Funciones

Funciones

- En Python una función es un bloque de código separado del bloque principal, el cual puede ser invocado en cualquier momento desde ésta u otra función.
- Cuando una función es invocada (llamada) el flujo del programa se redirige hacia el código de la función.
- Las funciones intercambian información con el programa principal con mediante argumentos y retorno.
- Hasta ahora solo hemos usado funciones. Ahora vamos a aprender a definirlas.

Definición de funciones

- La sintaxis básica para definir funciones en Python es:

```
def nombre_funcion([arg1, …, argN]):
    # Bloque de código
    [return out]
```

Ej:

```
def suma(a,b):
    return a + b
```

Definición de funciones

- Las funciones pueden o no aceptar argumentos y retornar valores. Ejemplos:

```
def foo1():
    print("Test")

def foo2():
    return True

def foo3(a, b):
    print(a*b)

def foo4(a,b):
    print(a+b)
    return True
```

- Una función sin retorno devuelve None.

Definición de funciones

- Los argumentos de las funciones pueden tener valores por defecto:

```
def foo(a, b=2):
    return a * b

print(foo(3))

print(foo(3,3))
```

- Podemos obviar el orden de los argumentos si los nombramos en la llamada.

```
print(foo(b=3,a=4))
```

Tipado estático de funciones

- Como mencionamos en la sesión anterior Python es un lenguaje de tipado dinámico. Por muy cómodo que resulte a la hora de trabajar esto puede ser un problema pues nos podemos encontrar con algo como esto:

```
def plus(a, b):
    return a + b
(…)
var = plus(“Hola”, 4)
```

Esto irremediablemente dará error.

Tipado estático de funciones

- ¿Quiere decir esto que la función está mal? No, simplemente que al no tener información de lo que hace la función no hemos tenido en cuenta el tipo de los argumentos de entrada.
- Una forma de solventar este problema es indicar qué tipos se espera de entrada y salida en una función. Esto se puede hacer desde Python 3.6:

```
def plus(a:int, b:int) → int:
    return a + b
```

Comentar funciones

- Podemos poner comentarios sobre nuestras funciones con un bloque de texto tras el encabezado de la función.

  Ej:
```
def cuadrado(a:int) → int:
    ‘‘‘
    Devuelve el cuadrado de un número entero.
    a -- número de entrada
    ’’’
    return a*a
```

- Existen una norma para la documentación de funciones (PEP257) recogida en la documentación.

Recursividad

- Python es un lenguaje que permite recursividad, es decir, una función puede llamarse a sí misma.
- Esto puede ser útil a veces para mejorar el rendimiento del programa. Pero cuidado, si el usar recursividad aumenta demasiado la complejidad tal vez deberíamos optar por una solución más legible.
- Ej:

```
def factorial(n: int) → int:
    if n <= 1:
        return n
    else:
        return n * factorial(n-1)
```

Ámbito de las variables

- Como podemos observar, las variables declaradas dentro de las funciones (incluyendo los argumentos) no se encuentran fuera del bloque del código de la función.
- Esto se debe a que son variables de ámbito local.
- Con ámbito de una variable nos referimos a las partes del programa donde existen.

Ámbito global

- Las variables de ámbito global son aquellas que se definen en el cuerpo principal del programa.
- Estas variables son accesibles desde todo el programa.
- Para poder cambiar el valor de una variable global en una función (y que no la confunda con la definición de una variable local) usamos la palabra reservada *global*.
- Ej:

```
def plusplus():
    global n
    n += 1
```

Ámbito local

- Son variables de ámbito local todas aquellas que están definidas dentro de definiciones de funciones o clases (ya las veremos).
- Las variables locales son accesibles solo en el ámbito donde han sido creadas.
- Si definimos una función dentro de una función, para acceder a las variables locales de la primera función usamos la palabra reservada nonlocal.

Ej:
```
def foo():
    a = 0
    def siguiente():
        nonlocal a
        a += 1
```

Variables estáticas

- Es posible crear variables que mantengan el valor entre llamadas. Llamamos a estas variables ‘atributos’ de la función.
- Hay dos maneras de definir esto:

Forma basic:

```
def siquiente():
    siguiente.var += 1
    return siguiente.var

siguiente.var = 0
```

Forma Pro:

```
def siguiente():
    if not hasattr(siguiente, ‘var’):
        siguiente.var = 0
    siguiente.var += 1
    return siguiente.var
```

Funciones como variables

- A estas alturas no nos sorprenderá saber que en Python las funciones son un tipo de variable más. Se pueden reasignar, copiar e incluso iterar.

Ej:

```
imprimir = print
imprimir('Hola Mundo!')
def suma(a,b):
  return a + b
def resta(a,b):
  return a - b
for f in [suma, resta]:
  print(f(1,1))
```

Funciones y estilo

- Usualmente un programa de Python tiene la siguiente estructura:
  - Importación de módulos
  - Definición de constantes, clases y funciones.
  - Flujo principal de información.

  En este esquema la mayoría del código estará en las funciones y usaremos el ámbito global solo para el intercambio de información entre funciones.

Funciones y estilo

- Ej:

```
def entrada_datos() → str:
    # entrada de datos
def tratamiento(datos: str) → str:
    # hacemos cosas
def salida_formato(datos: str) → None:
    # mostramos por pantalla los datos
datos = entrada_datos()
out = tratamiento(datos)
salida_formato(out)
```

Ejercicios

- Realiza una función que calcule los primeros N decimales de PI usando la Serie de Leibniz.
- Haz una función para extraer información que siga el siguiendo el siguiente formato:

  “var1:dato-var2:dato-(...)-varN:dato”

  Debe ser capaz mínimo de imprimir por pantalla cualquier ‘var’ que le indiquemos.