



BACKEND CON PYTHON



INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



TIPOS DE DATOS

Un **valor** es una de las cosas fundamentales que un programa manipula. Algunos ejemplos:

- “Hola mundo !”
- 10 (resultado de $7+3$)

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC
(AMD64)]
Type "copyright", "credits" or "license" for more

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: "HoLa mundo!"
Out[1]: 'Hola mundo!'

In [2]: 7+3
Out[2]: 10
```

TIPOS DE DATOS

Los valores y variables son clasificados en **tipos de datos**:



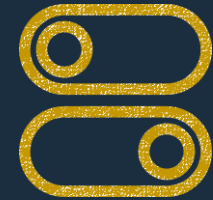
10 es un **entero**



-4.5 es un **decimal**






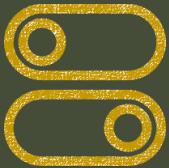
“Hola mundo !” es un **string** (o cadena de caracteres)



True es un **booleano**

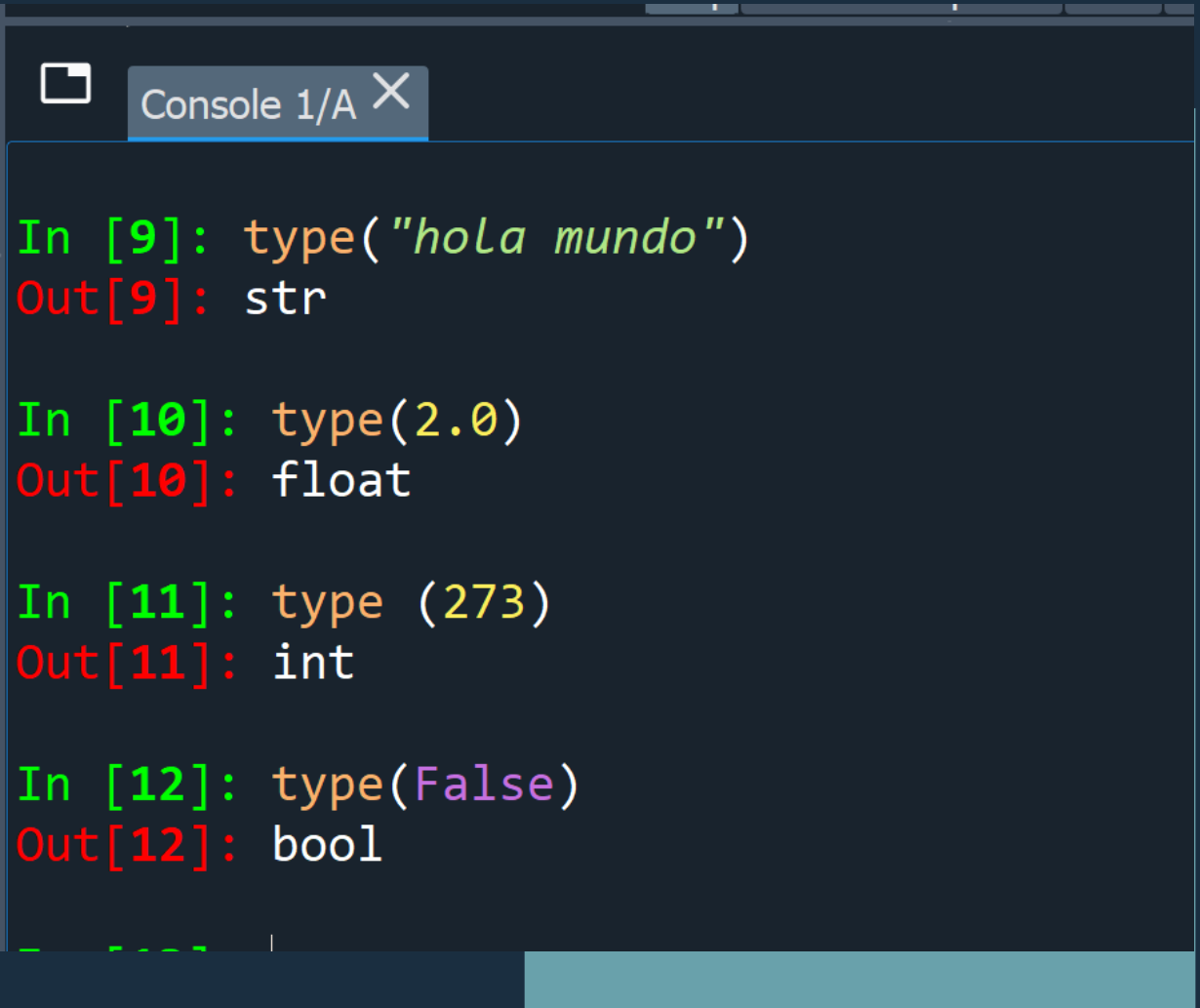
TIPOS DE DATOS

Los valores y variables son clasificados en **tipos de datos**:

	2	int	Ocupan menos memoria que los float y las operaciones son más rápidas
	2.0	float	Permite representar un número positivo/negativo con decimales
	“dos”	str	Cadena o secuencia de caracteres: letras, números, espacios, signos de puntuación, etc.
	False	bool	Dato lógico que solo puede representar dos valores: verdadero o falso

TIPOS DE DATOS

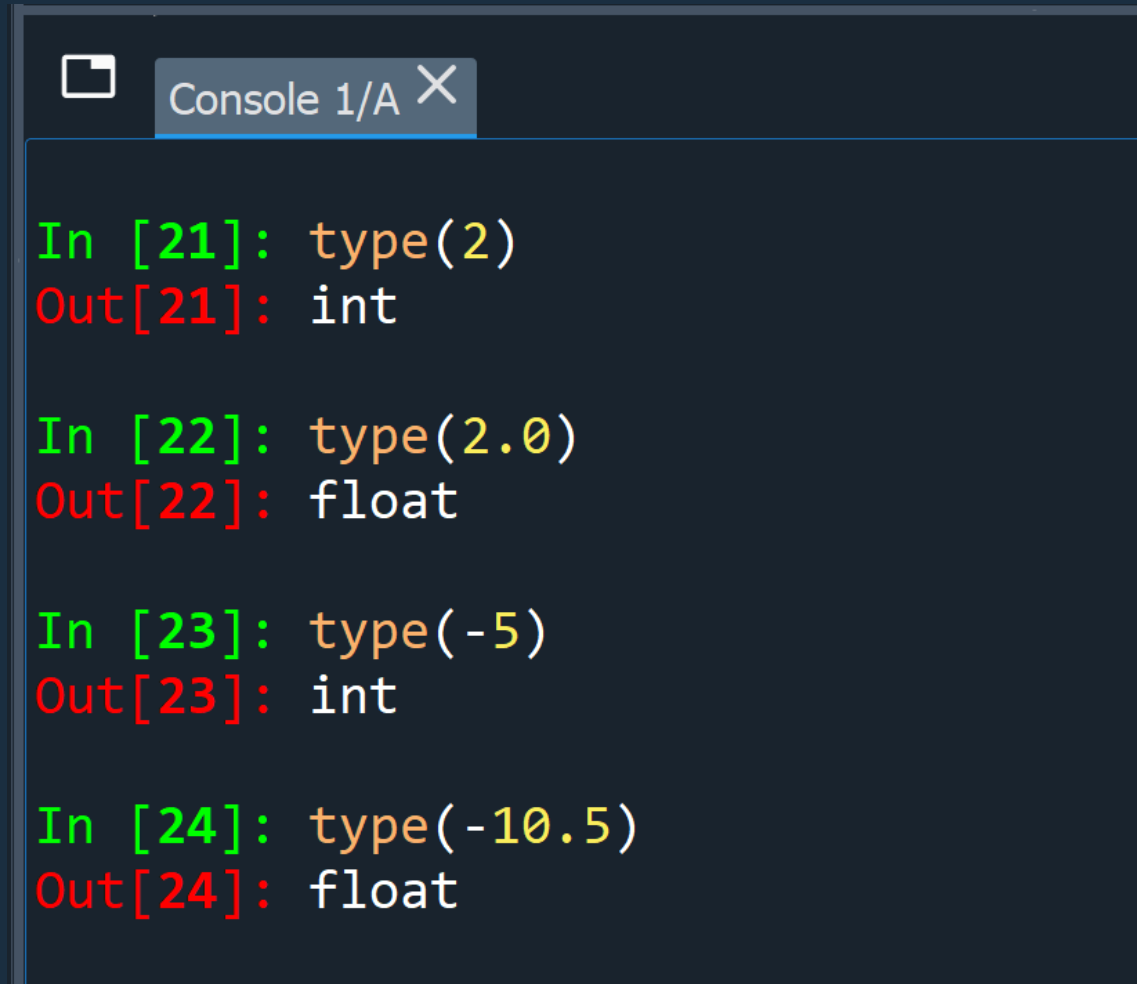
Para conocer el tipo de un valor, Python tiene una función llamada `type`



```
Console 1/A X  
  
In [9]: type("hoLa mundo")  
Out[9]: str  
  
In [10]: type(2.0)  
Out[10]: float  
  
In [11]: type (273)  
Out[11]: int  
  
In [12]: type(False)  
Out[12]: bool
```

TIPOS DE DATOS – INT Y FLOAT

- int sirve para representar enteros positivos y negativos.
- float sirve para representar números con decimales.
- 2 es tipo de dato diferente de 2.0



```
Console 1/A X  
  
In [21]: type(2)  
Out[21]: int  
  
In [22]: type(2.0)  
Out[22]: float  
  
In [23]: type(-5)  
Out[23]: int  
  
In [24]: type(-10.5)  
Out[24]: float
```


TIPOS DE DATOS - STRING

- A. Las simples sirven para valores sin ambigüedad
- B. Las dobles pueden contener cadenas con comillas sencillas y viceversa:

'Dije "no" ' o "It's today"

Pero qué pasa cuando una cadena contiene tanto comillas dobles como simples:

She said to me "That's mine!"

Podemos "protegerlas"

'She said to me "That\'s mine!"'

- C. Las triples se utilizan cuando las cadenas pueden tener cambios de línea:

- *"""She said to me...
"That\'s mine!""""*

```
Console 1/A X

In [4]: type("esto es un string")
Out[4]: str

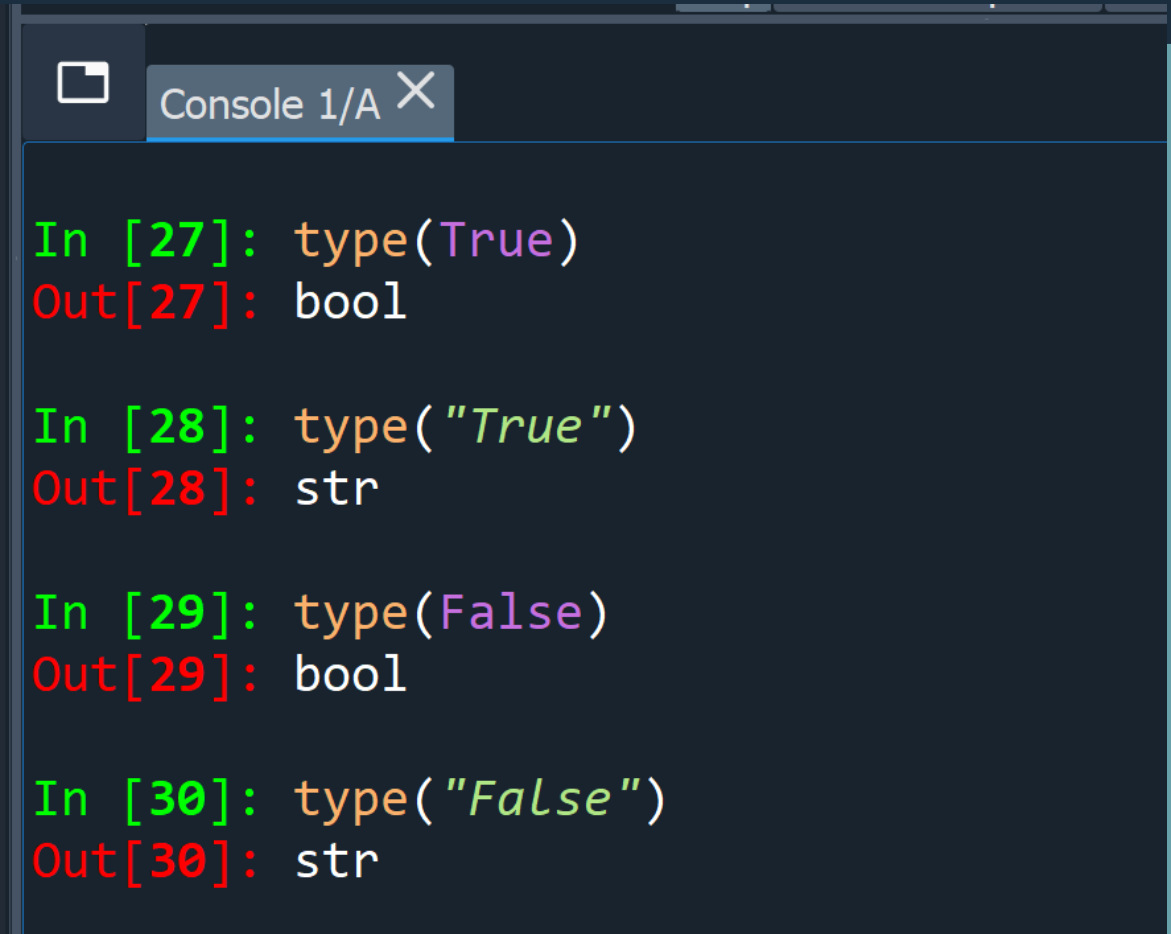
In [5]: type('esto tambien')
Out[5]: str

In [6]: type("""y este""")
Out[6]: str

In [7]: type(''''inclusive este''')
Out[7]: str
```

TIPOS DE DATOS - BOOLEAN

- Los bool son valores lógicos que solo pueden representar dos estados: True (verdadero) y False (falso)
- No tienen comillas como los str



```
Console 1/A X  
  
In [27]: type(True)  
Out[27]: bool  
  
In [28]: type("True")  
Out[28]: str  
  
In [29]: type(False)  
Out[29]: bool  
  
In [30]: type("False")  
Out[30]: str
```

TIPOS DE DATOS - VARIABLES

- Una de las características más poderosas de un lenguaje de programación es la capacidad de manipular variables
- Las variables se usan para **guardar valores** que se necesitarán más adelante en el programa
- La instrucción de **asignación (=)** se usa para guardar un valor en una variable

```
Console 1/A X

In [59]: pi=3.14159

In [60]: r=1.2983

In [61]: perimetro=2*pi*r

In [62]: area=pi*r**2

In [63]: perimetro
Out[63]: 8.157452594

In [64]: area
Out[64]: 5.2954103513951
```

INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



ESTRUCTURAS DE DATOS

- Hasta el momento las variables sirven para representar 1 único dato, pero si se necesita representar conjuntos de datos necesitaremos algunas estructuras de datos: listas, sets, diccionarios, árboles, grafos, tuplas, listas enlazadas, entre otras.
- En este curso nos concentraremos en 3:
 - Listas
 - Tuplas
 - Diccionarios

```
Console 1/A X
In [66]: lista= [1,3,4.0, False, "Hola"]
In [67]: lista
Out[67]: [1, 3, 4.0, False, 'Hola']
In [68]: type(lista)
Out[68]: list
```

```
Console 1/A X
In [77]: tupla = (1, 2.0, "tres")
In [78]: tupla
Out[78]: (1, 2.0, 'tres')
In [79]: type(tupla)
Out[79]: tuple
```

```
Console 1/A X
In [70]: diccionario={"llave1":"valor1", "llave2":2, 3:"valor3"}
In [71]: diccionario
Out[71]: {'llave1': 'valor1', 'llave2': 2, 3: 'valor3'}
In [72]: type(diccionario)
Out[72]: dict
```

ESTRUCTURAS DE DATOS

Listas

- Son colecciones ordenadas de valores y es un tipo de Python: **list**
- Los valores que conforman una **lista** son llamados **elementos** o **ítems**
- Las **listas** son similares a los strings, que son conjuntos ordenados de caracteres. Se diferencian en que los elementos de una **lista** pueden ser de cualquier tipo (enteros, flotantes, o incluso de cadenas)
- Las **listas** y los strings (y otras colecciones que mantienen el orden de sus ítems) se denominan **secuencias**

ESTRUCTURAS DE DATOS

Listas

Creación de listas.

- Se pueden crear poniendo los valores separados por , dentro de [] como en el ejemplo lista.
- Se pueden crear con un rango de valores (lista2)
- Se pueden crear multiplicando los elementos de otra lista (lista3)

```
Console 1/A X

In [81]: lista=[1,2,3,4]

In [82]: lista
Out[82]: [1, 2, 3, 4]

In [83]: lista2= list(range(1,10))

In [84]: lista2
Out[84]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [85]: lista3= [0]*5

In [86]: lista3
Out[86]: [0, 0, 0, 0, 0]
```

ESTRUCTURAS DE DATOS

Listas

Operaciones sobre listas.

- Se pueden sumar dos listas, como en el ejemplo la creación de la lista3.
- Se pueden agregar nuevos elementos al final con `.append`
- Se puede pedir el valor de un item dada su posición dentro de la lista
- Hay muchas más operaciones sobre las listas que se pueden consultar la documentación oficial de Python.

```
Console 1/A X

In [88]: lista1=[1,2]

In [89]: lista2=[3,4,5]

In [90]: lista3=lista1+lista2

In [91]: lista3
Out[91]: [1, 2, 3, 4, 5]

In [92]: lista3.append(6)

In [93]: lista3
Out[93]: [1, 2, 3, 4, 5, 6]

In [94]: lista3[4]
Out[94]: 5
```


ESTRUCTURAS DE DATOS

Diccionarios

- Un diccionario es un tipo de **dato compuesto** que nos permite manejar correspondencias entre **llaves** y **valores**
- Es un **tipo de dato** «**dict**», esto quiere decir que podemos tener variables y parámetros de tipo diccionario y también funciones que retornen diccionarios
- Tanto **llaves** como **valores** pueden ser de cualquier tipo de dato.

ESTRUCTURAS DE DATOS

Diccionarios

Creación de diccionarios

- Se pueden crear dando los elementos, separados por , donde primero se indica la llave y luego el valor correspondiente a dicha llave separados por :
- Se pueden crear vacíos e ir agregando elemento por elemento, como el directorio2 del ejemplo

```
Console 1/A X

In [96]: directorio={"Ana": "3245673", "Luis": "8765432"}

In [97]: directorio
Out[97]: {'Ana': '3245673', 'Luis': '8765432'}

In [98]: directorio2={}

In [99]: directorio2["Ana"]="1236547"

In [100]: directorio2["Luis"]="6574839"

In [101]: directorio2
Out[101]: {'Ana': '1236547', 'Luis': '6574839'}
```

ESTRUCTURAS DE DATOS

Diccionarios

Operaciones en diccionarios

- Para consultar un valor del diccionario debemos dar la llave. Si damos una llave inexistente tendremos un `KeyError`.
- Para evitar el `KeyError` podemos usar el método `get`.
- Los diccionarios tienen muchas operaciones que pueden ser consultadas en la documentación oficial de Python.

```
Console 1/A X

In [103]: diccionario={"Ana":1, "Luis":"B"}

In [104]: diccionario
Out[104]: {'Ana': 1, 'Luis': 'B'}

In [105]: diccionario["Ana"]
Out[105]: 1

In [106]: diccionario["beto"]
Traceback (most recent call last):

  File "C:\Users\German\AppData\Local\Temp/ipykernel_9816/1238470736.py", line 1, in <module>
    diccionario["beto"]

KeyError: 'beto'

Console 1/A X

In [109]: diccionario
Out[109]: {'Ana': 1, 'Luis': 'B'}

In [110]: diccionario.get("Ana","No existe")
Out[110]: 1

In [111]: diccionario.get("beto","No existe")
Out[111]: 'No existe'
```

ESTRUCTURAS DE DATOS

Tuplas

- Estructura de datos **lineal**
- Cada elemento se identifica con un **índice** o posición
- **Inmutables**: no se pueden modificar los elementos después de creados

```
Console 1/A X

In [113]: tupla= (1,2,3,"A")

In [114]: tupla[0]="B"
Traceback (most recent call last):

  File "C:\Users\German\AppData\Local\Temp/ipykernel_9816/3454843768.py", line 1, in <module>
    tupla[0]="B"

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

ESTRUCTURAS DE DATOS

Tuplas

Operaciones en tuplas

- Se crean asignando a una variable varios valores separados por ,
- El proceso de separar los valores se conoce como desempaquetado
- Se pueden crear tuplas de 1 valor poniendo una , después del valor o la variable

```
Console 1/A X
In [117]: tupla1=(1,2,3)
In [118]: a,b,c=tupla1
In [119]: a
Out[119]: 1
In [120]: b
Out[120]: 2
In [121]: c
Out[121]: 3
```

```
Console 1/A X
In [123]: tupla = (1,)
In [124]: type(tupla)
Out[124]: tuple
In [125]: tupla2=(1)
In [126]: type(tupla2)
Out[126]: int
```

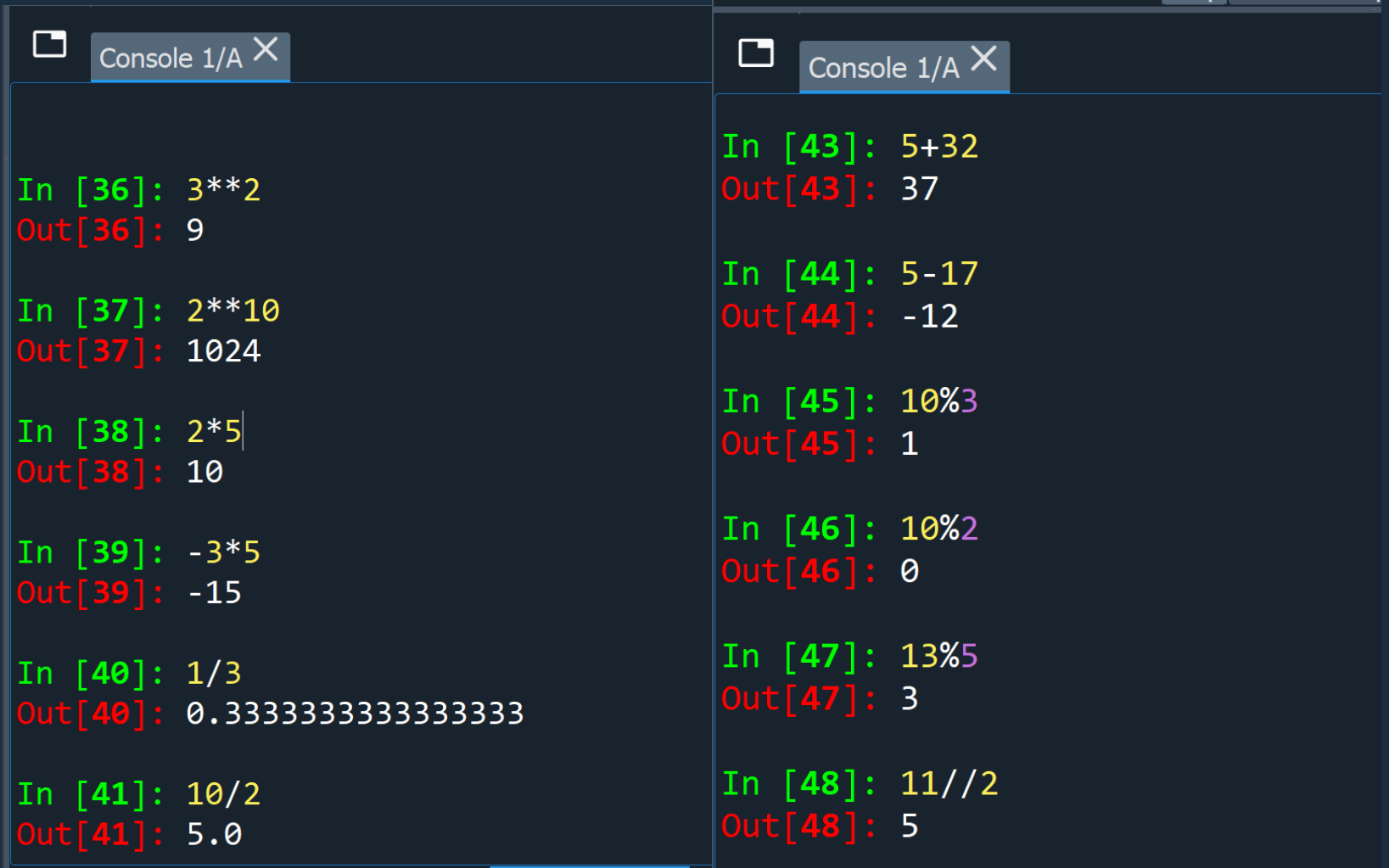
INTRODUCCIÓN A PYTHON

- Tipos de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



OPERADORES

- Exponenciación: $**$
- Multiplicación: $*$
- División: $/$
- Suma: $+$
- Resta: $-$
- Módulo: $\%$
- División entera: $//$



The image displays two side-by-side screenshots of a Jupyter Notebook's console window, titled 'Console 1/A'. The left screenshot shows the results of operations from input 36 to 41, and the right screenshot shows results from input 43 to 48. Each entry consists of an input line (In [X]) and an output line (Out[X]).

Input	Operation	Output
In [36]	$3**2$	Out[36]: 9
In [37]	$2**10$	Out[37]: 1024
In [38]	$2*5$	Out[38]: 10
In [39]	$-3*5$	Out[39]: -15
In [40]	$1/3$	Out[40]: 0.3333333333333333
In [41]	$10/2$	Out[41]: 5.0
In [43]	$5+32$	Out[43]: 37
In [44]	$5-17$	Out[44]: -12
In [45]	$10\%3$	Out[45]: 1
In [46]	$10\%2$	Out[46]: 0
In [47]	$13\%5$	Out[47]: 3
In [48]	$11//2$	Out[48]: 5

OPERADORES

Sobre los string también podemos aplicar los operadores + y *

```
Console 1/A X

In [51]: "hola " + "mundo"
Out[51]: 'hola mundo'

In [52]: "hola" * 5
Out[52]: 'holaholaholahohola'

In [53]: a="hola"

In [54]: a*3
Out[54]: 'holaholahola'

In [55]: a+"mundo"
Out[55]: 'holamundo'
```


INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



CONDICIONALES

Debemos comprender los operadores de comparación primero:

Es igual que	<code>=</code>	<code>x == y</code> es True si x es igual a y
Es diferente de	<code>!=</code>	<code>x != y</code> es True si x es diferente de y
Es menor que	<code><</code>	<code>x < y</code> es True si x es menor que y
Es mayor que	<code>></code>	<code>x > y</code> es True si x es mayor que y
Es menor o igual que	<code><=</code>	<code>x <= y</code> es True si x es menor o igual que y
Es mayor o igual que	<code>>=</code>	<code>x >= y</code> es True si x es mayor o igual que y

Console 1/A ✕

```
In [128]: 5==3+2
```

```
Out[128]: True
```

```
In [129]: 5>3
```

```
Out[129]: True
```

```
In [130]: 5<2
```

```
Out[130]: False
```

```
In [131]: 5!=3-2
```

```
Out[131]: True
```

```
In [132]: 5>=2
```

```
Out[132]: True
```

CONDICIONALES

Para condicionales más complejos podemos usar operadores `and`, `or` y `not`

Operador	Se lee como
<code>and</code> (conjunción)	y
<code>or</code> (disyunción)	o
<code>not</code> (negación)	no

<code>operando1 and operando2</code>	Es cierto, si ambos operandos son verdaderos
<code>operando1 or operando2</code>	Es cierto, si cualquiera de los dos operandos es verdadero
<code>not operando1</code>	Es cierto, si el operando es falso

```
Console 1/A X

In [134]: not 5 < 3
Out[134]: True

In [135]: 5 > 3 and 5 < 7
Out[135]: True

In [136]: 5 > 3 or 5 < 3
Out[136]: True
```

CONDICIONALES

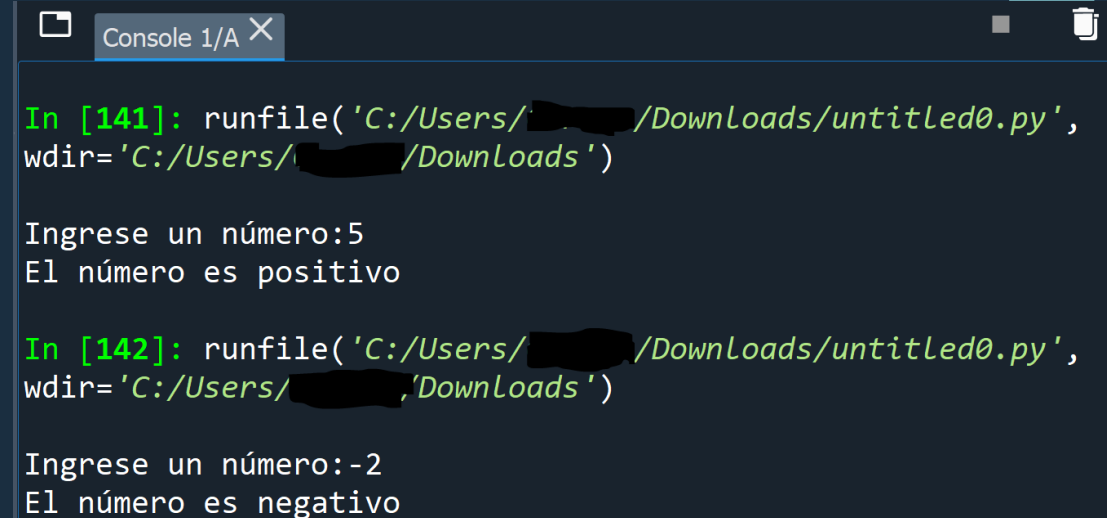
En Python los condicionales se expresan por medio de la instrucción if-else

```
if EXPRESIÓN BOOLEANA :  
    INSTRUCCIONES_1          # Se ejecutan si al evaluar la condición da True  
else:  
    INSTRUCCIONES_2          # Se ejecutan si al evaluar la condición da False
```

Para el siguiente código

```
3 a= int(input("Ingrese un número:"))  
4  
5 if a < 0:  
6     print("El número es negativo")  
7 else:  
8     print("El número es positivo")
```

Se obtiene el siguiente resultado



```
Console 1/A X  
  
In [141]: runfile('C:/Users/[redacted]/Downloads/untitled0.py',  
wdir='C:/Users/[redacted]/Downloads')  
  
Ingrese un número:5  
El número es positivo  
  
In [142]: runfile('C:/Users/[redacted]/Downloads/untitled0.py',  
wdir='C:/Users/[redacted]/Downloads')  
  
Ingrese un número:-2  
El número es negativo
```

CONDICIONALES

Para expresar más condiciones se usa la instrucción elif

```
3 a= int(input("Ingrese un número:"))
4
5 if a < 0:
6     print("El número es negativo")
7 elif a>=0 and a <100 :
8     print("El número es positivo y de dos cifras")
9 else:
10    print("El número es positivo y de más de dos cifras")
```

El resultado es el siguiente:

```
In [144]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
Ingrese un número:-5
El número es negativo
```

```
In [145]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
Ingrese un número:56
El número es positivo y de dos cifras
```

```
In [146]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
Ingrese un número:111
El número es positivo y de más de dos cifras
```

INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



BUCLES

Cuando necesitamos repetir una instrucción múltiples veces usamos los bucles.

En Python tenemos dos formas de hacer instrucciones repetitivas:

- while

```
while condición:  
    acción  
    acción  
    ...  
    acción
```

- for

```
for variable in serie de valores:  
    acción  
    acción  
    ...  
    acción
```

BUCLES

While

Se debe definir una condición y mientras que esta se cumpla, se repetirán las instrucciones dentro del while.

```
3 lista = [0,3,5,73,54,2,6]
4 i=0
5 while i<len(lista):
6     print(lista[i])
7     i+=1
8
```

En este caso la condición es $i < \text{len}(\text{lista})$

La línea $i+=1$ es fundamental, de lo contrario tendríamos un ciclo infinito

El resultado de ejecutar el ejemplo:

```
In [148]: runfile('C:/Users/[redacted]/Downloads/untitled0.py',
wdir='C:/Users/[redacted]/Downloads')
0
3
5
73
54
2
6
```


BUCLES

For

Sirve para repetir instrucciones sobre elementos de estructuras secuenciales (string, list, tuple, dict)

Con lista:

```
3 lista = [0,3,5,73,54,2,6]
4 for elem in lista:
5     print(elem)
```

Con string:

```
3 cadena="hola mundo"
4 for letra in cadena:
5     print(letra)
```

Con tupla:

```
3 tupla= 1,2,3
4 for num in tupla:
5     print(num)
```

```
In [149]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
0
3
5
73
54
2
6
```

```
In [150]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
h
o
l
a

m
u
n
d
o
```

```
In [151]: runfile('C:/Users/.../Downloads/untitled0.py',
wdir='C:/Users/.../Downloads')
```

```
1
2
3
```

BUCLES

For

Sirve para repetir instrucciones sobre diccionarios con for, se puede hacer por llaves, por valores, o por ítems (llaves + valores)

Por llaves:

```
3 diccionario={"Ana":"1234","Luis":"3456","Beto":"7890"}
4 for llave in diccionario.keys():
5     print(llave)
```

```
In [153]: runfile('C:/Users/.../Downl
wdir='C:/Users/.../Downloads')
Ana
Luis
Beto
```

Por valores:

```
3 diccionario={"Ana":"1234","Luis":"3456","Beto":"7890"}
4 for valor in diccionario.values():
5     print(valor)
```

```
In [154]: runfile('C:/Users/.../Downl
wdir='C:/Users/.../Downloads')
1234
3456
7890
```

Por ítems:

```
3 diccionario={"Ana":"1234","Luis":"3456","Beto":"7890"}
4 for llave,valor in diccionario.items():
5     print(llave+":"+valor)
```

```
In [155]: runfile('C:/Users/.../Downl
wdir='C:/Users/.../Downloads')
Ana:1234
Luis:3456
Beto:7890
```

INTRODUCCIÓN A PYTHON

- Tipos de datos
- Estructuras de datos
- Operadores
- Estructuras de control
 - *Condicionales*
 - *Bucles*
- Funciones



FUNCIONES

A lo largo de la presentación han aparecido varias funciones propias de Python. Recordemos algunas:

- `print`: imprime un mensaje en consola
- `input`: permite al usuario escribir algún valor
- `len`: retorna la cantidad de elementos (lista, diccionario, tupla, string)
- `str`: convierte un valor a string
- `int`: convierte un valor a entero
- `type`: dice el tipo de dato del valor/variable

Hay otras que no han aparecido pero serán de mucha utilidad

- `min`: encuentra el valor mínimo
- `max`: encuentra el valor máximo
- `abs`: calcula el valor absoluto de un número
- Muchas otras que pueden ser consultadas en la documentación oficial de Python

FUNCIONES

Ejemplo print e input

```
In [159]: print("hola mundo")  
hola mundo
```

```
In [160]: input("Escriba un número:")
```

Escriba un número:5

```
Out[160]: '5'
```

Ejemplo len, min y max

```
In [161]: lista=[0,1,2,3,4,5]
```

```
In [162]: len(lista)
```

```
Out[162]: 6
```

```
In [163]: min(lista)
```

```
Out[163]: 0
```

```
In [164]: max(lista)
```

```
Out[164]: 5
```

Ejemplo type, int, float

```
In [166]: a="5"
```

```
In [167]: type(a)
```

```
Out[167]: str
```

```
In [168]: b=int(a)
```

```
In [169]: type(b)
```

```
Out[169]: int
```

```
In [170]: c=float(b)
```

```
In [171]: type(c)
```

```
Out[171]: float
```

Ejemplo abs

```
In [173]: abs(5)
```

```
Out[173]: 5
```

```
In [174]: a=3
```

```
In [175]: abs(a)
```

```
Out[175]: 3
```

```
In [176]: abs(-4)
```

```
Out[176]: 4
```

FUNCIONES

Python permite la creación de funciones propias para ser reutilizadas.

Para esto usamos la instrucción `def`, seguida del nombre que queremos ponerle a la función.

Python es muy flexible al momento de crear funciones, pero por buenas prácticas de programación en el curso seguiremos el estándar:

```
def nombre_función (parámetro:tipo)->tipo_retorno:
```

Para Python no es obligatorio especificar el tipo de los parámetros o del retorno, pero en el curso lo haremos.

FUNCIONES

Para el siguiente ejemplo no hay retorno, por eso está el None:

```
3 def calcular_potencias (numero:int)->None:
4     cuadrado=numero**2
5     cubo=numero**3
6     cuatro=numero**4
7     print(cuadrado, cubo, cuatro)
```

Podemos ver como con solo una ejecución fue fácil reutilizar el código haciendo un simple llamado a la función y cambiando el valor del parámetro:

```
In [180]: runfile('C:/Users/[redacted]/DownLoa
wdir='C:/Users/[redacted]/Downloads')
```

```
In [181]: calcular_potencias(3)
9 27 81
```

```
In [182]: calcular_potencias(5)
25 125 625
```

```
In [183]: calcular_potencias(11)
121 1331 14641
```

```
In [184]: calcular_potencias(27)
729 19683 531441
```

MANOS A LA OBRA

Taller 1 disponible en BloqueNeon

