

Introducción a python

Índice

Introducción	2
Lenguaje interpretado	3
Usos de Python	4
Cómo empezar a programar en Python	5
Operadores	9
Variables	14
Recolector de basura	18
Conversión de tipos de datos	19

Python

Introducción

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue creado a finales de los 80 por Guido van Rossum y su nombre fue inspirado en el grupo de comediantes británicos Monty Python.

Python es uno de los lenguajes más populares y ampliamente utilizados en la actualidad, debido a su facilidad de aprendizaje, sintaxis legible y su amplia gama de aplicaciones en áreas como la ciencia de datos, inteligencia artificial, desarrollo web, entre otros.

Python es un lenguaje interpretado, lo que significa que no necesita ser compilado antes de ser ejecutado. Esto lo hace más fácil de utilizar y depurar, ya que los errores se pueden identificar de manera inmediata.

Python tiene una sintaxis simple y legible, que se enfoca en el uso de palabras en lugar de símbolos, lo que lo hace fácil de leer y escribir para los programadores de todos los niveles. Además, cuenta con una amplia biblioteca estándar y una gran cantidad de paquetes externos disponibles, lo que lo hace ideal para proyectos de cualquier tamaño y complejidad.

En resumen, Python es un lenguaje de programación de alto nivel, interpretado y de propósito general que es ampliamente utilizado debido a su facilidad de aprendizaje, sintaxis legible y su amplia gama de aplicaciones.

Python es un lenguaje de programación con una serie de características que lo hacen muy popular entre los programadores. A continuación se detallan algunas de las características más importantes de Python:

1. Sintaxis clara y legible: Python tiene una sintaxis clara y legible que hace que el código sea fácil de leer y comprender. Esto permite a los programadores escribir y mantener el código de manera más rápida y sencilla.
2. Multiplataforma: Python es compatible con múltiples sistemas operativos, lo que significa que el mismo código puede ejecutarse en diferentes plataformas, lo que lo hace muy versátil.
3. Orientado a objetos: Python es un lenguaje orientado a objetos, lo que significa que se pueden definir clases y objetos que permiten la reutilización del código y una mayor modularidad.
4. Tipado dinámico: significa que las variables no tienen que ser declaradas con un tipo de datos específico antes de su uso. Esto permite una mayor flexibilidad en la escritura de código.
5. Amplia biblioteca estándar: tiene una amplia biblioteca estándar que proporciona una gran cantidad de funciones y módulos útiles, lo que hace que sea más fácil y rápido desarrollar aplicaciones.
6. Interpretado: lo que permite una depuración de código más rápida y eficiente.
7. Comunidad activa: existe una gran comunidad de usuarios y desarrolladores que contribuyen con el desarrollo de paquetes, módulos y herramientas útiles, lo que lo hace una herramienta muy poderosa y versátil.

Lenguaje interpretado

Que Python sea un lenguaje de programación interpretado significa que el código fuente escrito en Python no se compila a un lenguaje de máquina ejecutable directamente, sino que se interpreta en tiempo de ejecución. En otras palabras, el intérprete de Python lee y

ejecuta el código línea por línea, traduciéndolo en instrucciones que la computadora puede entender y ejecutar.

Este enfoque presenta varias ventajas, entre ellas la facilidad para depurar y corregir errores en el código, ya que los errores se informan inmediatamente después de la ejecución de la línea de código correspondiente. Además, los lenguajes interpretados como Python generalmente tienen un proceso de desarrollo más rápido, ya que los cambios realizados en el código se pueden ver de inmediato sin necesidad de volver a compilar el programa.

Por otro lado, los lenguajes interpretados suelen tener una velocidad de ejecución menor que los lenguajes compilados, ya que el proceso de interpretación puede ser más lento que la ejecución de código de máquina compilado. Sin embargo, en la práctica, la velocidad de ejecución de Python es suficientemente rápida para la mayoría de las aplicaciones y se puede mejorar mediante el uso de bibliotecas y módulos de alto rendimiento.

Usos de Python

Python es un lenguaje de programación de propósito general que tiene una amplia variedad de aplicaciones en diferentes campos. Algunos de los usos más comunes de Python incluyen:

1. Desarrollo web: es muy utilizado en el desarrollo web, tanto para la creación de servidores como para la creación de sitios web dinámicos. Hay muchos frameworks de Python para el desarrollo web, como Django, Flask, Pyramid y Bottle.
2. Análisis de datos: es muy popular en el ámbito del análisis de datos. Con bibliotecas como NumPy, Pandas y Matplotlib, se pueden manipular grandes cantidades de datos y visualizarlos de manera efectiva.
3. Inteligencia artificial y aprendizaje automático: es uno de los lenguajes más utilizados en el campo del aprendizaje automático e inteligencia artificial. Bibliotecas como TensorFlow, Keras y PyTorch son muy utilizadas para crear modelos de aprendizaje automático.

4. Automatización de tareas: es un lenguaje muy útil para la automatización de tareas, ya que se puede utilizar para crear scripts y programas que realicen tareas específicas de manera automatizada.
5. Juegos y entretenimiento: también se utiliza para la creación de juegos y aplicaciones de entretenimiento, especialmente en la creación de juegos 2D.
6. Aplicaciones de escritorio: también se utiliza para el desarrollo de aplicaciones de escritorio, especialmente en el uso de frameworks como PyQt y wxPython.
7. Ciberseguridad: también es muy utilizado en el campo de la ciberseguridad, especialmente para la creación de herramientas de seguridad y análisis de vulnerabilidades.

En resumen, Python es un lenguaje muy versátil y se utiliza en una gran variedad de aplicaciones en diferentes campos, lo que lo hace una herramienta muy valiosa para cualquier desarrollador o usuario de tecnología.

Cómo empezar a programar en Python

Primero debemos instalar la versión de Python en nuestra computadora y luego un programa que funcionará como editor de código. Existen varios, como:

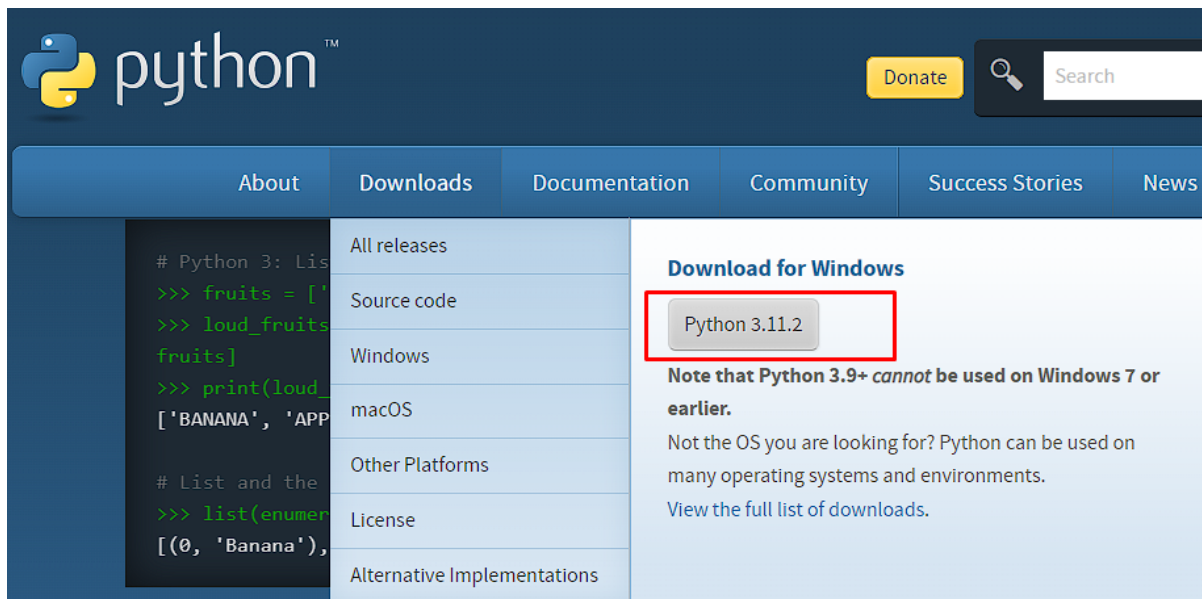
- Visual Studio Code.
- Sublime Text
- Eclipse
- Tony

A los fines de ayudarte a iniciar, te mostraremos como hacerlo con sublime text pero tu puedes elegir el que sea de tu agrado.

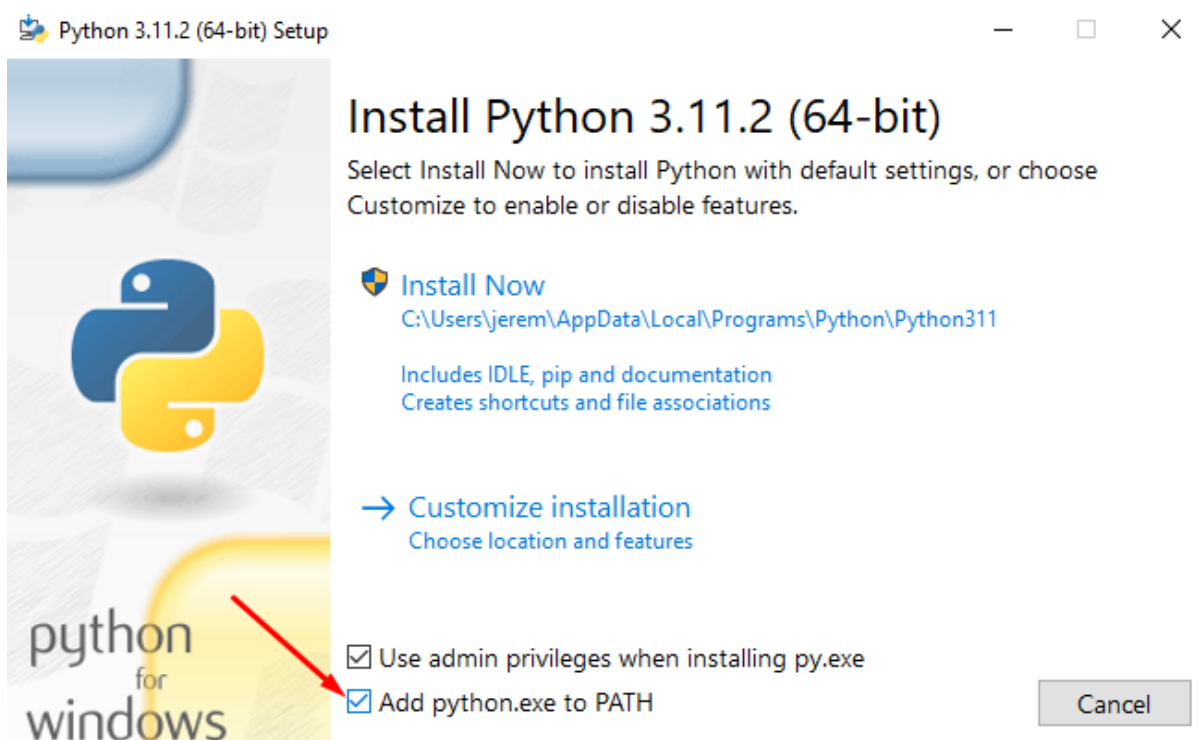
Primer paso - Instalar Python

Para instalar Python en tu computadora, sigue estos pasos:

1. Visita el sitio web oficial de Python en <https://www.python.org/downloads/> y descarga la versión más reciente del instalador para tu sistema operativo.

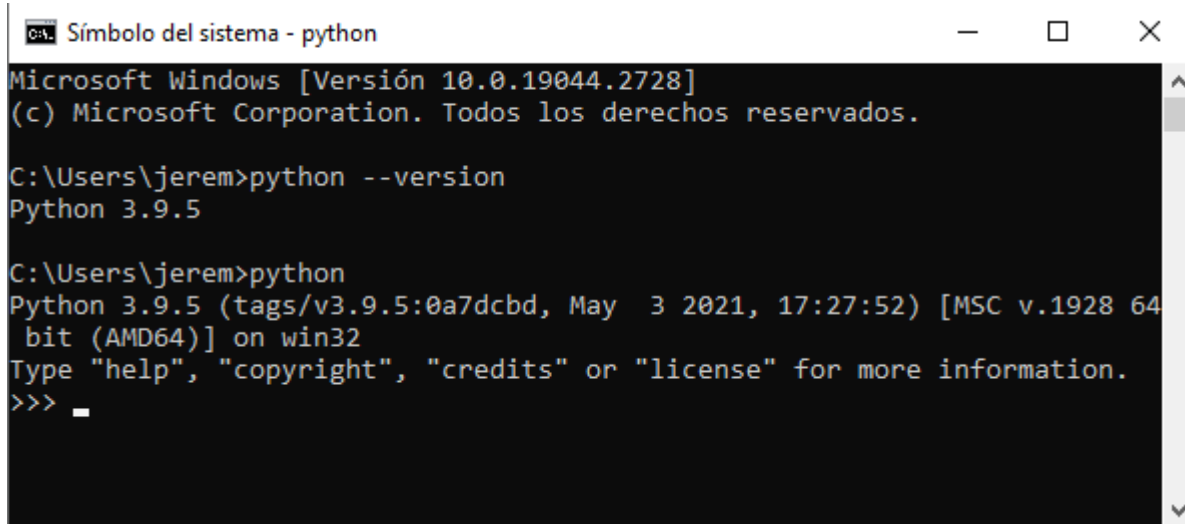


2. Ejecuta el archivo de instalación y sigue las instrucciones del asistente de instalación. Asegúrate de leer cuidadosamente cada paso antes de continuar.
3. En la ventana de instalación, selecciona la opción para agregar Python al PATH del sistema. Esto permitirá que puedas ejecutar comandos de Python desde cualquier ubicación en tu computadora.



4. Continúa con la instalación y espera a que se complete. El tiempo de instalación puede variar dependiendo de la velocidad de tu ordenador y de la versión de Python que hayas descargado.

5. Una vez que la instalación se haya completado, abre una ventana de línea de comandos (terminal en Linux/Mac o símbolo del sistema en Windows) y escribe "python --version" o "python". Si aparece la versión significa que Python se ha instalado correctamente.



```
Símbolo del sistema - python
Microsoft Windows [Versión 10.0.19044.2728]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jerem>python --version
Python 3.9.5

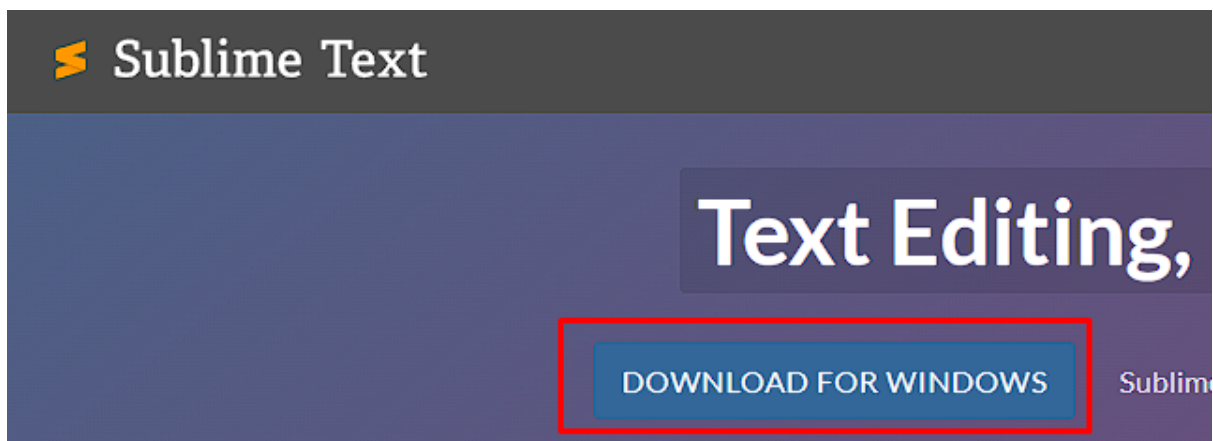
C:\Users\jerem>python
Python 3.9.5 (tags/v3.9.5:0a7dcdb, May 3 2021, 17:27:52) [MSC v.1928 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Segundo Paso - Instalar editor de código

Para empezar a programar en Python con Sublime Text, sigue estos pasos:

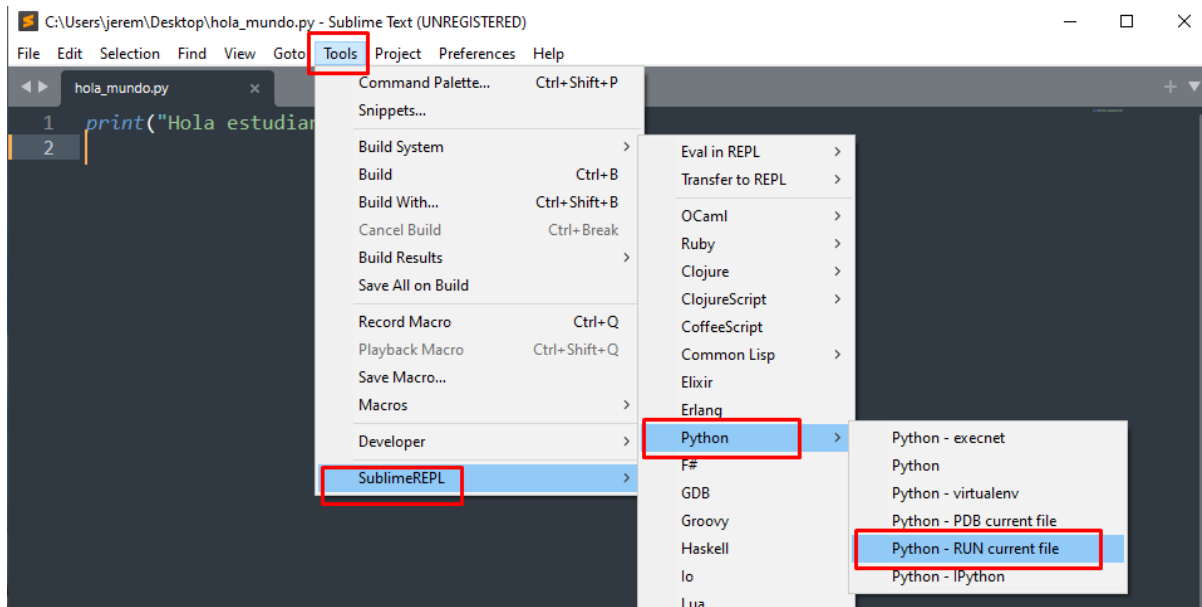
1. Descarga e instala Sublime Text en tu ordenador desde su página oficial:

<https://www.sublimetext.com/>.



2. Abre Sublime Text y crea un nuevo archivo en blanco.
3. Guarda el archivo con la extensión ".py" para indicar que es un archivo de código Python. Puedes guardar el archivo en cualquier ubicación en tu ordenador.
4. Escribe el código de Python en el archivo, comenzando con la línea "print('Hola estudiantes')" para imprimir el mensaje "Hola mundo" en la pantalla.

5. Para ejecutar el código, sigue los pasos que se muestran a continuación. Verás la salida del código en la ventana de la consola en la parte inferior de la pantalla.



6. En caso que no aparezca la opción de SublimeREPL, sigue los siguiente pasos:
- Abre Sublime Text y haz clic en "Tools" en la barra de menú.
 - Selecciona "Command Palette" en el menú desplegable.
 - En la ventana emergente, escribe "Package Control: Install Package" y selecciona esa opción.
 - Espera unos segundos hasta que aparezca una lista de paquetes disponibles.
 - Escribe "SublimeREPL" en el campo de búsqueda y selecciona el paquete que aparece.
 - Espera a que se instale el paquete y aparecerá un mensaje de confirmación.
7. Si deseas personalizar la configuración de Sublime Text para programar en Python, puedes instalar paquetes adicionales y plugins para tener un entorno más completo. Para ello, selecciona "Preferences" > "Package Control" en la barra de menú superior y luego busca e instala los paquetes relevantes.

Tipos de datos básicos

Python es un lenguaje de programación de tipado dinámico, lo que significa que las variables no necesitan ser declaradas con un tipo de dato específico antes de su uso. Los tipos de datos en Python se pueden clasificar en los siguientes tipos:

1. Números: los números en Python pueden ser enteros (int), flotantes (float) o complejos (complex). Ejemplos:

```
a = 5          # un entero
b = 3.14       # un flotante
c = 2 + 3j     # un número complejo
```

2. Cadena de caracteres: las cadenas de caracteres (str) son una secuencia de caracteres que se representan con comillas simples (') o dobles ("). Ejemplos:

```
nombre = 'Juan'          # una cadena de caracteres con comillas simples
apellido = "Pérez"       # una cadena de caracteres con comillas dobles
mensaje = 'Hola, ¿cómo estás?' # una cadena de caracteres con comillas simples
```

3. Booleanos: los valores booleanos (bool) en Python pueden ser True (verdadero) o False (falso). Ejemplos:

```
a = True
b = False
```

Operadores

En Python, los operadores son símbolos que se utilizan para realizar operaciones en variables y valores. A continuación, se muestran algunos de los operadores más comunes en Python junto con ejemplos:

1. Operadores aritméticos:

- Suma (+): se utiliza para sumar dos valores..

```
a = 5 + 3
print(a)  # salida: 8
```

- Resta (-): se utiliza para restar dos valores.

```
b = 10 - 7
print(b)    # salida: 3
```

- Multiplicación (*): se utiliza para multiplicar dos valores.

```
c = 2 * 4
print(c)    # salida: 8
```

- División (/): se utiliza para dividir dos valores

```
d = 15 / 3
print(d)    # salida: 5.0
```

- División entera (//): se utiliza para obtener la división entera de dos valores.

```
e = 15 // 4
print(e)    # salida: 3
```

- Módulo (%): se utiliza para obtener el resto de la división de dos valores.

```
f = 15 % 4
print(f)    # salida: 3
```

- Potencia (**): se utiliza para elevar un valor a una potencia.

```
g = 2 ** 3
print(g)    # salida: 8
```

2. Operadores de comparación:

- Igualdad (==): se utiliza para comprobar si dos valores son iguales.

```
a = 5
b = 5
print(a == b)    # salida: True
```

- Desigualdad (!=): se utiliza para comprobar si dos valores son diferentes.

```
c = 10
d = 7
print(c != d)    # salida: True
```

- Mayor que (>): se utiliza para comprobar si un valor es mayor que otro.

```
e = 15
f = 10
print(e > f)     # salida: True
```

- Menor que (<): se utiliza para comprobar si un valor es menor que otro.

```
g = 2
h = 5
print(g < h)     # salida: True
```

- Mayor o igual que (>=): se utiliza para comprobar si un valor es mayor o igual que otro.

```
i = 8
j = 8
print(i >= j)    # salida: True
```

- Menor o igual que (<=): se utiliza para comprobar si un valor es menor o igual que otro.

```
k = 3
l = 5
print(k <= l)    # salida: True
```

3. Operadores lógicos:

- Y (and): se utiliza para comprobar si se cumplen dos condiciones al mismo tiempo.

```
a = 5
b = 3
c = 10
print(a > b and c > a) # salida: True
```

- O (or): se utiliza para comprobar si se cumple al menos una de dos condiciones.

```
d = 15
e = 20
f = 25
print(d > e or f > e) # salida: True
```

- No (not): se utiliza para negar una condición.

```
g = 6
h = 5
print(not g > h) # salida: False
```

Operador de asignación

En Python, los operadores de asignación se utilizan para asignar valores a una variable. Los operadores de asignación más comunes son los siguientes:

1. El operador "=" (igual) se utiliza para asignar un valor a una variable.

Ejemplo:

```
x = 5

# La variable x ahora tiene el valor de 5
print(x)
```

2. El operador "+=" (más igual) se utiliza para agregar un valor a una variable existente.

Ejemplo:

```
x = 5
x += 3

# La variable x ahora tiene el valor de 8
print(x)
```

Operador "-=":

3. El operador "-=" (menos igual) se utiliza para restar un valor a una variable existente.

Ejemplo:

```
x = 5
x -= 2

# La variable x ahora tiene el valor de 3
print(x)
```

Jerarquía de operadores

Los operadores se ejecutan en un orden específico de acuerdo con su precedencia y asociatividad. A continuación, se muestra una lista de operadores en orden de precedencia, de mayor a menor:

- Operadores de paréntesis: ()
- Operadores de potenciación: **
- Operadores de multiplicación, división y módulo: *, /, //, %
- Operadores de suma y resta: +, -
- Operadores de comparación: <, <=, >, >=, !=, ==
- Operador de asignación: =
- Operadores lógicos: and, or, not

La asociatividad se refiere al orden en que se agrupan los operandos cuando hay varios operadores con la misma precedencia en una expresión. En Python, los operadores binarios son asociativos de izquierda a derecha, lo que significa que se evalúan de

izquierda a derecha. Por ejemplo, la expresión $4 * 3 / 2$ se evalúa primero como $4 * 3$, y luego el resultado se divide por 2.

Es importante tener en cuenta que los operadores de paréntesis tienen la mayor precedencia y pueden ser utilizados para modificar el orden de evaluación de los demás operadores. Por ejemplo, la expresión $(4 * 3) / 2$ se evalúa primero como $4 * 3$, y luego el resultado se divide por 2.

Otro ejemplo

$3 + 4 * 5 - 6 / 2$:

El operador de multiplicación tiene una precedencia mayor que el operador de suma y resta, por lo que se evalúa primero $4 * 5$, que da como resultado 20. Luego se suman 3 y 20, lo que da 23. Finalmente, se divide 6 entre 2, lo que da 3. La expresión completa se evalúa como $23 - 3$, lo que da como resultado 20.

Variables

Una variable es un espacio de memoria reservado para almacenar un valor o una referencia a un valor. Las variables en Python son dinámicamente tipadas, lo que significa que el tipo de datos de una variable puede cambiar durante la ejecución del programa.

Para crear una variable, simplemente se le da un nombre y se le asigna un valor utilizando el operador de asignación `"="`.

Ejemplo:

```
# Asignando un valor a la variable x
x = 5

# Asignando un valor a la variable y
y = "Hola mundo"
```

En este ejemplo, hemos creado dos variables: "x" y "y". La variable "x" contiene el valor numérico "5", mientras que la variable "y" contiene la cadena de texto "Hola mundo".

Es importante tener en cuenta que en Python, el nombre de una variable puede contener letras, números y guiones bajos, pero no puede comenzar con un número. Además, los nombres de variables son sensibles a mayúsculas y minúsculas, lo que significa que "x", "X" y "x1" son tres variables diferentes.

Una vez que se ha creado una variable, se puede utilizar en cualquier parte del programa. Por ejemplo, se puede utilizar una variable en una expresión matemática o en una cadena de texto.

Ejemplo:

```
x = 5
y = 10

# Imprimiendo la suma de x e y
print(x + y)

# Imprimiendo una cadena de texto que contiene el valor de la variable x
print("El valor de x es: " , x)
```

En este ejemplo, hemos utilizado las variables "x" y "y" en una expresión matemática para obtener la suma de los dos valores, y también hemos utilizado la variable "x" en una cadena de texto para imprimir su valor.

Definir nombres de variables

A continuación se presentan algunas buenas prácticas para definir nombres de variables en Python:

Utilice nombres de variables descriptivos:

1. Utilice nombres de variables que describan de manera clara y concisa el contenido de la variable. Utilice nombres de variables que sean fáciles de entender y que no sean demasiado largos.

Ejemplo:

```
# Buen ejemplo de nombre de variable
edad = 25

# Mal ejemplo de nombre de variable
e = 25
```

Utilice nombres de variables en minúsculas:

2. En Python, se recomienda utilizar nombres de variables en minúsculas. Si el nombre de la variable consta de varias palabras, puede separarlas con un guión bajo.

Ejemplo:

```
# Buen ejemplo de nombre de variable
nombre_completo = "Jeremias Lopez"

# Mal ejemplo de nombre de variable
nombrecompleto = "Jeremias Lopez"
```

Evite utilizar nombres de variables reservados:

3. Evite utilizar nombres de variables que palabras reservadas en Python, como "print", "input" o "list".

Ejemplo:

```
# Mal ejemplo de nombre de variable
print = "Hola mundo"
```

Utilice nombres de variables que reflejen el tipo de datos que contienen:

4. Utilice nombres de variables que reflejen el tipo de datos que contienen. Por ejemplo, utilice "lista_edades" en lugar de simplemente "lista".

Ejemplo:

```
# Buen ejemplo de nombre de variable
lista_edades = [20, 30, 40]

# Mal ejemplo de nombre de variable
lista = [20, 30, 40]
```


Siguiendo estas buenas prácticas, puede asegurarse de que su código sea fácil de entender y mantener, y puede ayudar a otros programadores a entender su código de manera más eficiente.

Ámbito de las variables

El ámbito de una variable en Python se refiere a la parte del programa donde la variable es accesible. En Python, existen dos tipos de ámbito de variables: global y local.

Una variable declarada fuera de una función se considera global y está disponible en todo el programa. Por otro lado, una variable declarada dentro de una función se considera local y solo está disponible dentro de esa función.

Ejemplo:

```
# Variable global
x = 5

def funcion():
    # Variable local
    y = 10
    print("El valor de x es: ", x)
    print("El valor de y es: ", y)

funcion()
print("El valor de x fuera de la función es: ", x)
```

En este ejemplo, la variable "x" se declara fuera de la función y, por lo tanto, es una variable global. La función "funcion()" declara la variable "y" dentro de ella, lo que la convierte en una variable local. Dentro de la función, ambas variables se imprimen en la consola y están disponibles. Fuera de la función, solo se puede acceder a la variable global "x".

Es importante tener en cuenta que si se intenta asignar un valor a una variable global dentro de una función, Python creará una nueva variable local en lugar de modificar la variable global. Para modificar la variable global, es necesario utilizar la palabra clave "global".

Ejemplo:

```
# Variable global
x = 5

def funcion():
    # Utilizando la palabra clave global para modificar la variable global
    global x
    x = 10
    print("El valor de x dentro de la función es: ", x)

funcion()
print("El valor de x fuera de la función es: ", x)
```

En este ejemplo, se utiliza la palabra clave "global" para indicar que se desea modificar la variable global "x" dentro de la función. Al imprimir el valor de "x" dentro y fuera de la función, se puede ver que el valor ha sido modificado en todo el programa.

Recolector de basura

En este punto no te preocupes si no se comprende totalmente el código que se muestra de ejemplo, sin embargo, es importante conocer el concepto.

El Garbage Collector o Recolector de Basura es una función en Python que administra la memoria en tiempo de ejecución. Su objetivo es liberar la memoria que ya no está siendo utilizada por el programa y así evitar problemas de memoria insuficiente.

En Python, el recolector de basura es automático y se encarga de eliminar objetos que ya no son utilizados por el programa. Esto significa que no es necesario liberar la memoria manualmente, como sucede en otros lenguajes de programación.

Un ejemplo simple de cómo funciona es el siguiente:

```

import sys

def mi_funcion():
    a = 10
    b = 20
    return a + b

# Creamos un objeto grande que no es utilizado en el programa
mi_lista = [i for i in range(1000000)]

# Imprimimos el tamaño en bytes del objeto "mi_lista"
print("El tamaño de la lista es: ", sys.getsizeof(mi_lista))

# Llamamos a la función
resultado = mi_funcion()

# Imprimimos el resultado de la función
print("El resultado de la función es: ", resultado)

```

En este ejemplo, se crea un objeto grande "mi_lista" que no se utiliza en la función "mi_funcion()". Después, se imprime el tamaño en bytes del objeto "mi_lista". Luego se llama a la función "mi_funcion()", que crea variables "a" y "b" y devuelve su suma. Finalmente, se imprime el resultado de la función.

Después de que se imprime el tamaño en bytes del objeto "mi_lista", el Garbage Collector elimina automáticamente la lista "mi_lista", ya que no se utiliza en el resto del programa. También elimina las variables "a" y "b" después de que se devuelve el resultado de la función.

En resumen, el Garbage Collector en Python se encarga de liberar automáticamente la memoria que ya no es utilizada por el programa, lo que hace que el proceso de programación sea más fácil y seguro.

Conversión de tipos de datos

En Python, es posible convertir entre diferentes tipos de datos utilizando las funciones de conversión incorporadas. Aquí hay algunos ejemplos:

1. Conversión de int a float (De entero a flotante):

```
numero_entero = 5
numero_flotante = float(numero_entero)
print(numero_flotante) # Salida: 5.0
```

2. Conversión de float a int (De flotante a entero):

```
numero_flotante = 3.14
numero_entero = int(numero_flotante)
print(numero_entero) # Salida: 3
```

3. Conversión de int a str (De entero a cadena de caracteres. str proviene de string que significa cadena):

```
numero_entero = 10
texto = str(numero_entero)
print(texto) # Salida: "10"
```

4. Conversión de str a int (De cadena a entero):

```
texto = "15"
numero_entero = int(texto)
print(numero_entero) # Salida: 15
```

5. Conversión de str a float (De cadena a flotante):

```
texto = "3.14"
numero_flotante = float(texto)
print(numero_flotante) # Salida: 3.14
```

Es importante tener en cuenta que la conversión de tipos de datos puede llevar a la pérdida de información si el valor original no se puede representar en el nuevo tipo de datos. Por ejemplo, la conversión de un número decimal a un número entero elimina la parte decimal.