# Soluciones - Primera clase de Python

## 2. Python en la actualidad

### ¿En qué posición se encuentra Python en la actualidad respecto a otros lenguajes de programación?

Según el índice TIOBE de octubre de 2025, **Python se mantiene en la primera posición** como el lenguaje de programación más popular, con un 24.45% de calificación. Ha dominado este índice desde finales de 2023 [1].

### ¿Para qué se usa?

Python es un lenguaje de programación de propósito general con una amplia gama de aplicaciones. Sus usos principales incluyen:

- **Desarrollo web:** Creación de aplicaciones web y APIs (con frameworks como Django y Flask).
- **Ciencia de datos y análisis de datos:** Procesamiento, análisis y visualización de grandes volúmenes de datos.
- **Aprendizaje automático (Machine Learning) e Inteligencia Artificial (IA):** Desarrollo de modelos y algoritmos de IA, incluyendo redes neuronales y procesamiento de lenguaje natural.
- **Automatización de tareas:** Scripting para automatizar procesos repetitivos en sistemas operativos y aplicaciones.
- **Desarrollo de software:** Creación de aplicaciones de escritorio y herramientas de línea de comandos.
- **Big Data:** Manejo y procesamiento de conjuntos de datos masivos.
- **Desarrollo de juegos:** Aunque no es su uso principal, se utiliza en el desarrollo de lógica de juegos y herramientas.
- **Finanzas:** Modelado financiero y análisis cuantitativo.

# 3. Conceptos clave de Python

## Explicar con ejemplos cada uno:

### Lenguaje interpretado

Un **lenguaje interpretado** es aquel cuyo código fuente es ejecutado directamente por un programa llamado intérprete, sin necesidad de ser compilado previamente a código máquina. Python es un lenguaje interpretado, lo que significa que el intérprete de Python lee y ejecuta el código línea por línea. Esto facilita la depuración y el desarrollo rápido, ya que los cambios se pueden probar inmediatamente [2].

**Ejemplo:**

```
# mi_script.py
print("Hola, mundo!")
x = 10
y = 20
print(x + y)
```

Para ejecutar este código, simplemente se invoca el intérprete de Python:

```
python mi_script.py
```

El intérprete leerá y ejecutará cada línea secuencialmente.

### Tipado dinámico

El **tipado dinámico** significa que el tipo de una variable se determina en tiempo de ejecución, no en tiempo de compilación. En Python, no es necesario declarar explícitamente el tipo de una variable; el intérprete asigna el tipo basándose en el valor que se le asigna. Además, una variable puede cambiar su tipo a lo largo de la ejecución del programa [3].

**Ejemplo:**

```
variable = 10          # variable es de tipo int
print(type(variable)) # Salida: <class 'int'>

variable = "Hola"      # Ahora variable es de tipo str
print(type(variable)) # Salida: <class 'str'>

variable = [1, 2, 3]  # Ahora variable es de tipo list
print(type(variable)) # Salida: <class 'list'>
```

### Es fuertemente tipado

Que un lenguaje sea **fuertemente tipado** significa que no permite operaciones entre tipos de datos incompatibles sin una conversión explícita. Python no realiza conversiones automáticas implícitas que puedan llevar a errores inesperados. Si intentas realizar una operación entre tipos que no son compatibles, Python generará un error [4].

### Ejemplo:

```
numero = 5
cadena = "10"

# Esto generará un TypeError porque no se pueden sumar un entero y una cadena
directamente
# resultado = numero + cadena
# print(resultado)

# Para realizar la operación, se necesita una conversión explícita:
resultado_correcto = numero + int(cadena)
print(resultado_correcto) # Salida: 15
```

### Es multiplataforma

Un lenguaje **multiplataforma** es aquel cuyo código puede ejecutarse en diferentes sistemas operativos (como Windows, macOS, Linux, etc.) sin necesidad de ser modificado o recompilado para cada plataforma. Python logra esto gracias a su intérprete, que está disponible para una amplia variedad de sistemas operativos [5].

### Ejemplo:

El siguiente código Python funcionará exactamente igual en Windows, macOS o Linux, siempre y cuando el intérprete de Python esté instalado en cada sistema:

```
import os

print(f"Sistema operativo actual: {os.name}")
print("Este script se ejecuta en cualquier plataforma compatible con Python.")
```

### Te atreves a escribir un código de ejemplo de un lenguaje tipado para que nos quede más claro a todos:

El enunciado pide un ejemplo de un lenguaje *tipado*. Considerando que Python es fuertemente tipado, el ejemplo anterior de la suma de `int` y `str` ya ilustra este concepto. Sin embargo, si se refiere a un lenguaje con **tipado estático** (donde los tipos se declaran y verifican en tiempo de compilación), un ejemplo en Java sería:

**Ejemplo (Java - Lenguaje de tipado estático):**

```java
// Ejemplo.java
public class Ejemplo {
    public static void main(String[] args) {
        int numero = 5; // Se declara explícitamente que 'numero' es un entero
        String cadena = "10"; // Se declara explícitamente que 'cadena' es un String

        // Esto causaría un error de compilación en Java, ya que no se pueden sumar int y String directamente
        // int resultado = numero + cadena;

        // Se requiere una conversión explícita:
        int resultadoCorrecto = numero + Integer.parseInt(cadena);
        System.out.println(resultadoCorrecto); // Salida: 15
    }
}
```

En Java, si intentaras asignar un `String` a una variable declarada como `int` (o viceversa sin conversión), el compilador te lo impediría antes de que el programa se ejecute, a diferencia de Python que lo detectaría en tiempo de ejecución.

## 4. El Zen de Python

### Ejecutar en clase:

```python
import this
```

**¿Qué nos muestra?**

Al ejecutar `import this` en Python, se muestra el "Zen de Python" (The Zen of Python), una colección de 19 principios guía para la escritura de código Python, escritos por Tim Peters. Estos principios reflejan la filosofía de diseño del lenguaje, enfatizando la legibilidad, la simplicidad y la claridad. Algunos de los principios más conocidos son:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- Aunque la practicidad le gana a la pureza.
- Los errores nunca deben pasar silenciosamente.
- A menos que se silencien explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debe haber una, y preferiblemente solo una, manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que seas holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que *ahora* mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres son una gran idea, ¡hagamos más de eso!

## Elige un principio y busca un ejemplo práctico que lo represente.

Principio elegido: **"Bello es mejor que feo."** y **"La legibilidad cuenta."**

Estos principios enfatizan la importancia de escribir código que sea fácil de leer y entender, no solo para el programador original, sino también para otros desarrolladores que puedan trabajar con él en el futuro. Un código "bello" y legible reduce la probabilidad de errores y facilita el mantenimiento.

**Ejemplo práctico:**

Consideremos dos formas de calcular la suma de los números pares en una lista:

**Opción 1 (Menos legible/"fea"):**

```
def calcular_suma_pares_feo(lista):
    s = 0
    for i in range(len(lista)):
        if lista[i] % 2 == 0:
            s += lista[i]
    return s

numeros = [1, 2, 3, 4, 5, 6]
print(calcular_suma_pares_feo(numeros))
```

**Opción 2 (Más legible/"bella" - Usando comprensión de listas y `sum`):**

```
def calcular_suma_pares_bello(lista):
    return sum(numero for numero in lista if numero % 2 == 0)

numeros = [1, 2, 3, 4, 5, 6]
print(calcular_suma_pares_bello(numeros))
```

La Opción 2 es más concisa, utiliza características idiomáticas de Python (comprensión de listas) y es más fácil de entender de un vistazo. Cumple con los principios de "Bello es mejor que feo" y "La legibilidad cuenta" al ser más directa y expresiva.

## 5. Actividad práctica: perdiendo el miedo a programar

Una vez que tienes tu programa `HolaMundo` funcionando haz los siguientes cambios:

### 1. Define una variable con el valor que quieras imprimir por pantalla

```
mensaje = "¡Hola, Kevin! Este es mi primer programa en Python."
print(mensaje)
```

### 2. Declara una segunda variable, realiza la suma de las dos variables e imprime por pantalla el resultado

```
numero1 = 15
numero2 = 27
suma = numero1 + numero2
print(f"La suma de {numero1} y {numero2} es: {suma}")
```

### 3. ¿Qué ocurre si queremos imprimir una variable que no hemos creado?

Si intentamos imprimir una variable que no ha sido creada o definida previamente, Python generará un error de tipo `NameError`. Este error indica que el nombre de la variable no ha sido encontrado en el ámbito actual del programa.

**Ejemplo:**

```
# print(variable_no_existente) # Esto causaría un NameError
```

**Salida esperada (si se ejecutara la línea comentada):**

```
NameError: name 'variable_no_existente' is not defined
```

### Reflexión: los errores son parte del aprendizaje. Python nos avisa de lo que pasa, y podemos aprender a corregirlo.

Esta reflexión es fundamental en la programación. Los errores, o `exceptions` en Python, no son fallos, sino indicaciones del intérprete sobre dónde y por qué algo no salió como se esperaba. Aprender a leer y entender los mensajes de error (`tracebacks`) es una habilidad crucial para cualquier programador. Python, al ser un lenguaje con un tipado fuerte y una filosofía que valora la claridad (como se ve en el Zen de Python), proporciona mensajes de error bastante descriptivos que guían al desarrollador hacia la solución.

Por ejemplo, un `NameError` nos dice que una variable no está definida, un `TypeError` que una operación se intentó con tipos incompatibles, y un `IndexError` que se intentó acceder a un índice fuera de los límites de una secuencia. Cada uno de estos errores es una oportunidad para comprender mejor el lenguaje y mejorar la lógica del código.

## Referencias

[1] TIOBE. (Octubre de 2025). *TIOBE Index*. Recuperado de https://www.tiobe.com/tiobe-index/ [2] AWS. (s.f.). *¿Qué es Python? - Explicación del lenguaje Python*. Recuperado de https://aws.amazon.com/es/what-is/python/ [3]

Interactive Chaos. (s.f.). *Tipado dinámico*. Recuperado de https://interactivechaos.com/es/manual/tutorial-de-python/tipado-dinamico [4] Blog Academia Desarrollo Web. (28 de agosto de 2025). *Tipado de datos en Python: resolviendo el gran problema*. Recuperado de https://blog.academiadesarrolloweb.com/2025/08/28/tipado-de-datos-en-python-resolviendo-el-gran-problema/ [5] Entrenamiento Python Básico. (s.f.). *1.2. Características*. Recuperado de https://entrenamiento-python-basico.readthedocs.io/es/2.7/leccion1/caracteristicas.html