

PRÁCTICA 01

Posta en Producción Segura

Secuencia de Fibonacci

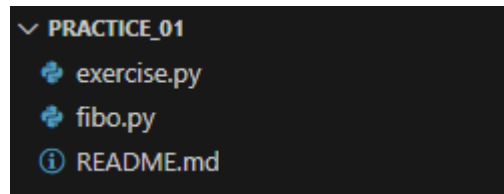
ÍNDICE

Posta en Producción Segura.....	1
Posta en marcha do proxecto.....	3
Pasos previos a creación do programa.....	3
Creación do programa.....	5
fibonacci.py.....	5
exercise.py.....	6
Proba do programa.....	9
Posta a punto da entrega.....	9
BIBLIOGRAFÍA.....	10

Posta en marcha do proxecto

Para comezar co programa o primeiro é definir en qué vai consistir. Saber se se vai usar un solo arquivo .py ou en cambio vaise facer un arquivo .py que execute un script e outro arquivo (main) que execute ese script importando.

Neste caso decidiuse que a mellor solución sería crear o script que crea a cadea de Fibonacci nun arquivo .py chamado *fibonacci.py* e logo chamalo cando fose necesario noutro arquivo .py chamado *exercise.py*.



Imaxe 1: Mostra dos arquivos do exercicio.

Para entrar en contacto co que debe facerse no programa en sí é importante leer varias veces o exercicio proposto e revisar a documentación de Python.

Tras elo, o mellor é elaborar un pseudocódigo no cal se realicen todas as funcións esixidas polo exercicio. A continuación se presenta o pseudocódigo de *fibo.py* e *exercise.py* (main):

1. Definición da función co parámetro da posición
2. Declaración de variables lista, "a" e "b"
3. Bucle for co parámetro da posición
4. Vaise engadindo a variable "a" a lista
5. Se incrementa a variable "a" en función de b (é o algoritmo de Fibonacci)
6. Devólvese a lista

Imaxe 2: Pseudocódigo de *fibo.py*.

1. Importase a librería unittest e o script fibo.py (co algoritmo de Fibonacci)
2. Declaración da clase Test (co uso da subclase TestCase de unittest)
3. Se define a función collendo self (obligatorio) como parametro
4. Dentro da función se recolle nunha lista local a lista da función de fibo pasando como parámetro 5
5. Se comproba (pídeo o exercicio) se o resultado do 5 número da lista equivale a 3
6. Logo se imprime o resultado (execútase se a comprobación foi válida)
7. Finalmente se usa `__name__` para indicar que *exercise.py* é o arquivo principal na execución

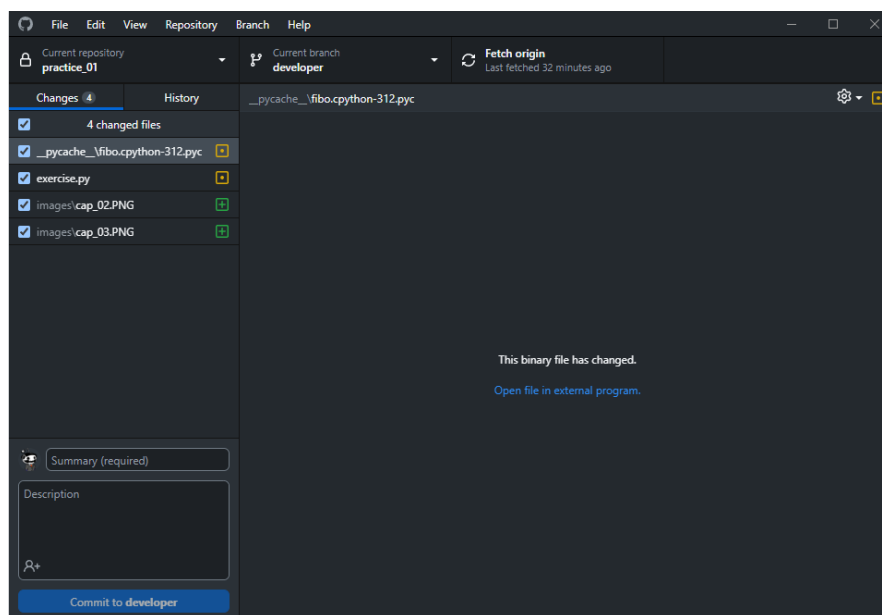
Imaxe 3: Pseudocódigo de *exercise.py*.

Pasos previos a creación do programa

Para a posta en produción do programa é preciso saber as ferramentas de software necesarias. Como se vai a usar *Windows 10* para a realización da práctica, a continuación se enumera todo o necesario para a posta en marcha do proxecto:

1. IDE *Visual Studio Code* (VSC).
2. *Python* (v3.12.0)
3. Extensión Python instalada en VSC.
4. *GitHub Desktop*.

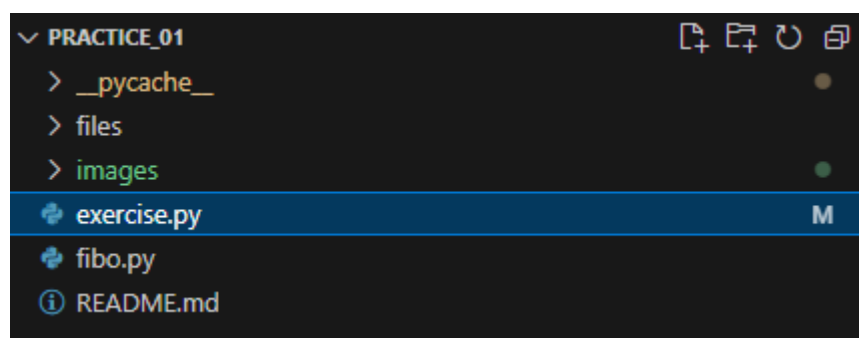
Tras ter todas esas ferramentas instaladas e correctamente configuradas, queda a parte mis “tediosa”. Preparar un entorno de desarrollo conectado mediante git a un repositorio de GitHub. Neste caso, ao estar no OS *Windows 10*, *GitHub* proporciona aos usuarios unha ferramenta moi potente: *GitHub Desktop*. Este programa permite, mediante unha interfaz gráfica, configurar unha carpeta para, mediante clone, conectala a un repositorio en *GitHub*. No presente caso, o repositorio chamarase **practice_01** e inicialmente contará cun **README.md** sen contido. Tamén é recomendado ter dúas ramas creadas no repositorio de GitHub, unha de desenvolvemento e outra a main. Sempre que se fagan commits, ata o último commit, se farán sempre a rama desenvolvemento.



Imaxe 5: Interfaz de *GitHub Desktop* durante o proxecto.

Tras ter todos estos pasos configurados se procede co paso inicial antes de comezar a programar: a creación dos elementos do proxecto.

Se creará un carpeta “files” para o pdf deste documento; e outra, de nome “images”, para as imaxes usadas neste documento. Agora solo queda crear os arquivos .py de “exercise” e “fibonacci”. Ademais estará o **README.md**, clonado anteriormente.



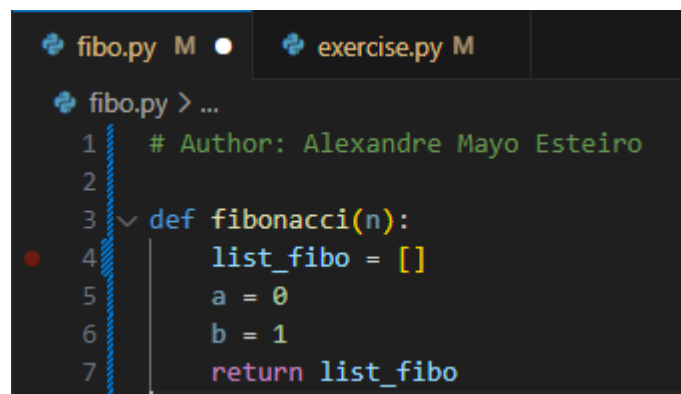
Imaxe 6: Todos os elementos dentro de Visual Studio Code

Creación do programa

fibonacci.py

Agora pódese comezar co código. Primeiro de todo comezo co script **fibonacci.py**. A continuación se amosan os pasos que se seguían na creación deste script:

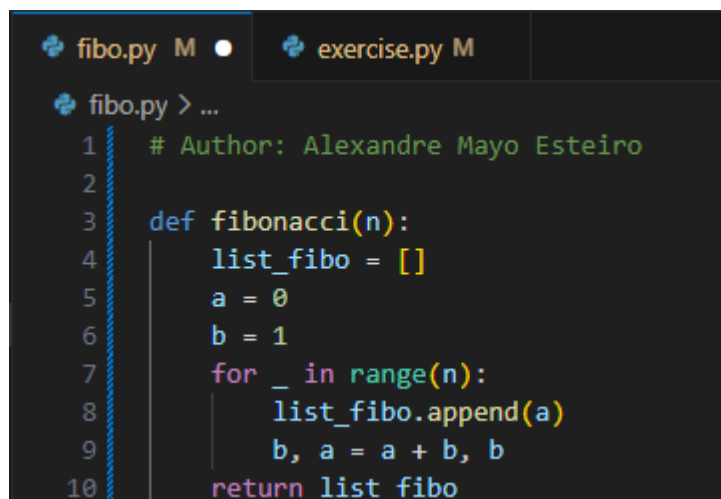
1. Primeiro se declara a función, as variables necesarias e o que retornará a función. Neste paso se crea a función cun parámetro *n* que será a lonxitude da cadea de *Fibonacci* que se creará nesta función. As variables serán a lista na que se almacenará a cadea e dúas variables tipo Integer (número enteiro). As variables Integer serán necesarias para xerar cada número da cadea.



```
fibonacci.py M • exercise.py M
fibonacci.py > ...
1 # Author: Alexandre Mayo Esteiro
2
3 def fibonacci(n):
4     list_fibo = []
5     a = 0
6     b = 1
7     return list_fibo
```

Código 1: Primeiro paso na creación do script fibonacci.py

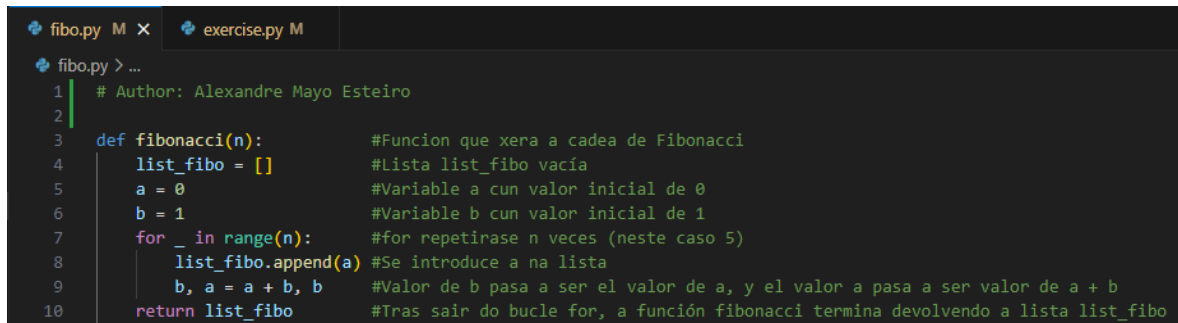
2. Logo créase o bucle no cal se recollerán os valores da cadea. Será un bucle for que se vai repetir *n* veces. Primeiro, a lista recollerá o valor de *a* (nun inicio será 0); e logo, se procederá a “avanzar” na cadea intercambiando o valor de *a* con *b*. Python permite isto sen necesidade dunha terceira variable (que sería temporal).



```
fibonacci.py M • exercise.py M
fibonacci.py > ...
1 # Author: Alexandre Mayo Esteiro
2
3 def fibonacci(n):
4     list_fibo = []
5     a = 0
6     b = 1
7     for _ in range(n):
8         list_fibo.append(a)
9         b, a = a + b, b
10    return list_fibo
```

Código 2: Segundo paso na creación do script fibonacci.py

3. Último paso. Se documenta o código o máis sinxelo posible para que o programador/revisor que lea o script poda comprender facilmente o como funciona.



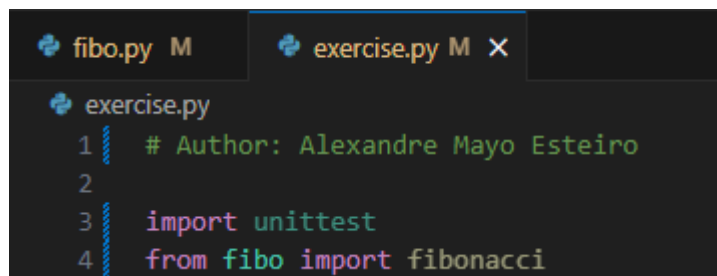
```
1 # Author: Alexandre Mayo Esteiro
2
3 def fibonacci(n):          #Funcion que xera a cadea de Fibonacci
4     list_fibo = []        #Lista list_fibo vacía
5     a = 0                 #Variable a cun valor inicial de 0
6     b = 1                 #Variable b cun valor inicial de 1
7     for _ in range(n):    #for repetirase n veces (neste caso 5)
8         list_fibo.append(a) #Se introduce a na lista
9         b, a = a + b, b    #Valor de b pasa a ser el valor de a, y el valor a pasa a ser valor de a + b
10    return list_fibo       #Tras sair do bucle for, a función fibonacci termina devolvendo a lista list_fibo
```

Código 3: Último paso na creación do script fibo.py

exercise.py

Agora solo queda facer o programa principal, **exercise.py**. A continuación se amosan os pasos na súa creación:

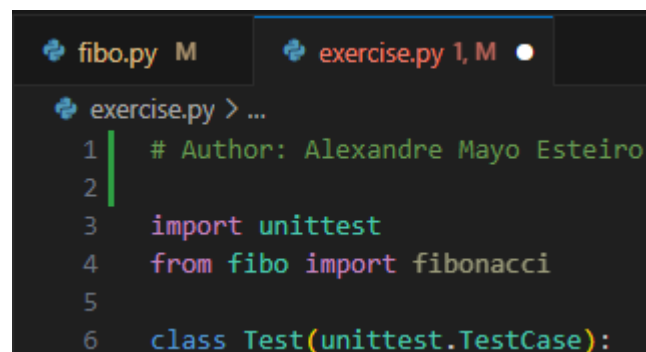
- Primeiro de todo, se importa a librería unittest e a función fibonacci dende o programa **fibo.py**.



```
1 # Author: Alexandre Mayo Esteiro
2
3 import unittest
4 from fibo import fibonacci
```

Código 4: Primeiro paso na creación do programa principal exercise.py

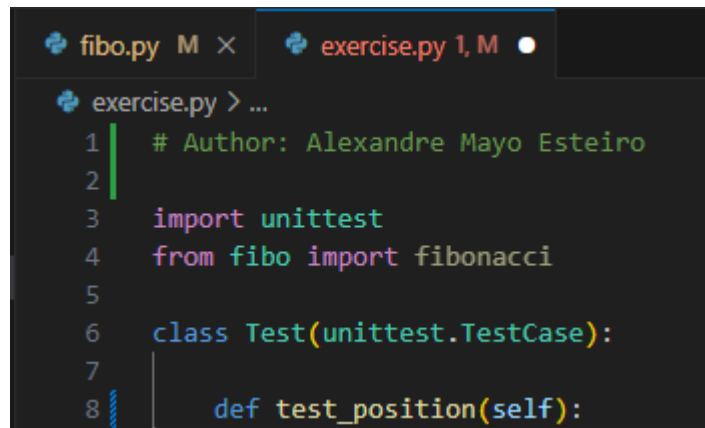
- A continuación, crease unha clase chamada **Test** que herda de **unittest.TestCase**. Esto permite definir métodos de proba (que é o que se esixe no exercicio).



```
1 # Author: Alexandre Mayo Esteiro
2
3 import unittest
4 from fibo import fibonacci
5
6 class Test(unittest.TestCase):
```

Código 5: Segundo paso na creación do programa principal exercise.py

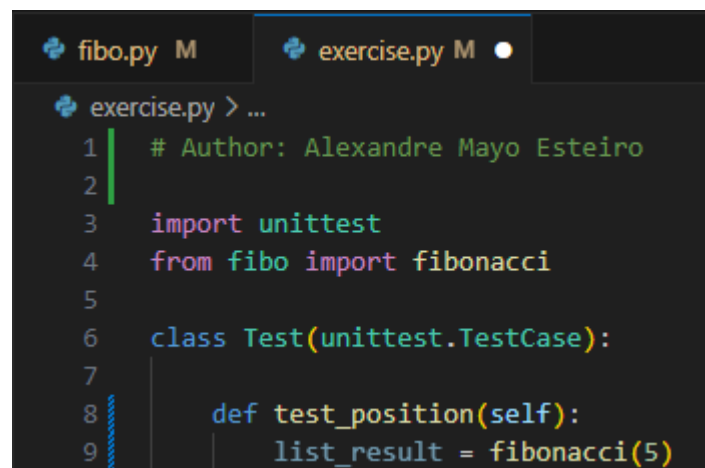
- Logo defínese o método **test_position** dentro da clase **Test**. Os nomes de métodos que comezan con "test" son identificados automaticamente como probas unitarias pola librería unittest.



```
fibonacci.py M x exercise.py 1, M ●
exercise.py > ...
1 | # Author: Alexandre Mayo Esteiro
2 |
3 | import unittest
4 | from fibo import fibonacci
5 |
6 | class Test(unittest.TestCase):
7 |
8 |     def test_position(self):
```

Código 6: Terceiro paso na creación do programa principal exercise.py

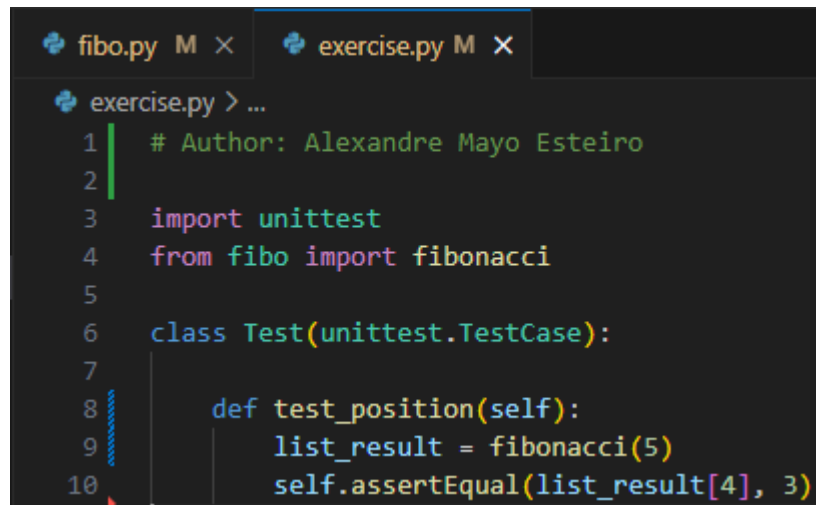
- A continuación invócase a función **fibonacci** cun argumento de 5. Presumiblemente, isto xerará a secuencia de números de Fibonacci 5 veces e devolve unha lista con esos números.



```
fibonacci.py M exercise.py M ●
exercise.py > ...
1 | # Author: Alexandre Mayo Esteiro
2 |
3 | import unittest
4 | from fibo import fibonacci
5 |
6 | class Test(unittest.TestCase):
7 |
8 |     def test_position(self):
9 |         list_result = fibonacci(5)
```

Código 7: Cuarto paso na creación do programa principal exercise.py

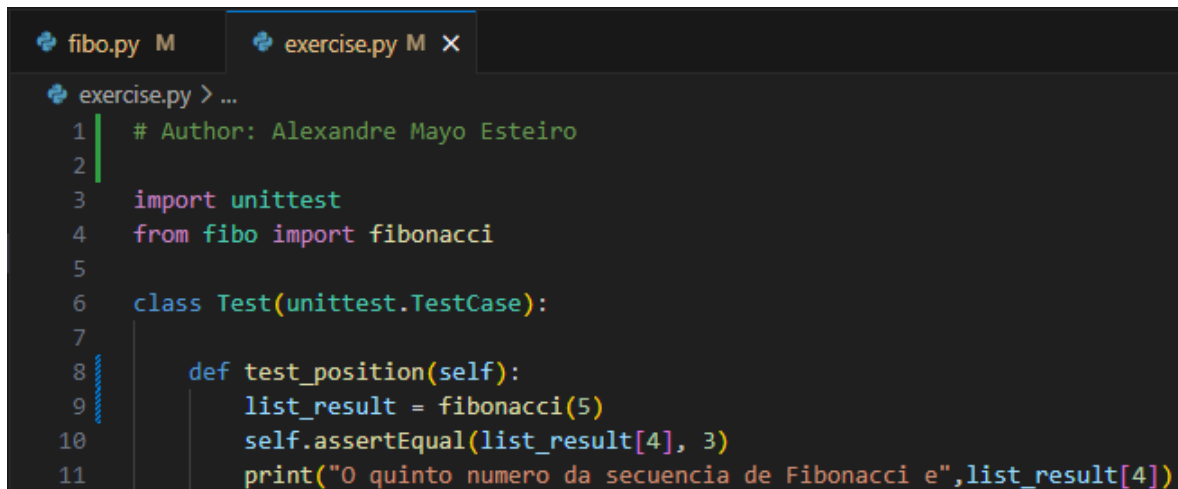
- Logo usa **self.assertEqual** para verificar que o quinto elemento (**list_result[4]**) da secuencia de Fibonacci xerada é igual a 3. É dicir, asegura que o quinto número da secuencia sexa 3. Isto chámase **proba de igualdade específica**.



```
exercise.py > ...
1 | # Author: Alexandre Mayo Esteiro
2 |
3 | import unittest
4 | from fibo import fibonacci
5 |
6 | class Test(unittest.TestCase):
7 |
8 |     def test_position(self):
9 |         list_result = fibonacci(5)
10 |         self.assertEqual(list_result[4], 3)
```

Código 8: Quinto paso na creación do programa principal exercise.py

- Case por terminar, antes de pechar a función se imprime o resultado (esto solo correrá se **self.assertEqual** non da erro).



```
exercise.py > ...
1 | # Author: Alexandre Mayo Esteiro
2 |
3 | import unittest
4 | from fibo import fibonacci
5 |
6 | class Test(unittest.TestCase):
7 |
8 |     def test_position(self):
9 |         list_result = fibonacci(5)
10 |         self.assertEqual(list_result[4], 3)
11 |         print("O quinto numero da secuencia de Fibonacci e",list_result[4])
```

Código 9: Sexto paso na creación do programa principal exercise.py

- Por último, se coloca **unittest.main()** no código, xa que é unha función que executa o conxunto de probas definido. Se este script (**exercise.py**) é o principal que se está executando, **unittest.main()** activará a execución das pruebas definidas na clase **Test**. Ademais se documenta todo o código como se fixo co script **fibo.py**.


```

1 # Author: Alexandre Mayo Esteiro
2
3 import unittest                                #Importase a biblioteca unittest para poder usar a subclase TestCase
4 from fibo import fibonacci                      #Importase o script fibo no cal se xera a cadea de Fibonacci
5
6 class Test(unittest.TestCase):                  #Clase Test que hereda de unittest.TestCase
7
8     def test_position(self):                    #Método de proba tomando self como parámetro. Vaise comprobar cal e o quinto numero da secuencia de Fibonacci
9         list_result = fibonacci(5)              #Chamada ao metodo fibonacci co argumento 5 e gardalo na lista list_result
10        self.assertEqual(list_result[4], 3)      #Comprobacion de que o quinto numero da secuencia de Fibonacci e 3
11        print("O quinto numero da secuencia de Fibonacci e", list_result[4]) #Imprimimos por pantalla o resultado do programa
12
13 if __name__ == '__main__':                      #Estas duas lineas verifican que o arquivo actual e o principal na execución
14     unittest.main()

```

Código 10: Último paso na creación do programa principal exercise.py

Proba do programa

Tras terminar con todo o código, compróbase que efectivamente funciona.

```

PS C:\> python.exe c:/
O quinto numero da secuencia de Fibonacci e 3
-----
Ran 1 test in 0.000s
OK

```

Imaxe 7: Impresión por terminal de VSC do programa executado

Posta a punto da entrega

Ao terminar co código só queda subilo ao git e formatear o README para que calquera que entre ao repositorio comprenda en qué consiste o contigo do mesmo.

BIBLIOGRAFÍA

Carremans, B. (2019, 13 junio). *What's in a (Python's) __name__?* freeCodeCamp.org.

<https://www.freecodecamp.org/news/whats-in-a-python-s-name-506262fe61e8/>

3.12.0 documentation. (s. f.).

<https://docs.python.org/3.12/>

UnitTest — Unit Testing framework. (s. f.). Python documentation.

<https://docs.python.org/3/library/unittest.html>

