Funciones

1. Funciones `def`

Una función es un bloque de código que tiene asociado un nombre, de manera que cada vez que se quiera ejecutar el bloque de código basta con invocar el nombre de la función.

Para declarar una función se utiliza la siguiente sintaxis:

```
    def <nombre_funcion> (<parámetros>):
       bloque código
         return <objeto>
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def bienvenida():
...     print('¡Bienvenido a Python!')
...     return
...
>>> type(bienvenida)
<class 'function'>
>>> bienvenida()
¡Bienvenido a Python!
```

1.1. Parámetros de una función

Una función puede recibir valores cuando se invoca a través de unas variables conocidas como *parámetros* que se definen entre paréntesis en la declaración de la función. En el cuerpo de la función se pueden usar estos parámetros como si fuesen variables.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def bienvenida(nombre):
...     print('¡Bienvenido a Python', nombre + '!')
...     return
...
>>> bienvenida('Mindhub')
¡Bienvenido a Python Mindhub!
```

1.2. Argumentos de la llamada a una función

Los valores que se pasan a la función en una llamada o invocación concreta de ella se conocen como argumentos y se asocian a los parámetros de la declaración de la función.

Los argumentos se pueden indicar de dos formas:

- **Argumentos posicionales:** Se asocian a los parámetros de la función en el mismo orden que aparecen en la definición de la función.
- **Argumentos por nombre:** Se indica explícitamente el nombre del parámetro al que se asocia un argumento de la forma `parametro = argumento`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def bienvenida(nombre, apellido):
...     print('¡Bienvenido a Python', nombre, apellido + '!')
...     return
...
>>> bienvenida('Patricia', 'Martucci')
¡Bienvenido a Python Patricia Martucci!
>>> bienvenida(apellido = 'Martucci', nombre = 'Patricia')
¡Bienvenido a Python Patricia Martucci!
```

1.3. Retorno de una función

Una función puede devolver un objeto de cualquier tipo tras su invocación. Para ello el objeto a devolver debe escribirse detrás de la palabra reservada **`return`**. Si no se indica ningún objeto, la función no devolverá nada.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def area_triangulo(base, altura):
...     return base * altura / 2
...
>>> area_triangulo(2, 3)
3
>>> area_triangulo(4, 5)
10
```

2. Argumentos por defecto

En la definición de una función se puede asignar a cada parámetro un argumento por defecto, de manera que si se invoca la función sin proporcionar ningún argumento para ese parámetro, se utiliza el argumento por defecto.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def bienvenida(nombre, lenguaje = 'Python'):
...     print('¡Bienvenido a', lenguaje, nombre + '!')
...     return
...
>>> bienvenida('Carla')
¡Bienvenido a Python Carla!
>>> bienvenida('Carla', 'Java')
¡Bienvenido a Java Carla!
```

3. Pasar un número indeterminado de argumentos

Por último, es posible pasar un número variable de argumentos a un parámetro. Esto se puede hacer de dos formas:

- **`*parametro`**: Se antepone un asterisco al nombre del parámetro y en la invocación de la función se pasa el número variable de argumentos separados por comas. Los argumentos se guardan en una lista que se asocia al parámetro.
- **`**parametro`**: Se anteponen dos asteriscos al nombre del parámetro y en la invocación de la función se pasa el número variable de argumentos por pares **`nombre = valor`**, separados por comas. Los argumentos se guardan en un diccionario que se asocia al parámetro.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def menu(*platos):
...     print('Hoy tenemos: ', end='')
...     for plato in platos:
...         print(plato, end=', ')
...     return
...
>>> menu('pasta', 'pizza', 'ensalada')
Hoy tenemos: pasta, pizza, ensalada,
```

---

4. Ámbito de los parámetros y variables de una función

Los parámetros y las variables declaradas dentro de una función son de **ámbito local**, mientras que las definidas fuera de ella son de **ámbito global**.

Tanto los parámetros como las variables del ámbito local de una función sólo están accesibles durante la ejecución de la función, es decir, cuando termina la ejecución de la función estas variables desaparecen y no son accesibles desde fuera de la función.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def bienvenida(nombre):
...     lenguaje = 'Python'
...     print('¡Bienvenido a', lenguaje, nombre + '!')
...     return
...
>>> bienvenida('Carla')
¡Bienvenido a Python Carla!
>>> lenguaje
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'lenguaje' is not defined
```

---

5. Ámbito de los parámetros y variables de una función

Si en el ámbito local de una función existe una variable que también existe en el ámbito global, durante la ejecución de la función la variable global queda eclipsada por la variable local y no es accesible hasta que finaliza la ejecución de la función.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> lenguaje = 'Java'
>>> def bienvenida():
...     lenguaje = 'Python'
...     print('¡Bienvenido a', lenguaje + '!')
...     return
...
>>> bienvenida()
¡Bienvenido a Python!
>>> print(lenguaje)
Java
```

6. Paso de argumentos por referencia

En Python el paso de argumentos a una función es siempre por referencia, es decir, se pasa una referencia al objeto del argumento, de manera que cualquier cambio que se haga dentro de la función mediante el parámetro asociado afectará al objeto original, siempre y cuando este sea mutable.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> primer_curso = ['Matemáticas', 'Física']
>>> def añade_asignatura(curso, asignatura):
...     curso.append(asignatura)
...     return
...
>>> añade_asignatura(primer_curso, 'Química')
>>> print(primer_curso)
['Matemáticas', 'Física', 'Química']
```

7. Documentación de funciones

Una práctica muy recomendable cuando se define una función es describir lo que la función hace en un comentario.

En Python esto se hace con un **docstring** que es un tipo de comentario especial se hace en la línea siguiente al encabezado de la función entre tres comillas simples `'''` o dobles `"""`.

Después se puede acceder a la documentación de la función con la función `help(<nombre-función>)`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def area_triangulo(base, altura):
... """Función que calcula el área de un triángulo.
...
... Parámetros:
...     - base: La base del triángulo.
...     - altura: La altura del triángulo.
... Resultado:
...     El área del triángulo con la base y altura especificadas.
... """
...     return base * altura / 2
...
```

```
>>> help(area_triangulo)
area_triangulo(base, altura)



    Función que calcula el área de un triángulo.

    Parámetros:
        - base: La base del triángulo.
        - altura: La altura del triángulo.
    Resultado:
        El área del triángulo con la base y altura especificadas.
```

8. Funciones recursivas

Una función recursiva es una función que en su cuerpo contiene una llama a si misma.

La recursión es una práctica común en la mayoría de los lenguajes de programación ya que permite resolver las tareas recursivas de manera más natural.

Para garantizar el final de una función recursiva, las sucesivas llamadas tienen que reducir el grado de complejidad del problema, hasta que este pueda resolverse directamente sin necesidad de volver a llamar a la función.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def factorial(n):
...     if n == 0:
...         return 1
...     else:
...         return n * factorial(n-1)
...
>>> f(5)
120
```

8.1. Funciones recursivas múltiples

Una función recursiva puede invocarse a si misma tantas veces como quiera en su cuerpo.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def fibonacci(n):
...     if n <= 1:
...         return n
...     else:
...         return fibonacci(n - 1) + fibonacci(n - 2)
...
>>> fibonacci(6)
8
```

8.2. Los riesgos de la recursión

Aunque la recursión permite resolver las tareas recursivas de forma más natural, hay que tener cuidado con ella porque suele consumir bastante memoria, ya que cada llamada a la función crea un nuevo ámbito local con las variables y los parámetros de la función.

En muchos casos es más eficiente resolver la tarea recursiva de forma iterativa usando bucles.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def fibonacci(n):
...     a, b = 0, 1
...     for i in range(n):
...         a, b = b, a + b
...     return a
...
>>> fibonacci(6)
8
```
