UD01.01: Introducción

Contenido

1. Introducción a Python

1.1 ¿Qué es Python?

- **Definición general del lenguaje:**
  Python es un lenguaje de programación de alto nivel, interpretado, diseñado para ser fácil de leer y escribir, con una sintaxis clara y concisa que facilita el desarrollo rápido de aplicaciones.
- **Contexto histórico y creador:**
  Fue creado a finales de los años 80 y lanzado en 1991 por Guido van Rossum, un programador holandés que buscaba un lenguaje que combinara la facilidad de uso con la potencia y flexibilidad necesarias para programación general.
- **Origen del nombre y evolución:**
  El nombre “Python” no proviene de la serpiente, sino que está inspirado en el grupo humorístico británico "Monty Python". Desde su creación, Python ha evolucionado significativamente, con versiones principales que han mejorado la sintaxis, la gestión de memoria, y las capacidades del lenguaje, destacando especialmente Python 2 y Python 3.

1.2 Características de Python

- **Lenguaje interpretado:**
  Python no requiere compilación previa; su código se ejecuta directamente línea por línea mediante un intérprete, lo que facilita el desarrollo y prueba rápida.
- **Lenguaje de alto nivel:**
  Proporciona abstracciones que permiten programar sin preocuparse por detalles de bajo nivel como la gestión manual de memoria.
- **Multiparadigma:**
  Soporta diversos estilos de programación, incluyendo programación estructurada, orientada a objetos y funcional.
- **Multiplataforma:**
  Python puede ejecutarse en diferentes sistemas operativos sin necesidad de modificar el código fuente.
- **Libre y de código abierto:**
  Python es distribuido bajo una licencia abierta que permite su uso, modificación y distribución sin coste.

1.3 Ventajas del lenguaje

- **Sintaxis limpia y sencilla:**
  Su código es legible y fácil de entender, lo que reduce errores y facilita el mantenimiento.
- **Alta productividad y rapidez en resultados:**
  Permite desarrollar prototipos y soluciones de manera rápida y eficiente.
- **Comunidad activa y recursos disponibles:**
  Existe una gran comunidad de usuarios y desarrolladores que contribuyen con documentación, tutoriales, paquetes y soporte.
- **Módulos y bibliotecas extensas:**
  Cuenta con una amplia colección de librerías estándar y externas que simplifican tareas complejas.
- **Popularidad y demanda en la industria:**
  Según índices como el TIOBE, Python está entre los lenguajes más usados en el mundo, especialmente en áreas emergentes como ciencia de datos, inteligencia artificial y desarrollo web.

Jun 2025	Jun 2024	Change	Programming Language	Ratings
1	1		Python	25.87%
2	2		C++	10.68%
3	3		C	9.47%
4	4		Java	8.84%
5	5		C#	4.69%
6	6		JavaScript	3.21%

2. Paradigmas de programación en Python

2.1 Programación estructurada (imperativa)

- **Estructuras de control básicas:**
  Uso de sentencias condicionales (if, else), bucles (for, while), y control de flujo para definir el comportamiento del programa.
- **Scripts simples y automatización de tareas:**
  Ideal para escribir pequeños programas o scripts para automatizar procesos repetitivos.
- **Ventajas y ejemplos:**
  Facilita la comprensión lógica y secuencial del código, con un ejemplo típico como recorrer una lista e imprimir elementos.

Este paradigma se enfoca en una secuencia de pasos para resolver un problema.

```
# Definimos una lista de números.
numeros = [1, 2, 3, 4, 5, 6]

# Usamos un bucle 'for' para recorrer cada elemento de la lista.
for numero in numeros:
    # Dentro del bucle, usamos una estructura 'if' para tomar una decisión.
    # El operador % (módulo) nos ayuda a verificar si un número es par.
    if numero % 2 == 0:
        # Si el número es par, lo imprimimos.
        print(numero)
```

2.2 Programación orientada a objetos (POO)

- **Clases y objetos:**
  Python permite definir clases que agrupan datos (atributos) y comportamientos (métodos), y crear objetos que son instancias de esas clases.

- **Métodos y atributos:**
  Los métodos son funciones dentro de las clases; los atributos almacenan datos propios de cada objeto.
- **Encapsulamiento, herencia y polimorfismo:**
  Nociones básicas: el encapsulamiento oculta datos internos; la herencia permite crear nuevas clases basadas en otras; el polimorfismo facilita que diferentes objetos respondan a la misma interfaz.
- **Ejemplo práctico con una clase:**
  Por ejemplo, una clase Persona con atributos nombre y edad y métodos para mostrar información.

La POO organiza el código en torno a objetos que tienen datos y comportamientos.

```
# Usamos 'class' para definir la plantilla de un objeto, en este caso, un Coche.
class Coche:
    # El método especial '__init__' se llama al crear un nuevo objeto.
    # 'self' se refiere al objeto que estamos creando.
    def __init__(self, marca, modelo):
        # Asignamos los datos (atributos) a la instancia del objeto.
        self.marca = marca
        self.modelo = modelo

    # Definimos un 'método', que es una función dentro de la clase.
    def mostrar_info(self):
        # 'f-string' es una forma sencilla de formatear texto en Python.
        print(f"Marca: {self.marca}, Modelo: {self.modelo}")

# Creamos un objeto 'mi_coche' a partir de la clase Coche.
# Le pasamos los valores para 'marca' y 'modelo'.
mi_coche = Coche("Ford", "Focus")
```

```
# Llamamos al método 'mostrar_info' del objeto 'mi_coche'.
mi_coche.mostrar_info()
```

2.3 Programación funcional

- **Concepto de funciones puras:**
  Funciones que no producen efectos secundarios y cuyo resultado depende solo de sus parámetros.
- **Inmutabilidad y ausencia de efectos colaterales:**
  Favorece la predictibilidad y facilita el paralelismo.
- **Funciones map(), filter(), reduce(), expresiones lambda:**
  Uso de funciones para transformar y procesar datos de forma declarativa y compacta.
- **Aplicaciones prácticas en procesamiento de datos:**
  Por ejemplo, aplicar transformaciones a listas sin modificar los datos originales.

Este paradigma se centra en el uso de funciones puras, evitando modificar datos.

```
# Definimos la lista original de números.
numeros = [1, 2, 3, 4, 5]

# Usamos la función 'map()' para aplicar una transformación a cada elemento de la lista.
# 'lambda' nos permite crear una función pequeña y anónima.
# 'x: x**2' es la función que eleva al cuadrado cada número.
# 'map' devuelve un objeto iterador, por lo que lo convertimos a una lista con 'list()'.
numeros_al_cuadrado = list(map(lambda x: x**2, numeros))

# Imprimimos la nueva lista. La lista original 'numeros' no fue modificada.
print(numeros_al_cuadrado)
```

3. Comparación con otros lenguajes

3.1 Python vs. C++

- **Diferencias de sintaxis:**
  Python es mucho más simple y legible; C++ requiere mayor detalle en la declaración de tipos y gestión de memoria.
- **Nivel de complejidad:**
  C++ es un lenguaje de bajo nivel cercano al hardware; Python es de alto nivel, abstrae detalles técnicos.

3.2 Python vs. Java

- **Verbosidad vs. simplicidad:**
  Java suele ser más verboso con código más extenso para tareas similares; Python favorece la simplicidad y rapidez.
- **Facilidad para principiantes:**
  Python se considera más amigable para quienes comienzan a programar debido a su sintaxis sencilla y menor curva de aprendizaje.

3.3 Conclusiones comparativas

- Python ofrece ventajas pedagógicas como su simplicidad y capacidad de enseñanza de conceptos básicos y avanzados con menor esfuerzo.
- En la práctica, es ideal para desarrollo rápido, prototipado y entornos donde la productividad es clave.
- Aunque C++ y Java tienen su lugar en sistemas donde el rendimiento o estructura estricta son prioritarios, Python es comúnmente la primera elección para muchos proyectos.

4. Aplicaciones de Python

4.1 Tipos de aplicaciones posibles

Python es un lenguaje muy versátil que se adapta a múltiples dominios, desde tareas simples hasta proyectos a gran escala.

- **Scripts y automatización:** Para tareas rutinarias, análisis de archivos, y administración del sistema. Por ejemplo, se usa para scripts de gestión de servidores con librerías como **os** y **subprocess**.
- **Análisis de datos y Machine Learning:** Ideal para manipular, analizar y visualizar grandes volúmenes de datos. Las librerías de referencia son **Pandas** para el análisis, **NumPy** para computación numérica y **Scikit-learn** para algoritmos de Machine Learning.
- **Desarrollo web:** Para crear el *backend* de sitios y aplicaciones web. Los *frameworks* más populares son **Django** (para proyectos grandes y complejos) y **Flask** (ideal para proyectos más pequeños o APIs).
- **Aplicaciones de escritorio:** Para construir interfaces gráficas de usuario (GUI). Se usan *frameworks* como **Tkinter**, **PyQt** o **Kivy**.

4.2 Entornos y herramientas de desarrollo

Trabajar con las herramientas adecuadas es clave para el desarrollo en Python.

- **Intérprete interactivo:** La consola de Python permite ejecutar código línea a línea para pruebas rápidas y experimentación.
- **IDEs recomendados:**
    - **IDLE:** Básico, integrado con Python.
    - **Thonny:** Orientado a principiantes, con un depurador visual.
    - **VS Code:** Potente y extensible, con muchas extensiones para Python.
    - **PyCharm:** Un IDE profesional y completo, muy popular entre los desarrolladores.

4.3 Python en el mundo profesional

La popularidad de Python se debe a su amplio uso en diversas industrias y contextos.

- **Casos de uso en empresas:** Grandes compañías como **Google**, **IBM**, **Nokia** y **Netflix** emplean Python para desarrollo interno, análisis de datos, automatización y más.
- **Contextos donde se emplea:** Se usa activamente en áreas como Inteligencia Artificial, ciencia de datos, desarrollo web, finanzas, educación, y automatización, entre otros.