Funciones y módulos

M2: Fundamentos de programación en Java

AE1: Implementar una suite de pruebas unitarias en lenguaje Java utilizando JUnit para asegurar el buen funcionamiento de una pieza de software

alkemy

Introducción 👩‍💻

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, que se ha convertido en una herramienta esencial en diversos campos como el desarrollo web, análisis de datos, inteligencia artificial y más.

En el corazón de Python, y de la programación en general, se encuentran las funciones y los módulos. Las **funciones permiten encapsular código reutilizable y organizar tareas específicas, lo que facilita la escritura de código más limpio y modular. Los módulos, por otro lado, permiten organizar y reutilizar conjuntos de funciones y variables, facilitando la gestión y el mantenimiento de proyectos más grandes.**

Este manual está diseñado para guiarte a través de los conceptos fundamentales y avanzados de las funciones y módulos en Python. Comenzaremos con una introducción a las funciones, explorando su sintaxis, cómo definirlas y utilizarlas, y luego profundizaremos en aspectos más avanzados como funciones anónimas, recursivas y el manejo de argumentos variables. Posteriormente, abordaremos el concepto de módulos, cómo utilizarlos y crearlos, y la importancia de la biblioteca estándar de Python. Finalmente, discutiremos las buenas prácticas de documentación y las convenciones de estilo que te ayudarán a escribir código profesional y mantenible.

Aprendizajes esperados

Cuando finalices la lección serás capaz de:

- Comprender en profundidad qué es una función y su utilidad en Python.
- Definir y utilizar funciones básicas y avanzadas, incluyendo funciones anónimas y recursivas.
- Manejar diferentes tipos de parámetros y argumentos en funciones.
- Utilizar y crear módulos en Python para organizar y reutilizar código.
- Aprovechar la biblioteca estándar de Python y conocer cómo importar módulos.
- Aplicar buenas prácticas de documentación y seguir las convenciones de estilo PEP8.
- Escribir código más limpio, eficiente y mantenible mediante el uso de funciones y módulos.

Introducción a las funciones en Python

¿Qué es una función?

Una función es un bloque de código autónomo diseñado para realizar una tarea específica. Las funciones son fundamentales en la programación, ya que permiten:

- **Reutilización de código**: Evitan la repetición, permitiendo llamar al mismo bloque de código múltiples veces en diferentes lugares del programa.
- **Modularidad**: Facilitan la organización del código en partes lógicas, lo que mejora la legibilidad y el mantenimiento.
- **Abstracción**: Permiten ocultar la complejidad, exponiendo solo la interfaz necesaria para interactuar con ellas.

En Python, las funciones son ciudadanos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos y retornadas por otras funciones.

Ejemplo básico de una función:

```
def saludar():
    print("Hola, bienvenido al mundo de Python.")
```

Para llamar a esta función, simplemente escribes `saludar()`, y se ejecutará el código dentro de ella.

SINTAXIS BÁSICA

La definición de una función en Python sigue la siguiente estructura:

```
def nombre_de_la_función(parámetros):
    """Docstring opcional que describe la función."""
    # Cuerpo de la función
    instrucciones
    return valor_de_retorno
```

- **`def`**: Palabra clave que indica el inicio de una función.
- **`nombre_de_la_función`**: Identificador que se usará para llamar a la función.
- **`parámetros`**: Variables que se pasan a la función (pueden ser opcionales).
- **`Docstring`**: Cadena opcional que documenta lo que hace la función.
- **`instrucciones`**: Código que ejecuta la función.
- **`return`**: Sentencia opcional que devuelve un valor al lugar donde se llamó la función.

Ejemplo con parámetros y retorno:

```
def sumar(a, b):
    """Devuelve la suma de dos números."""
    resultado = a + b
    return resultado

suma = sumar(5, 3)
print(f"La suma es: {suma}")  # Salida: La suma es: 8
```

PARÁMETROS Y ARGUMENTOS

Los parámetros son variables que se definen en la declaración de la función, y los argumentos son los valores reales que se pasan a la función cuando se llama.

- **Parámetros posicionales**: La asignación de valores se hace en orden.
- **Parámetros nombrados (keywords)**: Se especifican con el nombre del parámetro.

Ejemplo de parámetros posicionales y nombrados:

```
def presentar(nombre, edad):
    print(f"Me llamo {nombre} y tengo {edad} años.")

presentar("Ana", 25)  # Uso de parámetros posicionales
presentar(edad=30, nombre="Juan")  # Uso de parámetros nombrados
```

RETORNO DE VALORES

La sentencia `return` finaliza la ejecución de la función y devuelve opcionalmente un valor al llamador. Si no se especifica, la función devuelve `None`.

Ejemplo:

```
def es_par(numero):
    """Devuelve True si el número es par, False si es impar."""
    return numero % 2 == 0

resultado = es_par(4)
print(resultado)  # Salida: True
```

FUNCIONES PRECONSTRUIDAS

Python proporciona una serie de funciones predefinidas que facilitan tareas comunes.

Tabla de algunas funciones preconstruidas:

Función	Descripción	Ejemplo	Salida
`print()`	Imprime en la consola.	`print("Hola")`	Hola
`len()`	Retorna la longitud de un objeto.	`len([1, 2, 3])`	3
`type()`	Devuelve el tipo del objeto.	`type(3.14)`	`<class 'float'>`
`int()`	Convierte un valor a entero.	`int("42")`	42
`input()`	Captura entrada del usuario.	`nombre = input("Nombre: ")`	(Espera entrada)

Estas funciones son muy útiles y es recomendable familiarizarse con ellas para facilitar el desarrollo.

FUNCIONES AVANZADAS

FUNCIONES CON ARGUMENTOS PREDETERMINADOS

Permiten asignar valores por defecto a los parámetros, haciéndolos opcionales al llamar la función.

Ejemplo:

```
def potencia(base, exponente=2):
    return base ** exponente

print(potencia(5))       # Usa el exponente por defecto (2), Salida: 25
print(potencia(5, 3))    # Usa el exponente proporcionado, Salida: 125
```

FUNCIONES CON ARGUMENTOS VARIABLES (*ARGS Y **KWARGS)

***args (Argumentos variables no nombrados):**

Permite pasar un número variable de argumentos posicionales a una función.

```
def sumar_todos(*numeros):
    return sum(numeros)

print(sumar_todos(1, 2, 3, 4))  # Salida: 10
```

****kwargs (Argumentos variables nombrados):**

Permite pasar un número variable de argumentos nombrados (diccionarios).

```
def imprimir_info(**datos):
    for clave, valor in datos.items():
        print(f"{clave}: {valor}")

imprimir_info(nombre="Laura", edad=28, ciudad="Madrid")
# Salida:
# nombre: Laura
# edad: 28
# ciudad: Madrid
```

FUNCIONES ANÓNIMAS (LAMBDA FUNCTIONS)

Son funciones pequeñas y anónimas definidas con la palabra clave `lambda`. Se utilizan para operaciones simples y son útiles cuando se requiere una función para un corto periodo de tiempo.

Sintaxis:

```
lambda argumentos: expresión
```

Ejemplo:

```
doblar = lambda x: x * 2
print(doblar(5))  # Salida: 10

# Uso con funciones de orden superior como map(), filter()
numeros = [1, 2, 3, 4, 5]
doblados = list(map(lambda x: x * 2, numeros))
print(doblados)  # Salida: [2, 4, 6, 8, 10]
```

FUNCIONES RECURSIVAS

Una función recursiva es aquella que se llama a sí misma durante su ejecución. Se utiliza para resolver problemas que pueden dividirse en subproblemas del mismo tipo.

Ejemplo: Factorial de un número

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Salida: 120
```

Nota: Es importante definir una condición de parada para evitar una recursión infinita.

MÓDULOS EN PYTHON

¿Qué es un módulo y para qué sirve?

Un módulo es un archivo que contiene definiciones y declaraciones de Python (funciones, variables, clases) que pueden ser importadas y utilizadas en otros programas. Los módulos permiten:

- **Organización**: Dividir el código en partes lógicas.
- **Reutilización**: Utilizar el mismo código en diferentes programas.
- **Mantenimiento**: Facilitar actualizaciones y mejoras.

IMPORTACIÓN

Se puede importar un módulo completo o partes específicas de él.

Importar un módulo completo:

```
import math
print(math.pi)  # Salida: 3.141592653589793
```

Importar funciones específicas:

```
from math import sqrt, sin
print(sqrt(16))  # Salida: 4.0
```

Importar con alias:

```
import numpy as np
array = np.array([1, 2, 3])
```

CREACIÓN Y USO DE MÓDULOS PROPIOS

Para crear un módulo propio:

1. Crea un archivo con extensión `.py` (por ejemplo, `utilidades.py`).
2. Define funciones y variables en este archivo.
3. Importa el módulo en tu programa principal.

Ejemplo:

Archivo `utilidades.py`:

```
def saludo_personalizado(nombre):
    print(f"Hola, {nombre}. ¡Bienvenido!")
```

Archivo principal:

```
import utilidades

utilidades.saludo_personalizado("Carlos")
# Salida: Hola, Carlos. ¡Bienvenido!
```

LA LIBRERÍA STANDARD DE PYTHON

Python incluye una amplia biblioteca estándar que ofrece módulos para diversas tareas como manejo de archivos, comunicación en red, operaciones matemáticas, entre otros.

Algunos módulos estándar importantes:

- **`os`**: Interacción con el sistema operativo.
- **`sys`**: Parámetros y funciones del intérprete.
- **`datetime`**: Manipulación de fechas y horas.
- **`json`**: Manejo de datos en formato JSON.
- **`random`**: Generación de números aleatorios.

PAQUETES: ORGANIZACIÓN DE MÓDULOS EN DIRECTORIOS

Un paquete es una colección de módulos organizados en una estructura de directorios. Cada carpeta que representa un paquete debe contener un archivo `__init__.py`.

Estructura de un paquete:

```
mi_paquete/
    __init__.py
```

```
    modulo1.py
    modulo2.py
```

Puedes importar módulos de un paquete:

```
from mi_paquete.modulo1 import funcion_especial
```

MÓDULO MATH

El módulo `math` proporciona funciones matemáticas definidas por el estándar C.

Funciones y constantes comunes:

Función/Constante	Descripción	Ejemplo	Salida
`math.pi`	Constante pi	`math.pi`	3.141592653589793
`math.e`	Constante e	`math.e`	2.718281828459045
`math.sqrt(x)`	Raíz cuadrada de x	`math.sqrt(25)`	5.0
`math.sin(x)`	Seno de x (en radianes)	`math.sin(math.pi/2)`	1.0
`math.log(x[, base])`	Logaritmo de x en base especificada	`math.log(8, 2)`	3.0

DOCUMENTACIÓN Y BUENAS PRÁCTICAS

La documentación y seguir las convenciones de estilo son aspectos cruciales en el desarrollo de software, ya que facilitan la colaboración y el mantenimiento.

DOCSTRING: DOCUMENTACIÓN DE FUNCIONES Y MÓDULOS

Los docstrings son cadenas literales que aparecen justo debajo de la definición de una función, módulo o clase, y describen su funcionamiento.

Ejemplo:

```
def area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio.

    Parámetros:
```

```
    radio (float): El radio del círculo.

    Retorna:
    float: El área del círculo.
    """
    return math.pi * radio ** 2
```

Puedes acceder al docstring utilizando `help(area_circulo)` o `area_circulo.__doc__`.

CONVENCIONES DE NOMBRADO (PEP8)

PEP8 es la guía oficial de estilo para Python. Algunas recomendaciones son:

- **Nombres de variables y funciones**: Letras minúsculas, separadas por guiones bajos (`mi_variable`).
- **Nombres de clases**: Estilo CamelCase (`MiClase`).
- **Constantes**: Letras mayúsculas (`CONSTANTE`).
- **Indentación**: 4 espacios por nivel.
- **Líneas máximas de 79 caracteres**.

USO DE COMENTARIOS

Los comentarios son anotaciones en el código que no son ejecutadas y sirven para explicar partes del código.

- **Comentarios en línea**: Usando `#`.

```
# Esto es un comentario
total = 0  # Inicializar la variable total
```

- **Comentarios multilínea**: Usando `"""` o `'''` para documentar.

Cierre ✍️

Las funciones y los módulos son pilares fundamentales en la programación con Python. A través de este manual, hemos explorado desde los conceptos básicos hasta los avanzados, proporcionando una comprensión profunda y práctica de cómo utilizarlos eficazmente.

El uso adecuado de funciones permite escribir código más limpio, modular y reutilizable, lo que es esencial para proyectos de cualquier tamaño. Al aprovechar las funciones avanzadas, como las funciones anónimas y recursivas, puedes escribir código más eficiente y elegante.

Los módulos, por su parte, facilitan la organización del código y fomentan la reutilización, permitiéndote aprovechar la extensa biblioteca estándar de Python y compartir tu propio código con otros desarrolladores.

Además, seguir las buenas prácticas de documentación y estilo no solo mejora la legibilidad de tu código sino que también facilita la colaboración y el mantenimiento a largo plazo.

Te animamos a seguir practicando y explorando estos conceptos en tus propios proyectos. La programación es una habilidad que se perfecciona con la práctica constante y el aprendizaje continuo. Python, con su comunidad activa y recursos abundantes, es un excelente entorno para crecer como desarrollador.

Referencias 📚

- **Documentación oficial de Python**: https://docs.python.org/3/
    - La fuente más confiable y completa sobre Python, incluyendo tutoriales y referencias detalladas.
- **PEP8 – Guía de estilo para el código Python**: https://www.python.org/dev/peps/pep-0008/
    - Detalles sobre las convenciones de código que ayudan a mantener la uniformidad y legibilidad.
- **W3Schools Python Tutorial**: https://www.w3schools.com/python/
    - Recursos educativos con ejemplos interactivos para aprender Python.
- **Real Python**: https://realpython.com/
    - Artículos y tutoriales en profundidad sobre diversos temas de Python.
- **Programiz Python**: https://www.programiz.com/python-programming
    - Tutoriales y ejemplos claros y concisos para aprender Python.
- **Libro "Automate the Boring Stuff with Python"** de Al Sweigart
    - Un libro orientado a principiantes que enseña cómo utilizar Python para automatizar tareas comunes.

¡Muchas gracias!

Nos vemos en la próxima lección