



## GUÍA DE LABORATORIO 02

### Tipos de Datos I – Numbers, Strings y Boolean – Operadores

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

#### Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

#### 1. Objetivos:

- Escribir algoritmos y pseudocódigo para los casos planteados.
- Crear algoritmos

#### 2. Equipos, herramientas o materiales

- Computador
- Software: Python - VSCode
- Algoritmos

#### 3. Fundamento teórico

##### Conceptos clave tipos de datos:

##### Python Numbers

Hay tres tipos numéricos en Python:

- Int: Int, o entero, es un número entero, positivo o negativo, sin decimales, de longitud ilimitada.
- Float: Un número flotante o "número de punto flotante" es un número, positivo o negativo, que contiene uno o más decimales.  
Los flotantes también pueden ser números científicos con una "e" para indicar la potencia de 10.  
(x = 35e3)
- Complex: Los números complejos se escriben con una "j" como parte imaginaria.

Las variables de tipo numérico se crean cuando se les asigna un valor:

```
numbers.py
Semana02 > numbers.py > ...
1 # DATOS NUMERICOS
2 x = 1 # int
3 y = 2.8 # float
4 z = 1j # complex
5
6 print(type(x))
7 print(type(y))
8 print(type(z))
```

#### Type Conversion:

Puede convertir de un tipo a otro con los métodos int(), float() y complex():



```
02_conversion_type.py
Semana02 > 02_conversion_type.py -
1 x = 1 # int
2 y = 2.8 # float
3 z = 1j # complex
4
5 #convert from int to float:
6 a = float(x)
7
8 #convert from float to int:
9 b = int(y)
10
11 #convert from int to complex:
12 c = complex(x)
13
14 print(a)
15 print(b)
16 print(c)
17
18 print(type(a))
19 print(type(b))
20 print(type(c))
```

### Random Number:

Python no tiene una `random()` función para crear un número aleatorio, pero Python tiene un módulo incorporado llamado `random` que se puede usar para crear números aleatorios:

```
03_random_number.py
Semana02 > 03_random_number.py
1 # Import the random module, and display a random number between 1 and 9:
2
3 import random
4
5 print(random.randrange(1, 10))
```

## Python Strings

### Strings:

Las cadenas en Python están rodeadas por comillas simples o comillas dobles.

'hola' es lo mismo que "hola".

Puede mostrar un literal de cadena con la función `print()`:

```
04_strings.py
Semana02 > 04_strings.py
1 print("Hello")
2 print('Hello')
```

### Quotes Inside Quotes

Puedes utilizar comillas dentro de una cadena, siempre que no coincidan con las comillas que rodean la cadena:

```
05_quotes.py
Semana02 > 05_quotes.py
1 print("It's alright")
2 print("He is called 'Edem'")
3 print('He is called "Edem"')
```

### Multiline Strings:



Puede asignar una cadena de varias líneas a una variable utilizando tres comillas:

```
06_multiline_strings.py •
Semana02 > 06_multiline_strings.py > ...
1 a = """Lorem ipsum dolor sit amet,
2 consectetur adipiscing elit,
3 sed do eiusmod tempor incididunt
4 ut labore et dolore magna aliqua."""
5 print(a)
6
```

### **Strings are Arrays**

Al igual que muchos otros lenguajes de programación populares, las cadenas en Python son matrices de bytes que representan caracteres Unicode.

Sin embargo, Python no tiene un tipo de datos de carácter, un solo carácter es simplemente una cadena con una longitud de 1.

Se pueden utilizar corchetes para acceder a los elementos de la cadena.

```
07_string_arrays.py •
Semana02 > 07_string_arrays.py > ...
1 a = "Hello, World!"
2 print(a[1])
3
```

### **Looping Through a String**

Dado que las cadenas son matrices, podemos recorrer los caracteres de una cadena mediante un bucle for.

```
08_looping_string.py •
Semana02 > 08_looping_string.py > ...
1 for x in "sistemas":
2     print(x)
```

### **Check String**

Para comprobar si una determinada frase o carácter está presente en una cadena, podemos utilizar la palabra clave **in**.

```
09_check_string.py •
Semana02 > 09_check_string.py > ...
1 txt = "The best things in life are free!"
2 if "free" in txt:
3     print("Yes, 'free' is present.")
```

### **Check if NOT:**

Para comprobar si una determinada frase o carácter NO está presente en una cadena, podemos utilizar la palabra clave **not in**.

```
10_check_string_not.py •
Semana02 > 10_check_string_not.py > ...
1 txt = "The best things in life are free!"
2 print("expensive" not in txt)
```



### ***Slicing Strings***

Puedes devolver un rango de caracteres utilizando la sintaxis de segmentación.

Especifique el índice inicial y el índice final, separados por dos puntos, para devolver una parte de la cadena.

#### ***Slice From the Start***

Si omitimos el índice de inicio, el rango comenzará en el primer carácter:

#### ***Slice To the End***

Si omitimos el índice *final*, el rango irá hasta el final:

#### ***Negative Indexing***

Utilice índices negativos para iniciar la porción desde el final de la cadena:

```
11_slicing.py
Semana02 > 11_slicing.py > ...
1 # Slicing
2 b = "Hello, World!"
3 print(b[2:5])
4
5 #Slice From the Start
6 b = "Hello, World!"
7 print(b[:5])
8
9 #Slice To the End
10 b = "Hello, World!"
11 print(b[2:])
12
13 #Negative Indexing
14 b = "Hello, World!"
15 print(b[-5:-2])
```

### ***Modify Strings***

Upper Case: El upper() método devuelve la cadena en mayúsculas:

Lower Case: El lower() método devuelve la cadena en minúsculas:

Remove Whitespace: El espacio en blanco es el espacio antes y/o después del texto real, y muy a menudo es conveniente eliminar este espacio.

Replace String: El replace() método reemplaza una cadena con otra cadena:

Split String: El split() método devuelve una lista donde el texto entre el separador especificado se convierte en los elementos de la lista.<sup>01</sup>

```
12_modify_strings.py
Semana02 > 12_modify_strings.py > ...
1 a = "Hello, World!"
2 print(a.upper())
3
4 a = "Hello, World!"
5 print(a.lower())
6
7 a = " Hello, World! "
8 print(a.strip()) # returns "Hello, World!"
9
10 a = "Hello, World!"
11 print(a.replace("H", "J"))
12
13 a = "Hello, World!"
14 print(a.split(",")) # returns ['Hello', ' World!']
15
```



### **String Concatenation**

Para concatenar o combinar dos cadenas puedes utilizar el operador **+**.

```
13_concatenate.py
Semana02 > 13_concatenate.py > ...
1 a = "Hello"
2 b = "World"
3 c = a + b
4 print(c)
```

### **Format Strings**

podemos combinar cadenas y números utilizando f-strings o el método `format()`.

F-Strings: F-String se introdujo en Python 3.6 y ahora es la forma preferida de formatear cadenas.

Para especificar una cadena como una f-string, simplemente coloque un **f** delante del literal de cadena y agregue llaves **{}** como marcadores de posición para variables y otras operaciones.

Placeholders and Modifiers: Un marcador de posición puede contener variables, operaciones, funciones y modificadores para formatear el valor.

Un marcador de posición puede incluir un modificador para formatear el valor.

Se incluye un modificador agregando dos puntos **:** seguidos de un tipo de formato legal, como **.2f** que significa un número de punto fijo con 2 decimales.

```
14_format_strings.py
Semana02 > 14_format_strings.py > ...
1 age = 32
2 txt = f"My name is Edem, I am {age}"
3 print(txt)
4
5 price = 59
6 txt = f"The price is {price} dollars"
7 print(txt)
8
9 price = 59
10 txt = f"The price is {price:.2f} dollars"
11 print(txt)
```

### **Escape Characters (INVESTIGACIÓN PARA EL ESTUDIANTE)**

#### **Others String Methods:**

`Capitalize()`: Convierte el primer carácter a mayúsculas.

`Casefold()`: Convierte la cadena a minúsculas.

`Center()`: Devuelve una cadena centrada.

`Count()`: Devuelve el número de veces que un valor especificado aparece en una cadena.

`Encode()`: Devuelve una versión codificada de la cadena.

`Endswith()`: Devuelve verdadero si la cadena termina con el valor especificado.

`Expandtabs()`: Establece el tamaño de la tabulación de la cadena.

`Find()`: Busca en la cadena un valor especificado y devuelve la posición donde se encontró.

`Format()`: Formatea valores especificados en una cadena.

`Format_map()`: Formatea valores especificados en una cadena.

`Index()`: Busca en la cadena un valor especificado y devuelve la posición donde se encontró.





`Isalnum()`: Devuelve True si todos los caracteres de la cadena son alfanuméricos.

`Isalpha()`: Devuelve True si todos los caracteres de la cadena están en el alfabeto.

`Isascii()`: Devuelve True si todos los caracteres de la cadena son caracteres ascii.

`Isdecimal()`: Devuelve True si todos los caracteres de la cadena son decimales.

`Isdigit()`: Devuelve True si todos los caracteres de la cadena son dígitos.

`Isidentifier()`: Devuelve True si la cadena es un identificador.

`Islower()`: Devuelve True si todos los caracteres de la cadena están en minúsculas.

`Isnumeric()`: Devuelve True si todos los caracteres de la cadena son numéricos.

`Isprintable()`: Devuelve True si todos los caracteres de la cadena son imprimibles.

`Isspace()`: Devuelve True si todos los caracteres de la cadena son espacios en blanco.

`Istitle()`: Devuelve True si la cadena sigue las reglas de un título.

`Isupper()`: Devuelve True si todos los caracteres de la cadena están en mayúsculas.

`Join()`: Une los elementos de un iterable al final de la cadena.

`Ljust()`: Devuelve una versión justificada a la izquierda de la cadena.

`Lower()`: Convierte una cadena en minúsculas.

`Lstrip()`: Devuelve una versión recortada a la izquierda de la cadena.

`Makestrans()`: Devuelve una tabla de traducción para ser utilizada en traducciones.

`Partition()`: Devuelve una tupla donde la cadena se divide en tres partes.

`Replace()`: Devuelve una cadena donde un valor especificado se reemplaza por un valor especificado.

`Rfind()`: Busca en la cadena un valor especificado y devuelve la última posición donde se encontró.

`Rindex()`: Busca en la cadena un valor especificado y devuelve la última posición donde se encontró.

`Rjust()`: Devuelve una versión justificada a la derecha de la cadena.

`Rpartition()`: Devuelve una tupla donde la cadena se divide en tres partes.

`Rsplit()`: Divide la cadena en el separador especificado y devuelve una lista.

`Rstrip()`: Devuelve una versión recortada a la derecha de la cadena.

`Split()`: Divide la cadena en el separador especificado y devuelve una lista.

`Splitlines()`: Divide la cadena en saltos de línea y devuelve una lista.

`Startswith()`: Devuelve verdadero si la cadena comienza con el valor especificado.

`Strip()`: Devuelve una versión recortada de la cadena.

`Swapcase()`: Intercambia mayúsculas y minúsculas, las minúsculas se convierten en mayúsculas y viceversa.

`Title()`: Convierte el primer carácter de cada palabra a mayúsculas.

`Translate()`: Devuelve una cadena traducida.

`Upper()`: Convierte una cadena en mayúsculas.

`Zfill()`: Rellena la cadena con un número específico de valores 0 al principio.

## Boolean

Un tipo de dato booleano representa uno de los dos valores: Verdadero o Falso. El uso de estos tipos de datos será claro una vez que comencemos a usar el operador de comparación. La primera letra T para Verdadero (True) y F para Falso (False) debe ser mayúscula, a diferencia de JavaScript. Ejemplo: Valores Booleanos.



```
1 print(True)
2 print(False)
```

## Operadores



El lenguaje Python admite varios tipos de operadores. En esta sección, nos centraremos en algunos de ellos.

### Operadores Aritméticos

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

### Operadores de Asignación



Los operadores de asignación se utilizan para asignar valores a las variables. Tomemos el signo = como ejemplo. En matemáticas, el signo igual indica que dos valores son iguales, pero en Python significa que estamos almacenando un valor en una cierta variable y lo llamamos asignación o asignando un valor a una variable.

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
//=	$x //= 3$	$x = x // 3$
**=	$x ** = 3$	$x = x ** 3$
&=	$x \& = 3$	$x = x \& 3$
=	$x   = 3$	$x = x   3$
^=	$x \wedge = 3$	$x = x \wedge 3$
>>=	$x >> = 3$	$x = x >> 3$
<<=	$x << = 3$	$x = x << 3$

### Ejemplo: Enteros




```
1 # Operaciones aritméticas en Python
2 # Integers
3 print('Suma: ', 1 + 2) # 3
4 print('Resta: ', 2 - 1) # 1
5 print('Multiplicación: ', 2 * 3) # 6
6 print('División: ', 4 / 2) # 2.0 La división en Python da un número flotante
7 print('División: ', 6 / 2) # 3.0
8 print('División: ', 7 / 2) # 3.5
9 print('División sin el resto: ', 7 // 2) # 3, devuelve sin el número flotante o sin el resto
10 print('División sin el resto: ', 7 // 3) # 2
11 print('Módulo: ', 3 % 2) # 1, Devuelve el resto
12 print('Exponenciación: ', 2 ** 3) # 8 significa 2 * 2 * 2
```

#### **Ejemplo: Números de punto flotante**

```
1 # Números de punto flotante
2 print('Número de punto flotante, PI', 3.14)
3 print('Número de punto flotante, gravedad', 9.81)
```

#### **Ejemplo: Números complejos**

```
1 # Números complejos
2 print('Número complejo: ', 1 + 1j)
3 print('Multiplicando números complejos: ', (1 + 1j) * (1 - 1j))
```

-  Declararemos una variable y asignaremos un tipo de dato numérico. Voy a usar una variable de un solo carácter, pero recuerda que no debes desarrollar el hábito de declarar este tipo de variables. Los nombres de las variables deberían ser todo el tiempo mnemónico

#### **Ejemplo 01: Declarando la variable al principio**

```
1 # Declarando la variable al principio
2 a = 3 # a es un nombre de variable y 3 es un tipo de dato entero
3 b = 2 # b es un nombre de variable y 3 es un tipo de dato entero
4
5 # Operaciones aritméticas y asignando el resultado a una variable
6 total = a + b
7 diff = a - b
8 product = a * b
9 division = a / b
10 remainder = a % b
11 floor_division = a // b
12 exponential = a ** b
13
14 # Debería haber usado sum en lugar de total, pero sum es una función incorporada
15 # intenta evitar sobrescribir las funciones incorporadas
16 print(total) # si no etiquetas tu impresión con alguna cadena, nunca sabrás de dónde viene el resultado
17 print('a + b = ', total)
18 print('a - b = ', diff)
19 print('a * b = ', product)
20 print('a / b = ', division)
21 print('a % b = ', remainder)
22 print('a // b = ', floor_division)
23 print('a ** b = ', exponential)
```



**Ejemplo 02:**

```
1 # Declarando la variable al principio
2 print('== Suma, Resta, Multiplicación, División, Módulo ==')
3
4 # Declarando valores y organizándolos juntos
5 num_one = 3
6 num_two = 4
7
8 # Operaciones aritméticas
9 total = num_one + num_two
10 diff = num_two - num_one
11 product = num_one * num_two
12 div = num_two / num_one
13 remainder = num_two % num_one
14
15 # Imprimir valores con etiqueta
16 print('total: ', total)
17 print('diferencia: ', diff)
18 print('producto: ', product)
19 print('división: ', div)
20 print('resto: ', remainder)
```

🐍 Empecemos a conectar los puntos y empecemos a hacer uso de lo que ya sabemos para calcular (área, volumen, densidad, peso, perímetro, distancia, fuerza).

```
1 # Calculando el área de un círculo
2 radio = 10 # radio de un círculo
3 area_de_circulo = 3.14 * radio ** 2 # dos * significa exponente o potencia
4 print('Área de un círculo:', area_de_circulo)
```

```
1 # Calculando el área de un rectángulo
2 longitud = 10
3 ancho = 20
4 area_de_rectangulo = longitud * ancho
5 print('Área de rectángulo:', area_de_rectangulo)
6
7 # Calculando el peso de un objeto
8 masa = 75
9 gravedad = 9.81
10 peso = masa * gravedad
11 print(peso, 'N') # Agregando unidad al peso
12
13 # Calcular la densidad de un líquido
14 masa = 75 # en Kg
15 volumen = 0.075 # en metros cúbicos
16 densidad = masa / volumen # 1000 Kg/m^3
17 print('Densidad: ', densidad, 'Kg/m^3')
```

**Operadores de Comparación**

🐍 En programación comparamos valores, utilizamos operadores de comparación para comparar dos valores. Verificamos si un valor es mayor o menor o igual a otro valor. La siguiente tabla muestra los operadores de comparación de Python.



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

### Ejemplo 03: Operadores de Comparación

```

1 # Operadores de Comparación
2 print(3 > 2) # Verdadero, porque 3 es mayor que 2
3 print(3 >= 2) # Verdadero, porque 3 es mayor que 2
4 print(3 < 2) # Falso, porque 3 es mayor que 2
5 print(2 < 3) # Verdadero, porque 2 es menor que 3
6 print(2 <= 3) # Verdadero, porque 2 es menor que 3
7 print(3 == 2) # Falso, porque 3 no es igual a 2
8 print(3 != 2) # Verdadero, porque 3 no es igual a 2
9 print(len('mango') == len('avocado')) # Falso
10 print(len('mango') != len('avocado')) # Verdadero
11 print(len('mango') < len('avocado')) # Verdadero
12 print(len('leche') != len('carne')) # Falso
13 print(len('leche') == len('carne')) # Verdadero
14 print(len('tomate') == len('patata')) # Verdadero
15 print(len('python') > len('dragón')) # Falso

```

```

1 # Comparar algo da como resultado un verdadero o falso
2 print('Verdadero == Verdadero: ', True == True)
3 print('Verdadero == Falso: ', True == False)
4 print('Falso == Falso: ', False == False)

```



Además de los operadores de comparación anteriores, Python usa:

**is:** Devuelve verdadero si ambas variables son el mismo objeto (x is y)

**is not:** Devuelve verdadero si ambas variables no son el mismo objeto (x is not y)

**in:** Devuelve Verdadero si la lista consultada contiene un cierto elemento (x in y)

**not in:** Devuelve Verdadero si la lista consultada no tiene un cierto elemento (x in y)

```

1 # Valores
2 print('1 es 1', 1 is 1) # Verdadero - porque Los valores de los datos son iguales
3 print('1 no es 2', 1 is not 2) # Verdadero - porque 1 no es 2
4 print('A en Asabeneh', 'A' in 'Asabeneh') # Verdadero - A se encuentra en la cadena
5 print('B en Asabeneh', 'B' in 'Asabeneh') # Falso - no hay B mayúscula
6 print('coding' in 'coding para todos') # Verdadero - porque 'coding para todos' tiene la palabra 'coding'
7 print('a en an:', 'a' in 'an') # Verdadero
8 print('4 es 2 ** 2:', 4 is 2 ** 2) # Verdadero

```



## Operadores Lógicos

A diferencia de otros lenguajes de programación, python usa las palabras clave and, or y not para los operadores lógicos. Los operadores lógicos se utilizan para combinar declaraciones condicionales:

Operator	Description	Example
and	Returns True if both statements are true	<code>x &lt; 5 and x &lt; 10</code>
or	Returns True if one of the statements is true	<code>x &lt; 5 or x &lt; 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>

### Ejemplo 04:

```

1 # Declaraciones condicionales
2 print(3 > 2 and 4 > 3) # Verdadero - porque ambas declaraciones son verdaderas
3 print(3 > 2 and 4 < 3) # Falso - porque la segunda declaración es falsa
4 print(3 < 2 and 4 < 3) # Falso - porque ambas declaraciones son falsas
5 print('Verdadero y Verdadero: ', True and False)
6 print(3 > 2 or 4 > 3) # Verdadero - porque ambas declaraciones son verdaderas
7 print(3 > 2 or 4 < 3) # Verdadero - porque una de las declaraciones es verdadera
8 print(3 < 2 or 4 < 3) # Falso - porque ambas declaraciones son falsas
9 print('Verdadero o Falso:', True or False)
10 print(not 3 > 2) # Falso - porque 3 > 2 es verdadero, entonces not Verdadero da Falso
11 print(not True) # Falso - Negación, el operador not convierte verdadero en falso
12 print(not False) # Verdadero
13 print(not not True) # Verdadero
14 print(not not False) # Falso

```

## 4. Desarrollo y Actividades

### Ejercicio parte 01:

- Encuentra la longitud del texto python, convierte el valor a float y luego conviértelo a string
- Los números pares son divisibles por 2 y el residuo es cero. ¿Cómo comprobarías si un número es par o no usando python?
- Comprueba si la división entera de 7 entre 3 es igual al valor de 2.7 convertido a entero.
- Comprueba si el tipo de '10' es igual al tipo de 10
- Comprueba si `int('9.8')` es igual a 10
- Escribe un script que solicite al usuario que introduzca las horas y la tarifa por hora. ¿Cuál es el pago de la persona?  
Introduce las horas: 40  
Introduce la tarifa por hora: 28  
Tu salario semanal es 1120
- Escribe un script que solicite al usuario que introduzca el número de años. Calcula el número de segundos que una persona puede vivir. Supongamos que una persona puede vivir cien años.  
Introduce el número de años que has vivido: 100  
Has vivido durante 3153600000 segundos.



- 8) Escribe un script en Python que muestre la siguiente tabla

```
1 1 1 1 1
2 1 2 4 8
3 1 3 9 27
4 1 4 16 64
5 1 5 25 125
```

- 9) Invertir una cadena: Escribe un programa que pida al usuario una cadena y luego la imprima al revés.
- 10) Contar vocales: Escribe un programa que cuente el número de vocales en una cadena dada por el usuario.
- 11) Palíndromo: Escribe un programa que verifique si una cadena dada por el usuario es un palíndromo (se lee igual de izquierda a derecha que de derecha a izquierda).
- 12) Reemplazar espacios: Escribe un programa que reemplace todos los espacios en una cadena dada por el usuario con guiones bajos.
- 13) Contar palabras: Escribe un programa que cuente el número de palabras en una cadena dada por el usuario.
- 14) Mayúsculas y minúsculas: Escribe un programa que convierta una cadena dada por el usuario a mayúsculas y luego a minúsculas, imprimiendo cada resultado.
- 15) Eliminar espacios en blanco: Escribe un programa que elimine los espacios en blanco al principio y al final de una cadena dada por el usuario.
- 16) Extraer subcadena: Escribe un programa que extraiga una subcadena de una cadena dada por el usuario, especificando los índices de inicio y fin.
- 17) Verificar prefijo y sufijo: Escribe un programa que verifique si una cadena dada por el usuario comienza con un prefijo específico y termina con un sufijo específico.