Modulo 1

Introducción a la programación en Python

Centro de e-Learning
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Unidad 2: Tipos de datos y estructuras de control

Centro de e-Learning
UTN.BA

Organización del primer módulo

- **Clase 1:** Introducción al curso + Que es Python + Elección del editor de código + Definición de programa e instrucciones + Tipos de datos básicos

- **Clase 2:** Listas + Diccionarios + Tuplas + Estructuras de control condicionales y cíclicas + Funciones + Librería: NumPy

- **Clase 3:** Dataframes + Librería: Pandas + Concatenación y Merge de dataframes + Filtros + Limpieza de datos + Tipos de datos: fechas

- **Clase 4:** Visualización de datos + Principios generales del diseño analítico + Librería: Matplotlib + Gráficos mas populares

Tipos de datos: Listas

Las **listas** son una estructura de datos que permite almacenar una colección ordenada de elementos.

Se pueden pensar como una secuencia **mutable** de elementos, donde cada elemento puede ser de cualquier tipo (números, booleanos, cadenas de caracteres o incluso listas, etc…).

En **Python** las podemos reconocer ya que se declaran usando "**[ ]**"

```
In [1]:
lista_vacia = []

In [2]:
lista_de_enteros = [1,2,3,4,5]
print(lista_de_enteros)

[1, 2, 3, 4, 5]

In [3]:
lista_de_strings = ["Hola", "Mundo"]
print(lista_de_strings)

['Hola', 'Mundo']

In [4]:
lista_varios = [1,"texto",lista_de_enteros]
print(lista_varios)

[1, 'texto', [1, 2, 3, 4, 5]]

In [7]:
print(lista_de_enteros[0])

1
```

Tipos de datos: Diccionarios

Los **diccionarios** son otra estructura de datos que permiten almacenar y organizar datos de manera flexible. A diferencia de las listas, que están indexadas por números enteros, los diccionarios están indexados por **claves**.

Cada elemento en un diccionario consta de una **clave** y un **valor** asociado.

En **Python** los podemos reconocer ya que se declaran usando "**{ }**"

```
In [1]:
diccionario_vacio = {}

In [8]:
diccionario_persona = {"Nombre": "Juan", "Edad": 25}
print(diccionario_persona)

{'Nombre': 'Juan', 'Edad': 25}

In [9]:
print(diccionario_persona["Nombre"])

Juan

In [10]:
print(diccionario_persona["Edad"])

25

In [11]:
print(diccionario_persona.keys())
print(diccionario_persona.values())

dict_keys(['Nombre', 'Edad'])
dict_values(['Juan', 25])
```

Tipos de datos: Tuplas

Las **tuplas** son una estructura de datos similar a las listas, pero a diferencia de estas ultimas, las tuplas son **inmutables**, lo que significa que no se pueden modificar luego de ser creadas.

Se utilizan para almacenar colecciones ordenadas de elementos.

En **Python** los podemos reconocer ya que se declaran usando "**( )**"

```
In [12]:
tupla_vacia = ()

In [13]:
tupla_numeros = (1, 2, 3, 4, 5)
print(tupla_numeros)

(1, 2, 3, 4, 5)

In [14]:
tupla_strings = ("Hola", "Mundo")
print(tupla_strings)

('Hola', 'Mundo')

In [15]:
tupla_varios = (1, 2, 3, "texto", tupla_numeros)
print(tupla_varios)

(1, 2, 3, 'texto', (1, 2, 3, 4, 5))

In [16]:
print(tupla_numeros[0])
print(tupla_varios[3])

1
texto
```

Estructuras de control de flujo

Son el **conjunto de reglas** que permiten controlar el flujo de las acciones de un algoritmo o programa. Las mismas pueden clasificarse en secuenciales, condicionales e iterativas.

A continuación veremos:

- Condicionales: **IF**
  ELSE

- Ciclos: **WHILE**
  FOR

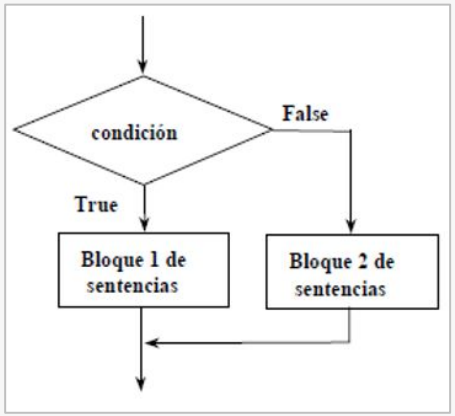

Diagrama de flujo: IF

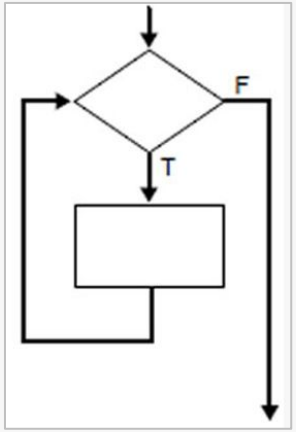

Diagrama de flujo: FOR

Condicionales: IF

La estructura general es la siguiente:

```
instruccion 1       # se ejecuta antes del if
if (condicion):
    instruccion 2   # podemos observar la identación
    instruccion 3   # significa que estamos dentro del if
instruccion 4       # salimos del if
```

Rama Condición
Sólo se ejecuta si `condición` es una expresión que evalúa a verdadero (TRUE).

Veamos un ejemplo! ->

```
In [23]:

numero = 10
if (numero > 0):
    print("¡Estamos adentro del if!")
    print(" El numero es:",numero,".")
print("¡Estamos afuera del if!")
```

```
¡Estamos adentro del if!
 El numero es: 10 .
¡Estamos afuera del if!
```

Condicionales: IF ELSE

La estructura general es la siguiente:

```
instruccion 1
if (condicion):
    instruccion 2  # podemos observar la identación
    instruccion 3  # significa que estamos dentro del if
else:
    instruccion 4  # estamos dentro del else
instruccion 5      # salimos del if/else
```

Rama Condición
Sólo se ejecuta si `condición` es una expresión que evalúa a verdadero (TRUE).

Rama Falsa
Sólo se ejecuta si la `condición` evalúa a falso (FALSE).

Veamos un ejemplo! ->

```
numero = 10
if (numero > 5):
    print("¡Estamos adentro del if!")
    print(" El numero es:", numero, ".")
else:
    print("¡Estamos adentro del else!")
    print(" El numero es mayor a 5!")
print("¡Salimos del else!")
```

```
¡Estamos adentro del if!
 El numero es: 10 .
¡Salimos del else!
```

Condicionales: FOR

La estructura general es la siguiente:

```
instruccion 1                # se ejecuta antes del for
for variable in interable:
    instruccion 2            # si todavia estamos en el iterable
    instruccion 3            # entonces se ejecutan estas instrucciones
instruccion 4                # salimos de la iteracion (for)
```

´**variable**´ es una variable que toma el valor de cada elemento en el ´iterable´ en cada iteración del bucle. El ´**iterable**´ es una secuencia de elementos, como una lista, tupla, cadena, rango, etc.

Veamos un ejemplo! ->

```
In [27]: lista = [1, 2, 3]
         for numero in lista:
             print(numero)
             print("¡Estamos dentro del for!")

         1
         ¡Estamos dentro del for!
         2
         ¡Estamos dentro del for!
         3
         ¡Estamos dentro del for!
```

Condicionales: WHILE

La estructura general es la siguiente:

```
instruccion 1                # se ejecuta antes del while
while condicion:
    instruccion 2            # si todavia se cumple la condición
    instruccion 3            # entonces se ejecutan estas instrucciones
instruccion 4                # salimos de la iteracion (while)
```

´**condicion**´ es una expresión booleana. Mientras la condición sea verdadera, el bloque de código dentro del bucle **while** se ejecutara repetidamente. Es importante asegurarse de que no se convierta en un **bucle infinito**!

Veamos un ejemplo! ->

```
In [29]: i = 0
         while i < 3:
             print(i)
             print("¡Estamos dentro del while!")
             i = i + 1 #Aumenta el valor de i en cada iteración

         0
         ¡Estamos dentro del while!
         1
         ¡Estamos dentro del while!
         2
         ¡Estamos dentro del while!
```

Funciones

Una **función** es un bloque de código que realiza una tarea especifica y puede ser **reutilizado** en diferentes partes de un programa. Las funciones ayudan a organizar y modularizar el código, lo que facilita su mantenimiento y comprensión.

En Python podemos definir funciones utilizando la palabra clave `**def**`. La estructura general es:

```
In [15]: def nombre_funcion(parametro1,parametro2,...):
             # Código de la función
             # Puede ser una o varias líneas de código
             return resultado
```

El **return** es una declaración opcional que permite que la función devuelva un resultado al lugar desde el cual fue llamada. Puede tener cero o mas declaraciones del tipo ´return´.

Funciones: Ejemplo

In [31]:
```
# Primero definimos a la funcion y la declaramos antes de utilizarla

def suma(numero1,numero2): # Se llama `suma` y recibe 2 parametros
    resultado = numero1 + numero2  # guardamos la suma en la variable ´resultado´
    return resultado   # Le pedimos que retorne el valor de ´resultado´

x = 10   # declaramos a x igual a 10
y = 20   # declaramos a y igual a 20

total = suma(x,y)   # invocamos a la funcion y como parametros usamos a X e Y

print("La suma de X e Y es igual a:", total)  # imprimimos el valor de la suma
print("Podemos llamar a la funcion tambien de esta manera, cuyo resultado es:", suma(11,22))
```

```
La suma de X e Y es igual a: 30
Podemos llamar a la funcion tambien de esta manera, cuyo resultado es: 33
```

NumPy

NumPy es una librería en Python utilizada para realizar operaciones numéricas y manipulación de arreglos (**arrays**) de manera eficiente. Proporciona estructuras de datos y **funciones** que son esenciales para la computación científica y el análisis de datos. Las características claves son:

- **Arrays N-dimensionales (ndarray):** NumPy introduce el objeto ´ndarray´ que es una estructura de datos eficiente para almacenar y manipular matrices multidimensionales.

```
In [33]: import numpy as np    #cargamos la libreria y la renombramos a "np" para usarla

         lista_numeros = [1, 2, 3]             # creamos una lista de numeros
         numpy_array = np.array(lista_numeros) # transformamos esa lista en un array

         print("La lista es:", lista_numeros)
         print("El NumPy array es:", numpy_array)

         La lista es: [1, 2, 3]
         El NumPy array es: [1 2 3]
```

NumPy

- **Operaciones Vectorizadas:** También permite realizar operaciones matemáticas y lógicas directamente sobre arrays, sin necesidad de utilizar bucles explícitos. Esto se conoce como **vectorizacion** y mejora la velocidad de ejecución de operaciones.

```
In [36]: import numpy as np     #cargamos la libreria y la renombramos a "np" para usarla

         lista_numeros = [4, 5, 6]       # creamos una lista de numeros

         array_a = np.arange(3)              # podemos crear un array de esta manera
         array_b = np.array(lista_numeros)  # transformamos esa lista en un array
         resultado = array_a + array_b      # hacemos la suma de ambos vectores

         print("El primer array es:", array_a)
         print("El segundo array es:", array_b)
         print("La suma de ambos es:" , resultado)

         El primer array es: [0 1 2]
         El segundo array es: [4 5 6]
         La suma de ambos es: [4 6 8]
```

NumPy

- **Funciones matemáticas y estadísticas:** NumPy proporciona un conjunto amplio de funciones matemáticas y estadísticas que pueden aplicarse de manera eficiente.

```
In [37]: print("El promedio es:", np.mean(array_a))
         print("El desvio estandar es:", np.std(array_a))

         El promedio es: 1.0
         El desvio estandar es: 0.816496580927726
```

- **Broadcasting**: facilita la operación entre arrays de diferentes formas y tamaños mediante un concepto llamado “broadcasting”, que extiende automáticamente los arrays para que tengan dimensiones compatibles

```
In [41]: print(np.array([[1, 2, 3], [4, 5, 6]]) + np.array([10, 20, 30]))

         [[11 22 33]
          [14 25 36]]
```