Tipos_Datos

January 9, 2026

INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Formamos tu PROPÓSITO DE VIDA

Introducción a Python

Nombre: Alejandro Bedoya

Comentarios

¿Qué son? Bueno, basicamente es un texto que está dentro del fichero donde el intérprete lo ignora y no es ejecutado

¿Para qué sirven? Sirven para poder documentar tu código y hacerlo más legible para otros programadores. En lo posible es tratar de hacer el código lo más fácil de entender y necesite pocos

comentarios, comentar linea por linea en un código que sea horrible puede ser fatal para las siguientes actualizaciones

¿Cómo se hacen los comentarios?

Para el código en una sola linea, se hace con #Hola soy un comentario

Para el código en bloque, se hace con “ “ “Hola, soy un comentario en bloque” “ ”

[60]:
```
#Esto es un comentario
print("Esto es un print complicado") #Se imprime el string
"""
Esto es un comentario en bloque, generalmente se lo usa para instrucciones
o también se lo suele usar para explicar de manera detallada algún método o␣
 ↪función
"""

"""
En esta función vamos a calcular el promedio entre dos datos (dato1 y dato2)
y retornaremos el resultado, para poder ejecutar la función simplemente
tenemos que hacer calcularPromedio(numero 1, numero 2)
"""
def calcularPromedio(dato1, dato2):
    promedio = (dato1 + dato2)/2
    return promedio

print("Hola, soy una función para sacar el promedio", calcularPromedio(40, 80))
```

```
Esto es un print complicado
Hola, soy una función para sacar el promedio 60.0
```

Literales, Variables y Tipos de Datos Básicos

Términos fundamentales

Objetos: Es cualquier tipo de dato (números, caracteres o datos más complejos)

Operaciones: Cualquier forma de manipular los datos (igual, sumar, restar, dividir, etc)

Ejemplo

[61]:
```
#4 es objeto de valor 4
#3 es objeto de valor 3
#+ es una operación (suma)
4+3
```

[61]: 7

Literales

Python tiene una serie de tipos de datos integrados en el propio lenguaje, basicamente son expresiones que generan objetos de estos tipos. ¿Qué objetos? bueno, pueden ser:

Simples o compuestos

Mutables o Inmutables

Literales Simples

Enteros

Decimales o floats

Booleanos

[62]:
```
print(4) #Entero (int)
print(6.9) #Float (float)
print("Hola, soy un string") #Cadena de texto (string)
print(False) #Booleano (boolean)
```

```
4
6.9
Hola, soy un string
False
```

Literales Compuestos

Listas:Se usan cuando quieres una colección ordenada donde se pueda modificar o recorrer de posición

Tuplas:Son inmutables, es decir, no se podrán modificar una vez que se coloquen de manera posterior y sirven cuando queremos valores fijos en esa colección

Set (Conjunto):Sirve cuando necesitas almacenar valores únicos y eliminar los duplicados

Diccionario:Sirve cuando necesitas guardar y buscar información usando clave - valor

[63]:
```
soy_Lista = [1,2,3,4,5] #Es mutable, tiene un orden y se puede recorrer de␣
 ↪posición
soy_Tupla = (1,2,3,4,5) #Es inmutable, sus valores son fijos después de ser␣
 ↪creado
soy_Set = {"Alejandro", "Alejandro", "Bedoya" , "Bedoya"} #
soy_Diccionario = {"nombre": "Alejandro", "apellido": "Bedoya"}

#type es para ver qué tipo de dato es
print(type(soy_Lista))
print(type(soy_Tupla))
print(type(soy_Set))
print(type(soy_Diccionario))

#También hay otra forma de hacer tuplas y es cuando no tiene una variable y son␣
 ↪datos que están ahí xd
69, 4
```

```
<class 'list'>
<class 'tuple'>
<class 'set'>
<class 'dict'>
```

[63]:
```
(69, 4)
```

Variables

Hace referencia a los objetos, las variables y los objetos se almacenan en una zona de la memoria, siempre harán referencia a objetos y nunca a otras variables y además los Objetos en si pueden referenciar a otros como en las listas

Sintaxis:

nombre_variable = objeto

[64]:
```
#Asignación de una variable
variable_uno = 12
print(variable_uno)
```

```
12
```

[65]:
```
numero_entero = 4 #int
numero_decimal = 4.1 #float
cadena_texto = "Soy un texto" #string
numero_complejo = 10+4j #Se usa para funciones imaginarias (no lo usaremos pero␣
 ↪es para tener conocimiento xd)
dato_booleano = True #boolean
dato_nulo = None #null/vacío

print(type(numero_entero))
print(type(numero_decimal))
print(type(cadena_texto))
print(type(numero_complejo))
print(type(dato_booleano))
print(type(dato_nulo))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'complex'>
<class 'bool'>
<class 'NoneType'>
```

Dado a que Python es lenguaje que contiene tipado dinámico, la misma variable puede cambiar de tipo de dato o contenido en momentos diferentes de la ejecución del programa

[66]:
```
primera_variable = 12
print(primera_variable)
print(type(primera_variable))
```

```
12
<class 'int'>
```

[67]:
```
primera_variable = "Ahora ya no soy un número"
print(primera_variable)
print(type(primera_variable))
```

```
Ahora ya no soy un número
<class 'str'>
```

[68]:
```
primera_variable = 14.2
print(primera_variable)
print(type(primera_variable))
```

```
14.2
<class 'float'>
```

Garbage Collection: Basicamente quiere decir que las funciones reemplazadas son borradas, por ejemplo: la variable "primera_variable" con objeto de valor 12 será borrada porque ya se reemplazo por el tipo de dato string "Ahora ya no soy un número"

Identificadores

Podemos tener un identificador único para los objetos referenciados en las variables y se obtiene a partir de la dirección de memoria

[69]:
```
"""
Aquí cada identificador va a cambiar debido a que es la misma variable pero
el tipo de dato y valor es distinto
"""

variable_id = 12
print("Este es el primer ID",id(variable_id))

variable_id = "Hola, seré un ejemplo para ID"
print("Este es el segundo ID", id(variable_id))

variable_id = 3.141632
print("Este es el tercer ID", id(variable_id))
print()
"""
Las variables que referencian al mismo objeto pueden tener el mismo ID

"""
variable_compartida = 4567
variable_diferente = variable_compartida
variable_random = 4567

print("Esta variable será la inicial: ",id(variable_compartida))
print("Esta variable será del mismo ID: ",id(variable_diferente))
#Esta tiene un ID diferente debido que cuando se hace "variable_diferente =␣
 ↪variable_compartida" su valor ya no es el mismo de antes
```

```
print("Esta variable tendrá un ID diferente: ",id(variable_random))
print()

"""
También debemos tomar en cuenta que los resultados se guardan en la memoria en base a los bits
¿Qué pasaría si supera los 256 bits (el num máximo de bits)? Simplemente se reinicia y se genera un nuevo ID
"""
#Pueden tener el mismo valor pero el id será distinto porque se reinicia la cantidad de bits
variable_grande = 258
segunda_variable_grande = 258

print("Primer resultado de variable grande: ",id(variable_grande))
print("Segundo resultado de variable grande: ",id(segunda_variable_grande))
print()
"""
¿Cómo podría hacer que sean iguales? Simplemente igualando entre variables

"""
variable_grandota = 400
segunda_variable_grandota = variable_grandota
tercera_variable_grandota = segunda_variable_grandota

print("Primera variable grande con mismo ID: ",id(variable_grandota))
print("Segunda variable grande con mismo ID: ",id(segunda_variable_grandota))
print("Tercera variable grande con mismo ID: ",id(tercera_variable_grandota))
```

```
Este es el primer ID 140708103042312
Este es el segundo ID 1533154101936
Este es el tercer ID 1533146598096

Esta variable será la inicial:  1533154056880
Esta variable será del mismo ID:  1533154056880
Esta variable tendrá un ID diferente:  1533154054288

Primer resultado de variable grande:  1533154057680
Segundo resultado de variable grande:  1533154057360

Primera variable grande con mismo ID:  1533154057456
Segunda variable grande con mismo ID:  1533154057456
Tercera variable grande con mismo ID:  1533154057456
```

Comentarios

¿Qué son? Bueno, basicamente es un texto que está dentro del fichero donde el intérprete lo ignora y no es ejecutado

¿Para qué sirven? Sirven para poder documentar tu código y hacerlo más legible para otros programadores. En lo posible es tratar de hacer el código lo más fácil de entender y necesite pocos comentarios, comentar linea por linea en un código que sea horrible puede ser fatal para las siguientes actualizaciones

¿Cómo se hacen los comentarios?

Para el código en una sola linea, se hace con #Hola soy un comentario

Para el código en bloque, se hace con “ “ “Hola, soy un comentario en bloque” “ ”

Asignación de Múltiples variables

Se pueden asignar variables en una sola linea en vez de ir linea por linea

[70]:
```
num1, num2, num3 = 1,2,3
print(num1,num2,num3)
print()
```

```
1 2 3
```

[71]:
```
#Otra forma de hacerlo es
conjunto = num1,num2,num3,14,2.5,"Hola, soy un string"
print(conjunto)
#Se hace una tupla debido a que hay multiples tipos de datos (int, float y el␣
 ↪string)
print(type(conjunto))
print()

"""
Las variables también pueden tener intercambio de valores

"""
variable_principal = 12
variable_secundaria = 22
#12 , 22 = 22 , 12
variable_principal,variable_secundaria = variable_secundaria,variable_principal
print(variable_principal, variable_secundaria)
print()
```

```
(1, 2, 3, 14, 2.5, 'Hola, soy un string')
<class 'tuple'>

22 12
```

Tipos de Datos Básicos

Booleano

[72]:
```
#Primera forma de hacer booleanos
primer_bool = True
segundo_bool = False

#Otra forma de hacerlos pero solo funciona con 0 y 1
tercer_bool = bool(1) #True
cuarto_bool = bool(0) #False

print(primer_bool)
print(type(primer_bool))
print(segundo_bool)
print(type(segundo_bool))
print(tercer_bool)
print(type(tercer_bool))
print(cuarto_bool)
print(type(cuarto_bool))

#Comparaciones
print(primer_bool is segundo_bool ) #Compara el ID en la memoria
print(primer_bool == segundo_bool ) #Compara valores
print()

print('Números')
print(2) #Número Entero
print(3.4) #Número Float
print(1/2) #Resultado en números racionales
print()

#Mismo valor, diferentes representaciones
entero = 58
binario =  0b00111010
octal = 0o72
hexadecimal = 0x3A

print(entero == binario == octal == hexadecimal) #true porque todos son 58 en␣
 ↪sus respectivos formatos
```

```
True
<class 'bool'>
False
<class 'bool'>
True
<class 'bool'>
False
<class 'bool'>
False
False
```

```
Números
2
3.4
0.5

True
```

Strings (Cadenas de Texto)

Son una cadena de caracteres en secuencias, eso quiere decir que la posición de los caracteres es importante y además de eso, son inmutables, las operaciones sobre strings no cambian el string original

Ejemplo:

[73]:
```
soy_string = "Soy Alejandro Bedoya"
print(soy_string[0]) #Sirve para ver el primer lugar de nuestro string
print(soy_string[-1]) #Sirve para ver el último lugar de nuestro string
#sintaxis: empieza en n hasta n2 (excluyendolo) y tiene un salto de n números
print(soy_string[0: 5 : 2]) #S y A
print(soy_string[ : ]) #Para que se vea todo el string
print(soy_string + " y tengo 20 años") #Concatenar: suma de strings
```

```
S
a
SyA
Soy Alejandro Bedoya
Soy Alejandro Bedoya y tengo 20 años
```

Conversión entre tipos (Parsear)

A veces, en el proceso de nuestro software, vamos a tener que hacer un cambio de tipo de dato de la versión actual a una que nosotros queramos. Un ejemplo sería al momento de usar input; el input devuelve el valor en strings, pero si necesitamos una cantidad deberemos convertirlo a int o float dependiendo del caso. Para eso sirve esta conversión.

[74]:
```
valor1 = int(2.8) #Solo devolverá el entero
valor2 = int("3") #Debe ser si o si un número en caso de un string
valor3 = float (14) #Si es entero, devolverá con un .0 (de 3 a 3.0)
valor4 = bool(1) #Solo funciona con 1 y con 0, 1 verdadero 0 falso
valor5 = bool(None) #Se hace falso porque no hay nada

print(valor1)
print(type(valor1))

print(valor2)
print(type(valor2))

print(valor3)
print(type(valor3))
```

```
print(valor4)
print(type(valor4))

print(valor5)
print(type(valor5))

print()
print("Tipos de Divisiones")
print(7/4)#División Normal : 1.75
print(7//4) #División absoluta, solo devuelve el entero : 1
print(int(7/4)) #Lo mismo que arriba pero parseando
```

```
2
<class 'int'>
3
<class 'int'>
14.0
<class 'float'>
True
<class 'bool'>
False
<class 'bool'>

Tipos de Divisiones
1.75
1
1
```

Operadores Aritméticos

Operador

Descripción

a + b

Suma

a - b

Resta

a / b

División

a // b

División Absoluta

a % b

Modulo / Resto

a * b

Multiplicacion

a ** b

Exponenciación

[75]:
```
print("Operadores Aritméticos")

primer_numero = 3
segundo_numero = 2

print("la suma de los números es: ", primer_numero + segundo_numero )
print("la resta de los números es: ", primer_numero - segundo_numero )
print("la multiplicación de los números es: ", primer_numero * segundo_numero )
print("la división de los números es: ", primer_numero / segundo_numero )
print("la división absoluta de los números es: ", primer_numero //␣
 ↪segundo_numero )
print("el módulo de los números es: ", primer_numero % segundo_numero )
print("la potencia de los números es: ", primer_numero ** segundo_numero )
```

```
Operadores Aritméticos
la suma de los números es:  5
la resta de los números es:  1
la multiplicación de los números es:  6
la división de los números es:  1.5
la división absoluta de los números es:  1
el módulo de los números es:  1
la potencia de los números es:  9
```

Operadores de Comparación

Operador

Desc

a > b

Mayor

a < b

Menor

a == b

Igualdad

a != b

Desigualdad

a >= b

Mayor o Igual

a <= b

Menor o Igual

[76]:
```
comparativa1 = 10
comparativa2 = 12

print("La primera variable es > que la segunda? ", comparativa1>comparativa2)␣
 ↪#var 1 es mayor
print("La primera variable es < que la segunda? ", comparativa1<comparativa2)␣
 ↪#var 2 es menor
print("La primera variable es == que la segunda? ", comparativa1==comparativa2)␣
 ↪#var 1 no es igual a var 2
print("La primera variable es >= que la segunda? ", comparativa1>=comparativa2)␣
 ↪#var 1 no es igual o mayor a var 2
print("La primera variable es <= que la segunda? ", comparativa1<=comparativa2)␣
 ↪#var 1 es igual o menor a var 2
print("La primera variable es != que la segunda? ", comparativa1!=comparativa2)␣
 ↪#var 1 no es igual a var 2
```

```
La primera variable es > que la segunda?  False
La primera variable es < que la segunda?  True
La primera variable es == que la segunda?  False
La primera variable es >= que la segunda?  False
La primera variable es <= que la segunda?  True
La primera variable es != que la segunda?  True
```

Operadores Lógicos

Operador

Desc

a and b

True, si ambos son True

a or b

True, si alguno de los dos es True

a ^ b

XOR - True, si solo uno de los dos es True

not a

Negación

[77]:
```
decision1 = True
decision2 = False

print("decision1 and decision2 son verdadero? ", decision1 and decision2) #Las␣
 ↪dos deben cumplirse para dar true
```

```
print("decision1 or decision2 son verdadero? ", decision1 or decision2) #Al␣
 ↪menos una de las dos debe ser verdadero
print('Al menos una de las dos variables es verdadero? ', decision1 ^ decision2␣
 ↪)# Es verdadero si al menos uno de los dos es verdadero
print("decision1 no es verdadero ", not decision1) #Simplemente cambia el valor
```

```
decision1 and decision2 son verdadero?  False
decision1 or decision2 son verdadero?  True
Al menos una de las dos variables es verdadero?  True
decision1 no es verdadero  False
```

Operadores de Asignación

Operador

Desc

=

Asignación

+=

Suma y asignación

-=

Resta y asignación

*=

Multiplicación y asignación

/=

División y asignación

%=

Módulo y asignación

//=

División entera y asignación

**=

Exponencial y asignación

&=

And y asignación

|=

Or y asignación

^=

Xor y asignación

»=

Despl. Derecha y asignación

«=

DEspl. Izquierda y asignación

[78]:
```
asignada1 = 2
asignada1 *= 3 #asignada = asignada * 3
asignada1 += 1 #No existe a++ o a-- como en otros lenguajes
print(asignada1)
```

```
7
```

[79]:
```
asignada2 = 6
asignada2 -= 2 #b = b-2
print(asignada2)
```

```
4
```

Operadores de Identidad

Operador

Desc

a is b

True, si ambos operadores son una referencia al mismo objeto

a is not b

True, si ambos operadores no son una referencia al mismo objeto

[83]:
```
identidad1 = 4444
identidad2 = identidad1
print(identidad1 is identidad2) #Identidad 1 es identidad 2? en base a memoria␣
 ↪= verdadero
print(identidad1 is not identidad2) # Identidad 1 no es identidad2? = falso, si␣
 ↪es identidad 2
```

```
True
False
```

Operadores de Pertenencia

Operador

Desc

a in b

True, si a se encuentra en la secuencia b

a not in b

True, si a no se encuentra en la secuencia b

[85]:
```
pertenencia1 = "Soy Alejandro"
pertenencia2 = {1: "a", 2: "b"} #Diccionario

print("S" in pertenencia1) #Aquí saldrá verdadero, en string se busca por índice
print("alejandro" not in pertenencia1) #Aquí también es verdadero, recuerda que␣
 ↪distingue entre mayúsculas y minúsculas

print()

print(1 in pertenencia2) #Verdadero, en los diccionarios se busca por clave, no␣
 ↪por valor
print("a" in pertenencia2) #Falso, no es válido es valor para esto, solo la␣
 ↪clave que sería 1 o 2
```

```
True
True

True
False
```

Entrada de Valores

Aquí nosotros usamos el input que nos ayudará a recibir los datos por teclado del usuario. Es importante conocer que los datos que se reciben por default en input es de tipo string.

[86]:
```
print("Programa para poder dividir dos números")

#Para poder tener los datos en enteros, necesitamos parsear
num1 = int(input("Escriba el primer número"))
num2 = int(input("Escriba el segundo número"))

print(f"La división entre los dos números es : {num1 / num2}")
```

```
Programa para poder dividir dos números

Escriba el primer número 12
Escriba el segundo número 2

La división entre los dos números es : 6.0
```

Tipos de Compuestos (Colecciones)

Listas

Se usa cuando necesites una colección ordenada, modificable y que permita elementos repetidos.

[87]:
```
#1) Formas de hacer lista
#Una lista es un conjunto de datos
listaUno = list()
listaDos = []
```

```
print(len(listaDos))

edades = [23,22,52,56,64,60,69]
print(len(edades))

#También se pueden mezclar los tipos de datos
informacion = ["Alejandro", "Bedoya", 23, 45.32, 25, 25]
print(informacion)
print(type(informacion))

#Se puede buscar por posición
print(informacion[0]) #Alejandro
print(informacion[1]) #Bedoya
print(informacion[-1]) #45.32 ---- este sirve para buscar el final

#Se puede contar el número de veces que hay algo
print(informacion.count(25))#El 25 se repite 2 veces

#También podemos asignar variables a los valores de la tabla
nombre,apellido,edad,dinero,num1,num2 = informacion
print(nombre) #Alejandro
print(apellido) #Bedoya
print(edad) #23
print(dinero) #45.32
print(num1) #25
print(num2) #25

#Añadir nuevo elemento, siempre irá al final
informacion.append("verdura")
print(informacion)

#Para poder especificar en dónde queremos añadir
informacion.insert(2, "añadirIndice") #posición , dato
informacion.insert(3, "añadirIndice2") #posición , dato
print(informacion)

#Información del índice
informacionNueva = ["Hola", "Soy", "hola"]

print(informacionNueva.index("Soy")) #Posición 1
print(informacionNueva.index("Hola")) #Posición 0
print(informacionNueva.index("hola")) #Posición 2

#Remover el item
informacion.remove("añadirIndice")
print(informacion)
```

```python
#Remover el último valor
informacion.pop()
print(informacion)

#también se puede especificar cual quieres hacer
informacion.pop(1)
print(informacion)

#Otra manera de eliminar
del informacion[0]
print(informacion)

#Limpiar los datos de la lista
informacion.clear()
print(informacion)

#Podemos ordenar al revés
informacion = ["Alejandro", "Bedoya", 23, 45.32, 25, 25]
informacion.reverse()
print(informacion)

#otro orden
edades.sort()
print(edades)

#Buscar entre atributos
informacionImportante = ["No soy importante","addrriel@hotmail.com",␣
 ↪"contraseña", "no soy importante"]
print(informacionImportante[1:3])
```

```
0
7
['Alejandro', 'Bedoya', 23, 45.32, 25, 25]
<class 'list'>
Alejandro
Bedoya
25
2
Alejandro
Bedoya
23
45.32
25
25
['Alejandro', 'Bedoya', 23, 45.32, 25, 25, 'verdura']
['Alejandro', 'Bedoya', 'añadirIndice', 'añadirIndice2', 23, 45.32, 25, 25,
'verdura']
1
```

```
0
2
['Alejandro', 'Bedoya', 'añadirIndice2', 23, 45.32, 25, 25, 'verdura']
['Alejandro', 'Bedoya', 'añadirIndice2', 23, 45.32, 25, 25]
['Alejandro', 'añadirIndice2', 23, 45.32, 25, 25]
['añadirIndice2', 23, 45.32, 25, 25]
[]
[25, 25, 45.32, 23, 'Bedoya', 'Alejandro']
[22, 23, 52, 56, 60, 64, 69]
['addrriel@hotmail.com', 'contraseña']
```

Diccionarios

Son una colección que destacan porque se hacen parejas entre clave y valor. Pueden ser modificadas y, desde la versión 3.7 de Python, ya están ordenadas. Se pueden ver como listas indexadas por cualquier objeto inmutable, no necesariamente por números enteros. A diferencia de las listas, esto no es una secuencia, es un mapping.

[93]:
```
#Creación de un diccionario simple, con expresión literal
cliente = {"cedula": "1726159260", "nombre": "Alejandro", "edad": 20}
print(cliente)
print()

#Creación uniendo dos colecciones
nombres = ["Pablito", "Clavo un" , "Clavito"]
edades = [12, 24, 21]

#Primero lo convertimos a diccionario el valor final, luego juntamos la clave y␣
 ↪luego el valor (clave, valor)
datos_conjunto = dict(zip(nombres, edades)) #{'Pablito': 12, 'Clavo un': 24,␣
 ↪'Clavito': 21}
print(datos_conjunto)
print()

#Creación pasando claves y valores a la función "dict"
persona2 = dict(nombre = "Alejandro", apellido = "Bedoya") #Aquí ponemos las␣
 ↪variables para transformar en diccionario
print(persona2)
print()

#Creación usando una lista de tuplas de dos elementos
persona3 = dict(
    #Aquí viene la lista
    [
        #Aquí vienen las tuplas
        ('nombre', 'Rosa'), ('apellido','Garcia')
    ]
)
```

```
#de manera horizontal se ve así
#persona2 = dict([('nombre', 'Rosa'), ('apellido','Garcia')])
print(persona3)
```

```
{'cedula': '1726159260', 'nombre': 'Alejandro', 'edad': 20}

{'Pablito': 12, 'Clavo un': 24, 'Clavito': 21}

{'nombre': 'Alejandro', 'apellido': 'Bedoya'}

{'nombre': 'Rosa', 'apellido': 'Garcia'}
```

[101]:
```
#Creación incremental por medio de asignación
#como las claves no existen, se crean nuevos items

ciudadano = {}
"""

Sintaxis : variable [clave] = "valor"

"""
#Ese valor le vamos asignar en forma de clave y damos un valor
ciudadano['cedula'] = '11111111D'
#Ese valor le vamos asignar en forma de clave y damos un valor
ciudadano['Nombre'] = 'Carlos'
#Ese valor le vamos asignar en forma de clave y damos un valor
ciudadano['Edad'] = 34
#Ese valor le vamos asignar en forma de clave y damos un valor
ciudadano['celular'] = '0939863058'

print(ciudadano)
print()

#Para buscar un valor a través de la clave
print(ciudadano["Nombre"])

#ojo: el acceso a claves que no existen o por índice dará error

#Modificación de un valor a través de la clave (lo que hicimos arriba)
ciudadano["Nombre"] = "Adriel" #Cambiamos por mi nombre
ciudadano["Edad"] += 1 # edad = edad + 1
print(ciudadano)

#Ejemplo Clave
diccionarioDatos = {"Nombre": "Alejandro Bedoya",
                    "Edad": 23,
```

```
                    "Ciudad": "Quito",
                    "Dinero": 20.15}
print(len(diccionarioDatos))
#Ejemplo Valor
diccionarioValor = {1:"Adriel", 2: "Suárez"}
print(len(diccionarioValor))

#Se pueden mezclar
diccionarioMezclado = {"Pais": "Ecuador", 1: "Quito", 2: "La Carolina",␣
 ↪"Residencia": "Casa"}
print(diccionarioMezclado)

#Se pueden hacer tipos de datos embebidos
diccionarioInterno = {
    "información": {"nombre", "apellido", "no sé que poner xd"} #set dentro de␣
 ↪diccionario
}

print(len(diccionarioInterno))

#Para buscar por clave o valor
print(diccionarioMezclado["Pais"])#clave
print(diccionarioMezclado[2])#valor

#Cambiar el resultado
diccionarioMezclado["Pais"] = "Brazuca"
print(diccionarioMezclado) #ahora es brazuca y no Ecuador

#Para visualizar por clave (ojo, si pones el valor no funcionará; solo es la␣
 ↪clave)
print("Quito" in diccionarioMezclado)#sale error porque es un valor
print("Pais" in  diccionarioMezclado)#sale true porque esa clave si existe

#Listado de los items
print(diccionarioMezclado.items()) #(clave: valor)

#Mostrar solo las claves
print(diccionarioMezclado.keys())

#Mostrar solo los valores
print(diccionarioMezclado.values())

#Crear un diccionario nuevo pero que no tiene valores
nuevoDiccionario = diccionarioMezclado.fromkeys(("Nombre", "Apellido", "edad",␣
 ↪"esSapo?"))
print(nuevoDiccionario)
```

```
{'cedula': '11111111D', 'Nombre': 'Carlos', 'Edad': 34, 'celular': '0939863058'}

Carlos
{'cedula': '11111111D', 'Nombre': 'Adriel', 'Edad': 35, 'celular': '0939863058'}
4
2
{'Pais': 'Ecuador', 1: 'Quito', 2: 'La Carolina', 'Residencia': 'Casa'}
1
Ecuador
La Carolina
{'Pais': 'Brazuca', 1: 'Quito', 2: 'La Carolina', 'Residencia': 'Casa'}
False
True
dict_items([('Pais', 'Brazuca'), (1, 'Quito'), (2, 'La Carolina'),
('Residencia', 'Casa')])
dict_keys(['Pais', 1, 2, 'Residencia'])
dict_values(['Brazuca', 'Quito', 'La Carolina', 'Casa'])
{'Nombre': None, 'Apellido': None, 'edad': None, 'esSapo?': None}
```

Sets (Conjuntos)

Es un conjunto de datos, se diferencian porque no es una estructura ordenada y tampoco se pueden repetir los datos. Se usa cuando necesites una colección sin orden, sin duplicados y para hacer operaciones de conjuntos (unión, intersección, etc.).

[102]:
```python
primerSet = set()
segundoSet = {"Primer Dato", "Segundo dato"} #El diccionario no tiene los datos␣
 ↪seguidos, es un conjunto de datos
tercerSet = {} #Diccionario
print(type(primerSet))
print(type(segundoSet))
print(type(tercerSet))

#Length par alos elementos dentro
print(len(segundoSet))

#Añadir un elemento al inicio
primerSet.add("Primer Dato")
print(primerSet)

#Verificar que el dato existe
print("Primer Dato" in segundoSet)#Verdadero porque si hay eso
print("Tercer Dato" in segundoSet)#Falso porque nunca se añadió ni escribió eso

#Eliminar un dato
primerSet.remove("Primer Dato")
print(primerSet)
```

```
#Borrar todos los elementos del set
cuartoSet = {1,2,3,4,5,6,7}
cuartoSet.clear() #Borramos los datos dentro del set
print(cuartoSet)
print(len(cuartoSet)) #Saldrá 0 porque no tiene valores dentro

#Unir dos sets
setPrueba = {"tu", "ñaña"}
setPrueba2 = {"calla", "sapo"}
unionSets = setPrueba.union(setPrueba2)
print(unionSets)

#Quitamos los elementos
print(unionSets.difference(setPrueba2))
```

```
<class 'set'>
<class 'set'>
<class 'dict'>
2
{'Primer Dato'}
True
False
set()
set()
0
{'sapo', 'tu', 'calla', 'ñaña'}
{'tu', 'ñaña'}
```

Secuencias

Son objetos que se pueden recorrer uno por uno, pero no almacenan todos los valores desde un inicio (Por eso se les llama iterables no materializados).

Aunque se parecen a las listas, no son listas reales como range() o enumerate(). Estos objetos no guardan valores, solo se sabe generar cuando se recorren

Si te gustaría materializar una secuencia

[129]:
```
soy_ejemplo = list(range(5))
print(soy_ejemplo) #cuando los transformas en lista se guardan en la memoria

#Por otro lado, el enumerate sirve para poder poner el índice y el valor
soy_ejemplo2 = enumerate(["a", "b", "c"])
print(list(soy_ejemplo2))    # [(0, 'a'), (1, 'b'), (2, 'c')]
#si te diste cuenta, se hacen en tuplas porque son valores que no se podrán␣
 ↪cambiar

# enumerate para iterar una colección (índice y valor)
lista = [10, 20, 30]
```

```python
#vamos recorriendo por cada i, el valor será un espacio del index en la lista
#index = nIndex. Value = n
for i, value in enumerate(lista):
    print('Index = ' + str(i) + '. Value = ' + str(value))

print()

#Otra forma de hacerlos es mediante dos datos, uno será la clave y el otro será␣
 ↪el valor
for index, value in [(3,10), (4,20), (5,30)]:
 print('Index = ' + str(index) + '. Value = ' + str(value))

print()

#enumerate para iterar una colección (índice y valor)
#en el range le estamos diciendo que empiece desde el índice 0 hasta el 9 y que␣
 ↪salte de 2 en 2
for index, value in enumerate(range(0,10,2)):
 print('Index = ' + str(index) + '. Value = ' + str(value))

print()

# zip para unir, elemento a elemento, dos colecciones, retornando lista de␣
 ↪tuplas
# útil para iterar dos listas al mismo tiempo
#unimos nombre y edad, hacemos un for para que vaya recorriendo por índice y␣
 ↪luego lo agregamos
nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34,45]
for nombre,edad in zip(nombres,edades):
  print('Nombre: ' + nombre + ', edad: ' + str(edad))

nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34]
for i in range(0,len(nombres)):
  print(f"Nombre es {nombres[i]} y edad es {edades[i]}")

print()

nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34]
jugadores = zip(nombres,edades) # genera secuencia
print(list(jugadores))
print(type(jugadores))

print()
```

```
# listas de diferentes longitudes
nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34,44,33] #los valores restantes no se toman en cuenta
jugadores = zip(nombres,edades)
print(list(jugadores))

print()

# unzip
nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34]
alturas = [120, 130, 140]

jugadores = list(zip(nombres, edades, alturas))  # materializamos

print(jugadores)

ns, es, al = zip(*jugadores)

print(list(ns)) #nombres
print(list(es)) #edades
print(list(al)) #altura

print()

nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34]
dd = [23,34,45]

jugadores_z = zip(nombres,edades, dd) #se puede recorrer solo una vez
print(list(jugadores_z))
jugadores_uz = zip(*jugadores_z)
print(list(jugadores_uz))

print()

#El correcto sería así
nombres = ['Manolo','Pepe','Luis']
edades = [31,34,34]
dd = [23,34,45]

jugadores_z = list(zip(nombres, edades, dd)) #Creamos la lista y recorremos

print(jugadores_z)

jugadores_uz = list(zip(*jugadores_z)) #Luego vamos a crear una nueva lista y␣
 ↪recorremos
```

```
print(jugadores_uz)
```

```
[0, 1, 2, 3, 4]
[(0, 'a'), (1, 'b'), (2, 'c')]
Index = 0. Value = 10
Index = 1. Value = 20
Index = 2. Value = 30

Index = 3. Value = 10
Index = 4. Value = 20
Index = 5. Value = 30

Index = 0. Value = 0
Index = 1. Value = 2
Index = 2. Value = 4
Index = 3. Value = 6
Index = 4. Value = 8

Nombre: Manolo, edad: 31
Nombre: Pepe, edad: 34
Nombre: Luis, edad: 34
Nombre es Manolo y edad es 31
Nombre es Pepe y edad es 34
Nombre es Luis y edad es 34

[('Manolo', 31), ('Pepe', 34), ('Luis', 34)]
<class 'zip'>

[('Manolo', 31), ('Pepe', 34), ('Luis', 34)]

[('Manolo', 31, 120), ('Pepe', 34, 130), ('Luis', 34, 140)]
['Manolo', 'Pepe', 'Luis']
[31, 34, 34]
[120, 130, 140]

[('Manolo', 31, 23), ('Pepe', 34, 34), ('Luis', 34, 45)]
[]

[('Manolo', 31, 23), ('Pepe', 34, 34), ('Luis', 34, 45)]
[('Manolo', 'Pepe', 'Luis'), (31, 34, 34), (23, 34, 45)]
```

Ejercicios

Escribe un programa que muestre por pantalla la concatenación de un número y una cadena de caracteres. Para obtener esta concatenación puedes usar uno de los operadores explicados en este tema. Ejemplo: dado el número 3 y la cadena 'abc', el programa mostrará la cadena '3abc'.

Escribe un programa que muestre por pantalla un valor booleano que indique si un número entero N está contenido en un intervalo semiabierto [a,b), el cual establece una cota inferior a (inclusive)

y una cota superior b (exclusive) para N.

Escribe un programa que, dado dos strings S1 y S2 y dos números enteros N1 y N2, determine si el substring que en S1 se extiende desde la posición N1 a la N2 (ambos inclusive) está contenido en S2.

Dada una lista con elementos duplicados, escribir un programa que muestre una nueva lista con el mismo contenido que la primera pero sin elementos duplicados.

Escribe un programa que, dada una lista de strings L, un string s perteneciente a L y un string t, reemplace s por t en L. El programa debe mostrar la lista resultante por pantalla.

Escribe un programa que defina una tupla con elementos numéricos, reemplace el valor del último por un valor diferente y muestre la tupla por pantalla. Recuerda que las tuplas son inmutables; tendrás que usar objetos intermedios.

Dada la lista [1,2,3,4,5,6,7,8], escribe un programa que, a partir de esta lista, obtenga la lista [8,6,4,2] y la muestre por pantalla.

Escribe un programa que, dada una tupla y un índice válido i, elimine el elemento de la tupla que se encuentra en la posición i. Para este ejercicio, sólo puedes usar objetos de tipo tupla; no puedes convertir la tupla a una lista, por ejemplo.

Escribe un programa que obtenga la mediana de una lista de números. Recuerda que la mediana M de una lista de números L es el número que cumple la siguiente propiedad: la mitad de los números de L son superiores a M y la otra mitad son inferiores. Cuando el número de elementos de L es par, se puede considerar que hay dos medianas. No obstante, en este ejercicio consideraremos que únicamente existe una mediana.

Soluciones

[135]:
```
"""
Escribe un programa que muestre por pantalla la concatenación de un número y␣
 ↪una cadena de caracteres.
Para obtener esta concatenación puedes usar uno de los operadores explicados en␣
 ↪este tema.
Ejemplo: dado el número 3 y la cadena 'abc', el programa mostrará la cadena␣
 ↪'3abc'.
"""
#Creamos las variables
num1 = 20
caracter1 = "Tengo"
edadString = "años"
#No se puede sumar entre distintos tipos de datos pero podemos concatenar con␣
 ↪la ","
print(caracter1, num1, edadString) #Tengo 20 años
```

```
Tengo 20 años
```

[138]:
```
"""
Escribe un programa que muestre por pantalla
```

```
un valor booleano que indique si un número entero N está contenido en un␣
 ↪intervalo semiabierto [a,b), el cual establece una cota inferior a␣
 ↪(inclusive) y una cota superior b (exclusive) para N.
"""

#Instrucciones
#Piden hacer un programa que:
#Debo tener un número entero N
#Debo tener dos límites dos límites
#Verificar si N está dentro del intervalo [limite1, limite2)
#Muestre en pantalla un valor booleano (True o False)
primer_numero = 12
limite1 = 8
limite2 = 13
#Creo una variable respuesta donde me devolverá un true o false dependiendo el␣
 ↪número de comparación
respuesta = primer_numero >= limite1 and primer_numero <= limite2
print(respuesta)
```

```
True
```

[141]:

```
"""
Escribe un programa que, dado dos strings S1 y S2 y dos números enteros N1 y N2
Determine si el substring que en S1 se extiende desde la posición N1 a la N2␣
 ↪(ambos inclusive) está contenido en S2.
"""

#Instrucciones
#Dos strings: S1 y S2
#Dos números: N1 y N2
#Verificar si el primer string se extiende desde el primer número al segundo␣
 ↪número

S1 = "Hola soy"
S2 = "Goku"
N1 = 2
N2 = 4

substring = S1[N1:N2+1]
resultado = substring in S2

print("Substring:", substring)
print(resultado)


"""
Escribe un programa que, dado dos strings S1 y S2 y dos números enteros N1 y N2,
```

```
determine si el substring que en S1 se extiende desde la posición N1 a la N2
(ambos inclusive) está contenido en S2.

"""

S3 = "Hola"
S4 = "Carabola"
N3 = 12
N4 = 4
#variable = primer string[inicio:final+1]
#el +1 porque necesitamos tener el último número porque dice que ambos deben␣
 ↪estar incluidos
string_slice = S3[N3: N4+1]
respuestaxis = string_slice in S4
print(respuestaxis)
```

```
Substring: la
False
True
```

[142]:
```
"""
Dada una lista con elementos duplicados, escribir un programa que muestre una␣
 ↪nueva lista con el mismo contenido que la primera pero sin elementos␣
 ↪duplicados.
"""

#Lo que debería hacer es parsear la lista a un set
#recuerda que el set no admite elementos duplicados
lista_duplicada = [1,2,2,2,2,3,4,5,5,6,7,7,7,8]
set_yo_te_elijo = set(lista_duplicada)
print(set_yo_te_elijo)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

[150]:
```
"""
Escribe un programa que, dada una lista de strings L, un string s perteneciente␣
 ↪a L y un string t, reemplace s por t en L.
El programa debe mostrar la lista resultante por pantalla.
"""

#Información
#Lista de String llamada "L"
#tenemos una variable de string "s" que pertenece a L
#Tenemos una variable de string llamada "t"

L = ["Soy Martin", "Soy Alejandro", "perro", "gato", "mueble", "sal"]
s = "perro"
t= "tenedor"
```

```
#Ahora necesito recorrer todos los espacios mediante el indice de la lista
for i in range(len(L)):
    #Hago condicional en caso que encuentre la palabra que está en el string
    if L[i] == "perro":
        #en caso que la encuentre, le digo que en el índice donde encontró, le␣
 ↪reemplace por t
        L[i] = t
print(L)

"""
Escribe un programa que, dada una lista de strings A, un string variable_string␣
 ↪perteneciente a A
y un string variable_string2, reemplace variable_string por variable_string2 en␣
 ↪A.
El programa debe mostrar la lista resultante por pantalla.

"""

A = ["Soy Bardock", "Soy Kakaroto", "Soy Trunks", "Soy Piccolo", "Soy Goku"]
variable_string = "Soy Goku"
variable_string2 = "Soy Vegeta"

for i in range(len(A)):
    if A[i] == variable_string:
        A[i] = variable_string2

print(A)
```

```
['Soy Martin', 'Soy Alejandro', 'tenedor', 'gato', 'mueble', 'sal']
['Soy Bardock', 'Soy Kakaroto', 'Soy Trunks', 'Soy Piccolo', 'Soy Vegeta']
```

[151]:
```
"""
Escribe un programa que defina una tupla con elementos numéricos,
reemplace el valor del último por un valor diferente
y muestre la tupla por pantalla. Recuerda que las tuplas son inmutables.␣
 ↪Tendrás que usar objetos intermedios.
"""

#Instrucciones
#Crear una tupla de números
#Reemplazar el último valor de la tupla a otro, pero hay que tener en cuenta␣
 ↪que las tuplas son inmutables
#transformar a una lista
#luego haremos que en el último valor se asigne uno distinto
#damos print
```

```
tupla_numerica = (1,2,3)
lista_numerica = list(tupla_numerica)
lista_numerica[-1] = 4
volviendo_tupla = tuple(lista_numerica)
print(volviendo_tupla)
```

```
(1, 2, 4)
```

[164]:
```
"""
Dada la lista [1,2,3,4,5,6,7,8] escribe un programa que, a partir de esta␣
 ↪lista, obtenga la lista [8,6,4,2] y la muestre por pantalla.
"""
#Creo la lista
lista_ejercicio6 = [1,2,3,4,5,6,7,8]
#Primero saco en orden los ejercicios
lista_menor = lista_ejercicio6[1:7+1:2]
#Luego le transformo al revés
lista_reves = lista_menor[-1: -9: -1]
#print
print(lista_reves)
```

```
[8, 6, 4, 2]
```

[171]:
```
"""
Escribe un programa que, dada una tupla y un índice válido i,
elimine el elemento de la tupla que se encuentra en la posición i.
Para este ejercicio sólo puedes usar objetos de tipo tupla. No puedes convetir␣
 ↪la tupla a una lista, por ejemplo.
"""
#Instrucciones
#crear una tupla y crear el indice que queremos borrar
#Hacemos una busqueda por indices donde vamos a excluir al número de indice
#print
tupla_borrada = (1,2,3,4)
indice_borrar = 2
#Primero excluyo el índice que quiero que se vaya en una nueva tupla
#pero saldrá así (1,2)
#para obtener el 4, necesito simplemente añadir 1 al número que quise borrar,␣
 ↪que en este caso es 3+1 = 4
tupla_actualizada = tupla_borrada[:indice_borrar] +␣
 ↪tupla_borrada[indice_borrar+1:]
print(tupla_actualizada)
```

```
(1, 2, 4)
```

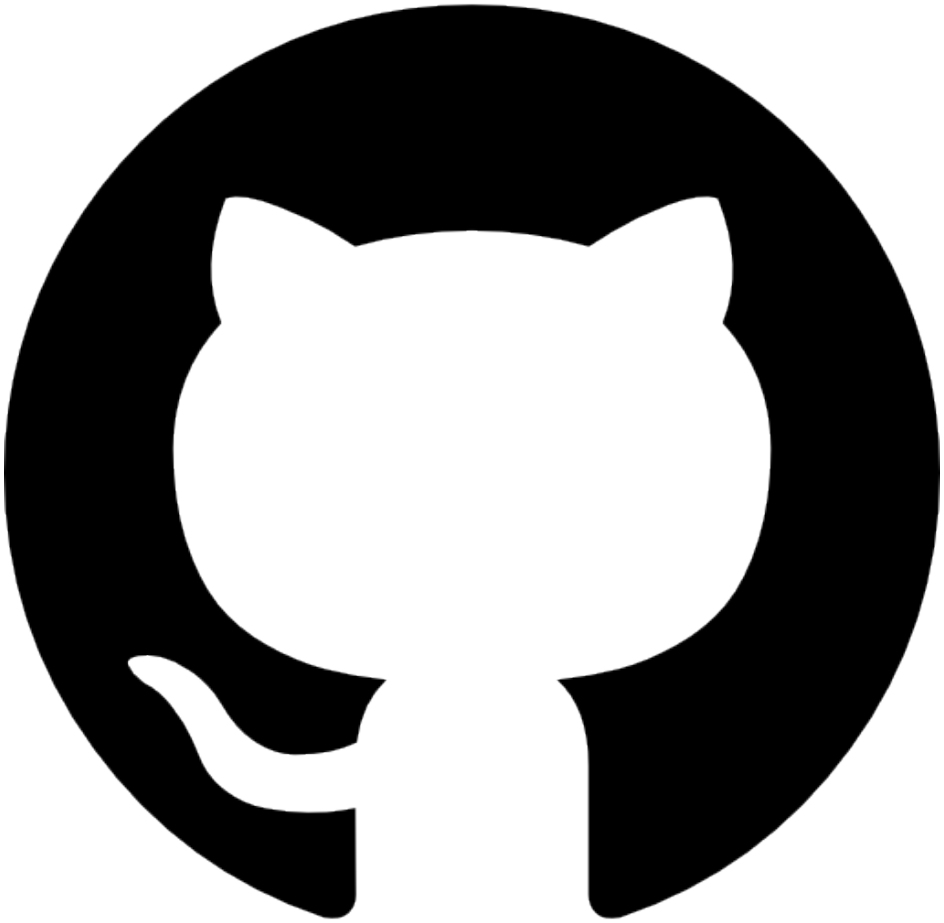
[175]:
```
"""
Escribe un programa que obtenga la mediana de una lista de números.
```

```
Recuerda que la mediana M de una lista de números L es el número que cumple la␣
 ↪siguiente propiedad:
la mitad de los números de L son superiores a M
y la otra mitad son inferiores.
Cuando el número de elementos de L es par, se puede considerar que hay dos␣
 ↪medianas.
No obstante, en este ejercicio consideraremos que únicamente existe una mediana.
"""

#Instrucciones
#Crear una lista
#Sacar la mediana
#Si es par, se considera que hay dos medianas
#El ejercicio solo considera una

#Proceso
#Crear la lista
age_awewes = [98,84,84,84,84,68,68,74,74,75,78,99,102,60,60,60,85,85]
#Ordenaremos los datos con sorted
age_awewes_ordenado= sorted(age_awewes)
#Calculamos la mediana: lo haremos mediante el índice del medio
indice_mediana = len(age_awewes_ordenado)//2
#Luego, instanciamos la mediana en una variable
mediana = age_awewes_ordenado[indice_mediana]
print("Ordenamos la lista de abuelitos: ", age_awewes_ordenado)
#recuerda contar desde el índice 0, ahí dará el 84
#Es el valor que está en el centro
#En caso que la mediana sea par, se suman y se dividen para 2
print("La mediana de awewes que tenemos es: ", mediana)

#Par
L = [7, 2, 5, 4, 9, 6]

# Ordenamos la lista
L_ordenada = sorted(L)

# Calculamos índice del medio
n = len(L_ordenada)

if n % 2 == 1: #si es impar entonces
    # Cantidad impar mediana es el elemento central
    mediana = L_ordenada[n // 2]
else: #si es par entonces
    # mediana es el promedio de los dos del medio
    mediana = (L_ordenada[n//2 - 1] + L_ordenada[n//2]) / 2

# Mostramos resultados
```

```
print("Lista ordenada:", L_ordenada)
print("Mediana:", mediana)
```

```
Ordenamos la lista de abuelitos:  [60, 60, 60, 68, 68, 74, 74, 75, 78, 84, 84,
84, 84, 85, 85, 98, 99, 102]
La mediana de awewes que tenemos es:  84
Lista ordenada: [2, 4, 5, 6, 7, 9]
Mediana: 5.5
```

Github



Puedes acceder al repositorio en: https://github.com/Addrriel/MachineLearningCuadernazo.git

En caso de que no funcione en el PDF, puedes usar directamente el enlace: https://github.com/Addrriel/MachineLearningCuadernazo.git

[ ]: