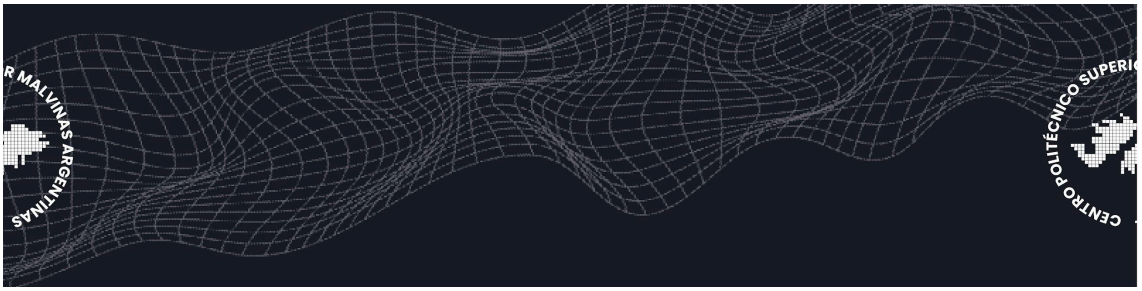


PROGRAMACIÓN 1

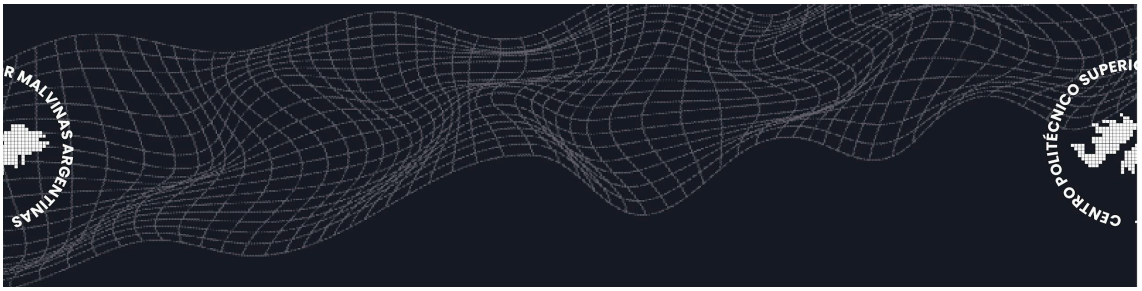
1° AÑO

Clase Nº 3: Fundamentos de la programación

Contenido

En la clase de hoy trabajaremos los siguientes temas:

- Lenguaje de programación Python: qué es, cómo funciona y cuales son sus aplicaciones.
- Elementos de un programa en Python: Sentencias, comentarios, palabras reservadas, identificadores.
- Variables: Declaración, tipos de datos, asignación de valores, literales, ámbito, conversión de tipos y modificadores de acceso.
- Operadores: Aritméticos, relacionales, lógicos, de asignación. Orden de los operadores.
- Expresiones lógicas.
- Expresiones y conversión de tipos.
- Entrada y salida por consola.



1. Presentación

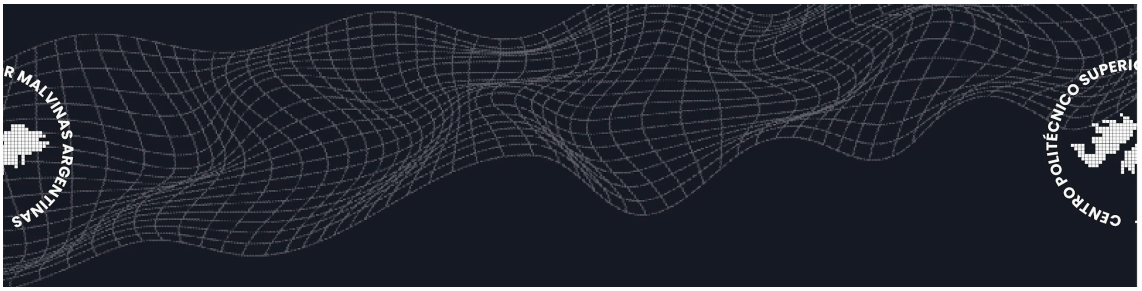
¡Bienvenidos a la tercera clase de Programación 1!

En esta sesión explicaremos los fundamentos del lenguaje de programación Python. Examinaremos los elementos esenciales de un programa en Python, desde las sentencias hasta los sistemas numéricos utilizados para codificar la información.

Nuestro objetivo es familiarizarnos con Python, un lenguaje reconocido por su facilidad de aprendizaje y su versatilidad en diversas áreas de la programación. A lo largo de esta clase, nos sumergimos en los conceptos clave que nos permitirán comprender cómo se estructuran y ejecutan los programas en Python.

Abordaremos temas como la sintaxis de las sentencias, el uso de comentarios para documentar el código, la importancia de las palabras reservadas y la creación de identificadores significativos. Además, exploraremos cómo Python codifica la información y cómo maneja diferentes sistemas numéricos, preparándonos para realizar operaciones y conversiones eficientes.

¡Iniciemos!



2. Desarrollo

¿Qué es Python?

Python es un lenguaje de programación de alto nivel creado por Guido van Rossum en 1991. Se caracteriza por su sintaxis clara y legible, lo que lo hace ideal para desarrollar una amplia gama de aplicaciones.

Características principales:

- **Simplicidad:** La sintaxis de Python es fácil de entender y leer, lo que lo convierte en un lenguaje ideal para principiantes y expertos por igual. El código es tan comprensible como el inglés simple.
- **Versatilidad**: Python es adecuado para una variedad de aplicaciones, desde desarrollo web hasta análisis de datos, inteligencia artificial, scripting y más. Adecuado para tareas cotidianas, permitiendo tiempos de desarrollo cortos.
- **Interpretado:** Python utiliza un enfoque de interpretación en lugar de compilación. Esto significa que el código se ejecuta línea por línea por el intérprete de Python.
- **Código abierto:** para que cualquiera pueda contribuir a su desarrollo.

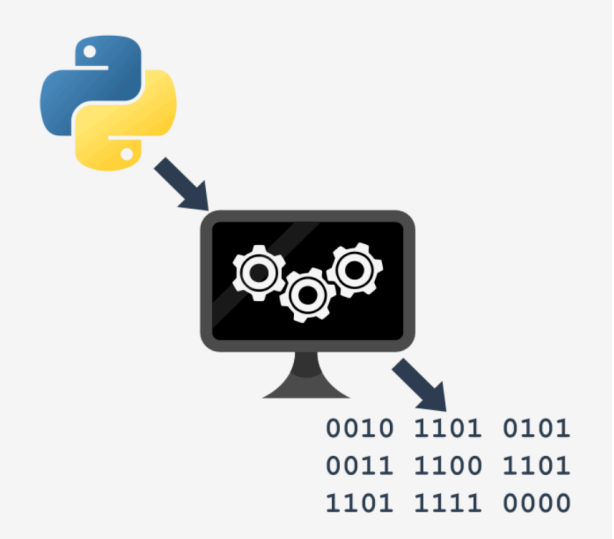
¿Para qué se utiliza Python?

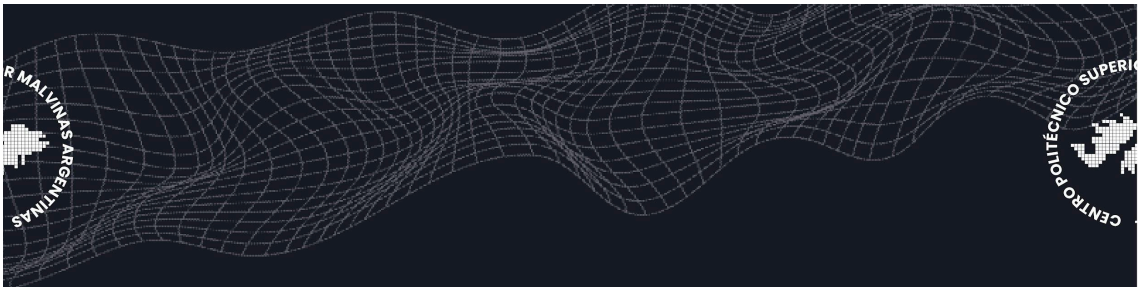
Python se utiliza en una amplia gama de aplicaciones, incluyendo:

- Desarrollo web con frameworks como Django y Flask.
- Análisis de datos y ciencia de datos con bibliotecas como Pandas, NumPy y Matplotlib.
- Desarrollo de juegos con Pygame.
- Automatización de tareas administrativas y scripting.
- Desarrollo de aplicaciones de escritorio con PyQt o Tkinter.
- Desarrollo de inteligencia artificial y aprendizaje automático con TensorFlow, Keras y PyTorch, entre otros.

¿Cómo funciona Python?

Python es un lenguaje de programación interpretado, lo que significa que el código fuente escrito en Python no se traduce directamente a código de máquina como en los lenguajes compilados como C. En cambio, el código Python se ejecuta línea por línea por un intérprete de Python.





Veamos en detalle cómo funciona este proceso:

1. Análisis Léxico y Sintáctico:

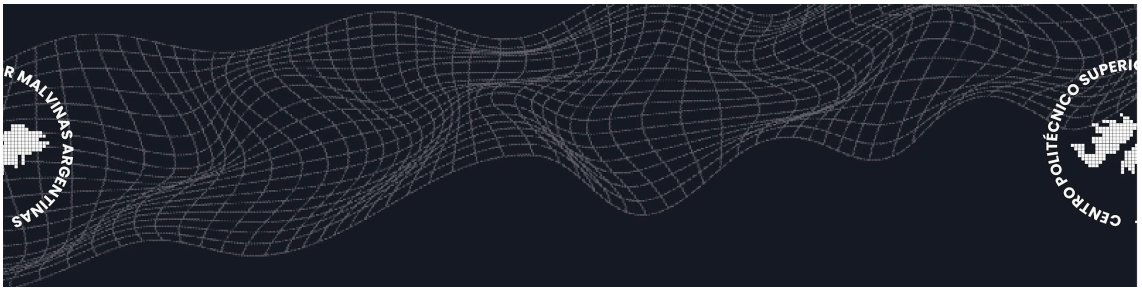
Cuando escribimos un programa en Python, el primer paso es el análisis léxico y sintáctico. Durante este paso, el intérprete de Python analiza el código fuente y verifica su estructura y sintaxis. Si encuentra algún error de sintaxis, como un nombre mal escrito o una instrucción incorrecta, el intérprete generará un mensaje de error.

Durante este proceso, Python también convierte el código fuente en una forma interna llamada "árbol de análisis sintáctico abstracto" (AST), que representa la estructura del programa de una manera más fácil de entender para el intérprete.

2. Generación de Bytecode:

Una vez que el código fuente se ha analizado correctamente, se compila en un conjunto de instrucciones de bajo nivel llamadas "bytecode". El bytecode de Python es un código intermedio que se ejecuta en la máquina virtual de Python (PVM).

El bytecode es independiente de la arquitectura de la máquina, lo que significa que el mismo bytecode se puede ejecutar en diferentes plataformas sin necesidad de recompilación. Esta portabilidad es una de las características clave de Python.



3. Interpretación por la Máquina Virtual de Python:

El bytecode generado se ejecuta en la Máquina Virtual de Python (PVM). La PVM es responsable de ejecutar el bytecode y realizar todas las operaciones necesarias para que el programa funcione correctamente.

La PVM toma las instrucciones bytecode una por una y las traduce a código de máquina nativo para la plataforma en la que se está ejecutando. Este proceso se conoce como "interpretación en tiempo de ejecución".

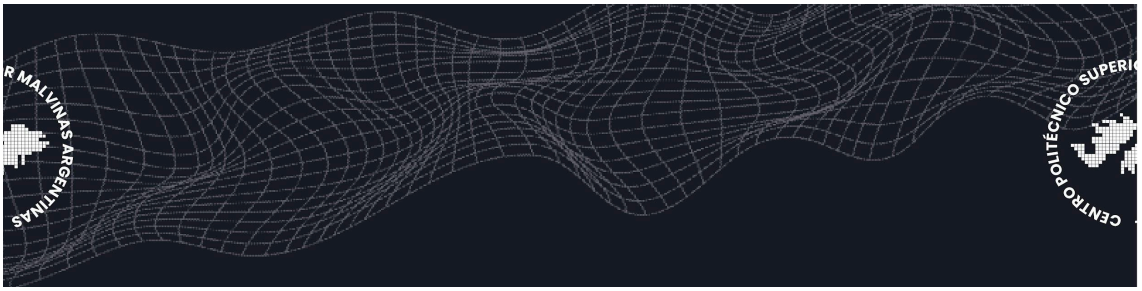
4. Gestión de Memoria y Recolección de Basura:

Python maneja automáticamente la gestión de memoria y la recolección de basura. Esto significa que no es necesario que el programador se preocupe por asignar y liberar memoria manualmente como en otros lenguajes de programación como C.

La recolección de basura en Python es un proceso automático que identifica y elimina objetos no utilizados para liberar espacio en la memoria. Esto simplifica el desarrollo de software en Python y reduce la posibilidad de errores relacionados con la gestión de la memoria.

Resumen comparación con C

Aspecto	Python	C
Velocidad de ejecución	Más lento debido a la interpretación en tiempo de ejecución.	Más rápido debido a la compilación previa.
Sintaxis	Más simple y legible.	Más compleja y detallada.
Tipado	Dinámico, no es necesario especificar el tipo de datos.	Estático, se deben declarar los tipos de datos.
Gestión de memoria	Automática, el intérprete maneja la memoria y la recolección de basura.	Manual, el programador es responsable de la gestión de memoria.



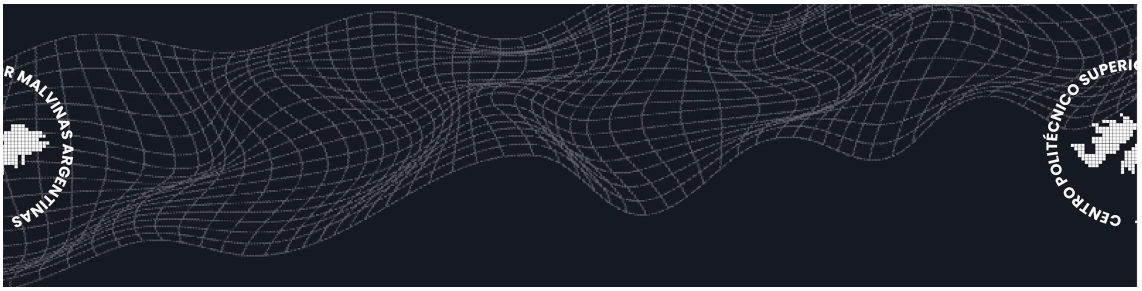
Elementos de un programa en Python

Sentencias

Los programas en Python se construyen utilizando sentencias, que son instrucciones individuales que realizan acciones específicas. En Python, las sentencias pueden estar dentro de funciones, pero también pueden estar en el nivel de módulo, lo que significa que no necesariamente están dentro de una función principal como en otros lenguajes.

Algunas características importantes sobre las sentencias en Python incluyen:

- **No se necesita la finalización con ; (punto y coma)**: A diferencia de algunos lenguajes como C, en Python no es necesario terminar cada sentencia con un punto y coma (;). Python permite el uso opcional del punto y coma al final de una sentencia, pero no es requerido.
- **Bloques de Sentencias:** En Python, los bloques de sentencias están definidos por la indentación, utilizando espacios en blanco o tabulaciones. Esto significa que la estructura del código se determina mediante la indentación, lo que mejora la legibilidad del código.
- **Distinción entre Mayúsculas y Minúsculas:** Al igual que en otros lenguajes de programación, Python distingue entre mayúsculas y minúsculas en los nombres de variables, funciones y otros identificadores. Por lo tanto, *miVariable* y *mivariable* serían consideradas como variables diferentes en Python.



Comentarios

Los comentarios en Python son piezas de texto que son ignoradas por el intérprete durante la ejecución del programa. Se utilizan para documentar el código y hacerlo más comprensible para otros programadores y para el propio desarrollador en el futuro.

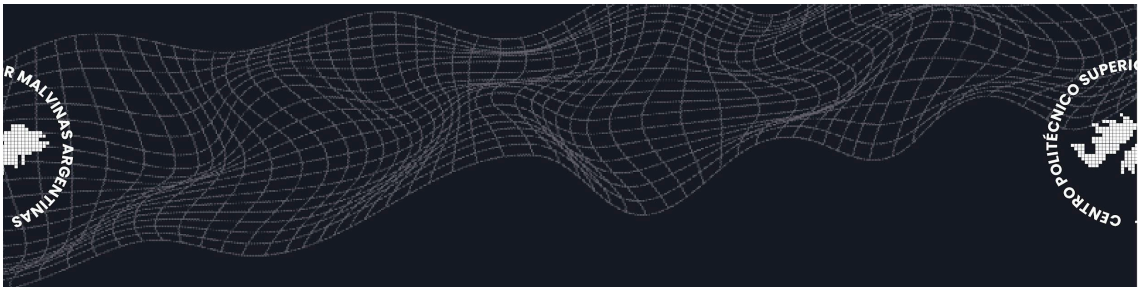
Algunas características de los comentarios en Python son:

- **Delimitación de Comentarios:** En Python, los comentarios de una línea comienzan con el símbolo #. Todo el texto después del símbolo # en una línea se considera un comentario y no se ejecuta.

```
# Este es un comentario de línea única en Python.
```

- **Comentarios Multilínea:** Python también permite comentarios multilínea utilizando triple comillas simples (''') o triples comillas dobles (""""). Esto es útil para comentarios extensos o documentación del código.

```
"""
Este es un comentario multilínea en Python.
Puede abarcar varias líneas y es útil para explicar
bloques de código extensos o complejos.
"""
```

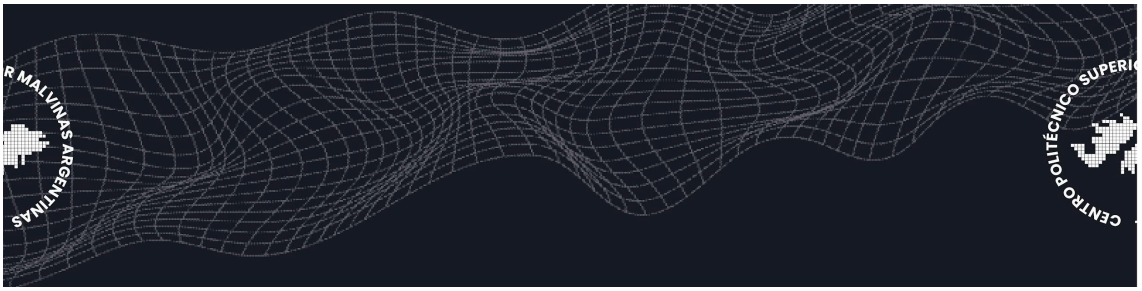


Palabras Reservadas

Las palabras reservadas en Python son aquellas que tienen un significado especial y están reservadas para su uso en el lenguaje. Estas palabras no pueden ser utilizadas como nombres de variables, funciones u otros identificadores en el código.

Ejemplos de palabras reservadas en Python incluyen ***if, else, while, for, def, class, import, from, return,*** entre otras.

Palabras Reservadas	
`False`	`class`
`None`	`return`
`True`	`finally`
`and`	`for`
`as`	`from`
`assert`	`global`
`async`	`if`
`await`	`import`
`break`	`in`
`class`	`is`
`continue`	`lambda`
`def`	`nonlocal`
`del`	`not`
`elif`	`or`
`else`	`pass`
`except`	`raise`
`finally`	`return`
`for`	`try`



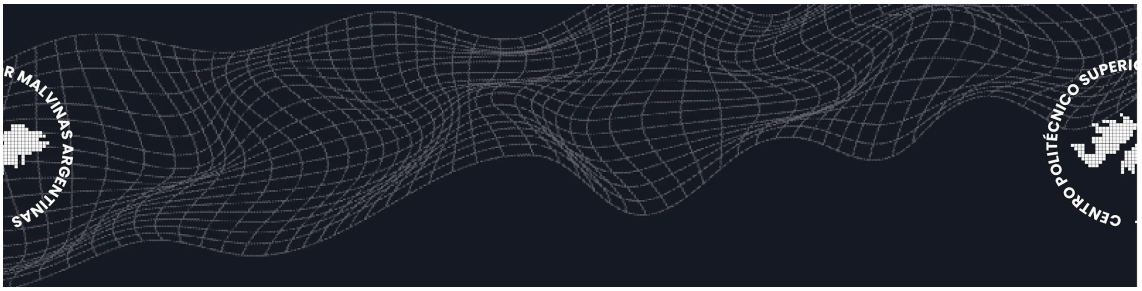
Identificadores

Los identificadores en Python son nombres que se utilizan para identificar variables, funciones, clases y otros elementos en el código. Algunas reglas para los identificadores en Python incluyen:

- Deben comenzar con una letra (a-z, A-Z) o un guion bajo (_), seguido de letras, números o guiones bajos.
- No pueden ser palabras reservadas.
- Python distingue entre mayúsculas y minúsculas en los identificadores, por lo que nombre, Nombre y NOMBRE serían considerados como identificadores diferentes.

Ejemplos de identificadores correctos:

1. `nombre_variable`
2. `numero123`
3. `variable_1`
4. `_mi_variable`
5. `NombreClase`
6. `funcion_principal`
7. `PI`
8. `lista_de_compras`
9. `__variable_privada`
10. `numero_de_telefono`



Variables

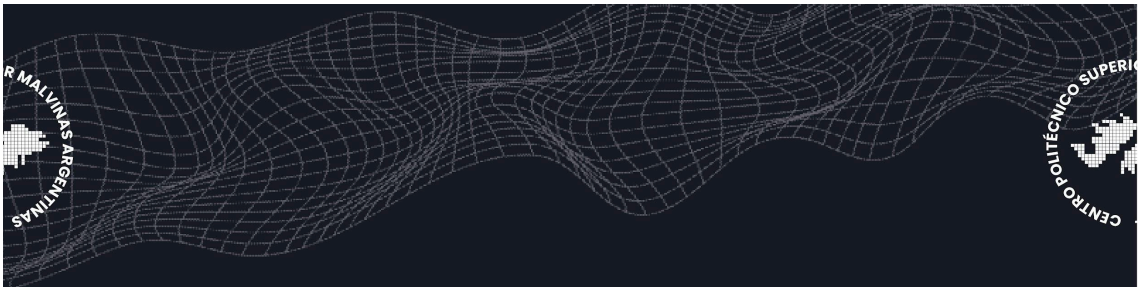
Una variable en Python es un identificador que se utiliza para representar un dato específico dentro de una parte del programa. En su forma más básica, una variable puede ser utilizada para almacenar valores como números, texto, listas u otros tipos de datos. El valor asignado a una variable puede cambiar durante la ejecución del programa, y Python es un lenguaje que permite que el tipo de datos asociado a una variable también cambie dinámicamente. Por ejemplo:

```
a = 3
b = 5
c = a + b
d = 'a'

a = 4
b = 2
c = a - b
d = 'W'
```

Declaraciones

En Python, no se requiere declarar explícitamente el tipo de datos de una variable antes de usarla. La declaración y asignación de valores a variables se pueden hacer en una sola línea de código, sin necesidad de especificar el tipo de datos. Por ejemplo:

```
a, b, e = 10, 20, 30
raiz1, raiz2 = 0.5, 2.5
```

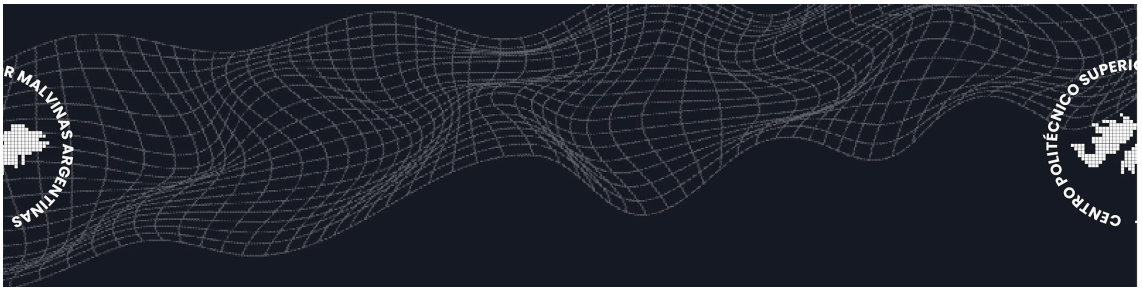


Tipos de datos

Python ofrece una variedad de tipos de datos incorporados, incluyendo:

- int: para números enteros.
- float: para números en coma flotante.
- str: para cadenas de caracteres.
- list: para listas de elementos.
- tuple: para tuplas inmutables.
- dict: para diccionarios clave-valor.
- bool: para valores booleanos True y False.

```
# Números enteros
numero_entero = 42
print(numero_entero)  # Output: 42
print(type(numero_entero))  # Output: <class 'int'>

# Números en coma flotante
numero_flotante = 3.14
print(numero_flotante)  # Output: 3.14
print(type(numero_flotante))  # Output: <class 'float'>

# Cadenas de caracteres
cadena = "Hola, mundo!"
print(cadena)  # Output: Hola, mundo!
print(type(cadena))  # Output: <class 'str'>

# Listas de elementos
lista = [1, 2, 3, 4, 5]
print(lista)  # Output: [1, 2, 3, 4, 5]
print(type(lista))  # Output: <class 'list'>
```



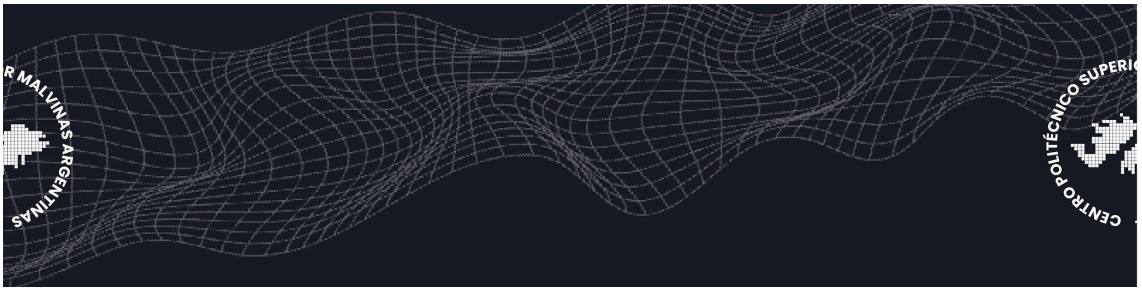

```python
# Tuplas inmutables
tupla = (1, 2, 3)
print(tupla)  # Output: (1, 2, 3)
print(type(tupla))  # Output: <class 'tuple'>

# Diccionarios clave-valor
diccionario = {"nombre": "Juan", "edad": 30}
print(diccionario)  # Output: {'nombre': 'Juan', 'edad': 30}
print(type(diccionario))  # Output: <class 'dict'>

# Valores booleanos
valor_booleano = True
print(valor_booleano)  # Output: True
print(type(valor_booleano))  # Output: <class 'bool'>
```

Asignación de valores

La asignación de valores en Python se realiza utilizando el operador de asignación =. Este operador permite asignar un valor a una variable.

```python
# Asignación de valores simples
x = 10
y = "Hola"
z = 3.14

# Asignación múltiple
a, b, c = 1, 2, 3

# Asignación a través de expresiones
resultado = 5 + 3
```

```python
# Declaración de una variable entera
num = 10

# Declaración y asignación simultánea de múltiples variables
a, b = 5, 10
```



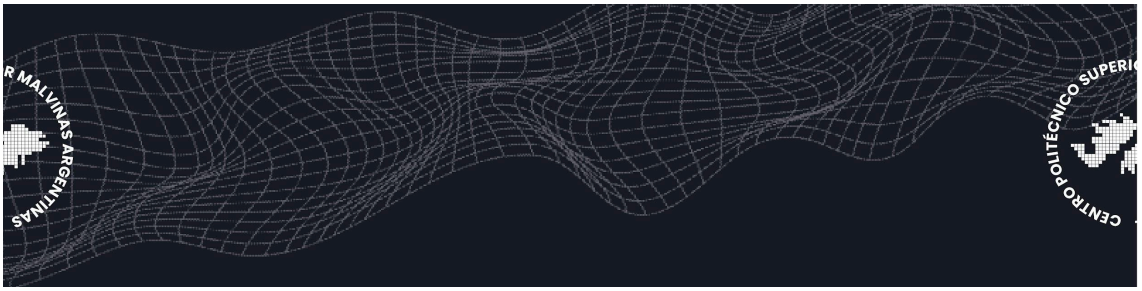
Operadores Aritméticos

Python admite una variedad de operadores aritméticos para realizar operaciones matemáticas:

- Suma +
- Resta -
- Multiplicación *
- División /
- Módulo %
- División entera //
- Exponente **

```
# Ejemplos de operaciones aritméticas
suma = 5 + 3
resta = 10 - 7
multiplicacion = 4 * 6
division = 20 / 5
modulo = 10 % 3
division_entera = 22 // 7
exponente = 2 ** 3
```

```
a = 10
b = 3
suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b
modulo = a % b
division_entera = a // b
exponente = a ** b
```

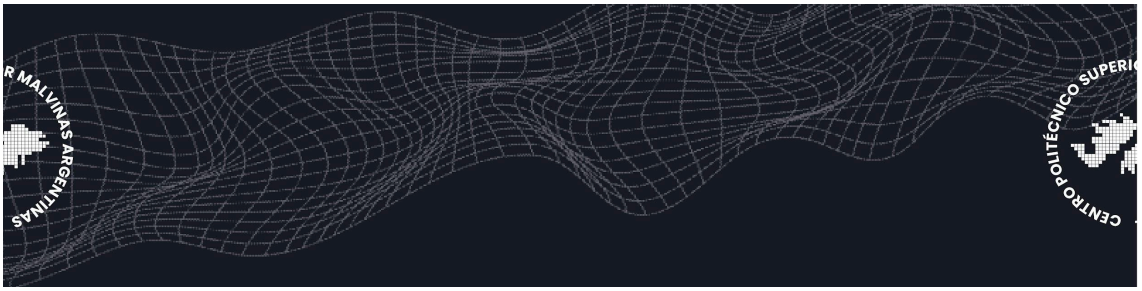


Operadores Relacionales

Los operadores relacionales en Python se utilizan para comparar dos valores y evaluar la relación entre ellos. Estos operadores devuelven un valor booleano (True o False) que indica si la relación es verdadera o falsa. Veamos los diferentes operadores:

- **Igualdad (==):** Este operador verifica si dos valores son iguales. Devuelve True si los valores son iguales y False si son diferentes.
- **Desigualdad (!=):** Este operador verifica si dos valores son diferentes. Devuelve True si los valores son diferentes y False si son iguales.
- **Mayor que (>):** Verifica si un valor es mayor que otro. Devuelve True si el primer valor es mayor que el segundo y False en caso contrario.
- **Menor que (<):** Verifica si un valor es menor que otro. Devuelve True si el primer valor es menor que el segundo y False en caso contrario.
- **Mayor o igual que (>=):** Verifica si un valor es mayor o igual que otro. Devuelve True si el primer valor es mayor o igual que el segundo y False en caso contrario.
- **Menor o igual que (<=):** Verifica si un valor es menor o igual que otro. Devuelve True si el primer valor es menor o igual que el segundo y False en caso contrario.

```
# Operadores relacionales
comparacion = a == b
desigualdad = a != b
mayor = a > b
menor = a < b
mayor_igual = a >= b
menor_igual = a <= b
```



Operadores Lógicos

Los operadores lógicos en Python se utilizan para evaluar condiciones y devolver un valor booleano (True o False) según el resultado de la evaluación.

- **Y lógico (and):**

Devuelve True si ambas condiciones son verdaderas.

Sintaxis: *condicion1* ***and*** *condicion2*

```
condicion1 = True
condicion2 = False
resultado = condicion1 and condicion2
print(resultado)  # Output: False
```

- **O lógico (or):**

Devuelve True si al menos una de las condiciones es verdadera.

Sintaxis: *condicion1* ***or*** *condicion2*

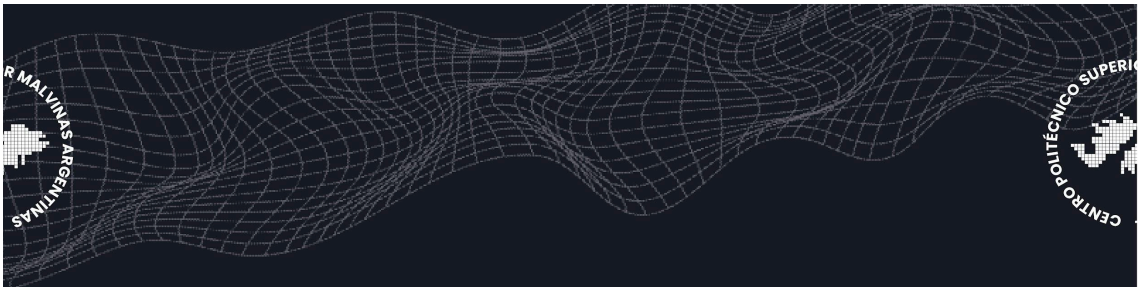
```
condicion1 = True
condicion2 = False
resultado = condicion1 or condicion2
print(resultado)  # Output: True
```

- **Negación lógica (not):**

Devuelve el valor opuesto de una condición.

Sintaxis: ***not*** *condicion*

```
condicion = True
resultado = not condicion
print(resultado)  # Output: False
```



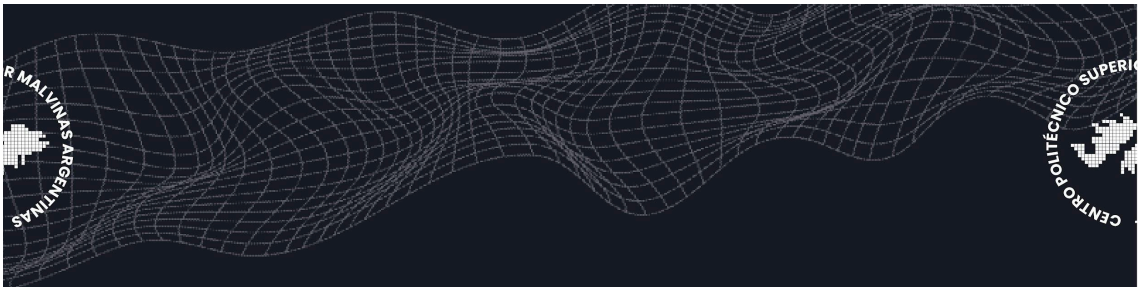
Orden de los Operadores

Los operadores se evalúan en un orden específico,Continuando con la explicación sobre el orden de los operadores en Python:

- Los operadores de mayor precedencia se evalúan primero.
- Los paréntesis se utilizan para agrupar expresiones y establecer el orden de evaluación.

El orden de precedencia de los operadores en Python, de mayor a menor, es el siguiente:

1. Paréntesis ()
2. Exponente **
3. Operadores de multiplicación, división y módulo (*, /, %, //)
4. Operadores de suma y resta (+, -)
5. Operadores de comparación (==, !=, >, <, >=, <=)
6. Operadores lógicos (not, and, or)
7. Operadores de asignación (=, +=, -=, *=, /=, %=, //=, **=)



Expresiones

En Python, una expresión es una combinación de valores, variables y operadores que se evalúa para producir un resultado.

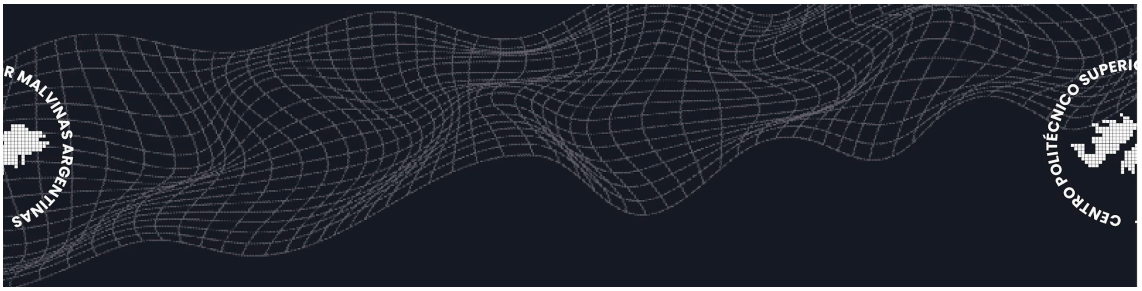
Algunos ejemplos prácticos:

```python
# Expresiones aritméticas
suma = 2 + 3  # suma es 5
producto = 4 * 5  # producto es 20
division = 10 / 2  # division es 5.0
modulo = 10 % 3  # modulo es 1
potencia = 2 ** 3  # potencia es 8

# Expresiones booleanas
condicion1 = True
condicion2 = False
resultado_and = condicion1 and condicion2  # resultado_and es False
resultado_or = condicion1 or condicion2  # resultado_or es True
resultado_not = not condicion1  # resultado_not es False

# Expresiones de concatenación de cadenas
nombre = "Juan"
apellido = "Perez"
nombre_completo = nombre + " " + apellido  # nombre_completo es "Juan Perez"
```

Conversión de tipos

En Python, puedes convertir valores de un tipo a otro utilizando funciones incorporadas específicas.

Algunos ejemplos prácticos:



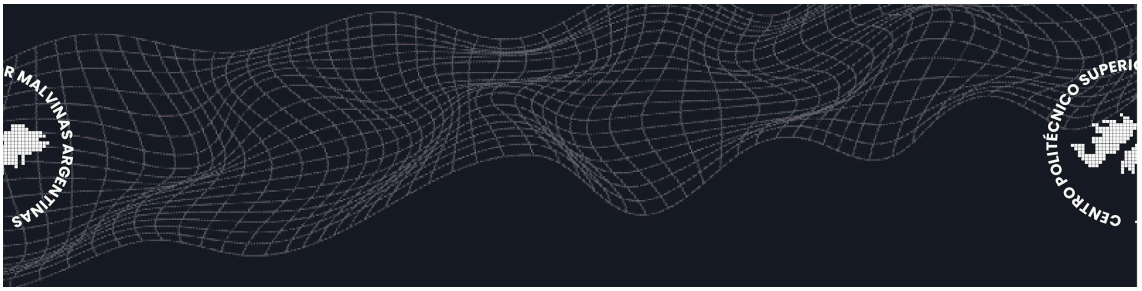

```python
# Conversión de tipo int
numero_entero = int("10")  # convierte "10" en 10
resultado = numero_entero + 5  # resultado es 15


# Conversión de tipo float
numero_decimal = float("3.14")  # convierte "3.14" en 3.14
resultado = numero_decimal * 2  # resultado es 6.28


# Conversión de tipo str
numero = 42
cadena = str(numero)  # convierte 42 en "42"
saludo = "Hola, " + str(numero)  # saludo es "Hola, 42"


# Conversión de tipo bool
valor = 0  # considerado False
resultado = bool(valor)  # convierte 0 en False
```

Entrada de Datos

En Python, la entrada de datos se puede realizar mediante la función input(). Esta función permite al usuario introducir datos desde el teclado. Por lo general, se utiliza para solicitar información al usuario durante la ejecución de un programa. Ejemplo:

```python
nombre = input("Por favor, ingrese su nombre: ")
print("¡Hola,", nombre + "!")
```

En este ejemplo, la función input() solicita al usuario que ingrese su nombre, que luego se almacena en la variable nombre. Luego, se imprime un saludo utilizando el nombre ingresado por el usuario.



Consideraciones Adicionales:

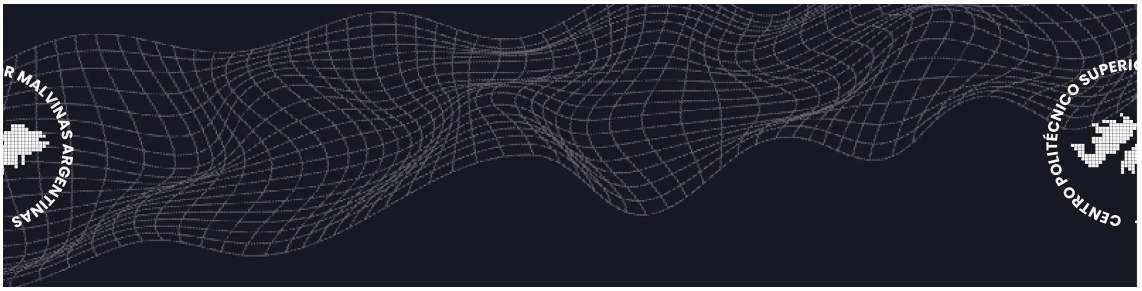
- Es importante tener en cuenta que la función input() devuelve siempre una cadena de texto (string), incluso si el usuario ingresa un número. En caso de necesitar convertir la entrada a otro tipo de dato, como un entero o un flotante, se debe realizar una conversión explícita utilizando las funciones int() o float(), respectivamente.
- La función input() detiene la ejecución del programa hasta que el usuario ingresa una entrada y presiona la tecla Enter. Por lo tanto, es útil utilizarla en combinación con mensajes claros que indiquen al usuario qué se espera de él.

Salida de Datos

Para mostrar resultados o mensajes al usuario en Python, se utiliza la función print(). Esta función toma uno o más argumentos y los muestra en la consola. Ejemplo:

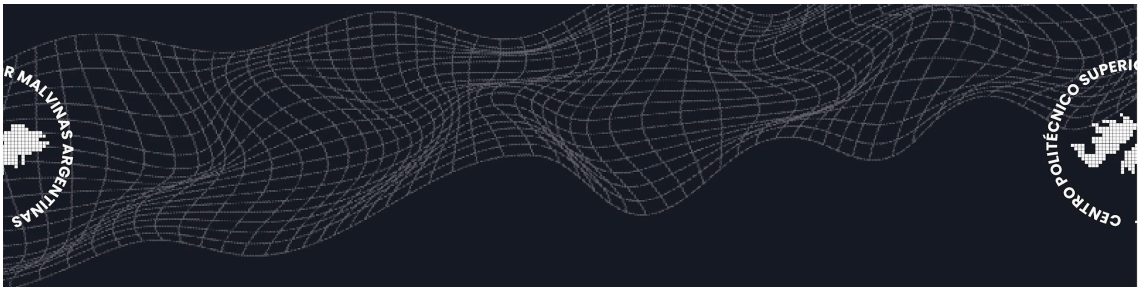
```
nombre = "Juan"
edad = 30
print("Hola,", nombre + ". Tu edad es", edad, "años.")
```

En este ejemplo, se utiliza la función print() para mostrar un mensaje de saludo junto con el nombre y la edad de una persona.



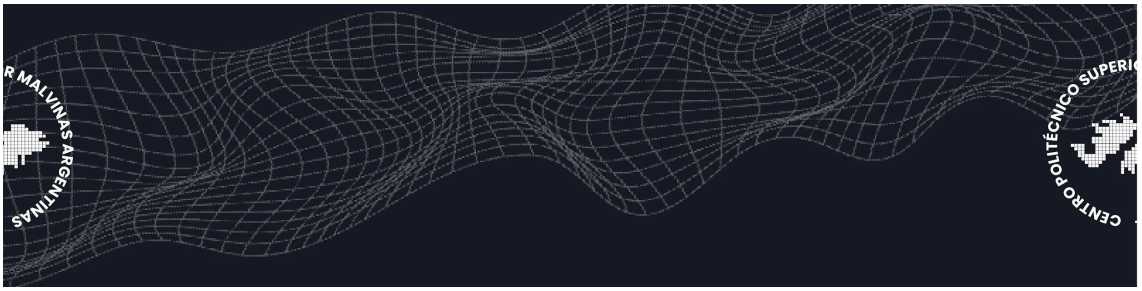
3. Actividades

1. **Operadores relacionales:** Crea un programa que compare dos números y determine si el primero es mayor que el segundo.
2. **Operadores lógicos:** Construye un programa que determine si un número ingresado por el usuario es par y positivo al mismo tiempo.
3. **Conversión de tipos:** Realiza una concatenación de una cadena y un número, convirtiendo el número a cadena.
4. **Expresiones condicionales:** Crea un programa que determine si un año ingresado por el usuario es un año bisiesto o no, usando expresiones lógicas.
5. **Expresiones complejas:** Escribe un programa que determine si un número es divisible por 3 y 5 al mismo tiempo.



4. Conclusión

Este contenido abarca los fundamentos esenciales de la programación, incluyendo cómo estructurar programas con sentencias y comentarios, trabajar con diferentes tipos de datos y variables, realizar operaciones con operadores, y tomar decisiones basadas en condiciones lógicas. También se abordan conceptos clave como la entrada y salida de datos. Estos conceptos son cruciales para construir una base sólida en programación y son aplicables en una amplia gama de lenguajes y proyectos.



Bibliografía Obligatoria

- ***FUNDAMENTOS DE PROGRAMACIÓN****. Algoritmos, estructura de datos y objetos-Cuarta edición - Luis Joyanes Aguilar-McGraw-Hill - 2008 - ISBN 978-84-481-6111-8*

Bibliografía sugerida de la unidad

- ***Python 3 – Los fundamentos del lenguaje*** *- Es un libro completo, bien escrito, claro, aunque un poco extenso, por lo que no es apto para lectores apurados.*
- ***Python para todos: explorando datos en Python 3*** *- Este libro es ideal para estudiantes y programadores que buscan especializarse en la exploración y el análisis de datos.*

Bibliografía adicional

- *Python Documentation: excelente referencia para comprender en profundidad los operadores y su uso en el lenguaje Python. Documentación de Python*
- *"Learning Python" de Mark Lutz: Este libro es una guía completa para aprender Python desde cero. Learning Python*
- *"Python Crash Course" de Eric Matthes: Este libro es ideal para principiantes que desean aprender Python de manera rápida y efectiva. Python Crash Course*
- *"Python for Data Analysis" de Wes McKinney: Si estás interesado en utilizar Python para el análisis de datos, este libro es una excelente referencia. Python for data analysis*