Curso de Python

Python es lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Python es muy elogiado por su elegante sintaxis y su código legible, si estás comenzando tu carrera de programación, Python se adapta a tus necesidades.

Python tienen una amplia gama de usos. Desde procesamiento de datos, al aprendizaje de máquina. Por ello, Python es elegido como el lenguaje de programación de muchas empresas y organizaciones.

**Campos de uso de Python:**

* Frontend: Se encarga de llevar el diseño de una aplicación o sitio web a código
* IoT: Se encarga de darle la capacidad de conectarse a internet a elementos que pueden estar a nuestro alrededor.
* IA: Se encarga de enseñarle a la computadora a resolver un determinado problema sin la necesidad de estar involucrados constantemente.
* Backend: Se encarga de crear la lógica con la cual va a funcionar una determinada aplicación y que va a ser almacenada en un servidor.
* DevOps: Se encarga de manejar la información almacenada en la nube de una determinada aplicación.
* Data Science: Se encarga de tomar la información relevante de un determinado ambiente y poder sacar conclusiones al respecto.
* Videojuegos: Se encarga de combinar la programación, el diseño y la música para generar grandes experiencias a los usuarios.
* Desarrollo móvil: Se encarga de crear aplicaciones que serán almacenadas en la PlayStore o AppStore, y que podremos hacer uso de ellas desde nuestros smartphones.

Dentro de todo lenguaje de programación existe un núcleo llamado **algoritmo**. Un algoritmo es una serie de pasos ordenados para resolver un problema. Este es finito, ordenado, y no ambiguo.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Definiciones de algoritmo**

* Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problema.
* Algoritmo: Se denomina algoritmo a un grupo finito de operaciones organizadas de manera lógica y ordenada que permite solucionar un determinado problema.
* Algoritmo: una serie de instrucciones o reglas establecidas que, por medio de una sucesión de pasos, permiten arribar a un resultado o solución.
* Algoritmo: una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema.

**¿Cómo se Diseña un Algoritmo?**

En programación, un algoritmo establece, de manera genérica e informal, la secuencia de pasos o acciones que resuelve un determinado problema y, para representarlo, se utiliza, fundamentalmente, dos tipos de notación: pseudocódigo y diagramas de flujo.

Partes de un Algoritmo  
Todo algoritmo debe obedecer a la estructura básica de un sistema, es decir: entrada, proceso y salida.

**Características de los Algoritmos**

Las características fundamentales que debe cumplir todo algoritmo son:

* Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
* Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
* Un algoritmo debe ser finito. el algoritmo se debe terminar en algún momento; o sea, debe tener un número finito de pasos.
* Un algoritmo debe ser legible: El texto que lo describe debe ser claro, tal que permita entenderlo y leerlo fácilmente.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Para empezar a programar con Python, necesitaremos las siguientes herramientas:

**Editor de código**

Facilita la escritura del código, ya que da ciertas ayudas o resalta palabras claves del lenguaje de programación.

**Ejemplos de editor de código**

* Visual Studio Code
* Sublime text
* Atom
* Pycharm

El que recomienda y el que se va a utilizar es Visual Studio Code, porque es el más popular en la industria de tecnología y varios desarrolladores lo utilizan.

**Consola**

Programa que sirve para manejar la computadora sin necesidad de emplear la interfaz gráfica.

Ejemplos de consola

* CMD
* Power Shell
* Cmder.  
  Se utilizará cmder, porque tiene comandos compatibles con sistemas operativos del tipo Unix y Windows.

**Lenguaje de programación**

Python, que es el lenguaje más usado en el mundo y es ampliamente considerado el más fácil de aprender.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

No necesitamos la interfaz gráfica de nuestra computadora para poder usarla. Para esto, nuestra mejor herramienta es la consola. La consola nos permite comunicarnos con el computador por medio de comandos y así realizar tareas sin la necesidad de utilizar el mouse en una interfaz, sino solamente a través del código.

Comandos básicos para usar en la consola:

Ctrl + L = Limpiar pantalla

CD = Change Directory

… = Carpeta padre

CD… = Cambiar de directorio a la carpeta padre

Alt + 92 =

ls = list

mkdir = Make directory

touch = para crear archivos

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Primero, para iniciar la consola interactiva de Python debemos escribir el comando \*\*py \*\*en Windows, pero en otros sistemas el comando es python3.  
Ahora, podemos comenzar.

En la consola nos permite escribir operaciones matemáticas como 5 + 5 sin escribir nada más, pero en el editor de código debemos “imprimir” el resultado, de la siguiente manera:

print(5 + 5). Con esto obtendremos el resultado.

Ahora veamos como se realiza cada operación aritmética:

**Operadores aritméticos en Python**

* Suma: 5 + 5
* Resta: 5 - 5
* Multiplicación: 5 \* 5
* División (con decimales): 5 / 5
* División (sin decimales): 21 // 5
* Resto de la división: 21 % 5
* Potencia: 2 \*\* 2
* Raíz cuadrada:

math.sqrt(9)

3.0

math.sqrt(11.11)

3.3331666624997918

math.sqrt(Decimal('6.25'))

2.5

Python respeta la separación de términos, por lo que si escribimos 5 + 5 \* 2 multiplicará primero 5 x 2. En el caso de que quisiéramos que primero sume 5 + 5 ponemos paréntesis:  
(5 + 5) \* 2.

Para recordar el orden de las operaciones en álgebra y en Python, es recomendable utilizar el orden **PEMDAS**:

* Paréntesis
* Exponentes o raíces
* Multiplicaciones
* Divisiones
* Adiciones y sustracciones

\_\_\_\_\_\_\_\_\_\_\_\_

Una **variable** es un lugar en memoria (una especie de caja) en el que podemos guardar objetos (números, texto, etc). Esta variable posee un identificador o nombre con el cual podemos llamarla más tarde cuando la necesitemos.

**Asignación de variables**

En Python, creamos las variables asignándoles un valor de la siguiente manera:  
<identificador> = <valor>  
en este caso el signo = se lee como “asignar”

**Reasignación de variables**

Podemos en cualquier momento cambiar el valor de nuestra variable volviendo a asignar un valor al mismo identificador:  
<identificador> = <nuevo\_valor>

**Reglas en el uso de identificadores de variable**

* No pueden empezar con un número.
* Deben estar en minúsculas
* Para separar las palabras usamos el guion bajo: \_
* Estas reglas son aplicadas al lenguaje Python, en otros lenguajes puede haber otras reglas.

**Tipos de variables en Python**

a = 28 → int (entero)  
b = 1.5 → float (decimales)  
c = “Hello” → str (string o cadena de texto)  
d = True → boolean (verdadero o falso)  
e = None → NoneType (Sin valor)  
f = “5” → str (5 y “5” no son lo mismo. La primera es un entero y la segunda una cadena de texto)

\_\_\_\_\_\_\_\_\_

Un **objeto** es una forma de modelar el mundo en programación. En los lenguajes de programación se caracterizan por tener métodos y atributos. En Python todo es un objeto.

Podemos encontrar cuatro tipos de datos que vienen definidos por defecto en Python, a estos tipos de datos los conocemos como **primitivos**.

**Tipos de datos primitivos en Python**

* Integers: números Enteros
* Floats: números de punto flotante (decimales)
* Strings: cadena de caracteres (texto)
* Boolean: boolenaos (Verdadero o Falso)

Algunos operadores aritméticos pueden funcionar para operar con otros tipos de datos. Por ejemplo: podemos sumar strings, lo que concatena el texto o multiplicar un entero por un *string*, lo que repetirá el \_string \_las veces que indique el entero.

**Tipos de dato adicionales**

* Datos en texto: str
* Datos numéricos: int, float, complex
* Datos en secuencia: list, tuple, range
* Datos de mapeo: dict
* Set Types: set, frozenset
* Datos booleanos: bool
* Datos binarios: bytes, bytearray, memoryview

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cómo convertir un tipo de dato a otro en Python:

Sintaxis Descripción

* int(var) variable a entero
* float(var) variable a flotante
* str(var) variable a texto
* bool(var)variable a booleano
* abs(var) variable a valor absoluto

**Ejemplo de conversión de datos en Python**

>>> number1 = input("Escribe un número: ")

Escribe un número: 4

>>> number2 = input("Escribe otro número: ")

Escribe un número: 5

>>> numero1 + numero 2

=> '45' <== Se concatenan

**Solución:**

>>> number1 = int(input("Escribe un numero: "))

Escribe un numero: 100

>>> number2 = int(input("Escribe otro numero: "))

Escribe otro numero: 300

>>> number1 + number2

=> 400

**Ejemplo 2:**

>>> numero1 = 4.5

int(numero1)

=> 4 <== Trunca el flotante

**Ejemplo 3:**

>>> numero1 = 4.5

**str**(numero1)

=> '4.5' <== Lo convierte **a** texto

\_\_\_\_\_\_\_\_

Mi pequeño resumen:

* **and** para comparar si dos valores son verdaderos.  
  -**or** para comparar si dos valores son falsos.  
  -**not** para invertir el valor booleano.  
  -**==** Compara dos valores y te dice si son iguales o no.  
  -**!=** Compara dos valores y te dice sin son diferentes o no.  
  -**>** Compara si es mayor que otro valor.  
  -**>** Compara si es menor que otro valor.  
  **>=** igual o mayor que el valor a comparar.  
  **<=** igual o menor que el valor a comparar.

Seguro que muchos lo podéis explicar mucho mas claro. Un saludo!

Conoce los operadores lógicos que tiene Python y cómo utilizarlos de manera adecuada:

**Operadores lógicos**

* **and** ( y ): compara dos valores, y si ambos son verdaderos, devuelve True.
* **or** ( ó ): si al comparar dos valores y uno de los dos se cumple, devuelve True. Solo devuelve falso cuando los dos valores no se cumplen.
* **not** ( no): invierte el valor de una variable, dando el valor contrario al de la variable evaluada.

**Operadores de comparación**

* **==** ( igual qué ): determina si dos valores son iguales o no.
* **!=** (diferente de): determina si dos valores son distintos o no. Si los valores son diferentes devuelve True, si son iguales devuelve False.
* **>** (mayor que): compara dos valores, y determina si es mayor que el otro.
* **<** (menor que): compara dos valores y determina si es menor que el otro.
* **>=** (mayor o igual): compara dos valores y determinas si es mayor o igual que el otro.
* **<=** (menor o igual): compara dos valores y determinas si es menor o igual que el otro.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Los condicionales son decisiones que se establecen desacuerdo a los parámetros que indiquemos, para obtener un tipo de resultado deseado.

**Ejemplo**: si un número es mayor o igual que otro, los números deberán sumarse, de lo contrario deberán restarse. Debe cumplirse una condición para saber cuál será el camino a seguir.

A continuación veremos cómo funcionan los condicionales en Python.

**if**

(Si) se usa para la condición principal.

**elif**

(Si no) en caso de que la condición principal o anterior no se cumpla, se puede utilizar para agregar otra condición.

**else**

(Sino) en caso de que la(s) condición(es) anterior(es) no se cumplan, se ejecuta como alternativa sin condicional.

**En lenguaje natural (español)**

* ‘Si’ introduce una oración en la que se indica una condición real o hipotética que se ha de cumplir necesariamente para que sea cierto o se produzca lo que se expresa: Si corres, lo alcanzarás.
* ‘Sino’ es una conjunción adversativa que se escribe en una sola palabra y se usa, principalmente, para contraponer un concepto a otro: No estudia, sino que trabaja.
* ‘Si no’\*\* introduce una oración condicional: Si no estudias, no aprobarás.

**Ejemplo de condicionales en Python**

nacionalidad = input("¿Eres peruano? (Responde sí o no) ")

**if** nacionalidad == "sí":

cambio\_a\_soles = 3.9865

dolares = str(round(**float**(input("¿Cuántos soles tienes? "))/cambio\_a\_soles, 2))

print("Entonces tienes " + dolares + " dólares.")

elif nacionalidad == "no":

cambio\_a\_dolares = 0.25298

soles = str(round(**float**(input("Bienvenido a Perú. ¿Cuántos dólares tienes? "))/cambio\_a\_dolares, 2))

print("Entonces tienes " + soles + " soles.")

**else**:

print("Escribe sólo sí o no, por favor. ")

\_\_\_\_\_\_\_\_\_\_\_\_

Un detalle muy importante en cualquier lenguaje de programación es conocer las diferencias entre los condicionales. En Python en particular, es crucial mencionar la diferencia entre if, elif y else.

**Diferencias entre if, else y elif**

**if:**

if se encarga de iniciar el condicional y solicitar un requisito para ejecutar todo el código por debajo, que conocemos como bloque de código.

**else:**

Si se desea ejecutar otro código en caso de que no se cumpla el if. Por ejemplo: el usuario no elige la opción 1, entonces (else)…

**elif:**

Se utiliza cuando utilizamos múltiples condiciones, lo que en el código de esta clase son la opción 2 y 3. En esta clase, teníamos la opción 1, pero debemos también evaluar qué pasa si el usuario elige la opción 2 o 3, por lo que decimos “que estamos evaluando múltiples condiciones”.

**Añadir comentarios en Python**

Para realizar un comentario (de una sola línea), empleamos el “#”. Un comentario es simplemente texto, el cual no es ejecutado y no afecta en absoluto en el código. Se utiliza para explicar las líneas de código que hemos creado y hacerlas más fáciles de entender.

Para qué veas cómo se utilizan los condicionales y comentarios, mira este ejemplo:

#convierte pesos mexicanos, argentinos y colombianos a dólares

# """ """ permite crear strings multilineas

menu = """

Bienvenido al conversor de monedas multipais

1-Pesos Mexicanos

2-Pesos Colombianos

3-Pesos Argentinos

Elige una opción:

"""

# de derecha a izquierda: llamo a la funcion input, le paso la variable menu para que la imprima y reciba el número que el usuario escogió, lo convierto a int y lo guardo en la variable 'opcion'

opcion = int(input(menu))

**if** opcion == 1: #pesos mexicanos

#pregunto al usuario la cantidad a convertir

pesos = input('¿Cuántos pesos mexicanos tienes?: ')

#convierto a float para mejor manejo de datos

pesos = float(pesos)

#escribo el valor del dolar en pesos mexicanos

tipo\_de\_cambio = 21.5

elif opcion == 2: #pesos colombianos

#pregunto al usuario la cantidad a convertir

pesos = input('¿Cuántos pesos colombianos tienes?: ')

#convierto a float para mejor manejo de datos

pesos = float(pesos)

#escribo el valor del dolar en pesos colombianos

tipo\_de\_cambio = 3715.01

elif opcion == 3: #pesos argentinos

#pregunto al usuario la cantidad a convertir

pesos = input('¿Cuántos pesos argentinos tienes?: ')

#convierto a float para mejor manejo de datos

pesos = float(pesos)

#escribo el valor del dolar en pesos argentinos

tipo\_de\_cambio = 74.44

**else**: #el usuario escribió algo diferente

print('Escribe una opción correcta: ')

#hago la conversión

dolares = pesos / tipo\_de\_cambio

#redondeo los dólares a dos decimales

dolares = round(dolares, 2)

#convierto el float de dolares a un string

dolares = str(dolares)

#imprimo el valor de la conversion. Se pueden sumar (concatenar) strings con '+'

print('Tienes $' + dolares +' dólares')

\_\_\_\_\_\_\_\_\_\_

Las funciones ayudan a optimizar el código. Es decir, utilizar la menor cantidad de líneas dentro del código y evitar escribir acciones repetitivas.

Esto nos sirve para entregar un código *más limpio* y con buenas prácticas, que no desperdicia recursos innecesariamente. En Python, para definir funciones empleamos def.

Gracias a def, podemos “definir” funciones que emplearemos más tarde. Una función, en programación, es un grupo de instrucciones con un objetivo en particular y que se ejecuta cuando es “invocada”.

Cuando la definimos, estamos dándole un conjunto de instrucciones o un algoritmo. Al ahorrar líneas de código con funciones logramos también que la legibilidad de este sea más fácil.

**Cómo usar def en Python**

**def** **nombredelafuncion**():

# instrucciones de la función

**Ejemplo de funciones con def en Python**

**def** **conversacion**(opcion):

print('Hola')

print('Cómo estás')

print('Elegiste la opcion: ' + str(opcion))

print('Adiós')

opcion = int(input('Ingrese una opción (1, 2, 3): '))

**if** opcion == 1:

conversacion(opcion)

**elif** opcion == 2:

conversacion(opcion)

**elif** opcion == 3:

conversacion(opcion)

**else**:

print('Escribe una opción correcta.')

\_\_\_\_\_\_\_\_\_\_

Para el siguiente ejemplo, crearemos el código para un conversor de monedas.

En la primera parte se define la función que resumirá muchos procesos del programa. Usando *def*, se agrega la función “conversor” con los parámetros que varían dependiendo de las respuestas, que son: tipo de pesos y valor del dólar. Es decir, dentro del programa se definirá el valor de cada parámetro.

**def** **conversor**(tipo\_pesos, valor\_dolar):

En la variable pesos se plantea que el usuario introduzca con *input* la cantidad de pesos que tiene, encontrando el primer parámetro, que es el tipo de pesos, que se establece más adelante por fuera de la función, ya que es un protocolo.

pesos = input("¿Cuántos pesos " + tipo\_pesos + " tienes?: ")

Ese dato ingresado en la variable pesos se convierte de un *string* a un número con *float*.

pesos = float(pesos)

• En este punto, la variable dólares aparece para definir cuánto cuesta, con base en los pesos ingresados anteriormente y el valor dólar que se definirá más tarde por fuera de la función, ya que es un protocolo.

dolares = pesos / valor\_dolar

El valor, que probablemente sea decimal, se reduce con el atributo *round* dependiento de la variable dolares a solo 2 decimales.

dolares = round(dolares, 2)

Debido a que los dólares son obtenidos como números, se traducen a *strings* por medio del operador str  
dolares = str(dolares)  
Una vez obtenido el valor de los dolares en *string*, se imprime el resultado entre cadenas de texto.  
print (“Tienes $ " + dolares + " dolares”)  
• Luego de establecer la función, se crea la variable menu que **no se imprime** y contiene texto de referencia.

menu = “”“

Tienes dinero 💲

Nosotros **la** convertimos 💰

1 - cop

2 - eur

3 - arg

Elige una opció**n**: “””

• Ya definida la variable menu, se crea la variable opción que dependerá de lo ingresado en *input* por el usuario en relación con la variable menu. Es decir, que se imprime menu y se deja el espacio para obtener un dato digitado por el usuario, que luego es traducido en número usando *int*.

opción = **int**(input(**menu**))

• Dependiendo de lo que este usuario ingrese en el input de la variable opción: Usando *if*, si es igual a 1, entonces se ejecuta la función anterior de conversor. Dentro de esta se establece el protocolo de tipo de *peso* para este caso y en relación con menu que es Colombianos. El protocolo de valor del dólar se define con base en el tipo de peso

**if** opción == 1:

**conversor**(“colombianos”, 3875)

Empleando *elif*, si se selecciona otra opción de la variable menu ingresado en el input de opción, se ajusta el protocolo en función a lo mostrado en menu.

elif opción == 2:

**conversor**(“euros”, 0.8)

elif opción == 3:

**conversor**(“argentinos”, 65)

Si no se ingresa ningún dato relacionado dentro de las 3 opción de menu, se imprime que ingrese una opción correcta.

**else**:

**print**(“ingresa una opción correcta”)

\_\_\_\_\_\_\_\_\_\_\_\_\_

En Python, los **slices**, traducidos al español como “rebanadas”, nos permiten dividir los caracteres de un *string* de múltiples formas. A continuación, realizaremos un ejemplo cómo utilizarlos:

**Cómo usar *slices* en Python**

nombre = "Francisco"

nombre

"Francisco"

nombre[0 : 3)

Arranca desde el primer índice hasta llegar antes del 3° índice.  
El resultado sería  
"Fra"

nombre[:3]

Va desde el principio hasta antes de llegar del 3° índice. Como no hay ningún parámetro en el primer lugar, se interpreta que arranca desde el principio. Recordemos que empezamos a contar desde cero como primer dígito.  
El resultado sería  
"Fra"

nombre[1:7]

Arranca desde el índice 1 hasta llegar antes del 7.  
El resultado sería  
"rancis"

nombre[1:7:2]

Arranca desde el índice 1 hasta llegar antes del 7, pero pasos de 2 en 2, ya que eso es lo que nos indica el 3er parámetro, el cual es 2.  
El resultado sería  
"rni"

nombre[1::3]

Arranca desde el índice 1 hasta el final del *string* (al no haber ningún 2° parámetro, significa que va hasta el final), pero en pasos de 3 en 3.  
El resultado sería  
"rcc"

nombre[::-1]

Al no haber parámetro en las 2 primeras posiciones, se interpreta que se arranca desde el inicio hasta el final, pero en pasos de 1 en 1 con la palabra al revés, porque es -1.  
El resultado sería  
"ocsicnarF"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Un *bucle* es un ciclo continuo en todos los lenguajes de programación que nos permite iterar sobre nuestros pasos: magina un contador cíclico (1,2,3,4,5,6…) donde puedes agregar un paso más sobre tu programa principal.

**Ejemplo de bucle en Python**

Para este ejemplo, utilizaremos las potencias hasta llegar a un número determinado:

**def** **potencia**(numero):

potencia = 1

**while** (potencia <= 10):

result = numero \*\* potencia

print('Potencia de {} elevado a la {} es {}'.format(numero, potencia, result))

potencia += 1

**def** **run**():

numero = int(input('Escribe el numero al cual quieres averiguarle la potencia: '))

potencia(numero)

**if** \_\_name\_\_ == "\_\_main\_\_":

run()

**Ejemplo de bucle en la vida real**

* Despertar
* Estudiar en Platzi
* Comer
* Dormir

Cuando repetimos estas acciones en ese orden, durante un tiempo determinado o infinito estamos hablando de un bucle.

\_\_\_\_\_\_\_\_\_\_

Un **bucle while** permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).

La sintaxis del bucle while es la siguiente:

**while** condicion:

cuerpo del bucle

Python evalúa la condición:

* Si el resultado es *True*, se ejecuta el cuerpo del bucle. Una vez ejecutado el cuerpo del bucle, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del bucle) una y otra vez mientras la condición sea cierta.
* Si el resultado es *False*, el cuerpo del bucle no se ejecuta y continúa la ejecución del resto del programa.

**Ejemplo de while en Python**

**def** **run**():

LIMITE = 1000000

contador = 0

potencia\_2 = 2\*\*contador

**while** potencia\_2 < LIMITE:

print('2 elevado a ' + str(contador) +

' es igual a: ' + str(potencia\_2))

contador = contador + 1

potencia\_2 = 2\*\*contador

**if** \_\_name\_\_ == "\_\_main\_\_":

run()

**Consejo**: se puede cancelar un ciclo infinito si presionas Ctrl + c.

\_\_\_\_\_\_\_\_\_\_

El ciclo for es un tipo de bucle usado cuando se conozcan la cantidad de veces a iterar.

Un contador es una variable que se encarga de contener valores que irán incrementando o decrementando cada vez que se ejecuta una acción que lo contenga. El incremento o decremento es llamado paso del contador y es siempre constante.

Ejemplo: El marcador de un partido de fútbol, cada vez que un equipo anote un gol, aumenta su marcador en una unidad.

Ejemplo 2: En las carreras de autos, cuando un vehículo pasa por la línea de meta, se incrementa en una unidad el número de vueltas dadas al circuito, o bien se decrementa el número de vueltas que faltan por realizar.

Entonces, el incremento es siempre constante, el paso del contador no necesariamente puede ser una unidad, también puede incrementarse o decrementarse de a dos, tres, cuatro, hasta n. Es decir, puede ser cualquier número que conserve siempre el mismo valor durante todo el programa.

Su sintaxis es:

variable = variable + constante(al incrementar)

variable = variable - constante(al decrementar)

o de manera resumida:

variable += constante

variable -= constante

Ejemplo:

gol\_local = 0 #si anotan gol: gol\_local = gol\_local +1

Consejo: Es importante inicializar en cero a la variable cuando aparezca a ambos lados del símbolo de asignación

Ejemplo del bucle for en Python

def imprimir\_numero(inicio, fin):

for inicio in range(fin+1):

print(f'Numero: {inicio}')

def imprimir\_numero\_while(inicio, fin):

while inicio <= fin:

print(f'Numero: {inicio}')

inicio += 1

def run():

while True:

print('')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*N U M E R O S\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('')

inicio = int(input('Digite el número inicial para la secuencial: '))

print('')

fin = int(input('Digite el número final para la secuencial: '))

print('')

if inicio < fin:

imprimir\_numero(inicio,fin)

else:

print(f'El numero inicial {inicio} debe ser ser menor al numero final {fin}.')

if \_\_name\_\_ == "\_\_main\_\_":

run()

\_\_\_\_\_\_\_\_\_\_\_

La **instrucción *continue* en Python** devuelve el control al comienzo del ciclo *while* o ciclo *for*. Esta instrucción rechaza todas las declaraciones restantes en la iteración actual del ciclo y mueve el control de regreso a la parte superior del mismo.

La instrucción *break* en Python termina el ciclo actual y reanuda la ejecución en la siguiente instrucción. En otras palabras, break rompe el ciclo entero mientras que *continue* solo rompe la vuelta actual.

**Ejemplos de interrupción de ciclos con *break* y *continue* en Python**

**Ejemplo1**

**def** **run**():

**for** i **in** range(10000):

print(i)

**if** i == 5678:

**break**

**if** \_\_name\_\_ == '\_\_main\_\_':

run()

**Ejemplo 2**

**def** **run**():

**for** contador **in** range(1000):

**if** contador % 2 != 0:

**continue**

print(contador)

**if** \_\_name\_\_ == '\_\_main\_\_':

run()

**Ejemplo 3**

**def** **run**():

texto= input('Escribe un texto: ')

**for** letra **in** texto:

**if** letra == 'o':

**break**

print(letra)

**if** \_\_name\_\_ == '\_\_main\_\_':

run()

\_\_\_\_\_\_\_\_\_

En este ejemplo, aprenderemos **cómo detectar si un número es primo en Python**. Esto se conoce como prueba de primalidad. Un número primo es todo número que puede dividirse únicamente por sí mismo y por 1. Todos los números primos, excepto el 2, son impares.

En la matemática aplicada, los números primos son utilizados para generar códigos criptográficos seguros. Esto se logra empleando los números primos de Mersenne (números muy grandes).

**Ejemplo: cómo averiguar si un número es primo en Python**

**def** **es\_primo**(numero):

**if** numero == 1:

**return** **False**

**else**:

contador = 0

**for** i **in** range(1 , numero+1):

**if** i == 1 **or** i == numero:

**continue**

**if** numero % i == 0:

contador += 1

**if** contador == 0:

**return** **True**

**else**:

**return** **False**

**def** **run**():

numero = int(input("Escribe un número: "))

**if** es\_primo(numero):

print(str(numero) + " es primo")

**else**:

print(str(numero) + " NO es primo")

**if** \_\_name\_\_ == "\_\_main\_\_":

run()

\_\_\_\_\_\_\_\_\_\_

Para el siguiente ejemplo, utilizaremos módulos en Python para crear la funcionalidad de nuestro código. Un módulo es un archivo con funciones ya predefinidas, que tenemos disponibles para ejecutarlas. Para traer o *invocar* un módulo, debemos escribir lo siguiente:

**import** random

En este caso, “importamos” la función random, que trae un conjunto de funciones que nos permiten trabajar con la aleatoridad.

numero\_aleatorio = random.

Gracias al punto ., accedemos a las funciones que trae el módulo. En este caso, generamos una variable y le asignamos random.randint(1, 100), lo cual nos genera un número aleatorio entero desde un número hasta otro (en este caso del 1 al 100).

**Ejemplo de juego de aleatoriedad con módulos en Python**

**import** random

**def** **run**():

numero\_aleatorio = random.randint(1, 100)

numero\_elegido = int(input("Elije un numero del 1 al 100 :"))

**while** numero\_elegido != numero\_aleatorio:

**if** numero\_elegido < numero\_aleatorio:

print("Busca un numero mas grande ")

**else**:

print("Busca un numero mas pequeño ")

numero\_elegido = int(input("Elije otro numero :"))

print("Ganaste!")

**if** \_\_name\_\_ == "\_\_main\_\_":

run()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Los **diccionarios** en Python son una estructura de datos mutable las cuales almacenan diferentes tipos de valores sin darle importancia a su orden. Identifican a cada elemento por una clave (Key). Se escriben entre {}.

**Operaciones en diccionarios**

* .keys():Retorna la clave de nuestro elemento.
* .values(): Retorna una lista de elementos (valores del diccionario).
* .items(): Devuelve lista de tuplas (primero la clave y luego el valor).
* .clear(): Elimina todos los items del diccionario.
* .pop(“n”): Elimina el elemento ingresado.

**Cómo trabajar con diccionarios**

* Definir función principal

**def** **run**():

# Defino el diccionario, agrego los valores.

mi\_diccionario = {

'llave1' : 1,

'llave2' : 2,

'llave3' : 3,

}

* Imprimir las llaves del diccionario utilizando un bucle for. Con ‘.keys()’ estamos llamando a imprimir las llaves, no los valores.

**for** llave **in** mi\_diccionario.keys():

print(llave)

* Imprimir los valores del diccionario empleando un bucle *for*. Con ‘.values()’ estoy llamando a imprimir los valores.

**for** valores **in** mi\_diccionario.values():

print(valores)

* Imprimir las llaves y los valores usando ‘.items()’. Para esto, coloco las variables llave e items.

**for** llave, **items** **in** mi\_diccionario.**items**():

print("La llave: '" + llave + "' contiene el item: " + str(**items**))

**if** \_\_name\_\_ == '\_\_main\_\_':

run()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_