Funciones en Python

Definición de funciones

def nombre_función**(**parámetros**):**
    acciones
    return valor

parámetros: identificador1, ..., identificadorN

¡No se puede dar el mismo nombre a una variable y una función!

Ejemplo

```
# Calcula el sumatorio entre a y b (a <= b)
def sumatorio(a, b): # Cabecera (con los parámetros formales)
    suma = 0
    for número in range(a, b+1):
        suma += número
    return suma # Valor de retorno
```

Llamada a funciones

nombre_función(argumentos)

argumentos: expresión1, expresión2, ..., expresiónN

Ejemplo

```
n1 = int(input('Dame un número: '))
n2 = int(input('Dame otro número: '))
suma = sumatorio(n1, n2)
print(suma)
print(sumatorio(100, 50*3))
```

Llamada a funciones

Ejemplo: Código estructurado en funciones + programa principal

```
# --- Funciones ---
# Calcula el factorial de n (n >= 0)
def factorial(n):
    productorio = 1
    for i in range(2, n+1):
        productorio *= i
    return productorio

# --- Programa principal ---
print('Dame números (<0 para acabar):')
número = int(input('¿Número? '))
while número >= 0:
    print('{0}! = {1}'.format(número, factorial(número)))
    número = int(input('¿Número? '))
...
print(factorial(factorial(3)))
...
```

Funciones sin parámetros

Ejemplo: Menú de opciones

```
def menú():
    opción = -1
    while opción < 1 or opción > 3:
        print('1. Suma de vectores')
        print('2. Resta de vectores')
        print('3. Producto de vectores')
        opción = int(input('Elige opción: '))
    return opción
```

Procedimientos

Ejemplo: Mostrar la tabla de multiplicar del número dado

```
def muestra_tabla(n):
    print('Tabla del {0}:'.format(n))
    for i in range(1, 11):
        print('{0} x {1} = {2}'.format(n, i, n * i))
```

Ejemplo: Invertir la lista dada

```
def invierte_lista(lista):
    for índice in range(len(lista) // 2):
        aux = lista[índice]
        lista[índice] = lista[len(lista) - 1 - índice]
        lista[len(lista) - 1 - índice] = aux
```

En realidad devuelven None (ausencia de valor)

Procedimientos

Los procedimientos admiten el uso de **return** sin valor de retorno

Ejemplo: Mostrar la nota de un alumno

```
def muestra_nota_de_alumno(alumnos, notas, alumno_buscado):
    for i in range(len(alumnos)):
        if alumnos[i] == alumno_buscado:
            print(alumno_buscado, notas[i])
            return
    print('El alumno no pertenece al grupo.')
```

Funciones que devuelven varios valores

Ejemplo: Máximo y mínimo de una lista (no vacía) de números

```
def máximo_y_mínimo(lista):
    máximo = lista[0]
    mínimo = lista[0]
    for elemento in lista:
        if elemento > máximo:
            máximo = elemento
        elif elemento < mínimo:
            mínimo = elemento
    return [máximo, mínimo]
```

```
[maxi, mini] = máximo_y_mínimo([6, 3, 7, 2])
```

Observa que parámetros y/o valor de retorno pueden ser secuencias

¿Parámetros&valor de retorno o teclado&pantalla?

Criterios

- Si nos piden una función que reciba uno o más datos, debemos ponerlos como parámetros formales y no leerlos de teclado
- Si nos piden una función que devuelva uno o más datos, debemos hacerlo con un **return** y no imprimirlos por pantalla
- Sólo si nos dicen que la función lea de teclado y/o imprima por pantalla usaremos **input** & **print**

¿Parámetros & valor de retorno o teclado & pantalla?

Ejemplo: Dado un número determinar si es par, versión incorrecta

```
def es_par():
    n = int(input('Dame un número: '))
    if n%2 == 0:
        print('El número ES par.')
    else:
        print('El número NO ES par.')
```

Ejemplo: Dado un número determinar si es par, versión correcta

```
def es_par(n):
    return n%2 == 0:
```

Variables locales y globales

Ejemplo

```
# Calcula el factorial de n (n >= 0)
def factorial(n):
    productorio = 1
    for i in range(2, n+1):
        productorio *= i
    return productorio

# -- Programa principal --
número = 5
print(factorial(número))
print(productorio)  # ERROR
```

productorio, i y n son variables locales a factorial
número es una variable global

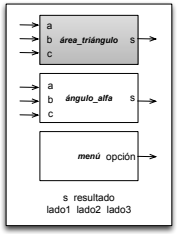
Variables locales y globales

Ejemplo: area_y_angulo.py

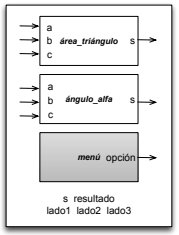
```
1  def área_triángulo(a, b, c):
2      s = (a + b + c) / 2
3      return sqrt(s * (s-a) * (s-b) * (s-c))
4  def ángulo_alfa(a, b, c):
5      s = área_triángulo(a, b, c)
6      return 180 / pi * asin(2 * s / (b*c))
7  def menú():
8      opción = 0
9      while opción != 1 and opción != 2:
10         print('1) Calcular área del triángulo')
11         print('2) Calcular ángulo opuesto al primer lado')
12         opción = int(input('Escoge opción: '))
13     return opción
14
15 # ---- Programa principal ----
16 lado1 = float(input('Dame lado a: '))
17 lado2 = float(input('Dame lado b: '))
18 lado3 = float(input('Dame lado c: '))
19
20 s = menú()
21
22 if s == 1:
23     resultado = área_triángulo(lado1, lado2, lado3)
24 else:
25     resultado = ángulo_alfa(lado1, lado2, lado3)
26
27 print('Escogiste la opción', s)
28 print('El resultado es:', resultado)
```

Variables locales y globales

Ámbito de la variable local `s` y de los parámetros formales `a`, `b`, y `c` de la función *área_triángulo*

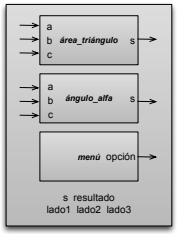


Ámbito de la variable local `opción` de la función *menú*

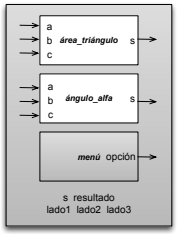


Variables locales y globales

Ámbito de las variables globales `resultado`, `lado1`, `lado2` y `lado3`



Ámbito de la variable global `s`



Observa que la variable global `s` queda "oculta" por las variables locales con el mismo nombre

Variables locales y globales

Tipos de ámbito y precedencia (de mayor a menor)

- **ámbito local**: todos los identificadores declarados en una función
- **ámbito global**: todos los identificadores declarados en un módulo/fichero y que no están dentro de funciones
- **ámbito predefinido**: todos los identificadores predefinidos Python que se pueden usar sin **import**

Criterios

- Utilizaremos parámetros/valores de retorno (**return**) para consultar/modificar variables globales
- No utilizaremos identificadores predefinidos Python como nombres para nuestras variables/funciones

Modificación de parámetros

Cuando en el cuerpo de una función se modifica el valor de un parámetro, ¿se modifica el argumento correspondiente?

- Si el parámetro es de tipo atómico o de tipo cadena → No
- Si el parámetro es de tipo lista → *Podría*, dependiendo de cómo escribamos el cuerpo de la función

Duplicar una lista con **append**

```
def duplica1(lista):
  for i in range(len(lista)):
    lista.append(lista[i])

l = [1, 2]
duplica1(l) # l es [1,2,1,2]
```

Duplicar una lista con +

```
def duplica2(lista):
  for i in range(len(lista)):
    lista = lista + [lista[i]]

l = [1, 2]
duplica2(l) # ¡No cambia!
```

Diseño de programas con funciones

Ejemplo: lee_positivos.py

```
1  a = int(input('Dame un número positivo: '))
2  while a<0:
3      print('Error, el número debe ser positivo.')
4      a = int(input('Dame un número positivo: '))
5
6  b = int(input('Dame un número positivo: '))
7  while b<0:
8      print('Error, el número debe ser positivo.')
9      b = int(input('Dame un número positivo: '))
10
11 c = int(input('Dame un número positivo: '))
12 while c<0:
13     print('Error, el número debe ser positivo.')
14     c = int(input('Dame un número positivo: '))
```

Diseño de programas con funciones

Ejemplo: lee_positivos_funcion.py

```
1  def lee_entero_positivo(texto):
2      n = int(input(texto))
3      while n<0:
4          print('Error, el número debe ser positivo.')
5          n = int(input(texto))
6      return n
7
8  a = lee_entero_positivo('Dame un número positivo: ')
9  b = lee_entero_positivo('Dame otro número positivo: ')
10 c = lee_entero_positivo('Dame otro número positivo: ')
```

Ventajas del uso de funciones

- Se escribe menos código (una función *vs.* código repetido)
- El código es más fácil de depurar & mantener
- El código es más legible

Diseño de programas con funciones

Reglas prácticas

Es aconsejable definir una función si tenemos:

- un fragmento de código que se utiliza más de una vez
- un fragmento de código que hace una cosa, o que lo que hace se puede describir con una sola frase
- una función que hace muchas cosas, o que lo que hace no se puede describir con una sola frase $\rightsquigarrow$ dividir la función en otras más pequeñas/simples

Módulos

Módulos

Colecciones de funciones relacionadas que se pueden utilizar desde nuestros programas

Algunos módulos Python

`calendar`	para trabajar con fechas
`math`	funciones matemáticas
`random`	para generar números aleatorios
`time`	para medir tiempos

¡El usuario puede crear y utilizar sus propios módulos!

Módulos

modulo.py

```
# Constantes
...

# Funciones

def sumatorio(a, b):
  ...

def factorial(n):
  ...

...

# Código que NO se ejecuta al importar
if __name__ == '__main__':
  ...
```

```
from modulo import factorial

from modulo import *
```

Ejercicios con listas

1. Escribe una función que, dada una lista, devuelva otra con los mismos elementos de la lista pero sin repetir ninguno.
2. Escribe una función que, dadas dos listas, devuelva otra con los elementos comunes a ambas, sin repetir ninguno (*intersección de conjuntos*).
3. Escribe una función que, dadas dos listas, devuelva otra con los elementos que pertenecen a una o a otra, pero sin repetir ninguno (*unión de conjuntos*).
4. Escribe una función que, dadas dos listas, devuelva otra con los elementos que pertenecen a la primera pero no a la segunda, sin repetir ninguno (*diferencia de conjuntos*).
5. Escribe una función que, dadas dos listas, intercambie su valor.

Ejercicios con cadenas y listas de cadenas

1. Escribe una función para determinar si un número es *apocalíptico*.
   Pista: Un número n es apocalíptico si 2^n contiene el número 666, p.e. 157 y 192 son apocalípticos.
2. Escribe una función para determinar si una cadena está bien formada como número binario.
3. Escribe una función que, dada una lista de cadenas, devuelva la más larga. Si dos o más cadenas miden lo mismo y son las más largas, la función deberá devolver cualquiera de ellas.
4. Escribe una función que, dada una lista de cadenas, devuelva una lista con las cadenas más largas. Si dos o más cadenas miden lo mismo y son las más largas, la lista deberá incluirlas a todas.
5. Escribe una función que, dada una lista de cadenas, devuelva el prefijo común más largo.
6. Escribe una función que, dada una lista de números y un parámetro numérico, devuelva otra lista con dos sublistas, la primera de ellas con los números menores que el número y la segunda con los números mayores que él.

Ejercicios con matrices

1. Supongamos que almacenamos en una matriz de dimensiones $m \times n$ las calificaciones de m estudiantes (a los que identificaremos por su número de lista) en los n ejercicios entregados (si el ejercicio no se ha entregado almacenaremos -1). Se piden funciones y procedimientos para:
   - 1.a Dado el número de un alumno, devolver el número de ejercicios entregados.
   - 1.b Dado el número de un alumno, devolver la media de la calificación de los ejercicios entregados si los entregó todos, y 0 si no.
   - 1.c Devolver el número de alumnos que han entregado todos los ejercicios.
   - 1.d Dado el número de un ejercicio, devolver el número de estudiantes que lo han entregado.
   - 1.e Dado el número de un ejercicio, devolver la calificación máxima de los estudiantes que lo han entregado. Si ningún estudiante lo ha entregado, devolver `None`.

Ejercicios con matrices

1. Escribe una función que reciba una matriz y devuelva una lista con dos enteros que serán, respectivamente, el número de filas y columnas de la matriz.
2. Escribe una función que reciba una matriz y determine si es cuadrada (es decir, con igual número de filas que de columnas).
3. Escribe una función que reciba una matriz y determine si es simétrica.
4. Escribe una función que reciba una matriz y, si es cuadrada, devuelva la suma de los elementos de la diagonal principal (los elementos $A_{i,i}$). Si la matriz no es cuadrada la función deberá devolver `None`.
5. Escribe una función que reciba dos matrices y, si tienen las mismas dimensiones, devuelva la matriz suma. Si las matrices no tienen las mismas dimensiones la función deberá devolver `None`.

Ejercicios con matrices (de exámenes) I

1. Se dispone de una matriz con una serie de resúmenes mensuales de los datos medidos en una estación meteorológica. Cada fila de la matriz almacena la información de un mes concreto y contiene: el año, el mes, la media de las temperaturas máximas de todos los días (en grados centígrados), la media de las temperaturas mínimas de todos los días (en grados centígrados), el número de días que ha helado, el total de precipitaciones (en mm) y el total de horas de sol. Un ejemplo de matriz podría ser:

```
[[1996, 1, 6.6, 3.1, 7, 35.5, 29.3],\
 [1996, 2, 6.2, -0.1, 17, 60.8, 103.4],\
 [1996, 3, 8.4, 1.8, 9, 27.1, 76.4],\
 [1996, 4, 13.8, 4.8, 5, 49.3, 148.3],\
 [1996, 5, 14.1, 5.6, 1, 35.7, 185.7],\
 [1996, 6, 21.2, 10.2, 0, 23.7, 290.7],\
 [1996, 7, 23.3, 12.7, 0, 25.5, 256.6],\
 [1996, 8, 22.4, 12.5, 0, 53.1, 211.6],\
 [1996, 9, 18.4, 10.3, 0, 16.1, 125.3],\
 [1996, 10, 15.8, 8.6, 0, 33.0, 130.8],\
 [1996, 11, 9.6, 2.9, 10, 81.6, 101.0],\
 [1996, 12, 5.5, 0.9, 13, 16.2, 55.5],\
 [1997, 1, 7.1, -0.2, 16, 11.9, 51.4],\
 [1997, 2, 10.4, 4.2, 3, 76.5, 64.2]]
```

Ejercicios con matrices (de exámenes) II

Se pide:

1.a Una función `media_histórica_datos_mes` que, dados una matriz como la que se describe arriba y un mes, devuelva una lista con el mes seguido de la media de los datos registrados ese mes a lo largo de los años. Por ejemplo, dados los datos de arriba y el mes `1` la función deberá devolver `[1, 6.85, 1.45, 11.5, 23.7, 40.35]`, y con los mismos datos y el mes `2` deberá devolver `[2, 8.3, 2.05, 10.0, 68.65, 83.8]`.

1.b Una función `año_más_caluroso_que_la_media` que, dada una matriz como la que se describe arriba y un año, devuelva un booleano que indique si todos los meses de ese año cumplen que la temperatura máxima registrada es mayor o igual que la temperatura máxima que devuelve la función `media_histórica_datos_mes` referida a ese mes. Si no hay datos registrados de ese año la función deberá devolver `None`. Por ejemplo, dados los datos de arriba y el año `1997` la función deberá devolver `True`.