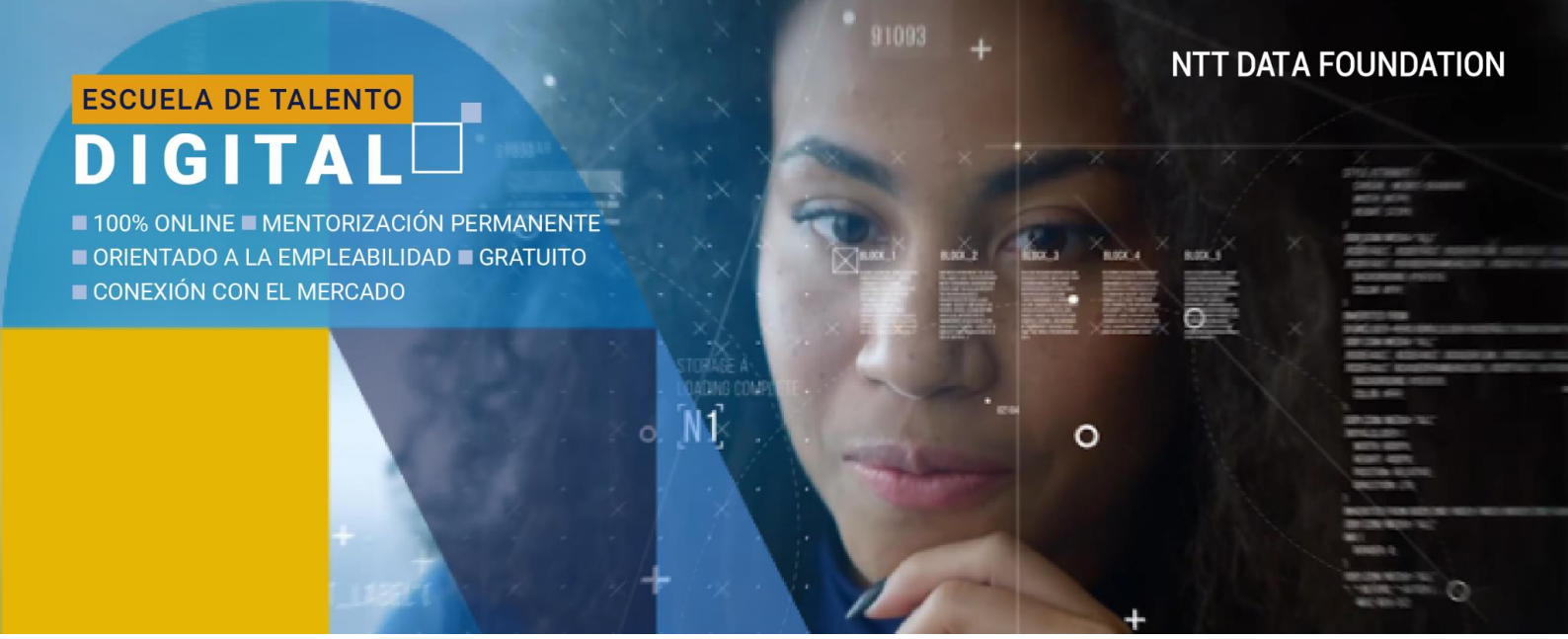

ESCUELA DE TALENTO DIGITAL

NTT DATA FOUNDATION

INTRODUCCIÓN A LA PROGRAMACIÓN

ÍNDICE

1. TIPOS DE DATOS

Los **tipos de datos** en programación se utilizan para definir el **tipo de información** que contiene un elemento. Si intentamos hacer una comparativa al lenguaje natural, esto podría ser similar a decir que, por ejemplo, 5 es un número o "a" es una letra, donde los tipos de datos serían número y letra respectivamente.

Algunos de los tipos de datos más utilizados en Python son:

Tipos	Explicación	Ejemplos
int	Números enteros	`-1, 0, 1, -45, 105, …`
float	Números decimales, de coma flotante o de punto flotante	`-1.05, 0.56, 2.23, …`
str	Texto, string o cadena de caracteres. Cualquier letra, palabra o frase. Este tipo siempre va entre comillas, simples o dobles.	`'A', 'a', "Hola", "Este tipo es texto", …`

list	Listas. Son conjuntos de elementos relacionados entre sí. Los elementos pueden ser del mismo tipo o no. Cada lista se representa entre corchetes.	`[1, 3, 5, 7], ['a', 'e', 'i', 'o', 'u'],`
		`[1, 'a', 2, 'b', 3, 'c'],`
		`["manzana", "pera", "naranja"]`

tuple	Tuplas. Son conjuntos de elementos, igual que las listas, pero se diferencian de estas en que sus valores no se pueden modificar. Cada tupla se representa entre paréntesis.	`(1, 3, 5, 7), ('a', 'e', 'i', 'o', 'u'),`
		`(1, 'a', 2, 'b', 3, 'c'),`
		`('rojo', 'verde', 'azul')`

dict	Diccionarios. Son conjuntos de pares (dos elementos) que contienen siempre una clave y un valor. Cada diccionario se representa entre llaves.	`{(1, 'hola'), (2, 'adios')}`
		Este diccionario tiene dos elementos. El primero sería `(1, hola)` donde 1 es la clave y hola el valor. El segundo elemento sería `(2, adios)` donde 2 es la clave y adios el valor. Otros ejemplos serían:
		`{("uno", 1), ("dos", 2), ("tres", 3), ("cuatro", 4)}`
		Donde uno, dos, tres y cuatro son las claves y 1, 2, 3 y 4 los valores.

set	Conjuntos. Son colecciones de elementos únicos no ordenados. En los conjuntos no hay elementos repetidos. Los elementos pueden ser del mismo tipo o no. Los conjuntos se representan entre llaves.	`{1, 2, 8}, {'a', 'b', 'c'},` `{'a', 1, 2, 'b', 'c'}, {'hola', 'adios'}`
bool	Booleanos. Este tipo de datos solo tiene dos valores posibles, True (verdadero) o False (falso).	`True, False`

Python permite hacer **conversiones de tipos**, es decir, permite transformar datos de un tipo en otro tipo utilizando funciones específicas para ello.

Por ejemplo, si tenemos un dato de tipo float, pero para la operación que vamos a realizar queremos convertirlo en un entero, utilizaremos la función `int( )`. Si nuestro tipo decimal es el número 2.4, la función `int (2.4)` nos devolverá el entero 2.

Si, por el contrario, tenemos un tipo entero, 2, pero necesitamos un decimal, usaremos la función `float ( )`. De esta forma, `float (2)` nos devolverá el decimal 2.0.

Hay otra función muy útil, `str( )`, que convierte el valor contenido entre paréntesis en un texto, pero veremos en el apartado Cadenas de texto, de este documento, cómo se utiliza.

2. VARIABLES

Las variables en programación son elementos, con un nombre específico, en los que se **guardan datos** durante la ejecución de un programa. Los datos almacenados en las variables pueden ser de los tipos que hemos visto anteriormente.

Por ejemplo, si tenemos la variable `numero = 10`, tenemos una variable que se llama numero y que tiene como valor el entero 10. Si, en cambio, nuestra variable es `coche = “rojo”`, nuestra variable se llamará coche y su valor es el texto “rojo”.

Python **distingue entre minúsculas y mayúsculas** en los nombres de las variables, por lo que coche sería una variable y Coche otra variable diferente.

La información contenida en una variable puede variar o se puede modificar a lo largo de un programa.

Por ejemplo, si tenemos la variable `x = 2`, es decir, tenemos una variable que se llama x y cuyo valor es 2, y después tenemos la sentencia `x = x + 1` (o lo que es lo mismo, la sentencia `x += 1`) estaremos diciendo que el nuevo valor de nuestra variable es 3, ya que, si sustituimos la x de la parte derecha de la ecuación por su valor real, tendremos que x = 2 + 1 = 3, por lo tanto, el nuevo valor de x será 3.

Python permite que sea el usuario de un programa el que le de valor a una variable a través del teclado. Para ello se utilizar el método `input( )`.

Por ejemplo, si tenemos la variable llamada coche, podemos no darle un valor inicial y preguntarle directamente al usuario con el siguiente fragmento de código:

```
coche = input (“¿De qué color es tu coche?”)
```

Aquí le estamos diciendo al programa que tenemos una variable que se llama coche y que será igual a los datos que introduzca el usuario en el teclado después de que se le muestre en pantalla el mensaje “¿De qué color es tu coche?”.

Si el usuario introduce la palabra rojo en el teclado, el valor de la variable coche será rojo.

3. OPERADORES

Los operadores son los elementos que nos permiten, en programación, hacer operaciones.

Los tipos de operadores que nos podemos encontrar son:

3.1. Aritméticos

Son los operadores que realizan un cálculo entre dos elementos.

Operador	Explicación	Ejemplo	Resultado
`+`	Adición. Suma dos elementos	`1 + 5`	6
`-`	Sustracción. Resta dos elementos	`5 - 1`	4
`*`	Multiplicación. Multiplica dos elementos	`5 * 2`	10
`/`	División. Divide dos elementos	`16 / 2`	8
`%`	Módulo o resto. Devuelve el resto de la división de dos números	`11 % 4`	3 (El resto de dividir 11 entre 4)
`**`	Exponenciación. Devuelve el primero de los números elevado al segundo de los números	`4**3`	64 (4 al cubo o 4 elevado a 3)
`//`	División con redondeo. Devuelve el entero del resultado de una división, aunque esta tenga decimales	`11 // 4`	2 (El resultado de la división es 2.75, pero este operador solo devuelve el entero)

3.2. Relacionales

Son los operadores que comparan dos elementos.

Operador	Explicación	Ejemplo	Resultado
`==`	Igual que. Compara dos elementos y devuelve True si ambos son iguales, si no devuelve False	`3 == 3` `3 == 4`	True False
`!=`	Distinto a. Compara dos elementos y devuelve True si ambos son diferentes, si no devuelve False	`3 != 4` `6 != 6`	True False

`>`	Mayor que. Devuelve True si el primer elemento es mayor que el segundo, si no devuelve False	`7 > 2`
		`7 > 7`
		`7 > 8`
			True
			False
			False

`>=`	Mayor o igual que. Devuelve True si el primer elemento es mayor o igual que	`7 >= 2`	True

	el segundo elemento, si no devuelve False	`7 >= 7` `7 >= 8`	True False

`<`	Menor que. Devuelve True si el primer elemento es menor que el segundo, si no devuelve False	`2 < 7`
		`2 < 2`
		`2 < 1`
			True
			False
			False

`<=`	Menor o igual que. Devuelve True si el primer elemento es menor o igual que el segundo, si no devuelve False	`2 <= 7`
		`2 <= 2`
		`2 <= 1`
			True
			True
			False

3.3. Lógicos

Son operadores que se utilizan para evaluar dos Booleanos y devolver otro Booleano en función de esa evaluación.

Operador	Explicación	Ejemplo	Resultado
`and`	`A and B`. Devuelve True si tanto A como B son verdaderas, es decir, si se cumplen ambas condiciones	`(5 > 2) and (5 < 8)`	True (se cumplen ambas, 5 es mayor que 2 y menor que 8)
		`(5 > 2) and (5 > 8)`	False (solo se cumple la primera condición)
`or`	`A or B`. Devuelve True si A o B son verdaderas, es decir, si se cumple solo una de las dos condiciones	`(5 > 2) or (5 > 8)`	True (se cumple la primera condición y no es necesario que se cumpla la segunda)
		`(5 < 2) or (5 > 8)`	False (no se cumple ninguna de las dos)
`not`	`not A`. Invierte el valor de A, es decir, devuelve True si A es False	`not (2 > 8)`	True (como 2 > 8 es False, devuelve lo contrario, True)
		`not (2 < 8)`	False (como 2 < 8 es True, devuelve False)

3.4. De pertenencia

Son operadores que se utilizan para evaluar si un elemento está en un conjunto de elementos.

Operador	Explicación	Ejemplo	Resultado
`in`	`A in B`. Devuelve True si A está contenido en el conjunto B.	`A = 5; B = [1, 3, 5]` `A in B`	True
		`A = 5; B = [1, 3, 6]` `A in B`	False
`not in`	`A not in B`. Es el opuesto del anterior y devuelve True si A no está en B.	`A = 5; B = [1, 3, 5]` `A not in B`	False
		`A = 5; B = [1, 3, 6]` `A not in B`	True

3.5. De identidad

Son operadores que evalúan si dos elementos hacen referencia al mismo objeto en memoria. Pero OJO, dos elementos pueden ser iguales, pero no hacer referencia al mismo elemento de memoria.

Operador	Explicación	Ejemplo	Resultado
`is`	`A is B`. Devuelve True si A y B se refieren al mismo objeto.	`A = x, B = x` `A is B`	True
		`A = x, B = y` `A is B`	False
`is not`	`A is not B`. Es el opuesto del anterior y devuelve True si A y B no se refieren al mismo elemento.	`A = x, B = x` `A is not B`	False
		`A = x, B = y` `A is not B`	True

3.6. Bit a bit

Son los operadores que realizan operaciones a nivel de bit en una expresión. Un bit es la unidad mínima de información en informática y solo puede valer 0 o 1.

Operador	Explicación	Ejemplo	Resultado

`&`	`A & B`. Realiza bit a bit la operación and entre A y B.	`A = 101` (no es el número 101, sino la cadena de 3 bits 1 0 1)
		`B = 110` (la cadena de 3 bits 1 1 0)
		`A & B` realizará las operaciones por cada bit así:
		`1` (el primer elemento de A) `and` `1` (el primer elemento de B) `= 1`
		`1` (2º de A) `and` `0` (2ª de B) `= 0`
		`1` (3º de A) `and` `0` (3º de B) `= 0`
			`100` (no es el número 100 sino la cadena 1 0 0)

`|`	`A | B`. Realiza bit a bit la operación or entre A y B.	`A = 101`
		`B = 110`
		`A | B` realizará las operaciones:
		`1 or 1 = 1`
		`0 or 1 = 1`
		`1 or 0 = 1`
			`111`

`^`	`A ^ B`. Compara bit a bit los elementos y devuelve 0 si ambos son iguales y 1 si son diferentes.	`A = 101`
		`B = 110`
		`A ^ B` comparará:
		`1 = 1 -> 0`
		`0 = 1 -> 1`
		`1 = 0 -> 1`
			`011`

`~`	`~ A`. Realiza bit a bit la operación not, es decir, invierte los valores de los bits uno a uno.	`A = 101`	`010`
`>>`	`A >> B`. Desplaza los bits de A a la derecha tantos bits como indica B.	`A = 010` `B = 1`	`A = 001`
		`A = 010` `B = 2`	`A = 000`
`<<`	`A << B`. Desplaza los bits de A a la izquierda tantos bits como indica B.	`A = 010` `B = 1`	`A = 100`

4. CADENAS DE TEXTO

La mayoría de los tipos de datos son un poco más complejos que int y float, por lo que vamos a ir viendo, durante el curso, como se opera con cada uno de ellos. Empezaremos con el tipo str o tipo texto.

Algunas de las operaciones más destacadas que podemos realizar en los tipos str son:

4.1. Acceso directo a cualquier carácter

Las cadenas de texto, en Python, nos permiten acceder directamente a un carácter situado en una posición específica. Es decir, si tenemos la variable `frase = "Me gusta este curso"`, podríamos extraer el carácter situado, por ejemplo, en la posición 3.

Esto se haría de la siguiente manera: `frase[3]`, que nos devolvería la g. Ojo que para que se vea por pantalla esta g, tienes que utilizar la función `print`. Es decir, `print(frase[3])`. En el resto de los ejemplos que verás en esta sección pasa lo mismo, para verlo por pantalla utiliza `print`.

Ten en cuenta en para Python el primer carácter es el 0, por lo tanto, en la posición 0 tendríamos la M, en la posición 1 la e, en la posición 2 el espacio en blanco y en la posición 3 la g.

4.2. Creación de subcadenas

Python nos permite seleccionar, no solo un carácter en una posición, sino una subcadena compuesta por varios caracteres. Para ello hay varias opciones que vamos a ver con ejemplos.

Si tenemos la misma variable de antes, `frase = "Me gusta este curso"`, imagina que quieres extraer la parte "gusta este", en este caso, lo harías así: `frase[3:13]`.

Como ves, la última vocal de este, la segunda e, está en la posición 12, pero debes tener en cuenta que Python llega hasta el carácter anterior al número indicado en segunda posición, en este caso si quieres llegar hasta el carácter 12, tendrás que indicarle al programa que recorra el texto hasta el siguiente carácter, el 13.

Es decir, si tienes un programa que diga:

```
frase = "Me gusta este curso"
```

La sentencia `print (frase[3:12])` sacará por pantalla el resultado gusta est

Si quieres que el resultado sea gusta este, la sentencia será `print (frase[3,13])`

También, puedes crear subcadenas desde un carácter cualquier hasta el final. En el caso de nuestro ejemplo, si quieres seleccionar la subcadena este curso, utilizarás `frase[9:]`. Es

decir, si no indicas un número después de los dos puntos, querrá decir que quieres la subcadena que va desde el índice indicado hasta el final del texto.

Python permite, además, recorrer la cadena de caracteres en sentido inverso, es decir, empezando por el final. Para ello solo tienes que indicar cuantos caracteres quieres desde el final de la frase. Si en nuestro ejemplo queremos la subcadena curso, se lo indicaremos al programa con `frase[-5:]`. Esto le dice que quieres extraer los últimos 5 caracteres.

Por último, podemos dividir la cadena de texto en varias subcadenas independientes. Por ejemplo, imagina que quieres obtener todas las palabras que están separadas por espacios en blanco. Para ello utilizarías la función Split sobre la variable frase que estamos utilizando, indicándole cómo hacer las subcadenas. En este caso, vamos a pedirle que haga subcadenas cada vez que haya un espacio en blanco. Se haría así: `frase.split (“ ”)`, lo que nos devolvería `[“Me”, “gusta”, “este”, “curso”]`. Ten en cuenta que el carácter por el que hace la separación no se guarda.

Por ejemplo, si le decimos que separe en subcadenas siempre que se encuentre una e, es decir, `frase.split (“e”)`, el resultado sería `[“M”, “ gusta”, “st”, “ curso”]`. Como ves, en este caso, no mantiene las e pero sí mantiene los espacios en blanco delante de gusta y curso.

4.3. Modificación de elementos

Hay diferentes funciones que nos permiten modificar los caracteres que tenemos en nuestro tipo str. Veamos algunas de ellas:

- **Reemplazar un carácter**. Imagina que, en nuestro ejemplo, `frase = “Me gusta este curso”`, queremos sustituir la letra u por la letra h. Para ello utilizaríamos la función `replace` de la siguiente manera, `frase.replace(“u”, “h”)`, con lo que obtendríamos `“Me ghsta este chrso”`
- **Poner todo el texto en mayúsculas o minúsculas**. Para cambiar todos los caracteres de un texto y ponerlos en mayúsculas, puedes utilizar la función `upper()`, y para ponerlos en minúsculas, la función `lower()`, que funcionan de esta manera:
  - `frase.upper()` devolverá `“ME GUSTA ESTE CURSO”`
  - `frase.lower()` devolverá `“me gusta este curso”`
- **Poner solo la primera letra de la frase en mayúsculas**. Si tenemos que frase es `“me gusta este curso”`, al ejecutar la función `capitalize`, `frase.capitalize()`, obtendrás `“Me gusta este curso”`.
- **Eliminar espacios en blanco**. La función `strip()` te permite eliminar todos los espacios en blanco que hay antes o después de una frase. Es decir, si tenemos que nuestro ejemplo es frase = “ Me gusta este curso “, frase.strip() nos devolverá `“Me gusta este curso”`

4.4. Concatenación de cadenas

Dos o varias cadenas de texto se pueden unir para dar otra cadena de texto. Es decir, si tienes tres variables de texto diferentes, por ejemplo, `texto1 = "Me gusta"`, `texto2 = "este"` y `texto3 = "curso"`, puedes unirlas para tener otra variable, frase, que sea "Me gusta este curso". Esto lo puedes hacer de varias formas:

- **Con el operador +**. Esto concatenaría las diferentes variables, pero no dejaría espacios en blanco entre ellas. Es decir, si tienes `print(texto1 + texto2 + texto3)` verías por pantalla `Me gustaestecurso`. Cuando usas este tipo de concatenación, deberías tenerlo en cuenta a la hora de definir la variable y dejar el espacio blanco donde consideres. Por ejemplo, en este caso, podrías definir `texto2 = " este "`, con espacio delante y detrás de este, con lo que ya tendrías `"Me gusta este curso"`.
- **Con el operador ,**. Este operador hace lo mismo que +, pero introduce directamente espacios en blanco entre cada una de las cadenas. Es decir, `print(texto1, texto2, texto3)` devolvería `"Me gusta este curso"`.
- **Con el operador incremental, +=**. Este operador va añadiendo elementos a una variable. Para utilizarlo, debemos definir primero una variable que será la que se va incrementando con el resto de las variables. Por ejemplo, si creamos la variable `texto = ""`, que es una variable tipo srt vacía, y queremos crear la frase "Me gusta este curso" con las variables texto1, texto2 y texto3, podremos hacerlo de la siguiente manera:

  `texto += texto1` (ahora texto no estaría vacía, tendría guardado "Me gusta")

  `texto += texto2` (a lo que había en texto ("Me gusta") se añadiría lo que hay en texto2 ("este") y tendrías que texto tiene guardado "Me gustaeste")

  `texto += texto3` (a lo que había en texto se añadiría texto3)

  Por lo que tendríamos al mostrarlo por pantalla, `print(texto)`, `"Me gustaestecurso"`. Ojo que el operador += no añade espacios en blanco, por lo que tendríamos que definirlos en las variables, tal y como pasaba con el operador +.

4.5. Concatenación de cadenas y números

Cuando queramos que un número se integre dentro de una cadena de texto, y se concatene con esta, tenemos que transformar ese número en texto con la función `str()`. Veámoslo con un ejemplo.

Imagina que preguntas por pantalla a un usuario cuántos años tiene y quieres devolver la frase "Tienes x años", sustituyendo la x por el número de años que ha introducido el usuario.

Esto se haría así:

```
numero = input("¿Cuántos años tienes? ")
```

`print(“Tienes”, str(numero), “años”)` o `print(“Tienes “ + str(numero) + “ años”)` o `print (“Tienes” + numero + “años”)`

Y por pantalla saldría:

`¿Cuántos años tienes?`

Introducirías, por ejemplo, 42, y te devolvería:

`Tienes 42 años`

Cuando queremos concatenar una variable con una cadena, como en el caso anterior, o incluso más de una variable, para no tener que usar tantas comillas y operadores para separar los textos, podemos usar la función `f ” “`. Si tenemos el mismo ejemplo que en el caso anterior, en lugar de `print(“Tienes “ + str(numero) + “ años”)`, podríamos usar la `f ” “` así, `print (f”Tienes {numero} años”)`, cuyo resultado sería Tienes x años (x sería el número que hayamos introducido en pantalla). Eso sí, la variable siempre irá entre llaves { }. Ten en cuenta que al usar `f “ “`, no tienes que transformar los números en texto, aquí sí se incluye directamente el valor de la variable.

Si tenemos más de una variable, tendríamos que incluir cada una entre llaves, por ejemplo:

```
numero = 4

texto = “equipo”

print(f”Nuestro {texto} tiene {numero} jugadores”)
```

Devolvería `Nuestro equipo tiene 4 jugadores`

La función `format` hace lo mismo que f, pero las variables se pasan al final en lugar de en el texto. Es decir, en el ejemplo anterior usando format tendríamos:

```
print(“Nuestro {} tiene {} jugadores”.format(texto, numero))
```

Devolvería `Nuestro equipo tiene 4 jugadores`

El operador `%` funciona exactamente igual que `format`, pero en lugar de llaves en el texto incluiremos `%s`. Veámoslo con el ejemplo anterior:

```
Print(“Nuestro %s tiene %s jugadores” % (texto,numero))
```

Devolvería `Nuestro equipo tiene 4 jugadores`, igual que en el caso anterior.

4.6. Introducción de caracteres especiales

Hay ocasiones en las que dentro de un texto queremos introducir un carácter especial que tiene un significado para el código y que el programa no lo va a reconocer como parte del texto sino como un operador, y nos puede dar problemas a la hora de ejecutarlo. Por ejemplo, imagina que en nuestra frase anterior “Me gusta este curso”, nosotros queremos que por pantalla aparezca Me gusta “este” curso, con la palabra este entre comillas. El programa si

ponemos `print(“Me gusta “este” curso”)` va a interpretar que las comillas de antes de este son las que cierran las comillas abiertas antes de Me, y va a lanzar un error, porque interpretaría que le estás diciendo que muestre `“Me gusta ” + este + “ curso”`, pero sin utilizar los operadores +, por lo que lanzaría un error.

Para poder introducir este tipo de caracteres especiales se utiliza el operador `\`. Este operador se coloca antes del carácter especial en ambas ocasiones, para decirle al programa “Eh, el carácter que voy a incluir ahora es un carácter que quiero que muestres por pantalla”. Por lo tanto, `print(“Me gusta \”este\” curso”)` devolverá `Me gusta “este” curso`