Introducción

Variables, expresiones, entrada/salida, condicional, bucles.

Repaso de los conceptos utilizando Python

Elementos fundamentales de un lenguaje de Programación

Constantes (o valores, o literales)

Variables

Sentencias

Expresiones

Constantes

Son valores fijos (Ej: números, letras o cadenas de caracteres) que no cambian su valor.

Cada constante tiene un tipo de dato asociado.

En Python no existen las constantes como tal.

La convención para nombrar constantes es hacerlo con todo el nombre en mayúsculas, de esta forma, aunque en realidad estemos trabajando con variables, quien esté trabajando con el programa, sabrá a simple vista que se trata de una constante y que no hay que reasignarle ningún valor.

Variables

- Es un espacio la memoria referenciado a partir de un nombre.
- El programador puede almacenar valores para recuperarlos posteriormente.
- El nombre es elegido por el programador.
- El proceso para asignar un valor a una variable, lo llamamos “asignación”

Reglas:

- Deben comenzar con una letra o ‘_’
- Deben estar formados por letras, números o ‘_’. **texto , _num1 , nombre**
  3ro , *tel , $modelo
- Python es Case Sensitive (sensible a mayúsculas y minúsculas)
  numero es distinto a **Numero**

Palabras reservadas

- No pueden usarse como nombres de variables
- Tienen significado propio dentro del contexto del lenguaje de programación

```
>>> import keyword
>>> print(keyword.kwlist)
```

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

Sentencias y expresiones

A = 10 Sentencia de asignación

B = A * 4 Sentencia de asignación (expresión)

print(“Hola!”) Sentencia print()

Expresiones numéricas. Operadores aritméticos

Operador	Nombre	Ejemplo
+	Suma	2 + 2
–	Resta	3 – 1
*	Multiplicación	5 * 3
/	División	5 / 2
%	Módulo	5 % 2
//	División entera	9 // 2
**	Potencia	2 ** 4

Expresiones numéricas. Orden de evaluación

Las expresiones (en general) están formadas por combinación de valores, operadores y variables.

Cuando las expresiones tienen múltiples operadores, debe quedar claro cuál es su orden de ejecución (precedencia de operadores).

Reglas:

1. Paréntesis
2. Potencia
3. Multiplicación/División
4. Suma/Resta

Misma precedencia: Izquierda a derecha

Expresiones. Ejemplo reglas de precedencia

10 + 4 ** 2 / 8 - **(2 + 2)**

10 + **4 ** 2** / 8 - 4

10 + **16 / 8** - 4

10 + 2 - 4

12 - 4

8

Tipos de datos en Python

Tipo	Clase	Notas	Ejemplo
`str`	Cadena	Inmutable	`'Cadena'`
`unicode`	Cadena	Versión Unicode de `str`	`u'Cadena'`
`list`	Secuencia	Mutable, puede contener objetos de diversos tipos	`[4.0, 'Cadena', True]`
`tuple`	Secuencia	Inmutable, puede contener objetos de diversos tipos	`(4.0, 'Cadena', True)`
`set`	Conjunto	Mutable, sin orden, no contiene duplicados	`set([4.0, 'Cadena', True])`
`frozenset`	Conjunto	Inmutable, sin orden, no contiene duplicados	`frozenset([4.0, 'Cadena', True])`
`dict`	Mapping	Grupo de pares clave:valor	`{'key1': 1.0, 'key2': False}`
`int`	Número entero	Precisión fija, convertido en *long* en caso de overflow.	`42`
`long`	Número entero	Precisión arbitraria	`42L` ó `456966786151987643L`
`float`	Número decimal	Coma flotante de doble precisión	`3.1415927`
`bool`	Booleano	Valor booleano verdadero o falso	`True` o `False`

Función type()

La función type nos devuelve el tipo de dato de un objeto dado. Por ejemplo:

```
>>> type(5)
<class 'int'>
>>> type(5.5)
<class 'float'>
>>> type([1,2])
<class 'list'>
>>> type(int)
<class 'type'>
```

Tipos de datos. Conversiones

- Cuando ponemos un **número entero** en una **expresión** junto a otro **número real**, el número entero se convierte implícitamente en un número **real**.
- Además, existen funciones como **int()** y **float()** para convertir de manera explícita un determinado valor.

- **Atención! La división de números enteros siempre genera un número real.**

Tipos de datos. Conversión de Strings

También pueden usarse las funciones **int()** y **float()** para convertir cadenas de caracteres en valores numéricos.

La función producirá un error si la cadena no es convertible al tipo requerido.

Análogamente, existe una función **str()** para convertir de valores numéricos a cadena de caracteres.

Entrada/Salida

Ya vimos que para mostrar información en la pantalla usamos la función **print()**.

Para leer datos de teclado, usaremos la función **input()**.

Importante! La función input() siempre retorna un String. Por lo tanto, si queremos leer un valor numérico, debemos convertirlo de manera explícita.

```
nombre = input("Ingrese su nombre ")

print("Usted se llama :", nombre)

edad = int( input( "Ingrese su edad: "))
```

Función print()

La función print() nos permite mostrar información en pantalla. La sintaxis es:

```
>>>print('texto')
texto
>>>cadena = "Esto es una cadena"
>>>print(cadena)
Esto es una cadena
>>>print(1 + 2)
3
nombre = "Alicia"
edad = 35
print("Me llamo", nombre, "y tengo", edad, "años.")
print(f"Me llamo {nombre} y tengo {edad} años.")
```

mismo funcionamiento, cadena interpolada

Función print()

La función print() siempre añade un salto de línea luego de escribir el string indicado.

Según la documentación, print() nos permitirá modificar el final de la impresión y la separación de los objetos a imprimir mediante los parámetros **end** y **sep**

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Ej:

```
saludo = "Hola"
print(saludo, end=", ")
print("¿Cómo estas?")
print("---")
```



```
Hola, ¿Cómo estas?
---
```

Cadenas interpoladas

Hay cuatro formas en las que podemos realizar la interpolación de cadenas: módulo (%), método format(), cadenas formateadas y clase de plantilla.

Se puede usar el módulo (**%**) para interpolar cadenas, los siguientes son los marcadores de posición más comunes:

%d = Integer

%f = Float

%s = String

%x = Hexadecimal

```
"<string with placeholders>" % (<comma separated values or variables>)
```

Cadenas interpoladas. %

Ejemplos:

```
nombre=input("Dime tu nombre:")
print ("Hola %s" % nombre)


perimetro = 2*base + 2*altura
area = base * altura
print ("Resultado: Area=%.2f Perimetro=%.2f" % (area,perimetro))
```

cadenas interpoladas. format()

El método format() se puede utilizar para formatear cadenas en Python. Este método es similar al anterior pero aquí, **{}** actúa como marcador de posición para cada tipo de valor.

```
"<string with placeholders>".format(<comma separated values and variables>)
```

pi = 3.14

nombre = “Damian”

```
print("El valor de PI o π: {}".format(pi)
```

```
print("Hola {}. El valor de PI o π: {}".format(nombre, pi)
```

Cadenas interpoladas. Cadenas F

Cuando se prefija con un carácter **f**, las cadenas regulares son cadenas formateadas.

Las cadenas formateadas también se conocen como cadenas f. Estas cadenas se utilizan para insertar dinámicamente representaciones de cadenas de variables y objetos dentro de cadenas.

Podemos agregar **{}** dentro de cadenas, y dentro de estos bloques, podemos agregar variables o lógica que devuelva algún valor.

```
nombre= "Joel"

texto= f"Mi nombre es {nombre}"
```

Función input()

input(*prompt*)

Si el argumento *prompt* está presente, se escribe a la salida estándar sin una nueva línea a continuación. La función lee entonces una línea de la entrada, la convierte en una cadena (eliminando la nueva línea), y retorna eso.

ej:

```
#!/usr/bin/env python

nombre=input("Dime tu nombre: ")

print (f"Hola {nombre}")
```

Condicional

```
#!/usr/bin/env python
# Sangrado con 4 espacios          (tabulación)
edad = 23
if edad >= 18:
    print('Es mayor de edad')
else:
    print('Es menor de edad')
```

Operadores de comparación

Operador	Significado
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
==	es igual
!=	es distinto (o no es igual)

Operadores lógicos

Operador	Funcionamiento	Ejemplos
and	`True` si las dos declaraciones son verdaderas	True and False -> False False and False -> False True and True -> True
or	`True` si una de las declaraciones es verdadera	True or False -> True True or True -> True False or False -> False
not	niega la declaración que le sigue	not True -> False not False -> True

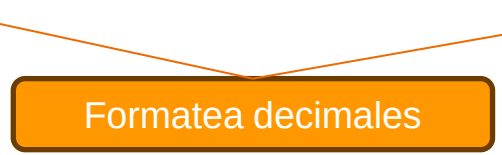
Calcular el perímetro y área de un rectángulo dada su base y su altura.

Tomense unos minutos para pensar como lo harían.

Deben pedirle al usuario la base y la altura y luego devolver por pantalla el resultado.

Calcular el perímetro y área de un rectángulo dada su base y su altura.

```
base=float(input("Dime la base:"))
altura=float(input("Dime la altura:"))
perimetro = 2*base + 2*altura
area = base * altura
print(f"Resultado: Area={area:.2f} Perimetro={perimetro:.2f}")
```



If - elif - else

```
distancia = 57
if distancia > 100 :
    print("Todavía faltan más de 100km")
elif distancia > 60 :
    print("Todavía faltan más de 60km")
elif distancia > 20 :
    print("Todavía faltan más de 20km")
else:
    print("Ya estas por llegar")
```

Condicionales anidados

```
x = ...
if x>10:
    print(“x es mayor a 10”)
    if x > 100:
        print (“x es demasiado grande”)
```

Bloques

Cada bloque de instrucciones dentro de una estructura de control debe estar tabulada

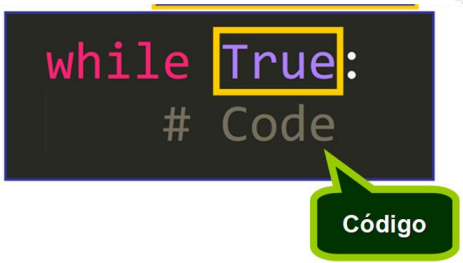
```
#!/usr/bin/env python

if num >=0:

        while num<10:

                print (num)

                num = num +1
```

Ciclos

En Python existen 2 tipos de ciclos:

- While
- For

Recordar que en el ciclo “**while**” Es EXTREMADAMENTE importante, **asegurarse que la condición sea falsa en alguna iteración**, de lo contrario tendremos un ciclo que no se detiene nunca.



- Papá, ¿qué es un bucle infinito?
- Pregunta a tu mamá.
- Mamá, ¿qué es un bucle infinito?
- Pregunta a tu papá.
- Papá, ¿qué es un bucle infinito?

Ciclo while

```
while <expresión_lógica>:
    Instrucción 1
    Instrucción 2
    …
    Instrucción n
Instrucción fuera del ciclo
```

```
n = 4
while n>0:
    print(n)
    n = n - 1
print(“Despegue!!!”)
```

Salida:
4
3
2
1
Despegue!!!

Ciclo for

```
for <variable> in <secuencia_de_elementos>:
    Instruccion 1
    ...
    Instrucción n
Instrucción siguiente al ciclo for
```

Ciclo for

Función **range()** para generar una secuencia iterable.

range(stop)

range(start, stop)

range(start, stop, step)

- start: Comienzo de la secuencia.
- stop: fin de la secuencia (sin incluir).
- step: “paso” entre los valores de la secuencia.

Ciclo for

```
for i in range(0, 20, 3):
    print(i)
```

¿Cómo será la salida?

Salida

0
3
6
9
12
15
18

Ciclo for

Con el ciclo for también podemos iterar colecciones que sean *iterables*. Una clase iterable es una clase que puede ser iterada. Dentro de Python hay gran cantidad de clases iterables como las **listas**, **strings**, **diccionarios** o **ficheros**. Si tenemos una clase iterable, podemos usarla a la derecha del for de la siguiente manera:

```
# for elemento in [clase_iterable]:
#    ...
```

la variable elemento irá tomando los valores de cada elemento presente en la clase iterable.

Ciclo for

```
ej:

lista = [5, 4, 9, 2]

for elemento in lista:

    print(elemento)

# Salida 5, 4, 9, 2
```

Que es “#!/usr/bin/env python”?

La línea #!/usr/bin/env python se conoce como shebang y se utiliza en los sistemas operativos tipo Unix (como Linux y macOS) para indicar qué intérprete de comandos debe usarse para ejecutar un script. En este caso específico, se está utilizando para scripts escritos en el lenguaje de programación Python.

En vez de llamar con python nombre.py se llama solo con ./nombre.py