Resumen para Quíz 1

September 23, 2022

1 AYUDANTÍA PYTHON

1.1 Tipos de datos

En Python existen distintos tipos de datos. Para saber a que elemento corresponde que tipo de dato, podemos utilizar la función **type()**. Sin embargo, también es posible detectar un tipo de dato si cumple ciertas caracteristicas.

1) int: Representa un número entero (Positivo, cero, negativo). Estos se escriben igual que en el lenguaje natural, solamente colocando el número y si es un número negativo, colocando el signo -

[1]:
```
type(10)
```

[1]:
```
int
```

[2]:
```
type(0)
```

[2]:
```
int
```

[3]:
```
type(-92)
```

[3]:
```
int
```

[4]:
```
type(-2147483648)
```

[4]:
```
int
```

2) float: Representa un número decimal positivo o negativo. Se denominan como número de "punto flotante". Estos se escriben similar al lenguaje natural, pero en vez de utilizar una coma para separar la parte real de la decimal, utilizamos un punto (.)

[5]:
```
type(0.1)
```

[5]:
```
float
```

[6]:
```
type(-9.7)
```

[6]:
```
float
```

[7]:
```
type(3.14159265359)
```

[7]:
```
float
```

[8]:
```
type(0.0)
```

[8]:
```
float
```

3) Strings (str): Los strings corresponden a cadenas de texto (palabras). Se escriben entre comillas, ya sea simples o dobles.

[9]:
```
type("Fundamentos de Programacion")
```

[9]:
```
str
```

[10]:
```
type('Python')
```

[10]:
```
str
```

[11]:
```
type("Don't forget")
```

[11]:
```
str
```

[12]:
```
type("Mañana es viernes, y el cuerpo lo sabe!")
```

[12]:
```
str
```

4) Listas (list): Las listas son un grupo de datos. Se escriben con corchetes ([]) y entre los corchetes deben ir los elementos de la lista, separados por una coma. Los elementos pueden ser de cualquier tipo de dato

[13]:
```
type([3,4,2,5])
```

[13]:
```
list
```

[14]:
```
type(["John", 13, 10, 2000])
```

[14]:
```
list
```

[15]:
```
type(["Funda Progra", 3.2, [3,4,5], 10])
```

[15]:
```
list
```

[16]:
```
type((3,2,5,6))
```

[16]:
```
tuple
```

[17]:
```
type({3,4})
```

[17]:
```
set
```

[18]:
```
type({3:2,5:4})
```

[18]:
```
dict
```

5) Booleanos (bool): Valores lógicos. Solo existen dos: True o False

[19]:
```
type(True)
```

[19]:
```
bool
```

[20]:
```
type(False)
```

[20]:
```
bool
```

1.2 Sintaxis básicas

print(): Imprime un mensaje por pantalla (consola). Debemos escribir el mensaje dentro de los parentesis, con comillas simples (' ') o dobles(" ")

[21]:
```
print("Hola mundo!")
```

```
Hola mundo!
```

[22]:
```
print('Hoy es la ayudantia de Python 3')
```

```
Hoy es la ayudantia de Python 3
```

Podemos colocar numeros y otros tipos de datos para acompañar un mensaje. Debemos escribir los elementos separados por una coma (,) y siempre tener en consideracion que todo lo que sea letra (texto) debe ir en comillas simples o dobles

[23]:
```
print("Hoy es el dia: ", 11, "/", 8, "/", 2022)
```

```
Hoy es el dia:  11 / 8 / 2022
```

input(): Pide una entrada al usuario. Todo lo que se ingrese a traves de input pasa a ser una cadena de texto (String). Se puede escribir un mensaje dentro del parentesis para que el usuario sepa que debe ingresar especificamente.

[24]:
```
input("Ingrese su edad: ")
```

```
Ingrese su edad: 21
```

[24]:
```
'21'
```

Si queremos que el elemento ingresado a traves de input no sea un String, podemos utilizar **Funciones de transformación de tipo de datos**. Para ello, antes del input debemos escribir el tipo de dato a transformar Las funciones que existen considerando los 4 tipos de datos importantes son

1) int()

2) float()

3) str()

4) list()

5) bool()

[25]:
```
int(input("Ingrese el anio de nacimiento: "))
```

```
Ingrese el anio de nacimiento: 2000
```

[25]:
```
2000
```

Debemos tener cuidado al transformar elementos, ya que hay casos donde es imposible hacer la transformación de un tipo de dato a otro. Por ejemplo, **no podemos transformar strings que representen texto a números!**

[26]:
```
int("hola")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12168/754444761.py in <module>
----> 1 int("hola")

ValueError: invalid literal for int() with base 10: 'hola'
```

[27]:
```
int('6')
```

[27]:
```
6
```

[28]:
```
int(5.3)
```

[28]:
```
5
```

[29]:
```
str([3,4,2])
```

[29]:
```
'[3, 4, 2]'
```

[30]:
```
int(True)
```

[30]:
```
1
```

[31]:
```
bool(20)
```

[31]:
```
True
```

[32]:
```
bool(0)
```

[32]:
```
False
```

1.3 Variables

Las variables son una herramienta muy util a la hora de programar. En las variables podemos guardar elementos para su posterios uso

[33]:
```
x = 5
print(x)
```

```
5
```

[34]:
```
x = "Hello gente"
print(x)
```

```
Hello gente
```

Para cumplir con buenas practicas de programacion (no solamente para la asignatura) es importante que el **nombre de nuestras variables representen lo que estamos guardando y que estas esten escritas en minusculas y con un guion bajo (_) para separar cada palabra**

[35]:
```
nombre = input("Ingresa tu nombre: ")
print(nombre)
```

```
Ingresa tu nombre: John
John
```

Las variables:

1) NO deben ser escritas en mayusculas, a excepción que estas sean constantes

2) NO deben partir con algun otro caracter que no sea una letra. No pueden partir con números u otros simbolos

3) NO pueden contener espacios

[36]:
```
edad = int(input("Ingrese su edad: "))
```

```
Ingrese su edad: 21
```

[37]:
```
PI = 3.1415 # lo anterior es una CONSTANTE
print(PI)
```

```
3.1415
```

1.4 Buenas practicas de la programación

Además de respetar las buenas practicas de la programación con nuestras variables, también es importante hacerlo con el resto de nuestro código. Para ello, existe lo que se conoce como **Comentarios** lo cual se simboliza con un simbolo de hashtag (#). Los comentarios son ignorados al ejecutar el código de nuestro programa, pero sirven para que alguien externo a nuestro programa pueda entender nuestro código. Siempre es bueno ir comentando para no perder la idea de como funciona nuestro código y también dividir nuestro código en tres partes:

1) ENTRADA

2) PROCESAMIENTO

3) SALIDA

[38]:
```
# ENTRADA
edad = int(input("Ingrese su edad: "))
# PROCESAMIENTO
resultado = edad + 15
# SALIDA
print("En 15 anios mas, vas a tener", resultado, "anios")
```

```
Ingrese su edad: 21
En 15 anios mas, vas a tener 36 anios
```

También podemos realizar comentarios multilinea utilizando comillas triples

1.5 Operaciones aritmeticas

Con Python, podemos aplicar todo lo anterior para poder realizar distintas operaciones aritmeticas y lógicas. Cada una tiene un simbolo asociado. Podemos realizar operaciones con ints, floats, strings y lists

1) Suma: Se simboliza con el simbolo **+**.

[39]:
```
15 + 16
```

[39]: 31

[40]:
```
3.6 + 8
```

[40]: 11.6

[41]:
```
"Hola " + "mundo"
```

[41]: 'Hola mundo'

[42]:
```
[3,4,5] + [9,8,7]
```

[42]: [3, 4, 5, 9, 8, 7]

[43]:
```
True + True
```

[43]: 2

2) Resta: Se simboliza con el simbolo **-**. No podemos realizar restas de strings o listas

[44]:
```
3-9
```

[44]: -6

[45]:
```
-9.9-9.9
```

[45]: -19.8

[46]:
```
euler_number = 2.71828182845904523536
resultado = euler_number - euler_number
print(resultado)
```

```
0.0
```

[47]:
```
promedio = 7.0-0.2
print(promedio)
```

```
6.8
```

[48]:
```
True - True
```

[48]: 0

3) Multiplicación: Se simboliza con el simbolo *

[49]:
```
12*3
```

[49]: 36

[50]:
```
0.0*2
```

[50]: 0.0

[51]:
```
"hola"*3
```

[51]: 'holaholahola'

[52]:
```
[3,4,2]*2
```

[52]: [3, 4, 2, 3, 4, 2]

[53]:
```
True * True
```

[53]: 1

4) División: Se simboliza con el simbolo /. No podemos realizar divisiones de Strings o de Listas. **El resultado siempre es un flotante (FLOAT)**

[54]:
```
3.14 / 2
```

[54]: 1.57

[55]:
```
10 / 2
```

[55]: 5.0

[56]:
```
0.1/10
```

[56]: 0.01

[57]:
```
0.99999999999999999999999999999999999999999999999999999999999999999/1
```

[57]: 1.0

[58]:
```
False / True
```

[58]: 0.0

5) División Entera: Se simboliza con el simbolo //. Aplican las mismas reglas que la división normal, **pero ahora es posible tener resultados integers (INT)**

[59]:
```
3.14 // 2
```

[59]: 1.0

[60]:
```
10 // 2
```

[60]: 5

[61]:
```
0.1 // 10
```

[61]: 0.0

[62]:
```
0.99999999999999999999999999999999999999999999999999999999999999999 // 1
```

[62]: 1.0

[63]:
```
False // True
```

[63]: 0

6) Exponente (Potencia): Se simboliza con el simbolo ** . No podemos realizar potencias con Strings o Listas

[64]:
```
25**2
```

[64]: 625

[65]:
```
4.3**2
```

[65]: 18.49

[66]:
```
99999999999999999999999999**0
```

[66]: 1

[67]:
```
0**0
```

[67]: 1

[68]:
```
True**False
```

[68]:
```
1
```

7) Modulo (Resto de la división): Se simboliza con el simbolo %. No podemos sacar restos con Strings o Listas

[69]:
```
'''
10 / 2 = 5.
5 * 2 da exactamente 10
'''
10%2
```

[69]:
```
0
```

[70]:
```
'''
¿Por que?
3 / -2 = -1.5
Python aproxima el resultado a -2
-2 * -2 = 4
Para llegar a 3, debemos restar 1
'''
3% -2
```

[70]:
```
-1
```

[71]:
```
'''
Notemos que es similar a lo anterior
-3 / 2 = -1.5
2 * -2 = -4
Para llegar a -3, debemos sumar 1
'''
-3 % 2
```

[71]:
```
1
```

[72]:
```
3 % True
```

[72]:
```
0
```

1.6 Operaciones Lógicas

1): Mayor o menor que: Se simbolizan con **>** o **<**

[73]:
```
2 > 3
```

[73]:
```
False
```

[74]:
```
-9 > -18
```

[74]: True

2) Mayor o igual que o menor o igual que: Se simbolizan con **>=** o **<=**

[75]:
```
9>=9
```

[75]: True

[76]:
```
"32" < "32"
```

[76]: False

3) Igual que: Se simboliza con **==**. Recordar que el **=** se utiliza solamente para asignacion!

[77]:
```
[3,4,2] == [3,4,2]
```

[77]: True

[78]:
```
"Friday Night" == "Funkin"
```

[78]: False

4) Distinto que: Se simboliza con **!=**.

[79]:
```
1.0 != 1
```

[79]: False

[80]:
```
"hola" != "HOLA"
```

[80]: True

5) Negación: Se simboliza con **not()**

[81]:
```
not(True)
```

[81]: False

[82]:
```
not(False)
```

[82]: True

6) Conjunción: Se simboliza con **and**

[83]:
```
True and True
```

[83]: True

[84]:
```
True and False
```

[84]: False

[85]:
```
(True and False) and False
```

[85]:
```
False
```

7) Disyunción: Se simboliza con **or**

[86]:
```
True or False
```

[86]:
```
True
```

[87]:
```
False or False
```

[87]:
```
False
```

Las operaciones tanto lógicas como aritmeticas tienen su orden y reglas: La precedencia de las operaciones son:

1) Parentesis

2) Exponentes

3) Cambio de signo

4) Multiplicación

4) División

4) Resto

5) Suma

5) Resta

6) Igualdad

6) Distinto que

6) Menor que

6) Menor o igual que

6) Mayor que

6) Mayor o igual que

7) Negación

8) Conjunción

9) Disyunción

[88]:
```
3**2 + 9*2 * (10 / 5)
```

[88]:
```
45.0
```

1.7 Condicionales

Cuando programamos, a veces necesitamos que se cumpla algo si se cumple cierta condicion. Los condicionales nos ayudan con esa tarea

[89]:
```
numero = int(input("Ingrese un numero entero: "))
if(numero >= 100):
    print("Es un numero mayor o igual a 100!")
elif(numero > 50 and numero < 100):
    print("Es un numero entre 50 y 100")
else:
    print("Es un numero menor o igual a 50!")
```

```
Ingrese un numero entero: 1000
Es un numero mayor o igual a 100!
```

Cada "if" se puede ver como un bloque. Esto quiere decir, que podemos tener ifs sin elses y también que podemos tener tantos elifs como condiciones tengamos. Notemos después de cada condicion, la consecuencia de esa condición esta separada. Esta separación se llama **identacion**

[91]:
```
numero = int(input("Ingrese un numero entero: "))
if numero >= 0:
    print("Es un numero positivo!")
if numero >= 100:
    print("Es un numero grande")
if numero > 1000:
    print("Es un numero muy grande!")
```

```
Ingrese un numero entero: 10000
Es un numero positivo!
Es un numero grande
Es un numero muy grande!
```

1.8 Ciclo While

El ciclo while repite una acción mientras la condición del while sea verdadera

[2]:
```
i = 0 # ITERADOR
lista_nombres = ["jose", "cristian", "john", "fernando"]
n = len(lista_nombres) # Len obtiene el LARGO (numero de elemenos) de la lista
while i < n: # CONDICION es que i sea menor a n
    print("Se repite el while")
    i += 1  # Se aumenta el iterador. Si no se aumenta, la iteración se repite␣
 ↪hasta el infinito

print("Se acabo el while")
```

```
Se repite el while
Se repite el while
Se repite el while
```

```
Se repite el while
Se acabo el while
```

1.9 Malas practicas

NO debemos aplicar las siguientes practicas: * While True: Se considera mala practica ya que puede provocar un ciclo infinito * break: No se puede utilizar break en FPI