia de la clase en la que se definió, en este caso TestFibonacci. Para asegurarnos de que o valor esperado (3) é igual a recibido utilizaremos a función assertEquals da clase TestFibonacci que herda da clase TestCase. Neste caso o comprobaremos que o quinto elemento (Fibonacci(5)[4]) é igual a 3:

```
import unittest
from fibonacci import Fibonacci
#Instanciamos a clase TestFibonacci que herda da clase unittest.TestCase
class TestFibonacci(unittest.TestCase):
    #Definimos a función do test que se asegure de que o 5 elemento da sucesión é igual a 3, debe iniciar por test
    def test_quinto_elemento_fibonacci(self):
        esperado = 3
        resultado = Fibonacci(5)[4]
        self.assertEqual(resultado, esperado, 'A quinta posición da sucesión debería ser {esperado}, pero recibíuse: {resultado}')

#Executa o código main() da biblioteca unittest que se encarga de executar os test que se definiron anteriormente si o modulo se ex
if __name__ == '__main__':
    unittest.main()
```

O mensaxe que lle pasamos como 3º parámetro á clase assertEquals trátase dun mensaxe de error que se mostrará se o valor recibido non coincide co esperado.

Seguindo o mesmo principio añadense dous tests máis que se executarán segundo o valor introducido polo usuario no script fibo.py que se importa coa línea de código `from fibo import num`.



```

Fibonacci > test.py > TestFibonacci > test_segundo_elemento_fibonacci
1  import unittest
2  from fibo import Fibonacci
3  from fibo import num
4  #Instanciamos a clase TestFibonacci que hereda da clase unittest.TestCase
5  class TestFibonacci(unittest.TestCase):
6      #Definimos a función do test que se asegure de que o 5 elemento d
7      if num>=5:
8          def test_quinto_elemento_fibonacci(self):
9              esperado = 3
10             resultado = Fibonacci(5)[4]
11             self.assertEqual(resultado, esperado, f'A quinta posición
12         if num<5 and num>1:
13             def test_segundo_elemento_fibonacci(self):
14                 esperado = 1
15                 resultado = Fibonacci(4)[1]
16                 self.assertEqual(resultado, esperado, f'A primeira posici
17         if num>=10:
18             def test_decimo_elemento_fibonacci(self):
19                 esperado = 34
20                 resultado = Fibonacci(10)[9]
21                 self.assertEqual(resultado, esperado, f'A decima posición
22
23
24
25  #Executa o código main() da biblioteca unittest que se encarga de exe
26  if __name__ == '__main__':
27      unittest.main()
28

```