Clase 1-La sintaxis de Python 1

August 10, 2023

1 Iniciando en Python

1.1 Hola Mundo en Python

En cualquier introducción a un nuevo lenguaje de programación, no puede faltar el famoso Hola Mundo. Se trata del primer programa por el que se empieza, que consiste en programar una aplicación que muestra por pantalla ese texto. Si ejecutas el siguiente código, habrás cumplido el primer hito de la programación en Python.

```
print("Hola Mundo")
```

Por lo tanto ya te puedes imaginar que la función print() sirve para imprimir valores por pantalla. Imprimirá todo lo que haya dentro de los paréntesis. Fácil ¿verdad? A diferencia de otros lenguajes de programación, en Python se puede hacer en 1 línea.

[ ]:
```
print("Hola mundo desde Python")
```

```
Hola mundo desde Python
```

1.2 Comentarios

- Los comentarios pueden ser de una sola línea o de varias líneas.

[ ]:
```
# Esto es un comentario en Python
print("Curso de Python desde Cero")
print("Hola desde python")
print(4+5)

"""
Esto es un comentario de varias
lineas en Python
print("Esto no se muestra")
"""
```

```
Curso de Python desde Cero
Hola desde python
9
```

[ ]:
```
'\nEsto es un comentario de varias \nlineas en Python\nprint("Esto no se
muestra")\n'
```

1.3 Definiendo variables en Python

Vamos a complicar un poco más las cosas. Creemos una variable que almacene un número. A diferencia de otros lenguajes de programación, no es necesario decirle a Python el tipo de dato que queremos almacenar en x. En otro lenguajes es necesario especificar que x almacenará un valor entero, pero no es el caso. Python es muy listo y al ver el número 5, sabrá de que tipo tiene que ser la x.

```
x = 5
```

Ahora podemos juntar el print() que hemos visto con la x que hemos definido, para en vez de imprimir el Hola Mundo, imprimir el valor de la x.

```
print(x) # Salida: 5
```

En el anterior fragmento habrás visto el uso #. Se trata de la forma que tiene Python de crear los denominados comentarios. Un comentario es un texto que acompaña al código, pero que no es código propiamente dicho. Se usa para realizar anotaciones sobre el código, que puedan resultar útiles a otras personas. En nuestro caso, simplemente lo hemos usado para decirte que la salida de ese comando será 5, ya que x valía 5.

1.4 Sumando Variables en Python

Vamos a sumar dos variables e imprimir su valor. Lo primero vamos a declararlas, con nombres a y b. Declarar una variable significa "crearla".

```
# Declaramos las variables a, b
# y asignamos dos valores
a = 3
b = 7
```

Ahora Python ya conoce a y b y sus respectivos valores. Podemos hacer uso de + para sumarlos, y una vez más de print() para mostrar su valor por pantalla.

`print(a+b)` Es importante que sólo usemos variables que hayan sido definidas, porque de lo contrario tendremos un error. Si hacemos:

```
print(z)
# Error! La variable no existe
```

Tendremos un error porque Python no sabe que es z, ya que no ha sido declarada con anterioridad.

[ ]:
```
# Variables y tipos:

texto = "Programación Competitiva"
print("Esto es un texto:")
print(texto)

entero = 10

print("Esto es un entero:")
print(entero)
```

```python
decimal = 10.0

print("Esto es un decimal:")

print(decimal)

logico = True

print("Esto es un valor logico:")

print(logico)
```

```
Esto es un texto:
Programación Competitiva
Esto es un entero:
10
Esto es un decimal:
10.0
Esto es un valor logico:
True
```

1.5 Concatenar en Python

[ ]:
```python
# Definicion de variables:

cad1 = "Esto"
cad2 = "es"
cad3 = "Python"

# Mostrar variables con print:

print("Pasando las cadenas como parámetros al print.")
print(cad1, cad2, cad3)

# Concatenar una sola cadena:

cadena = cad1 + " " + cad2 + " " + cad3
print("Utilizando una variable")
print(cadena)

# También puede hacerse desde el print:
print("Formando la cadena en el print")
print(cad1 + " " + cad2 + " " + cad3)

# Tercera manera, utilizando f:
print("Utilizando formateo - interpolado")
print(f"{cad1} {cad2} {cad3}")
```

```
# Cuarta forma, con format:
print("Utilizando la cláusula format:")
print("Cadena 1: {}, Cadena 2: {}, Cadena 3: {}".format(cad1,cad2,cad3))
```

```
Pasando las cadenas como parámetros al print.
Esto es Python
Utilizando una variable
Esto es Python
Formando la cadena en el print
Esto es Python
Utilizando formateo - interpolado
Esto es Python
Utilizando la cláusula format:
Cadena 1: Esto, Cadena 2: es, Cadena 3: Python
```

1.6 Mas acerca de los tipos de datos en Python

[ ]:
```
# Tipos de datos de variables:
nada = None
cadena = "Programación Ucatec"
entero = 7
decimal = 10.50
logico = False
listaNumeros = [10, 20, 30, 40, 50, 60]
listaNombres = ["Juan", "Carlos", "María", "Fernanda"]
# La tupla  es similar a una lista pero no cambia
tupla = ("Programación", "en", "Python")
# Un diccionario es una lista con clave y valor.
diccionario = {
    "nombre": "Juan",
    "apellido": "Perez",
    "ocupacion": "Programador"
}
rango = range(9)
dato_byte = b"Ucatec"

# Imprimiendo datos de variables:
print(nada)
print(cadena)
print(entero)
print(decimal)
print(logico)
print(listaNumeros)
print(listaNombres)
print(tupla)
print(diccionario)
```

```
print(rango)
print(dato_byte)

# Mostrando tipos de datos:
print(type(nada))
print(type(cadena))
print(type(entero))
print(type(decimal))
print(type(logico))
print(type(listaNumeros))
print(type(listaNombres))
print(type(tupla))
print(type(diccionario))
print(type(rango))
print(type(dato_byte))
```

```
None
Programación Ucatec
7
10.5
False
[10, 20, 30, 40, 50, 60]
['Juan', 'Carlos', 'María', 'Fernanda']
('Programación', 'en', 'Python')
{'nombre': 'Juan', 'apellido': 'Perez', 'ocupacion': 'Programador'}
range(0, 9)
b'Ucatec'
<class 'NoneType'>
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'list'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
<class 'range'>
<class 'bytes'>
```

1.7 Conversión de Tipos de Datos

[ ]:
```
texto = "Ucatec Programación"
anio = 2023

# Se puede mostrar enviando parametros:
print(texto, anio)
```

```
# Pero para concatenar se debe convertir el tipo de dato:
print(texto + " " + str(anio))

# Otras conversiones de tipos:
numero = int(7)

print(numero)
print(type(numero))
print(type(str(numero)))

# Reasignando numero:
numero = float(7)

print(numero)
print(type(numero))
print(type(str(numero)))
```

```
Ucatec Programación 2023
Ucatec Programación 2023
7
<class 'int'>
<class 'str'>
7.0
<class 'float'>
<class 'str'>
```

1.8 Strings

[ ]:
```
texto1 = "Programación"
texto2 = "\"Ucatec\""

cadena1 = texto1 + " " + texto2

print(cadena1)

texto3 = 'Club'
texto4 = '"Python"'
cadena2 = texto3 + " " + texto4

print(cadena2)

# Valores especiales:
print(cadena1 + '\t' + cadena2)
print(cadena1 + '\n' + cadena2)
print(cadena1 + '\r' + cadena2)
```

```
Programación "Ucatec"
Club "Python"
```

```
Programación "Ucatec"   Club "Python"
Programación "Ucatec"
Club "Python"
Club "Python""Ucatec"
```

1.9 Operadores

[ ]:
```python
# Operadores de asignación:
a = 2
b = 2

# Operadores aritméticos:
print("Operaciones Aritméticas")
print(f"La suma es: {a + b}")
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print(a ** b)

# Asignaciones complejas, incremento y decremento:
num = 7

print("Asignaciones complejas")
num += 1
print(num)
num -= 1
print(num)
num *= 2
print(num)
num //= 2
print(num)
num /= 2
print(num)

print("incremento y decremento")
anio = 2023
print(anio)
anio = anio + 1
print(anio)
anio = anio - 1
print(anio)
```

```
Operaciones Aritméticas
La suma es: 4
0
4
```

```
1.0
1
0
4
Asignaciones complejas
8
7
14
7
3.5
incremento y decremento
2023
2024
2023
```

1.10 Entrada y Salida de Datos

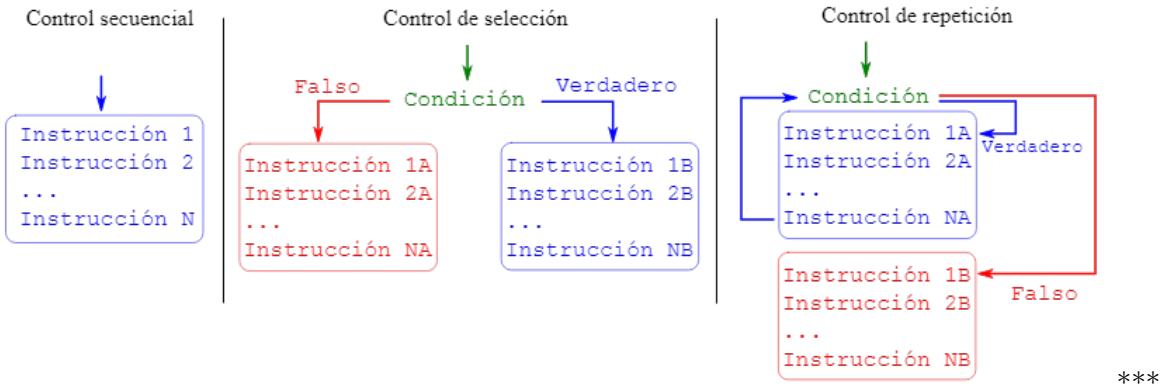
[ ]:
```python
# Entrada de datos:
num1 = input("Ingresa un número: ")
num2 = input("Ingresa otro número: ")

suma = int(num1) + int(num2)

# Salida de datos:
print("La suma es: " + str(suma))
```

```
La suma es: 12
```

2 Estructuras de Control

En un lenguaje de programación, las estructuras de control permiten modificar el flujo de la ejecución de un conjunto de instrucciones. Se pueden distinguir tres tipos básicos de control de flujo, a saber:

- Control secuencial
- Control de selección
- Control de repetición



***

En el control secuencial las instrucciones se ejecutan de manera secuencial desde el inicio hasta el fin del programa. En el control de selección se tiene una condición que puede ser falsa o verdadera, dependiendo de esto se ejecutará uno u otro bloque de instrucciones. En el control de repetición, un bloque de instrucciones se ejecuta de manera repetitiva mientras una condición sea verdadera, en caso contrario el flujo de ejecución se pasará a otro conjunto de instrucciones.

2.1 Estructuras Condicionales

El condicional if-elif-else es una estructura de control de selección que sirve para tomar decisiones, basándose en la evaluación de condiciones y/o comparaciones, en el flujo del programa. La sintaxis más general para if-elif-else es:

```
if cond1:
    # hacer algo
elif cond2:
    # hacer otra cosa
...
elif condn:
    # hacer algo más
else:
    # hacer algo por default
```

a es menor que b

Donde cond1, cond2, … condn son valores lógicos que resultan de una comparación. Esta estructura se evalúa secuencialmente hasta encontrar una condición que se cumpla, si ninguna lo hace, entonces se ejecuta la instrucción colocada en el caso por default else.

Hay que tener especial cuidado con las indentaciones de los bloques de código, en Python las indentaciones no son opcionales, tienen un significado sintáctico. La mayoría de los editores de código de Python indentarán de manera automática la siguiente línea, cada vez que coloques los dos puntos al final de una línea.

En el siguiente fragmento de código se muestra un programa que verifica si el valor de a es mayor, menor o igual al valor de b.

```
a = 10
```

```
b = 30

if a > b:
    print("a es mayor que b")
elif a < b:
    print("a es menor que b")
else:
    print("a es igual a b")
```

Es muy común que en algunos casos, por cuestiones inherentes a lo que se está programando, únicamente se decida entre dos posibilidades que son excluyentes, en esos casos basta con una estructura reducida if-else.

Por ejemplo, el siguiente código determina si un número es par o impar (un número entero cualquiera o es par o es impar, no hay otra posibilidad).

```
n = 1001

if (-1)**(n) > 0:
    print("El número es par")
else:
    print("El número es impar")
```

El número es impar

La verificación que realiza el código anterior se basa en el hecho de que las potencias pares de -1 siempre serán 1, y las impares -1, es decir:

$(-1)^n = \{ 1 \text{ ; si n es par } -1 \text{ ; si n es impar}$

Supongamos ahora otro caso en donde las posibilidades son mutuamente excluyentes: en un curso la calificación mínima aprobatoria es de 70, la escala de calificación es de 0 a 100, ¿cómo podríamos implementar un código que decida si un alumno aprobó o reprobó la asignatura? Observa lo siguiente:

```
calificacion = 100

if calificacion >= 70:
    print("Aprobado")
else:
    print("No aprobado")
```

Sobre datos y validaciones: >Usualmente un programa de computadora recibe datos de forma externa, es decir, alguien los proporciona de forma manual o se leen desde algún archivo de datos o desde algún sensor. Esto implica la posibilidad de que se reciban datos que no corresponderían con lo esperado, por ejemplo, en el caso del programa anterior, imagina ¿qué pasaría si un usuario inserta una calificación de 150? El programa seguiría funcionando y te imprimiría un mensaje de aprobado. Una manera de solventar esta situación sería adicionar una verificación de que el valor pasado esté en el rango de valores esperado, por ejemplo:

```
calificacion = 150
```

```
if calificacion < 0 or calificacion > 100:
    print("La calificación debe estar en la escala de 0 a 100")
elif calificacion >= 70:
    print("Aprobado")
else:
    print("No aprobado")
```

2.2 Operadores de Comparación

[ ]:
```
"""
--------------------------------
| Operador |     Descripción    |
--------------------------------
|    ==    | Igual que          |
|    !=    | Diferente          |
|     <    | Menor que          |
|     >    | Mayor que          |
|    <=    | Menor o igual que  |
|    >=    | Mayor o igual que  |
--------------------------------
"""
## Ejemplo:
## Programa que calcula el mayor de tres numeros sin operadores lógicos.

while True:
    aux = input("Ingresa 3 números: ")
    a, b, c = aux.split(" ")
    a = int(a)
    b = int(b)
    c = int(c)

    # Controlar finalización:
    if a == 0 and b == 0 and c == 0:
        break
    else:
        if a == b:
            if a == c:
                print(f"Son Iguales a={a} b={b} c={c}")
            elif a > c:
                print(f"Mayores: a={a} y b={b}")
            else:
                print(f"Mayor: c={c}")
        elif a > b:
            if a == c:
                print(f"Mayores: a={a} y c={c}")
            elif a > c:
```

```
                print(f"Mayor: a={a}")
            else:
                print(f"Mayor: c={c}")
        elif b > a:
            if b == c:
                print(f"Mayores: b={b} y c={c}")
            elif b > c:
                print(f"Mayor: b={b}")
            else:
                print(f"Mayor: c={c}")
```

```
Mayor: c=5
Mayor: c=7
Mayor: c=7
```

2.3 Operadores Lógicos

[ ]:
```
"""
Operadores Lógicos: Permiten realizar varias comparaciones en una expresión␣
 ↪lógica.
|-----------|-------------------|-----------------------------------------------|
|  Operador |    Descripción    | Forma de utilización                          ␣
 ↪|
|-----------|-------------------|-----------------------------------------------|
|  and (&)  | Conjunción Lógica | True cuando todas las condiciones se cumplen.|
|   or (|)  | Disyunción Lógica | False cuando ninguna condicionesse cumple.    |
|  XOR (^)  | O Exclusivo       | True solo cuando se cumple una condición.     |
|    not    | No Lógico         | True cuando todas las condiciones se cumplen  |
|-----------|-------------------|-----------------------------------------------|
"""
# Verificar si un número es mayor a cero y menor que diez:
n = int(input("Ingresa un número: "))
bandera = False

if n > 0 and n < 10:
    bandera = True
else:
    bandera = False

print(bandera)

# Verificar si un número es mayor a 10 o menor a -10:
n = int(input("Ingresa otro número: "))
bandera = False

if n > 10 or n < -10:
    bandera = True
```

```
else:
    bandera = False

print(bandera)

if not bandera:
    print("Upss bandera esta negada...")

if n > -10 & n < 10:
    print(f"El valor era {n}")

if n ^ 0: # En realidad este operador funciona como un !=
    print("Prueba XOR")
```

```
True
False
Upss bandera esta negada…
Prueba XOR
False
Upss bandera esta negada…
Prueba XOR
```

2.4 Bucle for

[ ]:
```
# Imprime 10 primeros números decimales, sabiendo que se inicia en cero.
print("Imprime los 10 primeros números decimales")
for i in range(0, 10):
    print(f"{i}\t", end="")


print("\nPrograma que lee varios nuemros separados por espacio y los muestra")
cad = input()
numeros = [int(num) for num in cad.split()]

print(numeros)
```

```
Imprime los 10 primeros números decimales
0       1       2       3       4       5       6       7       8       9
Programa que lee varios nuemros separados por espacio y los muestra
[1, 3, 2, 6, 5]
```

2.5 Bucle While

[ ]:
```
# Programa que genera la tabla de multiplicar de 1 a M de un número N.

n = int(input("Ingresar n: "))
m = int(input("Ingresar m: "))
```

```python
i = 0

while True:
    i+=1
    print(f"{n} * {i} = {n*i}")

    if i == m: break

# Ejemplo N=5; M=10
```

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```