# **Módulo 1**

# **Conceptos fundamentales de la programación en Python**

**Características básicas de Python:**

Aplicaciones de Python:

Tipo de objetos:

* Objetos tipo int
* Objetos tipo Punto Flotante ( float )
* Objetos tipo complex
* Objetos tipo bool

**Secuencias y colecciones**

* **Mapeo**
* **Conjuntos**
* Concepto de Mutable e Inmutable
* Variables y constantes:
* Constantes incorporadas:
* Casting:

Operadores

* Operadores aritméticos.
* Reglas de precedencia en operaciones aritméticas.
* Operadores de comparación o relacionales.
* Operadores lógicos.
* Operadores de asignación.
* Operadores de identidad.

Bloques

* Indentación de bloques de código.

Condicionales If.

Estructura de coincidencia match case.

Bucles:

* Bucles While:
* Modificación de ejecución en un bucle while.
* break:
* continue

Conceptos fundamentales de la programación en Python

**Características básicas de Python:**

|  |
| --- |
| import sys  print(sys.version)  Muestra la versión de Python que tienes instalada.  >>python --version |
| license():  Muestra la licencia bajo la cual está distribuida la versión de Python que tienes instalada. |
| credits():  Muestra información sobre los desarrolladores y los equipos que contribuyeron a la creación de Python |

Las características que hacen de Python un lenguaje tan popular y poderoso son:

* Sintaxis muy clara y legible.
* Fuerte capacidad de introspección.
* Orientación a objetos intuitiva.
* Expresión del código procedimental.
* Altamente modular, soporta paquetes jerárquicos.
* Uso de excepciones para el manejo de errores.
* Tipos de datos dinámicos de alto nivel.
* Extensa biblioteca estándar (STL) y módulos de terceros para prácticamente todas las tareas.
* Extensiones y módulos fácilmente escritos en C, C++ (o Java para Jython, o. NET para IronPython).
* Altamente integrable dentro de las aplicaciones como una interfaz de scripting.

Aplicaciones de Python:

Al ser un lenguaje multipropósito y altamente portable, Python se ha utilizado para desarrollar

* Aplicaciones de escritorio.
* Aplicaciones web.
* Análisis de datos.
* Administración de servidores.
* Seguridad y análisis de penetración.
* Cómputo en la nube.
* Cómputo científico.
* Procesamiento de lenguaje natural.(NLP)
* Visión artificial.
* Animación, videojuegos e imágenes generadas por computadora.
* Aplicaciones móviles.
* Machine Learning y Deep Learning
* Y mucho mas.

**Tipo de objetos:**

En Python todo son objetos. En un primer momento diferenciaremos los objetos tipo funciones de los usados para guardar datos.

Dentro de los usados para guardar datos tendremos los objetos que permiten guardar solo uno y las colecciones que permiten guardar múltiples objetos

En los objetos de un dato tenemos los numéricos enteros y flotantes, los booleanos y los strings (str)

En las colecciones veremos las tuplas, listas, diccionarios, set y frozensets.

Ademas hay objetos que se agregan mediante de las librerías (DataFrames, Arrays, etc.)

|  |
| --- |
| **Cuadro básico de tipos de objetos** |
| * int: Representa números enteros. * float: Representa números de punto flotante (con decimales). * complex: Representa números complejos (parte real e imaginaria). * bool: Representa valores de verdad (True o False). * str: Representa cadenas de texto. * list, tuple, range: Representan secuencias de elementos. * dict: Representa un diccionario de clave-valor. * set, frozenset: Representan conjuntos mutables e inmutables.   Mutabilidad e Inmutabilidad:  Objetos mutables pueden cambiar después de su creación (ejemplos: listas, diccionarios).  Objetos inmutables no pueden cambiar después de su creación (ejemplos: cadenas, tuplas).  La elección entre mutabilidad e inmutabilidad afecta al rendimiento y la seguridad del programa.  Variables y Constantes:  Una variable es un nombre que se asigna a un dato en memoria.  Las variables en Python son dinámicamente tipadas.  Las constantes son valores que no cambian durante la ejecución.  En Python, no existe una forma de declarar constantes, pero se usa convención de nomenclatura (mayúsculas) para indicarlas. |

**Objetos tipo int:**

Python int son objetos para representar números enteros (aquellos números que no tienen parte decimal o fracciones ni decimales)

Definición y Creación:

El interprete reconoce estos datos numéricos enteros y asigna el tipo de objeto

Operaciones Aritméticas:

Los enteros admiten operaciones aritméticas estándar (suma, resta, multiplicación, división exponenciación, radicación, etc)

El tamaño de un int en Python no es fijo y puede variar según la plataforma. Sin embargo, los enteros son representados internamente usando una cantidad de bytes fija. Por ejemplo, en una plataforma de 64 bits, el rango típico para un int es de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

**Operaciones Bit a Bit:**

Los enteros también admiten operaciones bit a bit, como AND, OR, XOR y desplazamientos:

a = 5 # Representación binaria: 101

b = 3 # Representación binaria: 011

resultado\_and = a & b # AND bit a bit: 001 (resultado = 1)

resultado\_or = a | b # OR bit a bit: 111 (resultado = 7)

resultado\_xor = a ^ b # XOR bit a bit: 110 (resultado = 6)

desplazamiento\_izq = a << 2 # Desplazamiento izquierdo: 10100 (resultado = 20)

Objetos tipo Punto Flotante ( float ):

Los números de punto flotante son una aproximación de los números reales en matemáticas. Deben ser utilizados teniendo en cuenta las consideraciones de su implementación y precisión (que no son una limitante de Python en sí mismo, sino de los sistemas computacionales).

Pueden especificarse también en notación científica.

Peligro con los flotantes

**﻿﻿**Hay que estar atentos a las operaciones y lógica realizadas con punto flotante. Al comparar sin tener en cuenta la precisión, podemos encontrar resultados 'inesperados'

Python float son objetos para representar números reales o de punto flotante, es decir, números con una parte entera y una parte fraccionaria (decimal).

**Precaución:** En operaciones matemáticas con números float pueden surgir problemas de precisión y redondeo en ciertas operaciones. Estas son limitaciones de precisión inherentes a la aritmética de punto flotante.

**Representación numérica:** Los números de punto flotante en Python siguen la representación estándar IEEE 754 para aritmética de punto flotante, lo que permite representar números con precisión limitada.

**Sintaxis:** Los números de punto flotante se pueden escribir en notación decimal utilizando el punto decimal para separar la parte entera de la parte fraccionaria. También se pueden escribir en notación científica utilizando la letra "e" o "E" para indicar la potencia de 10.

num = 2.5e3 # Equivalente a 2500.0

**Comparación:** Debido a la representación finita de los números de punto flotante, es importante tener en cuenta las limitaciones de comparación exacta. A menudo, es recomendable utilizar una tolerancia (epsilon) para comparar números de punto flotante en lugar de hacer comparaciones exactas.

**Limitaciones de precisión:** Los números de punto flotante tienen una precisión limitada. Esto significa que no todos los números reales pueden representarse con precisión. Al realizar cálculos, especialmente operaciones repetidas, puede acumularse un error debido a la representación finita.

**Funciones y métodos:** Python proporciona varias funciones y métodos incorporados para trabajar con números de punto flotante. Algunos ejemplos incluyen round(), abs(), math.sqrt(), math.sin(), etc.

**Infinito y NaN:** El tipo float también puede representar valores especiales como infinito (positivo o negativo) y NaN (Not-a-Number), que se utilizan para indicar resultados no válidos o indefinidos en cálculos.

|  |
| --- |
| import sys  print('Información tipo de dato float: { sys.float\_info }')  print('Información tipo de dato int: { sys.int\_info}')  print('Tamaño máximo entero: { sys.maxsize}') |

-

**Objetos tipo complex:**

Python complex son objetos para representar números complejos, que son números que constan de una parte real y una parte imaginaria. Un número complejo se denota en la forma a + bj, donde a es la parte real y b es la parte imaginaria, y j es la unidad imaginaria que cumple con la propiedad j^2 = -1.

Los objetos complex son especialmente útiles en matemáticas y en aplicaciones científicas y de ingeniería donde se tratan números complejos, como análisis de señales, sistemas lineales y procesamiento de imágenes.

**Creación de números complejos:** utilizando la función complex() o directamente escribiendo la parte real e imaginaria con la unidad imaginaria j.

dato = complex(3, 4)

dato = 3 + 4j

real\_part = dato.real # 3.0

imag\_part = dato.imag # 4.0

Funciones matemáticas y módulo:

Python proporciona funciones matemáticas para trabajar con números complejos, como abs(), que devuelve la magnitud (distancia al origen) del número complejo, y cmath para funciones trigonométricas, exponenciales y logarítmicas.

dato = 3 + 4j

magnitude = abs(dato) # 5.0

phase = cmath.phase(dato) # 0.93 (en radianes)

Objetos tipo bool:

Objeto bool o boleano

Ver álbebra de Boole en electrónica y en especial en cpu

El objeto bool o boleano que en python son: True o False , heredan de la clase int , o sea True toma valor 1 y False valor 0 y se pueden hacer todas las operaciones aritméticas.

Python bool son objetos para representar un valor de verdad, es decir, un valor que puede ser verdadero (True) o falso (False). Los valores booleanos son fundamentales en la programación, ya que se utilizan para tomar decisiones y controlar el flujo del programa.

True:

El valor True representa la afirmación verdadera. Se utiliza para indicar que una condición es cierta o que algo es verdadero.

False:

El valor False representa la afirmación falsa. Se utiliza para indicar que una condición no es cierta o que algo es falso.

El resultado de una condición siempre es una booleana. Si no hay condición explícita se toma como is True. Son esenciales para construir estructuras de control y tomar decisiones en la programación, permitiendo que los programas reaccionen de manera inteligente a diferentes condiciones y entradas.

Objetos tipo strings:

Python string o cadena de texto son objetos para representar secuencias de caracteres. Son fundamentales en Python y se utilizan ampliamente en la programación para manipular y representar información textual.

Tengan en cuenta que los caracteres pueden ser números, letras, simbolismo, etc.

Representación:

Una cadena se representa colocando el texto entre comillas simples (') o comillas dobles (")

Si se genera Con un conjunto el string tenga una linea, con triple juegos de comillas (''' o """) para múltiples lineas .

Codificaciones:

Las cadenas se manejan internamente como secuencias de Unicode, lo que permite trabajar con caracteres de varios idiomas y símbolos. Para codificaciones específicas (como UTF-8), puedes usar funciones como encode() y decode() para convertir entre cadenas Unicode y bytes codificados.

|  |
| --- |
| cadena\_unicode = u'\u00a1\u0050\u0079\u0074\u0068\u006f\u006e \u0065\u0073 \u0074\u0072\u0065\u006d\u0065\u006e\u0064\u006f\u0021'  print(cadena\_unicode)  frase = 'Python es tremendo!'  print(frase)  print(type(frase))  cadena\_bytes = bytes(frase, 'utf-8')  print(cadena\_bytes)  print(type(cadena\_bytes)) |

Indexación y slicing:

Puedes acceder a caracteres individuales en una cadena utilizando índices numéricos. El primer caracter tiene un índice de 0 el segundo un 1 hasta el ultimo siempre con némeros enteros positivos.

Puedes segmentar una cadena con slicing para obtener subcadenas usando la notación [inicio:fin].

Inmutabilidad:

Las cadenas en Python son inmutables, lo que significa que no se pueden modificar después de su creación.

Longitud de la Cadena:

Puedes obtener la longitud de una cadena utilizando la función len().

**Comparación de Cadenas**:

Las cadenas se pueden comparar utilizando operadores de comparación como ==, !=, <, >, <= y >=. Las comparaciones se basan en el orden lexicográfico (orden alfabético).

Raw Strings:

Los "raw strings" (cadenas sin procesar) se crean colocando una r antes de las comillas iniciales. En estas cadenas, las secuencias de escape (como \n o \t) se interpretan literalmente, lo que es útil cuando se trabaja con expresiones regulares y rutas de archivos en sistemas operativos.

Veremos en profundidad todos los métodos y atributos de la clase string en el módulo 2

Resumen de tipos de objetos:

En Python, existen varios tipos de datos el interprete encasilla dentro de objetos predefinidos, incluyen números, cadenas de texto, coleciones (listas, tuplas, conjuntos, diccionarios, etc – módulo 2 y 3). Los tipos de datos se utilizan para representar diferentes objetos con diferentes métodos y atributos. Las librerías agregan nuevas clases de objetos con sus métodos y atributos propios

﻿﻿**bool:** de verdadero o falso, para comprobar si un elemento cumple una condición o para usarlo en bucles.

**Númericos**

**int:** para números enteros, es decir, sin decimales.

**float:** de coma flotante, es decir, números con decimales (la coma flotante es la expresión para referirse a ellos).

**complex:** números complejos, es decir, números con parte imaginaria. Son raramente usados en programación.

**Secuencias y colecciones**

**str:** para cadenas de caracteres.

**list:** para hacer listas.

**tuple:** para hacer tuplas, listas inmodificables.

**range:** para hacer listas inmutables con números enteros en sucesión aritmética.

**array:** mediante librerías Ej array, numpy

**dataframes:** de Pandas

**Mapeo**

**dict:** diccionario; para relacionar un nombre o definición a una clave. Al escribir la clave te saldrá el nombre o definición correspondiente.

**Conjuntos**

**set:** para conjuntos mutables (modificables).

**frozenset:** para conjuntos inmutables (invariables).

**Concepto de Mutable e Inmutable:**

En Python todo son objetos, desde un numero entero a una función. Los objetos de almacenamiento de datos se clasifican en dos categorías principales: **mutables e inmutables**. Esta clasificación se refiere a la capacidad de un objeto de cambiar su valor o contenido después de su creación.

Un objeto mutable es aquel cuyo valor o contenido puede ser modificado después de su creación. En otras palabras, los objetos mutables pueden ser alterados mediante la modificación de su contenido sin cambiar su identidad. Por ejemplo, las listas y los diccionarios son objetos mutables en Python. Podemos agregar, eliminar o modificar elementos de una lista o diccionario sin crear un nuevo objeto.

Por otro lado, un objeto inmutable es aquel cuyo valor o contenido no puede ser modificado después de su creación. En otras palabras, los objetos inmutables no pueden ser alterados mediante la modificación de su contenido sin cambiar su identidad. Por ejemplo, las cadenas y las tuplas son objetos inmutables en Python. No podemos modificar los caracteres de una cadena o los elementos de una tupla después de su creación.

Es importante tener en cuenta que la mutabilidad e inmutabilidad de un objeto en Python también puede afectar su rendimiento y uso de memoria, por lo que es importante elegir la estructura de datos adecuada según los requisitos de cada caso de uso.

Importancia de la Mutabilidad:

**Eficiencia:** Los objetos inmutables pueden ser más eficientes en términos de memoria y rendimiento. Como no pueden cambiar, es más fácil para Python optimizar su uso en memoria.

**Seguridad:** Los objetos inmutables son más seguros en situaciones donde no deseas que los datos se modifiquen accidentalmente o de manera no deseada. Esto puede ser importante en la programación concurrente o en la manipulación de datos críticos.

**Asignación y Copias:** Comprender la mutabilidad es fundamental al tratar con asignaciones y copias de objetos. En objetos mutables, si asignas un objeto a otra variable, ambas variables se referirán al mismo objeto en memoria. En objetos inmutables, se crea una nueva copia del objeto cuando asignas o modificas.

**Inmutabilidad de Cadenas:** Las cadenas inmutables son útiles para garantizar que los datos de texto no cambien accidentalmente, lo que es crucial en situaciones como manipulación de contraseñas o comparaciones constantes.

|  |
| --- |
| Objetos Mutables:  Los objetos mutables permiten cambios en su contenido después de la creación.  Cuando se modifica un objeto mutable, este se modifica en su lugar en la memoria, sin necesidad de crear un nuevo objeto.  Ejemplos de objetos mutables en Python incluyen listas, diccionarios y conjuntos.  Objetos Inmutables:  Los objetos inmutables no permiten cambios en su contenido después de la creación.  Cuando se intenta modificar un objeto inmutable, se crea un nuevo objeto con el valor modificado en lugar de cambiar el original.  Ejemplos de objetos inmutables en Python incluyen cadenas (strings), tuplas y números (enteros, flotantes, etc.). |

Variables y constantes:

En Python, una variable es un nombre que se asigna a un dato en memoria y se utiliza para referirse a ese valor en el programa. El valor puede ser un número, una cadena de texto, una lista o cualquier otro objeto en Python.

Python posee un puñado de constantes. Y no se pueden definir por el usuario.

Nombres de variables:

* Los nombres de variables en Python deben comenzar con una letra o un guion bajo.
* No pueden comenzar con un número o contener caracteres especiales, como @, !, o $.
* Además, los nombres de variables no pueden ser palabras reservadas en Python, como if, while, for, etc. ( ver max, min, etc)
* Los nombres de los objetos deben ser descriptivos, en minúsculas (salvo que se desee aclarar que el contenido es constante) y en snake case. Las clases van con las primer letra en mayúscula, en una mezcla de Pascal case y snake case

En este caso, x es una variable que se asigna al valor 10, y nombre es una variable que se asigna a la cadena de texto "Juan".

nombre = “Juan“ # se guarda en memoria Juan el interprete lo designa como string y se le da la etiqueta nombre

valor\_eje\_x=10 # se guarda en memoria 10 el interprete lo designa como int y se le da la etiqueta valor\_eje\_x

En Python, las variables son dinámicamente tipadas, lo que significa que el tipo de datos de la variable se determina automáticamente en tiempo de ejecución. Se ve claramente como se cambiaron los valores en la tercer celda de código.

Una constante, por otro lado, es un valor que no cambia durante la ejecución del programa. En Python, no existe una forma de declarar explícitamente una constante, pero se puede utilizar una convención de nomenclatura para indicar que un valor no debe cambiar.

Constantes incorporadas:

El programador no puede crear constantes como en otros lenguajes (“C” por ejemplo).

Ejemplos en C y Java

|  |
| --- |
| #define PI 3.14159  const int edad = 30;  Características de las constantes en C:  Las constantes definidas con #define son reemplazadas en tiempo de preprocesamiento.  Las constantes definidas con const son almacenadas en memoria.  final double PI = 3.14159;  Las constantes en Java son utilizadas para representar valores fijos y predefinidos que no cambian durante la ejecución del programa. |

Estos objetos simplemente no existen en Python

En python hay un pequeño número de constantes viven en el espacio de nombres incorporado.

Los nombres: **None, False, True y \_\_debug\_\_** no se pueden reasignar (asignaciones a ellos, incluso como un nombre de atributo, lanza SyntaxError ), por lo que pueden considerarse constantes «verdaderas».

**False:**El valor falso del tipo bool.

**True:** El valor verdadero del tipo bool.

Las asignaciones a False o True son ilegales y generan un SyntaxError.

**None:**  None is the sole instance of the NoneType type.

**NotImplemented:**  DeprecationWarning

**Ellipsis:** Lo mismo que la elipsis literal «...». Valor especial que se utiliza principalmente junto con la sintaxis de segmentación extendida para tipos de datos de contenedor definidos por el usuario.``Ellipsis`` es la única instancia del tipo types.EllipsisType.

**\_\_debug\_\_:** Esta constante es verdadera si Python no se inició con una opción -O. Vea también la instrucción assert.

Otras se pueden agregar mediante librerías

Constantes agregadas por el módulo site

El módulo site (que se importa automáticamente durante el inicio, excepto si se proporciona la opción -S en la línea de comandos) agrega varias constantes al espacio de nombres integrado. Son útiles para el intérprete interactivo y no deben usarse en programas.

quit(code=None)

exit(code=None)

Objetos que cuando se imprimen, muestra un mensaje como «Use quit() o Ctrl-D (i.e. EOF) to exit», y cuando se llama, lanza SystemExit con el código de salida especificado.

**PI = 3.1416 # no es lo mismo que math.py**

En este caso, PI es una variable que se utiliza como una constante, ya que se espera que su valor no cambie durante la ejecución del programa. Para que no quede como esperanza la comunidad tiene la regla que si el objeto es en mayúsculas – upper() se considera que no debe (pero si puede) ser modificada. Es una convención de la comunidad Python, una buena práctica de programación para hacer el código más legible y fácil de entender.

En notas al final encontraras una listas de constantes principalmente usadas en física

Casting:

El casting de datos, también conocido como conversión de tipos, se refiere a la acción de cambiar el tipo de objeto y valor de un datos a otro tipo y valor siempre que se cumplan ciertas reglas. En Python, puedes realizar conversiones de tipos utilizando funciones incorporadas que permiten cambiar entre tipos numéricos, cadenas y otros tipos de datos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | # Casting de **string a entero**  numero\_str = "10"  numero\_int = int(numero\_str)# casting directo de string a int  print(f"Resultado: {numero\_int}") |   Salida esperada por consola   |  | | --- | | Resultado: 10 |   ·  Código Python   |  | | --- | | # Casting de **punto flotante a entero**  numero\_float = 3.14  numero\_int = int(numero\_float)# casting directo de float a int  print(f"Resultado: {numero\_int}") |   Salida esperada por consola   |  | | --- | | Resultado: 3 |   ·  Código Python   |  | | --- | | # Casting de **booleano a entero**  booleano = True  numero\_int = int(booleano)# casting directo de bool a int  print(f"Resultado: {numero\_int}") |   Salida esperada por consola   |  | | --- | | Resultado: 1 |   · |

·

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ·Código Python   |  | | --- | | # Casting de **entero a punto flotante**  numero\_int = 10  numero\_float = float(numero\_int)# casting directo de int a float  print(f"Resultado: {numero\_float }") |   Salida esperada por consola   |  | | --- | | Resultado: 10.0 |   ·  Código Python   |  | | --- | | # Casting de **string a punto flotante**  numero\_str = "3.14"  numero\_float = float(numero\_str))# casting directo de string a float  print(f"Resultado: {numero\_float }") |   Salida esperada por consola   |  | | --- | | Resultado: 3.14 |   Código Python   |  | | --- | | # Casting de **booleano a punto flotante**  booleano = False  numero\_float = float(booleano))# casting directo de bool a float  print(f"Resultado: {numero\_float }") |   Salida esperada por consola   |  | | --- | | Resultado: 0.0 |   · |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código Python   |  | | --- | | # Casting de **entero a string**  numero\_int = 10  numero\_str = str(numero\_int))# casting directo de int a string  print(f"Resultado: {numero\_str }") |   Salida esperada por consola   |  | | --- | | Resultado: "10" |   ·  Código Python   |  | | --- | | # Casting de **booleano a string**  booleano = True  numero\_str = str(booleano))# casting directo de string a bool  print(f"Resultado: {numero\_ str }") |   Salida esperada por consola   |  | | --- | | Resultado: "True" |   · |

Recuerda que el casting solo es posible si la conversión tiene sentido semántico. Por ejemplo, no es posible convertir una cadena de caracteres que no representa un número válido a entero o punto flotante.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting de **string a binatio**  cadena = '1001'  entero = int(cadena, base=2)  print(f"Resultado: {entero }")  #1001 => 1\*10\*\*3 + 0\*10\*\*2 + 0\*10\*\*1 + 1\*10\*\*0 => 8 + 0 + 0 + 1 => # Ejercicio 60: Calcular la longitud de la hipotenusa de un triángulo rectángulo.  from math import sqrt  ab = float(input('Escriba el valor de la longitud del vértice AB: '))  bc = float(input('Escriba el valor de la longitud del vértice BC: '))  hipotenusa = sqrt(ab\*\*2 + bc\*\*2)  print('La longitud de la hipotenusa es: {}'.format(hipotenusa)) 9 |   Salida esperada por consola   |  | | --- | | Resultado: 9 |   · |

.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting de **string(float) a int**  numero\_str = "3.14"  numero\_int = int(numero\_str))# casting directo de string con . decimal a int  print(f"Resultado: {numero\_int}") |   Salida esperada por consola   |  | | --- | | Traceback (most recent call last):  File "ejercicios.py", line xx, in <module>  numero\_int = int(numero\_str)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ValueError: invalid literal for int() with base 10: '3.14' |   · |

·

|  |  |  |
| --- | --- | --- |
| **divmod**() es una función, que devuelve el cociente y el resto al dividir el número **“A”** por el número **“B”** . (no puede ser un número complejo.)  El valor de retorno será el par de números positivos que consiste en el cociente y el resto obtenido al dividir **“A”** por **“B”**. En el caso de tipos de operandos mixtos, se aplicarán las reglas para los operadores aritméticos binarios.  Para los argumentos de números enteros, el valor de retorno será el mismo que (**“A”** // **“B”**, **“A”** % **“B”**.).  Código Python   |  | | --- | | tupla\_salida=**divmod**(5,2)  print(f"{tupla\_salida=}")  print ("\*"\*50)  tupla\_salida= **divmod**(13.5,2.5)  print(f"{tupla\_salida=}")  print ("\*"\*50) |   Salida esperada por consola   |  | | --- | | tupla\_salida=(2, 1)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  tupla\_salida=(5.0, 1.0)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting de negativo a valor **absoluto**  numero = -3.14159  absoluto = **abs**(numero)# casting directo de entero a binario  print(f"Resultado: {absoluto }") |   Salida esperada por consola   |  | | --- | | Resultado: 3.14159 |   · |

·

|  |  |
| --- | --- |
| Código Python   |  | | --- | | # Casting de **entero** que deseas convertir a formato **binario**  numero = 10  numero\_binario = **bin**(numero)# casting directo de entero a binario  print(f"Resultado: {numero\_binario } con prefijo")  print(f"Resultado: {numero\_binario[2:]} sin priefijo") |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting de **entero** que deseas convertir a formato **binario**  numero = 10  numero\_binario = **bin**(numero)# casting directo de entero a binario  print(f"Resultado: {numero\_binario } con prefijo")  print(f"Resultado: {numero\_binario[2:]} sin priefijo") |   Salida esperada por consola   |  | | --- | | Resultado: 0b1010 con prefijo  Resultado: 1010 sin priefijo |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting a un número **complejo** a partir de dos enteros (parte real y una parte imaginaria).  # Crear un número complejo con parte real e imaginaria  complejo1 = complex(3, 4)  # Crear un número complejo solo con parte real (imaginaria es 0)  complejo2 = complex(2)  # Crear un número complejo con parte imaginaria (real es 0)  complejo3 = complex(0, 1)  # Crear un número complejo desde una cadena  string = "2+3j"  complejo4 = complex(string)  print(f"{complejo1=}")  print(f"{complejo2=}")  print(f"{complejo3=}")  print(f"{complejo4=}") |   Salida esperada por consola   |  | | --- | | complejo1=(3+4j)  complejo2=(2+0j)  complejo3=1j  complejo4=(2+3j) |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | # Casting de entero a valor **hexadecimal** (base 16)  numero = 255  hexadecimal = **hex**(numero)# casting directo de entero a hexadecimal  print(f"Resultado: { hexadecimal }") |   Salida esperada por consola   |  | | --- | | Resultado: 0xff |   · |

·

|  |  |  |
| --- | --- | --- |
| ·Código Python   |  | | --- | | # Casting de entero a valor **octal** (0 a7)  numero = 15  octal = **oct**(numero)# casting directo de entero a octal  print(f"Resultado: {octal }") |   Salida esperada por consola   |  | | --- | | Resultado: 0o17 |   · |

·

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ·La función ascii() en Python devuelve una cadena que contiene una representación legible de los caracteres ASCII en la cadena especificada. Los caracteres no ASCII se representan utilizando la secuencia \x, \u o \U.  Código Python   |  | | --- | | # Casting de **unicode** a formato **ASCII**  texto = """¡¡¡Python es genial!!!  y es re facil"""  salida\_ascii = **ascii**(texto)# casting unicode a ASCII  print(f"Resultado: {salida\_ascii }") |   Salida esperada por consola   |  | | --- | | Resultado: '\xa1\xa1\xa1Python es genial!!!\n y es re facil' |   Código Python   |  | | --- | | # Casting de **caracter** a formato valor tabla **ASCII**  caracter = "A"  salida\_ascii = **ord**(caracter)# casting caracter a valor tabla ASCII  print(f"Resultado valor tabla ASCII: {salida\_ascii }") |   Salida esperada por consola   |  | | --- | | Resultado valor tabla ASCII: 65 |   Código Python   |  | | --- | | # Casting de **ASCII** a formato **caracter**  entrada\_valor\_ascii = 65  salida\_caracter = **chr**(entrada\_valor\_ascii)# casting ASCII a caracter  print(f"Resultado caracter: {salida\_caracter }") |   Salida esperada por consola   |  | | --- | | Resultado caracter: A |   · |

·

|  |  |  |
| --- | --- | --- |
| ·Colecciones a tuplas (**tuple**)  a conjuntos (**set**)  a ¿conjuntos congelado? (**frozenset**)  a listas (**list**)  a diccionarios (**dict**)  Código Python   |  | | --- | | objeto = [1,5,9,4,1,3,7,4,6,8,7,1,3,2,9,1,0]  print(f"objeto: {objeto}\n\t\t{type(objeto)}")  print ("\*"\*50) #---------------------------------------------------------------------  tupla = tuple(objeto)  print(f"objeto: {tupla}\n\t\t{type(tupla)}")  print ("\*"\*50) #---------------------------------------------------------------------  conjunto = set(objeto)  print(f"objeto: {conjunto}\n\t\t{type(conjunto)}")  print ("\*"\*50) #---------------------------------------------------------------------  conjunto\_congelado = frozenset(objeto)  print(f"objeto: {conjunto\_congelado}\n\t\t{type(conjunto\_congelado)}")  print ("\*"\*50) #---------------------------------------------------------------------  lista = list(conjunto)  print(f"objeto: {lista }\n\t\t{type(lista)}")  print ("\*"\*50) #---------------------------------------------------------------------  dic=dict.fromkeys(lista,"valor a cargar")  print(f"objeto: {dic }\n\t\t{type(dic)}")  print ("\*"\*50) #--------------------------------------------------------------------- |   Salida esperada por consola   |  | | --- | | objeto: [1, 5, 9, 4, 1, 3, 7, 4, 6, 8, 7, 1, 3, 2, 9, 1, 0]  <class 'list'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto: (1, 5, 9, 4, 1, 3, 7, 4, 6, 8, 7, 1, 3, 2, 9, 1, 0)  <class 'tuple'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}  <class 'set'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto: frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})  <class 'frozenset'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  <class 'list'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  objeto: {0: 'valor a cargar', 1: 'valor a cargar', 2: 'valor a cargar', 3: 'valor a cargar', 4: 'valor a cargar', 5: 'valor a cargar', 6: 'valor a cargar', 7: 'valor a cargar', 8: 'valor a cargar', 9: 'valor a cargar'}  <class 'dict'>  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

·

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ·bytearray es un tipo de dato mutable que representa una secuencia de bytes.  Puedes crear objetos bytearray para almacenar y manipular datos en forma de bytes.  Los objetos bytearray son similares a las listas, pero en lugar de almacenar elementos genéricos, almacenan bytes.  Creación de un bytearray:  Puedes crear un objeto bytearray de varias formas, como pasar una cadena de bytes, una lista de enteros o incluso especificar su longitud.  Código Python   |  | | --- | | # Crear un bytearray a partir de una cadena de bytes  byte\_array\_1 = bytearray(b'hello')  # Crear un bytearray a partir de una lista de enteros  byte\_array\_2 = bytearray([65, 66, 67, 68, 69])  # Crear un bytearray con una longitud específica  byte\_array\_3 = bytearray(10) # Crea un bytearray de 10 bytes con valores iniciales en 0 |   Acceso y modificación de elementos:  Puedes acceder a los elementos individuales de un bytearray utilizando índices y también modificarlos, ya que los objetos bytearray son mutables:  Código Python   |  | | --- | | byte\_array = bytearray(b'hello')  print(byte\_array[0]) # Imprime el valor del primer byte (104 en ASCII)  byte\_array[1] = 111 # Modifica el segundo byte a 111 ('o' en ASCII) |   Métodos útiles:  Los objetos bytearray tienen varios métodos que puedes utilizar para realizar diferentes operaciones, como concatenación, búsqueda y reemplazo:  Código Python   |  | | --- | | byte\_array = bytearray(b'hello')  byte\_array.append(100) # Agrega un byte al final  byte\_array.extend([101, 102]) # Extiende con varios bytes  index = byte\_array.index(108) # Encuentra el índice del byte 'l'  byte\_array.insert(index, 109) # Inserta el byte 'm' en el índice encontrado  byte\_array.remove(101) # Elimina la primera aparición del byte 101  byte\_array.replace(b'h', b'H') # Reemplaza todos los bytes 'h' con 'H' |   Conversión a otros tipos:  Puedes convertir un bytearray en una cadena de bytes (bytes) o en una cadena de caracteres (str) utilizando los métodos bytes() y decode() respectivamente:  Código Python   |  | | --- | | byte\_array = bytearray(b'hello')  byte\_string = bytes(byte\_array) # Convierte en una cadena de bytes  char\_string = byte\_array.decode('utf-8') # Decodifica a una cadena de caracteres |   Recuerda que bytearray es útil cuando necesitas manipular datos en forma de bytes de manera mutable. Si solo necesitas una secuencia inmutable de bytes, es mejor utilizar el tipo bytes.   |  | | --- | |  |   · |

**Operadores aritméticos.**

|  |  |
| --- | --- |
| operandos (objetos numéricos) | |
| + Suma | adicióna dos operandos. |
| - Resta | sustracción al valor del operando de la izquierda  - el valor del de la derecha.  Cambia el signo sobre un único operador. |
| \* | Multiplicación de dos operandos. |
| / | Divide el operando de la izquierda por el de la derecha.  La salida siempre es un float (genera un casting) |
| // | División entera se obtiene el cociente 'entero' de dividir el operando  de la izquierda por el de la derecha. |
| % | Módulo es el residuo que se obtiene el resto de dividir el entero del  operando de la izquierda por el de la derecha. |
| \*\* n | Potencia - Exponenciación eleva el operando de la izquierda a la  potencia del operador del de la derecha. |
| \*\* (1/n) | Radicación eleva el operando de la izquierda a la potencia de 1 (uno)  dividido el valor operador del de la derecha. |

Ejemplos :﻿﻿

|  |
| --- |
| t=[["","Addición- suma","x + y"],  ["-","Subtracción -resta","x - "],  ["\*","Multiplicación","x \* y"],  ["/","División","x / y"],  ["//","división entera","x // y"],  ["% ","Modulo resto"," x % y"],  ["\*\*","Exponenciación","x \*\* y"],  ["\*\*(1/n)","Radicación","x \*\*(1/y)"]]  print (tabulate(t))  a=8  print ("a=8")  print(f"el valor de {a=}")  #------------------------------  a=a+12  print ("a=a+12")  print(f"ahora el valor de {a=} Adicción- suma ")  #------------------------------  a=a-5  print ("a=a-5")  print(f"ahora el valor de {a=} Sustracción -resta")  #------------------------------  a=a\*2  print ("a=a\*2")  print(f"ahora el valor de {a=} Multiplicación ")  #------------------------------  a=a/3  print ("a=a/3")  print(f"ahora el valor de {a=} División")  #------------------------------  a=a%3  print ("a=a%3")  print(f"ahora el valor de {a=} Modulo resto")  #------------------------------  a=10  print ("a=10")  print(f"el valor de {a=}")  #------------------------------  a=a//3  print ("a=a//3")  print(f"ahora el valor de {a=} división entera")  #------------------------------  a=a\*12  print ("a=a\*12")  print(f"ahora el valor de {a=} Exponenciación ")  #------------------------------  a=a\*\*(1/2)  print ("a=a\*\*(1/2)")  print(f"ahora el valor de {a=} Radicación") |

La Radicación o Raiz.

Podemos considerar la radicación como un caso particular de la potenciación. En efecto, la raíz cuadrada de un numero (por ejemplo a) es igual que𝑎 elevado al 1/2, del mismo modo la raíz cúbica de a es a elevado al 1/3 y en general, la raíz enésima de un numero a es a elevado al 1/n.

****Nota:**** Aún cuando no afecta a la sintaxis, el uso de espacios entre los operadores aritméticos mejora la comprensión de las operaciones.

**Reglas de precedencia en operaciones aritméticas.**

Los operadores se apegan a la siguiente regla de precedencia siguiendo una secuencia de izquierda a derecha.

En Python, el orden de ejecución en una operación matemática sigue las reglas convencionales de la aritmética, siguiendo el concepto de "**PEMDAS**" (Paréntesis, Exponentes, Multiplicación y División, Adición y Substracción). Esto se refiere al orden en el que se evalúan las partes de una expresión matemática.

A continuación, desgloso cada parte del acrónimo "**PEMDAS**":

1. Paréntesis: Las operaciones dentro de paréntesis se evalúan primero. Si hay múltiples niveles de paréntesis anidados, se resuelven de adentro hacia afuera.

2. Exponentes: Las operaciones de exponentes se evalúan después de los paréntesis. Esto incluye operaciones como elevar un número a una potencia.

3. Multiplicación y División: Las multiplicaciones y divisiones se evalúan después de los exponentes. Se ejecutan de izquierda a derecha, en el orden en que aparecen en la expresión.

4. Adición y Substracción: Finalmente, las operaciones de suma /adición y substracción/ resta se evalúan después de las multiplicaciones y divisiones. Al igual que en el caso anterior, se ejecutan de izquierda a derecha, en el orden en que aparecen en la expresión.

Ejemplos:

Como se observa Python soporta números complejos y la parte imaginaria se representa con j ( en matemáticas se utiliza la letra i ).

Operadores de relación.

Los operadores de relación evalúan si dos valores/objetos cumplen con una condición específica. El resultado de esta evaluación es un objeto de tipo ****bool****.

|  |
| --- |
| == condición Igual a == b False  != condición Diferente a != b True  > mayor que a > b True  >= mayor igual que a >= b True  < menor que a < b False  <= menor igual que a <= b False |

·

|  |
| --- |
| t=[["==","Igual","x == y"],  ["!=","No igual / diferente","x != y"],  [">","mayor que","x > y"],  ["<","menor que","x < y"],  [">=","mayor igual que","x >= y"],  ["<=","menor igual que","x <= y"]]  print (tabulate(t))  a=8  b=9  c=10  d=5+5  print (f"{(a>b)=}")  print (f"{(a<b)=}")  print (f"{(a>=b)=}")  print (f"{(a<=b)=}")  print (f"{(a==b)=}")  print (f"{(a!=b)=}")  print (f"{((a<=b) and (c<=d))=}")  print (f"{((a>=b) and (c>=d))=}")  print (f"{((a<=b) or (c<=d))=}")  print (f"{((a>=b) or (c>=d))=}") |

,

**Operadores lógicos.**

|  |  |  |
| --- | --- | --- |
| **álgebra de Boole** | | |
| not | not True = False  not False = True | niegan la condición |
| and | True and True=True  True and False= False  False and True= False  False and False= False | True si todos los segmentos de la condición son True  False si al menos un segmento es False. |
| or | True or True=True  True or False= False  False or True= False  False or False= False | True si al menos un segmentos de la condición es True.  False si ninguno segmento es True. |

Estos operadores permiten la realización de las operaciones lógicas. Por lo general se realizan con objetos de tipo bool, pero Python también permite operaciones lógicas con otros tipos de datos.

En Python, los operadores lógicos tienen la siguiente precedencia (de mayor a menor):

1. not
2. and
3. or

Esto significa que el operador not se evalúa antes que and, y and se evalúa antes que or. Sin embargo, es recomendable usar paréntesis para evitar confusiones en expresiones complejas.

Por ejemplo, en la expresión not A and B or C, primero se evalúa not A, luego and, y finalmente or. Si queremos evaluar A and B antes de not, debemos usar paréntesis de la siguiente manera: not (A and B) or C.

|  |
| --- |
| #Operadores lógicos de Python (sentencias condiciónales):  x = 5  y = 5  and x > 0 and y < 18 # devuelve True ambos segmentos de la condición son True  or x < 0 or y < 18 # devuelve True alguno de los segmentos de la condición es True  not not(x < 0 or y > 18) # devuelve el booleano inverso a la condición, si es True devuelve False  y viceversa""") |

**Operadores de asignación.**

|  |  |  |
| --- | --- | --- |
| = asignación de int  = asignación de float  = asignación de str | a = 8  a = 3.14159  a = "Hola" | salida  real  booleana |
| + Suma  - Resta  \* Multiplicación  / División regular  // División entera  % Módulo  \*\* Exponenciación  \*\* 1 /radicación | a += b  a -= b  a \*= b  a /= b a  a //= b  a %= b  a = a \*\* b  a = a \*\* (1/b) | a = a + b  a = a - b  a = a \* b  a = a / b  a = a // b  a = a % b  a = a \*\* b  a = a \*\* (1/b) |
| walrus := Permite asignar un valor a una variable como parte de una expresión  while (user\_input := input("Ingresa un número (o 'salir' para terminar): ")) != 'salir':  print(f"Ingresaste: {user\_input}") | | |

Ejemplo :

Operadores de identidad.

·

|  |
| --- |
| Operadores de identidad  x=5  y=5  z=6  x is y devuelve True si el valor del primer objeto es el mismo que es segundo ver id  y not is z devuelve el booleano inverso a la condición, si es True devuelve False y viceversa""") |

**Bloques**

**Indentación de bloques de código.**

En Python la indentación forma parte de la sintaxis.

La indentación se utiliza para delimitar bloques de código agrupadas juntas dentro de un contexto (condicional, ciclo, función, etc.) que se ejecuta en un contexto particular- Ambitos. sin necesidad de utilizar caracteres delimitadores como ocurre en otros lenguajes de programación.

Por convención se utilizan cuatro espacios para indentar en vez de tabuladores. Para mayor referencia ir al PEP8.

Los bloques son una parte fundamental de la estructura del lenguaje y se utilizan para definir la estructura de control de flujo y la organización del código.

Condicionales If.

En los primeros años del cómputo, los programas consistían una secuencia lineal de instrucciones que eran ejecutadas por una máquina, tal como lo hace una pianola o una vieja caja de música. Sin embargo conforme esta disciplina se fue refinando, se volvió imperativo que el flujo de un programa de cómputo fuese lineal, sino que a partir de una serie de decisiones se ejecutaran diferentes bloques de código.

Mediante las estructuras derivadas de if, Python puede evaluar ciertas expresiones lógicas que resultarían finalmente en un valor booleano True o False , el cual ejecutaría el código correspondiente.

**Modificación de flujo condicional if**

|  |
| --- |
| Hasta ahora el flujo del programa fue de arriba hacia abajo y de izquierda a derecha. En occidente eso es muy normal. Pero hay lenguajes como árabes, hebreo, etc que se escriben de derecha a izquierda. Incluso hay ejemplos de poesía oriental incluso manga done se escribe el columnas de abajo hacia arriba. |

·

|  |  |  |
| --- | --- | --- |
| En IDE | comentarios | Salida esperada por consola |
| print (0)  print (1)  print (2)  print (3)  print (4)  print (5)  print (6)  print (7)  print (8)  print (9)  print (10) | El flujo del programa  es de arriba hacia abajo | 0  1  2  3  4  5  6  7  8  9  10 |

if else elif

* anidación
* lógica booleana

El flujo condicional "if" en el lenguaje Python se utiliza para tomar decisiones basadas en una condición (única o múltiple que veremos luego). La salida de la condición es una booleana, solo puede ser True o False.

Python utiliza la estructura condicional if-else-elif para tomar decisiones basadas en condiciones. Esta estructura te permite ejecutar diferentes bloques de código según el resultado de una o varias condiciones.

Puede modifica

**If-else:** Permite especificar una acción alternativa en caso de que la condición del "if" no se cumpla. El bloque de código dentro del "else" se ejecutará si la condición del "if" es falsa.

**If-elif-else:** Se puede encadenar varios bloques "elif" después de un "if" para evaluar múltiples condiciones en secuencia. El primer bloque "elif" cuya condición sea verdadera se ejecutará, o si ninguna condición es verdadera, se ejecutará el bloque "else" final.

**La sintaxis básica es la siguiente:**

|  |
| --- |
| if (condición):  # Código que se ejecuta si la condición es verdadera - True  else:  # Código que se ejecuta si la condición es falsa - False |

La condición debe ser una expresión que se evalúa como verdadera o falsa. Si la condición es verdadera, se ejecutará el código dentro del bloque de código del if. Si la condición es falsa, se ejecutará el código dentro del bloque de código del else.

También se puede utilizar una serie de estructuras condicionales elif para evaluar múltiples condiciones. La sintaxis sería la siguiente:

|  |
| --- |
| if (condición1):  # Código que se ejecuta si la condición1 es verdadera  elif (condición2):  # Código que se ejecuta si la condición2 es verdadera y la condición1 es falsa  elif (condición3):  # Código que se ejecuta si la condición3 es verdadera y las condiciones 1 y 2 son falsas  else:  # Código que se ejecuta si todas las condiciones anteriores son falsas |

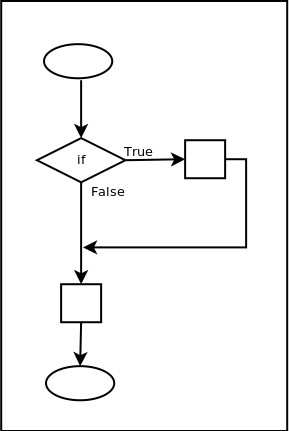
En este caso, se evalúa cada condición en orden. Si alguna de las condiciones es verdadera, se ejecutará el código dentro del bloque de código correspondiente y se saldrá de la estructura condicional. Si todas las condiciones son falsas, se ejecutará el código dentro del bloque de código del else final.

Cabe destacar que la estructura condicional if-else puede anidarse para crear decisiones más complejas en el código.

Estructura if simple.

La palabra clave ****if**** siempre evalúa una expresión lógica y en caso de que dicha expresión de por resultado el valor ****True****, se ejecutará el código indentado justo por debajo del ****if****. En caso de que la declaración resulte en el valor ****False****, el intérprete ignorará el bloque de código indentado y éste continuará con la instrucción siguiente inmediata a la indentación.

|  |
| --- |
| <flujo principal>  ...  if <expresión lógica>:  <bloque inscrito a if>  <flujoprincipal> |

A continuación se muestra un diagrama de flujo que ejemplifica al uso del condicional if en su modo más simple:

Ejemplo :

El modulo 2 de 0 es 0 y cero es neutro, ni par ni impar, ni positivo ni negativo

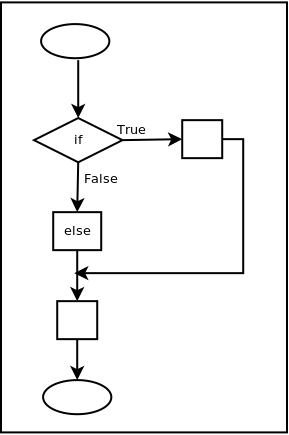
Acá se ve un ejemplo de un if simple y a su vez vemos el uso de la función input la cual le pide un dato al usuario, ese dato siempre es un string (cadena de caracteres), y como necesitamos evaluarlo en el if como si fuera un entero lo convertimos con int(). Una vez obtenido, el if verifica la condición y si es True se ejecuta todo el bloque de código que esta identado dentro del if. Y si da False se ignora y continua con el flujo del programa.

Estructura if con else.

Si el resultado de la expresión lógica evaluada por if da por resultado False, se puede utilizar else para ejecutar el bloque de código indentado debajo de esta expresión.

|  |
| --- |
| <flujo principal>  if <expresión lógica> :  <bloque inscrito al if>  else:  <bloque inscrito al else>  <flujo principal> |

A continuación se muestra un diagrama de flujo que ejemplifica al uso del condicional if con else :



ejemplo :

|  |  |  |  |
| --- | --- | --- | --- |
| Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 2  nota\_3er\_p = 7  nota\_4to\_p = 6  suma = nota\_1er\_p + nota\_2do\_p + nota\_3er\_p + nota\_4to\_p  media= suma/4  print (f"la media de las notas es {media}")  if (media <7):# desde -infinito a 6.999  print (f"no alcanzo el mínimo de 7 para aprobar el curso")  else:# todo valor que no haya entrado en la condición if, media debe ser mayor o igual a 7  print (f"curso aprobado") |   Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 7  nota\_4to\_p = 6 |   Salida esperada por consola   |  | | --- | | la media de las notas es 7.50 -  curso aprobado |   · |

ver mayúsculas y minúsculas (métodos de strings)

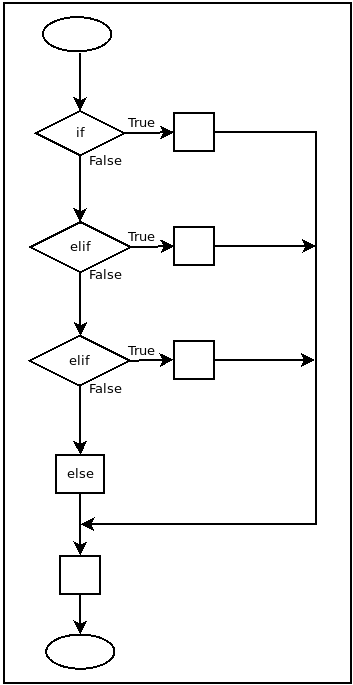
Estructura if...elif...else.

Es posible evaluar más de una expresión lógica mediante el uso de elif. En el caso de que exista más de una expresión lógica que de por resultado True, Python ejecutará solamente el código delimitado por la primera que ocurra.

En caso de que ninguna de las condiciones de por resultado True se puede utilizar else al final de la estructura.

|  |
| --- |
| <flujo principal>  if <expresión lógica> :  <bloque inscrito al if>  elif <expresión lógica> :  <bloque inscrito al elif>  elif <expresión lógica> :  <bloque inscrito al elif>  ...  else:  <bloque inscrito al else>  <flujo principal> |

A continuación se muestra un diagrama de flujo que ejemplifica al uso del condicional if con elif y else :



ejemplo :

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 2#<---------------------------------modificado  nota\_3er\_p = 7  nota\_4to\_p = 6 |   Salida esperada por consola   |  | | --- | | la media de las notas es 6.00 -  no alcanzo el mínimo de 7 para aprobar el curso |   · |

·

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 7  nota\_4to\_p = 6  suma = nota\_1er\_p + nota\_2do\_p + nota\_3er\_p + nota\_4to\_p  media= suma/4  print (f"la media de las notas es {media}")  if (media <7):# desde -infinito a 6.999  print (f"no alcanzo el mínimo de 7 para aprobar el curso")  elif (media <9):# todo valor que no haya entrado en la condición if,  # media debe ser mayor o igual a 7 pero debe ser menor a 9  print (f"curso aprobado")  else: # todo valor que no haya entrado en la condición if,  # media debe ser mayor o igual a a 9  print (f" felicitaciones. Al cuadro de honor") |   Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 7  nota\_4to\_p = 6 |   Salida esperada por consola   |  | | --- | | la media de las notas es 7.50 -  curso aprobado |   Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 2#<---------------------------------modificado  nota\_3er\_p = 7  nota\_4to\_p = 6 |   Salida esperada por consola   |  | | --- | | la media de las notas es 6.00 -  no alcanzo el mínimo de 7 para aprobar el curso |   Código Python   |  | | --- | | nota\_1er\_p = 10#<---------------------------------modificado  nota\_2do\_p = 9#<----------------------------------modificado  nota\_3er\_p = 10#<---------------------------------modificado  nota\_4to\_p = 9#<----------------------------------modificado |   Salida esperada con   |  | | --- | | la media de las notas es 9.50 -  felicitaciones. Al cuadro de honor |   . |

·

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Código Python   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 2  nota\_4to\_p = 10  suma = nota\_1er\_p + nota\_2do\_p + nota\_3er\_p + nota\_4to\_p  media= suma/4  print (f"la media de las notas es {media}")  if (media <7):# desde -infinito a 6.999  print (f"no alcanzo el mínimo de 7 para aprobar el curso")  elif (nota\_1er\_p>=4) and (nota\_2do\_p>=4) and (nota\_3er\_p>=4) and (nota\_4to\_p>=4) :  # todo valor que no haya entrado en la condición  # media mayor a 7  # todos los bimestres aprobados  print (f"curso aprobado")  else:  # media mayor a 7  # al menos un bimestre menos a 4  print (f"tenes que recuperar un bimestre") |   con   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 7  nota\_4to\_p = 6 |   Salida esperada por consola   |  | | --- | | la media de las notas es 7.50 -  curso aprobado |   con   |  | | --- | | nota\_1er\_p = 9  nota\_2do\_p = 8  nota\_3er\_p = 2  nota\_4to\_p = 10 |   Salida esperada por consola   |  | | --- | | la media de las notas es 7.25 -  tenes que recuperar un bimestre |   . |

Estructura de coincidencia match case.

A partir de Python 3.10, se ha añadido una nueva característica llamada match case, que es una nueva forma de expresión de control de flujo. La sintaxis de la expresión match case se asemeja a la de la sentencia switch de otros lenguajes de programación.

La expresión match case se utiliza para comparar un valor con varias opciones y ejecutar un bloque de código correspondiente a la opción coincidente. Es similar a la sentencia if-elif-else, pero con una sintaxis más simple y expresiva.

La sintaxis básica de match case es la siguiente

|  |
| --- |
| match <valor>:  case <opcion1>:  <Bloque de código para la opcion1>  case opcion2:  <Bloque de código para la opcion2>  ...  case opcionN:  <Bloque de código para la opcionN > <<  case \_:  <Bloque de código para cualquier otra opción> |

ejemplo:

La sintaxis básica es la siguiente:

|  |
| --- |
| match (objeto):  case 'A':  print (f" opción abrir")  case 'B':  print (f" opción borrar")  case 'C':  print (f" opción copiar")  case 'S':  print (f" adios")  case other:  print (f" opción no valida") |

La estructura match case en lenguaje Python se utiliza para realizar selecciones múltiples basadas en el valor de una expresión.

Aquí tienes un ejemplo básico de cómo usar match case:

En este ejemplo, se le pide al usuario que seleccione una opción del 1 al 3. Luego, se utiliza match (optar) para evaluar el valor de optar y ejecutar el bloque de código correspondiente al caso coincidente.

|  |  |  |  |
| --- | --- | --- | --- |
| Código Python   |  | | --- | | optar = input("Ingresa una opción:  1) para Abrir  2) para Borrar  3) para Copiar  4) para Salir")  match (int(optar)):  case 1:  print (f" opción abrir")  case 2:  print (f" opción borrar")  case 3:  print (f" opción copiar")  case 4:  print (f" adios")  case other:  print (f" opción no valida") |   ·  Case en Python permite or   |  | | --- | | match entero:  case (1) | (2) | (3): |   Case en python permite colecciones   |  | | --- | | match string:  case (["A", "B", "C", "D","E","F" ]): |   · |

Bucles:

Otras Estructuras de control de flujo son los bucles. En Python existen dos tipos de bucles :

Bucles While:

Un bucle while es una estructura de control de flujo en Python que se utiliza para repetir una serie de instrucciones mientras se cumple una determinada condición.

La sintaxis del bucle while es la siguiente:

|  |
| --- |
| <flujo de programa>  ...  while <condición>:  <Bloque de código>  <continua flujo del programa> |

Ejemplo de bucle while simple:

En este ejemplo el while comprueba que el valor de la variable sea mayor igual a 0, si esto se cumple imprime el valor y reduce el mismo para poder salir en algún momento del bucle, si no se introduce ese conteo el bucle pasa a hacer un bucle infinito bloqueando el programa hasta que el usuario lo cierre a la fuerza.

Condiciones y Bucles

La sintaxis básica es la siguiente:

|  |
| --- |
| while (condición):  # Código que se ejecuta **MIENTRAS** la condición es verdadera - True  Código que se ejecuta cuando la condición pasa a ser falsa - False |

En Python, la función while se utiliza para crear bucles o ciclos que se ejecutan mientras una determinada condición sea verdadera. El bloque de código dentro del while se repetirá continuamente hasta que la condición se evalúe como falsa.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | contador = 0  while contador < 5:  print(f"El contador es: {contador}")  contador =contador + 1  print("Adios...") |   Salida esperada por consola   |  | | --- | | El contador es: 0  El contador es: 1  El contador es: 2  El contador es: 3  El contador es: 4  Adios... |   · |

En el ejemplo anterior, creamos una variable contador inicializada en 0. Luego, utilizamos la función while para repetir un bloque de código mientras la condición contador < 5 sea verdadera.

Dentro del bucle while, imprimimos el valor actual del contador y luego incrementamos su valor en 1 utilizando contador = contador + 1.

Esto asegura que el bucle eventualmente terminará cuando la condición se evalúe como falsa.

**Modificación de ejecución en un bucle while.**

En ciertas circunstancias es necesario interrumpir el flujo lógico de un programa. Python cuenta con los siguientes recursos para hacerlo.

**Break:**

Interrupciones de ejecución de un bloque while.

La palabra reservada break termina prematuramente la ejecución del bloque de código en el que se encuentra y restablece el flujo de ejecución al bloque de código que lo precede.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numero =0  while True :# bucle infinito……. while True is True  numero += 1:  print (f"el valor de {numero =}")  if numero == 6:  print("break")  break |   Salida esperada por consola   |  | | --- | | el valor de numero =1  el valor de numero =2  el valor de numero =3  el valor de numero =4  el valor de numero =5  el valor de numero =6  break |   · |

·

**break:**- En el siguiente trabajamos a la inversa

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | contador = 15  while contador !=0:  print(f"El contador es: {contador}")  contador = contador - 1  if contador == 5:  print("break")  break  print("Adios...") |   Salida esperada por consola   |  | | --- | | El contador es: 15  El contador es: 14  El contador es: 13  El contador es: 12  El contador es: 11  El contador es: 10  El contador es: 9  El contador es: 8  El contador es: 7  El contador es: 6  break  Adios... |   · |

.

Continue

La palabra reservada continue termina de forma prematura la ejecución de un bloque dentro de un ciclo y vuelve al inicio del bucle.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numero =0  while numero<=10:  numero += 1:  print (f"el valor de {numero =}")  if numero%2 == 0:  print("\t\tcontinue x par")  continue  print("\timpar") |   Salida esperada por consola   |  | | --- | | el valor de numero =1  impar  el valor de numero =2  continue x par  el valor de numero =3  impar  el valor de numero =4  continue x par  el valor de numero =5  impar  el valor de numero =6  continue x par  el valor de numero =7  impar  el valor de numero =8  continue x par  el valor de numero =9  impar  el valor de numero =10  continue x par |   · |

·

Continue:

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | contador = 15  while contador !=0:  print(f"El contador es: {contador}")  contador = contador - 1  if contador%2 == 0:  print("\t\tcontinue x par")  continue  print("\timpar")  print("Adios...") |   Salida esperada por consola   |  | | --- | | El contador es: 15  continue x par  El contador es: 14  impar  El contador es: 13  continue x par  El contador es: 12  impar  El contador es: 11  continue x par  El contador es: 10  impar  El contador es: 9  continue x par  El contador es: 8  impar  El contador es: 7  continue x par  El contador es: 6  impar  El contador es: 5  continue x par  El contador es: 4  impar  El contador es: 3  continue x par  El contador es: 2  impar  El contador es: 1 |   · |

Anidación:

Código Python

En esta modificación del código anterior incluye un break que rompe el bucle si ciclo supero un valor que imponemos antes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código Python   |  | | --- | | valor=""  while True:  while not valor.isdigit() or int(valor) <1 or int(valor) >99:  valor = input ("Ingrese in valor entre 1 y 99 :")  print("ok")  valor = int (valor)  break |   Salida esperada por consola   |  |  |  | | --- | --- | --- | | Ingrese in valor entre 1 y 99 :100  Ingrese in valor entre 1 y 99 :0   |  |  | | --- | --- | | Ingrese in valor entre 1 y 99 : | 5 |   ok |   · |

·

|  |  |
| --- | --- |
| Código Python   |  | | --- | | optar = ""  while optar < 5:  optar = input("Ingresa una opción:  1) para Abrir  2) para Borrar  3) para Copiar  4) para Salir")  match (int(optar)):  case 1:  print (f" opción abrir")  case 2:  print (f" opción borrar")  case 3:  print (f" opción copiar")  case 4:  print (f" adios")  case other:  print (f" opción no valida") |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | externo=0  while externo <10:  interno=0  while interno <10:  print (f"{externo} - {interno}", end=" | ")  interno+=1  print()  externo+=1 |   Salida esperada por consola   |  | | --- | | 0 - 0 | 0 - 1 | 0 - 2 | 0 - 3 | 0 - 4 | 0 - 5 | 0 - 6 | 0 - 7 | 0 - 8 | 0 - 9 |  1 - 0 | 1 - 1 | 1 - 2 | 1 - 3 | 1 - 4 | 1 - 5 | 1 - 6 | 1 - 7 | 1 - 8 | 1 - 9 |  2 - 0 | 2 - 1 | 2 - 2 | 2 - 3 | 2 - 4 | 2 - 5 | 2 - 6 | 2 - 7 | 2 - 8 | 2 - 9 |  3 - 0 | 3 - 1 | 3 - 2 | 3 - 3 | 3 - 4 | 3 - 5 | 3 - 6 | 3 - 7 | 3 - 8 | 3 - 9 |  4 - 0 | 4 - 1 | 4 - 2 | 4 - 3 | 4 - 4 | 4 - 5 | 4 - 6 | 4 - 7 | 4 - 8 | 4 - 9 |  5 - 0 | 5 - 1 | 5 - 2 | 5 - 3 | 5 - 4 | 5 - 5 | 5 - 6 | 5 - 7 | 5 - 8 | 5 - 9 |  6 - 0 | 6 - 1 | 6 - 2 | 6 - 3 | 6 - 4 | 6 - 5 | 6 - 6 | 6 - 7 | 6 - 8 | 6 - 9 |  7 - 0 | 7 - 1 | 7 - 2 | 7 - 3 | 7 - 4 | 7 - 5 | 7 - 6 | 7 - 7 | 7 - 8 | 7 - 9 |  8 - 0 | 8 - 1 | 8 - 2 | 8 - 3 | 8 - 4 | 8 - 5 | 8 - 6 | 8 - 7 | 8 - 8 | 8 - 9 |  9 - 0 | 9 - 1 | 9 - 2 | 9 - 3 | 9 - 4 | 9 - 5 | 9 - 6 | 9 - 7 | 9 - 8 | 9 - 9 | |   · |

Los bucles al ser controladores de flujo incluyen la instrucción else por si la condición llega a no cumplirse.

En este ultimo ejemplo se ve como hay un else en la misma linea que el while, este else va tomar el flujo una vez que la condición del while no se cumple. Acá hay una bandera para saber si entro o no al while para poder hacer la impresión exacta, no confundir con el otro else que corresponde al bloque del if.

**Validación de datos : while**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | while entrada != 'S':  entrada = input ('¿ desea salir ? (S/N) :').upper()  print('seguimos en el while')  print ('Salimos, la variable entrada es igual a S') |   Salida esperada por consola   |  | | --- | | ¿ desea salir ? (S/N) : n  seguimos en el while  ¿ desea salir ? (S/N) : s  seguimos en el while  Salimos, la variable entrada es igual a S |   · |

·

**Operadores de identidad en condicionales:**

**is**  True si ambos operandos hacen referencia al mismo objeto.

False en caso contrario.

**is not** True si ambos operandos NO hacen referencia al mismo objeto.

False en caso contrario.

|  |
| --- |
| Registramos un objeto string para poder usar sus métodos isnumber() o isdigit() o isnumeric()  Generamos un while MIENTRAS el objeto string ingresado NO este compuesto solo por números.  Ingresamos un objeto string mediante un input dentro de un while  El bucle while se mantiene mientras la condición sea True por lo que si la niego o pido que sea falsa False obtendré una salida en el momento que ingrese un numero |

·

|  |
| --- |
| Un string es una cadena de caracteres.  Cada eslabón es un caracter.  isnumber() o isdigit() o isnumeric() evalua que cada caracter de la cadena sea un numero (0 a 9) |

·

|  |
| --- |
| Creo un objeto tipo str para poder usar sus métodos  Todo input devuelve un string  Para validar usamos algún método booleano de string  Si es necesario validar números .isdecimal() o isnumeric() o isdigit() por ahora son similares  casting de str a int |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | entrada=""  while entrada.isdecimal() is False:  entrada = input ('un numero entero :')  print ('Salimos')  print (f"El tipo de dato es {type(entrada)}")  entrada = int(entrada)# casting de str a int  print (f'la variable entrada es igual a {entrada}')  print (f"El tipo de dato es {type(entrada)}") |   Salida esperada por consola   |  | | --- | | un numero entero : puntos 3 # no aceptada  un numero entero : 3 puntos # no aceptada  un numero entero : 8 # aceptada  Salimos  El tipo de dato es <class 'str'>  la variable entrada es igual a 8  El tipo de dato es <class 'int'> |   · |

·

|  |
| --- |
| Si es un numero con punto decimal  usamos replace dentro del while donde si encontramos puntos ('.') lo cambiamos por un string vacio ('')  Un string es una cadena de caracteres.  Cada eslabón es un caracter.  En un numero decimal tengo un caracter punto (.) que separa la parte entere de la decimales  pi= 3.14159  \*\*no es numero  isnumber() o isdigit() o isnumeric() evaluá que cada caracter de la cadena sea un numero (0 a 9)  pi= 3.14159  pi.replace(".","")  se lee como 314159 sin punto, Esto permite usar isnumber() o isdigit() o isnumeric() |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | entrada=""  while entrada.replace(".","").isdecimal() is False:  entrada = input ('un numero flotante :')  print ('Salimos')  print (f"El tipo de dato es {type(entrada)}")  entrada = float(entrada)# casting de str a float  print (f'la variable entrada es igual a {entrada}')  print (f"El tipo de dato es {type(entrada)}") |   Salida esperada por consola   |  | | --- | | un numero flotante : 3 puntos # no aceptada  un numero flotante : 3 puntos 14159 # no aceptada  un numero flotante : 3.14159 # aceptada  Salimos  El tipo de dato es <class 'str'>  la variable entrada es igual a 3.14159  El tipo de dato es <class 'float'> |   · |

·

|  |
| --- |
| Se puede usar la misma función evaluada e formas distintas.  while entrada.replace(".","").isdecimal() **is False:**  while entrada.replace(".","").isdecimal() **is not True:**  while **not** entrada.replace(".","").isdecimal() **:**  comprobar que el bucle mientras no se cumple deja encerrado al usuario. |

·

|  |
| --- |
| Tengan en cuenta que la función **while** lleva una condición como un **if** pero el bloque de información que esta indentada se repite, itera, hasta que la condición sea **True**  Es decir. para poder salir del bucle, busco un False - not True |

·

Bucle For módulo 2 luego de colecciones

|  |
| --- |
| En física, existen varias constantes fundamentales que juegan un papel crucial en diversas teorías y ecuaciones. Aquí tienes algunos ejemplos de constantes importantes en física:  ¿que estructura de datos utilizarías?  Velocidad de la luz en el vacío (c): Es la velocidad máxima a la que puede propagarse la luz en el vacío y es una constante fundamental en la teoría de la relatividad. Su valor aproximado es 299792458 metros por segundo.  Constante gravitacional (G): Es una constante que aparece en la ley de gravitación universal de Newton y determina la fuerza gravitatoria entre dos objetos masivos. Su valor aproximado es 6.67430 \* 10\*\*-11 m³/(kg·s²).  Carga elemental (e): Es la carga eléctrica fundamental de un electrón o un protón. Su valor es aproximadamente 1.602 \* 10\*\*-19 culombios.  Constante de Planck (h): Es una constante que está relacionada con la cuantización de la energía y se utiliza en la mecánica cuántica. Su valor es aproximadamente 6.62607015 \* 10\*\*-34 julios-segundo.  Constante de Boltzmann (k): Es una constante que relaciona la temperatura con la energía en la física estadística y termodinámica. Su valor es aproximadamente 1.380649 \* 10\*\*-23 julios por kelvin.  Número de Avogadro (NA): Es el número de átomos o moléculas en un mol de sustancia. Su valor aproximado es 6.022 \* 10\*\*23 moléculas por mol.  Masa del electrón (me): Es la masa de un electrón, una partícula subatómica con carga negativa. Su valor es aproximadamente 9.10938356 \* 10\*\*-31 kilogramos.  Masa del protón (mp): Es la masa de un protón, una partícula subatómica con carga positiva. Su valor es aproximadamente 1.67262192 \* 10\*\*-27 kilogramos.  Masa del neutrón (mn): Es la masa de un neutrón, una partícula subatómica sin carga eléctrica. Su valor es aproximadamente 1.674927471 \* 10\*\*-27 kilogramos.  Constante de la ley de Coulomb (k\_e): Es una constante que aparece en la ley de Coulomb, que describe la fuerza electrostática entre dos cargas eléctricas. Su valor es aproximadamente 8.9875517923 \* 10\*\*9 N m\*\*2/C\*\*2.  Constante de permeabilidad del vacío (μ0): Es una constante que está relacionada con la magnetostática y determina la fuerza magnética entre corrientes eléctricas. Su valor es aproximadamente 4π \* 10\*\*-7 T m/A.  Constante de la ley de Stefan-Boltzmann (σ): Es una constante que aparece en la ley de Stefan-Boltzmann, que relaciona la radiación emitida por un cuerpo negro con su temperatura. Su valor es aproximadamente 5.670374419 \* 10\*\*-8 W/(m\*\*2 K\*\*4). |