Practica 1: Tipos de datos

A continuación se muestran los tipos de datos nativos de Python. Se utilizarán las funciones *type* e *isinstance* pa verificar el tipo de datos

Tipo de datos Numéricos: enteros (int)

Los enteros se inicializan con literales numéricos sin parte decimal. La palabra reservada 'int', es el literal del tip dato entero.

```
numero = 5
tipoDatoNumero = type(numero)
print(tipoDatoNumero)
```

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

Verificamos que el tipo de dato de la variable 'numero', sea entera, comparando su tipo de dato, o usando la func *isinstance*.

```
#Operador binario de comparación ==, compara dos variables, y si ambos son iguales, su valor es T
esEntero = tipoDatoNumero == int
#isinstance recibe como entradas la variable y el  nombre del tipo de datos
esEntero2 = isinstance(numero, int)

print("Es entero? ", esEntero)
print("De verdad es entero? ", esEntero2)
```

```
Es entero?  True
De verdad es entero?  True
```

Tipos de datos numéricos: booleanos (bool)

Un dato de tipo booleano, o variable booleana puede tomar únicamente dos valores, True o False.

```
variableBooleana = True
print(variableBooleana)
#verificamos el tipo de datos
esBooleana = isinstance(variableBooleana, bool)
print("Es booleana?", esBooleana)
```

```
True
Es booleana? True
```

Los operadores relacionales como ==, >, <, <=, >=, dan como resultado siempre un valor booleano

```
operacionBooleana1 = 7 > 3
operacionBooleana2 = type(7.0) == int
print(operacionBooleana2)
```

```
False
```

Tipos de datos numéricos: Números reales (float)

Los números reales incluyen una parte decimal, por lo que posibilita la representación de fracciones.

```
numeroReal = 3.4
numeroReal2 = 3 / 5
#se puede usar notacion exponencial para representar numeros flotantes
numeroReal3 = 5e-3

print("contenido de la variable ", numeroReal3)
print("Tipo de datos de la variable ", type(numeroReal3))
```

```
contenido de la variable  0.005
Tipo de datos de la variable  <class 'float'>
```

Al realizar una operación con operandos numéricos de distintos tipos, el resultado tendrá el tipo de datos más preciso.

```
resultado = 3.0 + 1
print(resultado)
print("Tipo de datos del resultado: ", type(resultado))
```

```
4.0
Tipo de datos del resultado:  <class 'float'>
```

Conversión de datos

Para convertir una variable a otro tipo de datos, se invoca al constructor del tipo de datos, con el nombre del tipo datos y la variable a convertir como entrada.

```
resultadoEntero = int(resultado)
print("Resultado de conversion ", resultadoEntero)
```

```
Resultado de conversion  4
```

Tipo de datos numéricos: Números complejos (complex)

Los números complejos están compuestos por una parte real, y una imaginaria. La parte imaginaria se expresa sufijo 'i'.

```
numeroComplejo = 3 + 5j
print(numeroComplejo)
tipoDatosNumeroComplejo = type(numeroComplejo)
print("Tipo de datos de la variable ", tipoDatosNumeroComplejo)
```

```
(3+5j)
Tipo de datos de la variable  <class 'complex'>
```

El cuadrado de la variable imaginaria $j = \sqrt{-1}$ es $j^2 = -1$

```
imaginariaCuadrada = (1j) ** 2
print("j^2 = ", imaginariaCuadrada)
```

```
j^2 =  (-1+0j)
```

Secuencias inmutables: Hileras (Strings o str)

Las secuencias inmutables corresponden a series o secuencias de valores, no modificables. Una hilera represer una secuencia de caracteres, representados con el estandar ASCII

```
palabra = "hola"
palabra2 = 'casa'
print('Contenido en palabra ', palabra)
#tipo de datos
tipoDatosPalabra = type(palabra)
esHilera = tipoDatosPalabra == str
print("Es de tipo str? ", esHilera)
#Extraccion de un caracter
primeraLetra = palabra[0]
print("Primera letra: ", primeraLetra)
```

```
Contenido en palabra  hola
Es de tipo str?  True
Primera letra:  h
```

Las secuencias inmutables no permiten la modificacion de alguno de sus elementos

```
palabra[1] = "a"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-7963c5a222e5> in <module>()
----> 1 palabra[1] = "a"

TypeError: 'str' object does not support item assignment
```

SEARCH STACK OVERFLOW

Secuencias inmutables: Tuplas

Las tuplas son secuencias de valores con cualquier tipo de dato. Son inmutables.

```
#Tupla de enteros
tuplaEnteros = (1, 3, 4)
#tupla de numeros complejos
tuplaComplejos = (1 + 1j, 3 + 4j, 0 + 1j)
```

```
print("Tupla de complejos ", tuplaComplejos)
print("tercer elemento de la tupla ", tuplaComplejos[2])
#tupla de tuplas
tuplaDeTuplas = ((1, 2, 3), (1 + 1j, 58.9))
print("Tupla de tuplas ", tuplaDeTuplas)
#consultar el segundo elemento de la segunda tupla en la tupla
print("Segundo elemento de la segunda tupla", tuplaDeTuplas[1][1])
```

```
Tupla de complejos  ((1+1j), (3+4j), 1j)
tercer elemento de la tupla  1j
Tupla de tuplas  ((1, 2, 3), ((1+1j), 58.9))
Segundo elemento de la segunda tupla 58.9
```

Secuencias inmutables: Arreglo de bytes (bytearray)

Los arreglos de bytes son secuencias de bytes (8 bits, valores enteros de 0 a 255). El arreglo de bytes definido c antes del a hilera, usa el esquema ASCII, y no es modificable.

```
#formas de construir un arreglo de bytes
arregloBytesInmutable = b'abc'

print("Arreglo de bytes: ", arregloBytes)
print(arregloBytesInmutable[0])
```

```
Arreglo de bytes:  b'abc'
El otro arreglo de bytes  bytearray(b'hola')
Codificacion ascii de la primer letra del segundo arreglo de bytes  104
```

Secuencias mutables: Arreglo de bytes mutable (bytearray)

El tipo de datos *bytearray* es un arreglo de bytes mutable mutable.

```
#se definen dos parametros o argumentos, la hilera y el tipo de codificacion
arregloBytes2 = bytearray("hola", 'ascii')
print("Arreglo de bytes modificable", arregloBytes2)
print("Codificacion ascii de la primer letra del segundo arreglo de bytes ", arregloBytes2[0])
arregloBytes2[0] = 99
print("Arreglo de bytes modificado ", arregloBytes2)
```

```
Arreglo de bytes modificable bytearray(b'hola')
Codificacion ascii de la primer letra del segundo arreglo de bytes  104
Arreglo de bytes modificado  bytearray(b'cola')
```

Secuencias mutables: Lista (list)

Una lista es una secuencia de objetos de cualquier tipo, y es modificable. Las listas son indexables como las tup con valores de 0 a $n - 1$, donde n es el largo de la lista.

```
lista = [10, 2, 4, 5]
print("Lista ", lista)
```

```
#conversion de una tupla a una lista
lista2 = list(tuplaEnteros)
print("Lista2 ", lista2)


#lista de listas y tuplas
listaDeListasYTuplas = [(2, 3.5), ["hola", "nombre"], (1, 3)]
print("Segundo elemento", listaDeListasYTuplas[1])

#Hilera a lista de caracteres
lista4 = list("hola")
print("Lista de caracteres  ", lista4)
```

```
Lista  [10, 2, 4, 5]
Lista2  [1, 3, 4]
Segundo elemento ['hola', 'nombre']
Lista de caracteres   ['h', 'o', 'l', 'a']
```

Conjuntos mutables

Los conjuntos son "bolsas" de objetos, donde cada objeto es único, sin orden y no indizables.

```
conjuntoFrutas = {"naranja", "naranja", "limon", "sandia"}
print("Conjunto Frutas ", conjuntoFrutas)
conjuntoFrutas2 = {"limon", "melon", "papaya"}
#se llama la funcion intersection para calcular la interseccion de dos conjuntos
conjuntoInterseccion = conjuntoFrutas.intersection(conjuntoFrutas2)
print("Interseccion ", conjuntoInterseccion)
#se elimina el elemento limon del conjunto interseccion
conjuntoInterseccion.remove("limon")
print("Interseccion modificado ", conjuntoInterseccion)
```

```
Conjunto Frutas  {'limon', 'sandia', 'naranja'}
Interseccion  {'limon'}
Interseccion modificado  set()
```

Diccionarios

Los diccinarios son un conjunto de pares ordenados, donde el primer elemento se refiere como llave, y el segun valor correspondiente a tal llave.

```
preciosDeArticulos = {"Televisor":50, "Celular":300, "Refrigerador":1500}
print("Diccionario completo ", preciosDeArticulos)
print("Precio del televisor ", preciosDeArticulos["Televisor"], "libras esterlinas")
```

```
Diccionario completo  {'Televisor': 50, 'Celular': 300, 'Refrigerador': 1500}
Precio del televisor  50 libras esterlinas
```

Practica

1. Realice la operación $5 + 8 \times 10^{-3}$, y muestre el resultado en pantalla como un número complejo.
2. Realice la operación en números complejos "(5 + 3j) - (3 + 7j)", y muestre el resultado en pantalla de la par real únicamente.
3. Realice la operación True + True, y muestre el resultado con un número real.

4. Cree una variable de tipo hilera, para la palabra "muro", muestrela en pantalla, y cambie la letra "r", por la le "d", para mostrar la hilera "mudo".
5. Cree una lista de números complejos (5 + 3j), (3 + 7j), (8 + 3j), y súmelos todos. Muestre el resultado en pantalla.
6. Cree una hilera con el contenido "hola", y modifique los dos primeros caracteres, sumandole el valor de 10 código ASCII. Muestre el resultado en pantalla.
7. Cree una lista de listas, para almacenar los datos de la siguiente tabla:

Numero de canasta	Frutas
121	naranja, banano
452	sandia, papaya
320	melon, mango
455	piña, uva

8. Cree un diccionario de diccionarios, para representar el contenido de la siguiente tabla:

Casa	Caracteristicas
121	2 cuartos, 2 garajes
452	3 cuartos, 1 garaje
320	4 cuartos, 4 garajes
455	4 cuartos, 2 garajes

De forma que por ejemplo, para consultar la cantidad de cuartos de la casa 121, se realice lo siguiente:

diccionario[121]["cuartos"]

9. Cree un diccionario con las traducciones a portugués de las palabras: hola, yo, usted, ellos, ella, él, soy, so casa, vivo, vivimos, viven, en, una, ciudad, pueblo. Cree además una lista de hileras con la siguiente oració vivo en una ciudad", y emplee el diccionario construído para crear una nueva lista de hileras con la traduc portugués.

```
a = True + True
print(a)
```

```
2
```