

2. Conceptos básicos (1/2)

Introducción a Python para Deep Learning

JUNTA DE EXTREMADURA
Consejería de Economía, Empleo y Transformación Digital

ETD
EXTREMADURA
Estrategia de Transformación
Digital de Extremadura


INTIA
INSTITUTO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

uex



Contenidos

1. Documentación
2. Configuración de VSC
3. Entrada y salida básica
4. Variables y tipos de datos
5. Operadores y expresiones
6. Estructuras de control
7. Cadenas de caracteres

0. Introducción

- En las siguientes dos sesiones veremos los conceptos básicos de Python, desde variables hasta clases, pasando por tipos de datos
- Sin una base sólida sería imposible afrontar lo que queda por delante de este curso, y mucho menos los cursos más avanzados del programa



1. Documentación

- Nuestro código debería ser limpio
- Idealmente, el código debe estar autodocumentado
- Pero, hasta llegar a ese punto, es necesario mucho esfuerzo



1. Documentación

```
def process_data(data):  
    # Filter data  
    data = [x for x in data if x > 0]  
  
    # Calculate average  
    average = sum(data) / len(data)  
  
    # Print results  
    print(f"Average: {average}")  
    return data
```

1. Documentación

```
def filter_positive_data(data):  
    return [x for x in data if x > 0]
```

```
def calculate_average(data):  
    return sum(data) / len(data)
```

```
def print_results(average):  
    print(f"Average: {average}")
```

```
data = filter_positive_data(raw_data)  
average = calculate_average(data)  
print_results(average)
```

1. Documentación

- Nuestro código debería ser limpio
- Idealmente, el código debe estar autodocumentado
- Pero, hasta llegar a ese punto, es necesario mucho esfuerzo
- Mientras alcanzamos la perfección: pragmatismo
- Solución: documentación



1. Documentación

- Los comentarios permiten añadir líneas no ejecutables de código
- Imprescindibles para explicar el funcionamiento de nuestros programas...
 - ...a otras personas
 - ...¡y a nuestro futuro yo!



Fuente: [Reddit](#)

1. Documentación

Tipos de comentarios en Python

1. Líneas: #
2. Bloques: ¡no hay!
3. Pero Python ignora cadenas sin asignación: bloques `""" """`
4. Consejo: documentar a medida que escribimos código
5. Utilidad: comentarios para cancelar ciertas instrucciones

Y ya que estamos, vamos a configurar VSC

2. Configuración de VSC

- La configuración de Visual Studio Code es... árida
- Pero, a vez, ¡muy flexible!
- Podemos usar su editor (**File > Preferences > Settings**)



2. Configuración de VSC

Mi configuración inicial aconsejada:

1. Mostrar caracteres no imprimibles
Editor: Render Whitespace: all
2. Marcador de columna 80
Editor: Rulers (no ofrece interfaz, luego lo vemos)
3. Eliminar espacios al final
Files: Trim Trailing Whitespace
4. Añadir línea al final al guardar
Files: Insert Final New Line
5. Tabulador son cuatro espacios
Editor: Insert Spaces
Editor: Tab Size: 4



2. Configuración de VSC

1. Y esto puede ir en el archivo principal...
2. ...o podemos guardarlo en un archivo especial dentro de una carpeta
3. Lo ideal: crear una carpeta para vuestros proyectos (o para esta sesión)
4. Dentro crear la carpeta **.vscode**
5. Dentro crear el archivo **settings.json**
6. Y dentro, escribir lo que viene a continuación
7. Así tenemos el código y la configuración del editor guardados dentro del mismo proyecto
8. Y entonces, abrimos una carpeta, no un archivo

2. Configuración de VSC

Mi configuración inicial aconsejada:

1. Mostrar caracteres no imprimibles
`"editor.renderWhiteSpace": "all"`
2. Marcador de columna 80
`"editor.rulers": [80]`
3. Eliminar *trailing spaces* (espacios al final)
`"files.trimTrailingWhitespace": true`
4. Añadir línea al final al guardar
`"files.insertFinalNewline": true`
5. Tabulador son cuatro espacios
`"editor.insertSpaces": true`
`"editor.tabSize": 4`



2. Configuración de VSC

```
{  
  "editor.renderWhitespace": "all",  
  "editor.rulers": [80],  
  "files.trimTrailingWhitespace": true,  
  "files.insertFinalNewline": true,  
  "editor.insertSpaces": true,  
  "editor.tabSize": 4  
}
```

3. Entrada y salida básica

1. `print()`

1. Ya conocemos un ejemplo ("Hola, mundo")
2. ¿Para qué podemos usar esta instrucción?
3. Haced la prueba en vuestros equipos, que cada uno con mensaje diferente (vuestro nombre, por ejemplo)
4. Y ahora, ¿qué?

```
print("Hola, Andy")
```

3. Entrada y salida básica

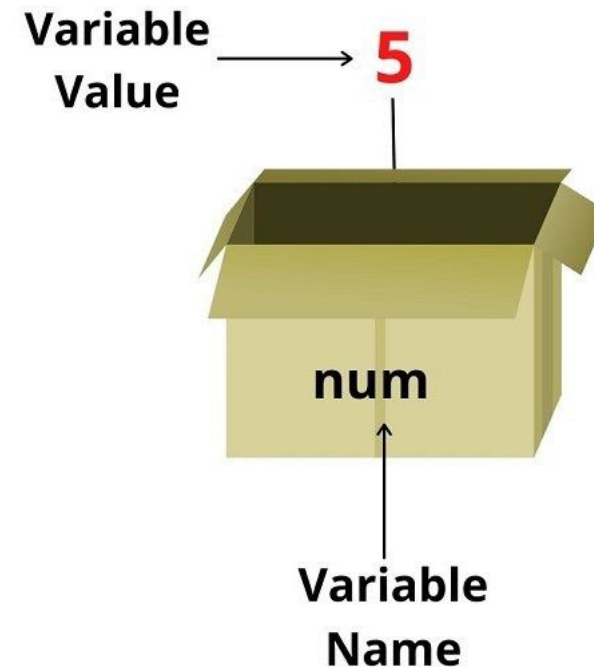
2. `input()`

1. Como `print()` ... bueno, casi
2. Hasta que el usuario no ponga algo, no se vuelve al programa
3. Haced un programa básico que pida vuestro nombre (`input`) y luego dé las gracias (`print`)
4. Y ahora... ¿qué?

```
input("Pulsa cualquier tecla para continuar...")  
print("Sigamos, pues")
```


4. Variables y tipos de datos

- En Python, las variables tienen un nombre, y se les asigna un valor
- No es necesaria una declaración explícita
- No es necesario indicar su tipo explícitamente
- Podemos saber el tipo de una variable con la función `type ()`
- Sin embargo, es un lenguaje con tipado dinámico



Fuente: [Pinterest](#)

4. Variables y tipos de datos

Python es de tipado dinámico

No necesitas declarar el tipo de las variables

```
x = 10          # x es un entero
```

```
print(type(x))  # <class 'int'>
```

```
x = "Hola"      # Ahora x es una cadena de caracteres
```

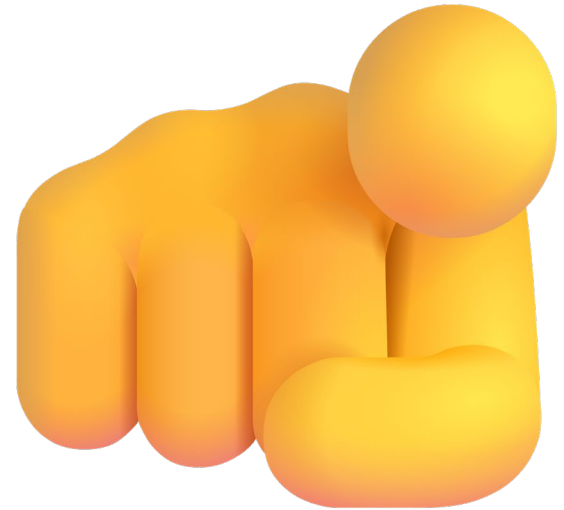
```
print(type(x))  # <class 'str'>
```

```
x = 3.14         # Ahora x es un número en coma flotante
```

```
print(type(x))  # <class 'float'>
```

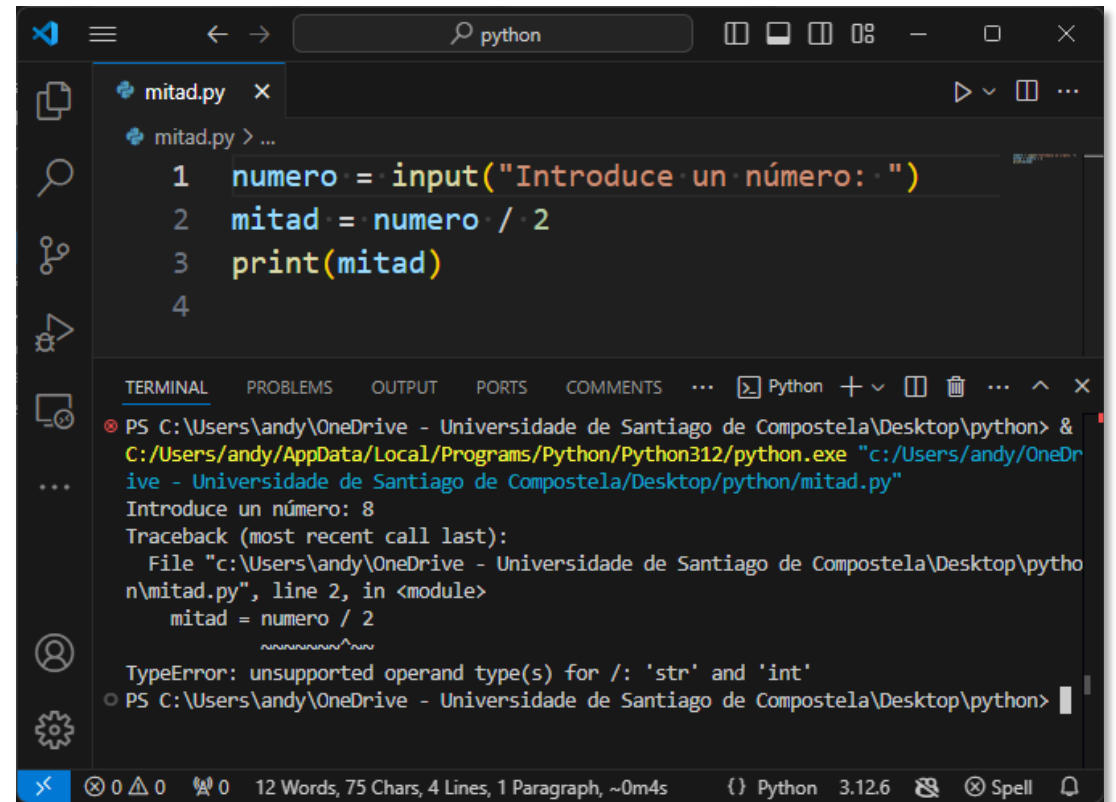
4. Variables y tipos de datos

- **Ejercicio 1:** pedir el nombre por pantalla y mostrarlo
- **Ejercicio 2:** pedir un número por pantalla y mostrarlo
- **Ejercicio 3:** pedir un número por pantalla, dividirlo entre 2 y mostrarlo



4. Variables y tipos de datos

- Debemos leer los mensajes de error con atención
- En este caso, está explicado qué ocurre
- Lo que pasa es que quizá no sea fácil entenderlo...
- Además, VSC nos ayuda a conocer los tipos de las variables...



The screenshot shows the Visual Studio Code interface with a file named `mitad.py` open. The code in the editor is:

```
1 numero = input("Introduce un número: ")
2 mitad = numero / 2
3 print(mitad)
4
```

Below the editor, the TERMINAL panel shows the command used to run the script and the resulting error message:

```
PS C:\Users\andy\OneDrive - Universidade de Santiago de Compostela\Desktop\python> & C:/Users/andy/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/andy/OneDrive - Universidade de Santiago de Compostela/Desktop/python/mitad.py"
Introduce un número: 8
Traceback (most recent call last):
  File "c:\Users\andy\OneDrive - Universidade de Santiago de Compostela\Desktop\python\mitad.py", line 2, in <module>
    mitad = numero / 2
            ~~~~~^~~~~
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

The error message indicates a `TypeError: unsupported operand type(s) for /: 'str' and 'int'`, which occurs because the `input()` function returns a string, and it is being divided by an integer.

4. Variables y tipos de datos

Tipos básicos

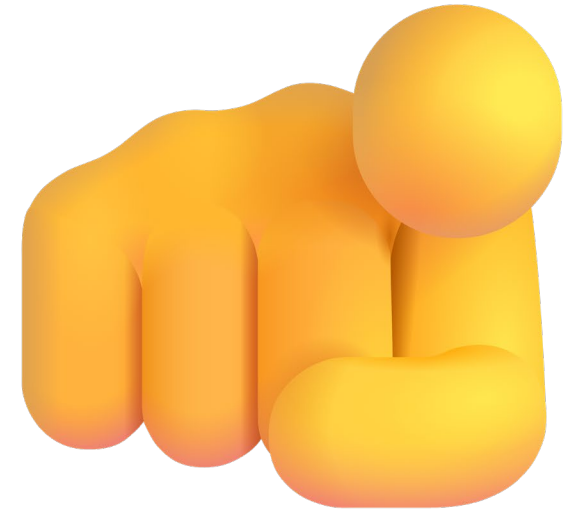
1. Enteros: `int`
2. Coma flotante: `float`
3. Booleanos: `bool`
4. Cadenas de caracteres: `str`
5. Nulo: `None`

4. Variables y tipos de datos

Escribe un programa que reciba cinco entradas del usuario y las convierta a los siguientes tipos:

- Entero (`int`)
- Flotante (`float`)
- Booleano (`bool`)
- Cadena de texto (`str`)

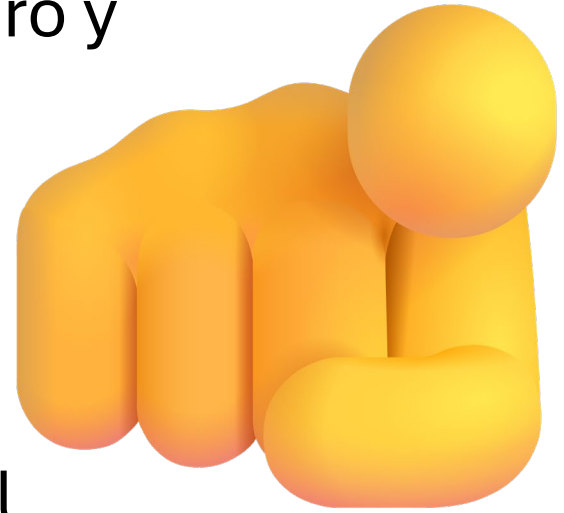
Para terminar, muestra los valores



4. Variables y tipos de datos

Escribe un programa que reciba un número entero y realice las siguientes operaciones:

- Convierte el número a flotante y muestra el resultado
- Convierte el número a una cadena y muestra el resultado
- Convierte el número a un booleano y muestra el resultado



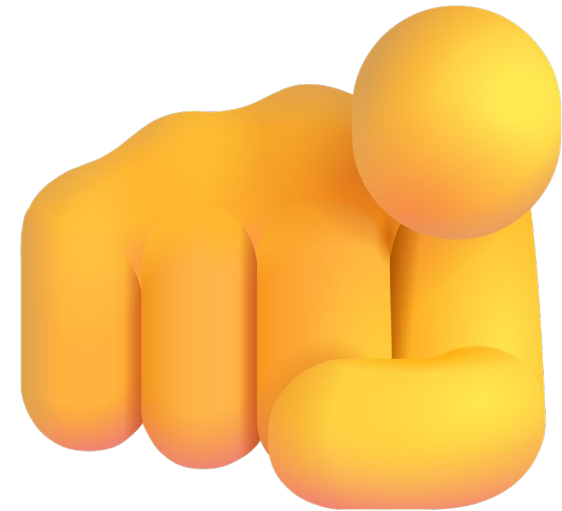
5. Operadores y expresiones

1. **Asignación:** =
2. **Comparación:** ==, !=, <, >, <=, >=
3. **Aritméticos:** +, -, *, /, %, **, //
4. **Asignación (de nuevo):** =, +=, -=, etcétera
5. **Lógicos:** and, or, not
6. **Identidad:** is, is not
7. **Bit:** & (and), | (or), ^ (xor), ~ (not), << (desplazamiento a la izquierda), >> (desplazamiento a la derecha)

5. Operadores y expresiones

Escribe un programa que reciba dos números del usuario y realice las siguientes comparaciones:

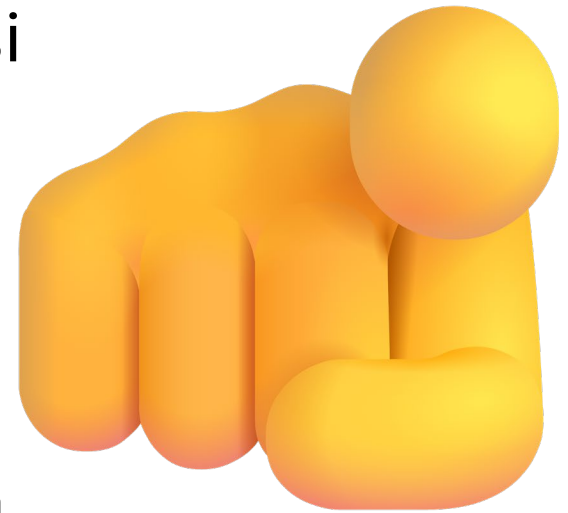
- Mayor que ($>$)
- Menor que ($<$)
- Igual a ($==$)
- Distinto de ($!=$)



5. Operadores y expresiones

Escribe un programa que compare dos cadenas de texto introducidas por el usuario. Debe verificar si las cadenas:

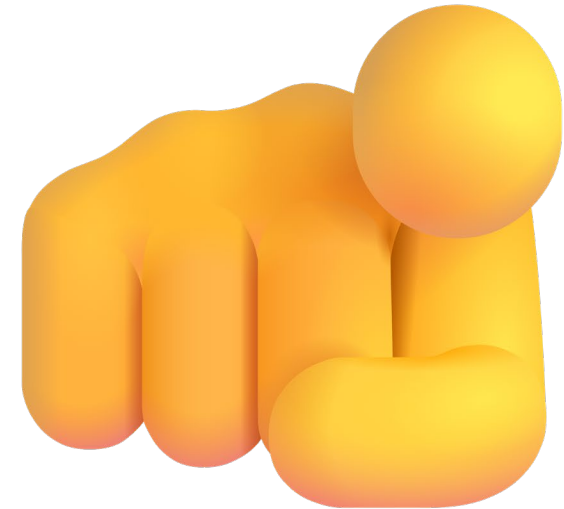
- Son iguales (==)
- Son diferentes (!=)
- Si la primera cadena es "mayor" que la segunda (alfabéticamente) (>)
- Si la primera cadena es "menor" que la segunda (<)



5. Operadores y expresiones

Escribe un programa que reciba dos valores booleanos (True o False) y realice las siguientes comparaciones:

- Verifica si ambos son True
- Verifica si alguno es True
- Compara si son iguales



6. Estructuras de control

En principio, sólo tenemos:

1. Condicional: If
2. Condicional: match
3. Bucle: For
4. Bucle: While

¿Para qué sirve cada una?
¿Os falta algo?



6. Estructuras de control

Condicional

Las estructuras condicionales permiten ejecutar diferentes bloques de código en función de una condición

```
if condicion:  
    # código si la condición  
    # es True  
elif otra_condicion:  
    # código si la otra  
    # condición es True  
else:  
    # código si ninguna  
    # condición es True
```

6. Estructuras de control

Condicional

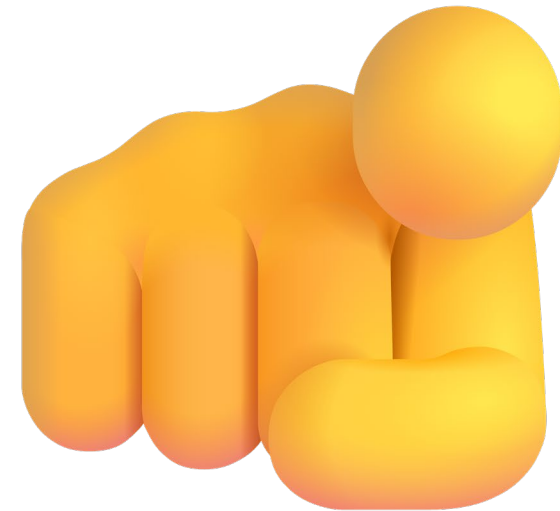
- Con match podemos tener código más limpio
- Parece que han tardado en incluirlo (3.10)

```
valor = 1
match valor:
    case 1:
        print("Es 1")
    case 2:
        print("Es 2")
    case _:
        print("Es...")
```

6. Estructuras de control

Condicional

Revisemos uno de los ejemplos anteriores: pidamos un número, comprobemos si lo que nos han dado lo es. Usad la propiedad `isdigit()` de las cadenas de texto antes de dividirlo entre dos



6. Estructuras de control

Bucle For

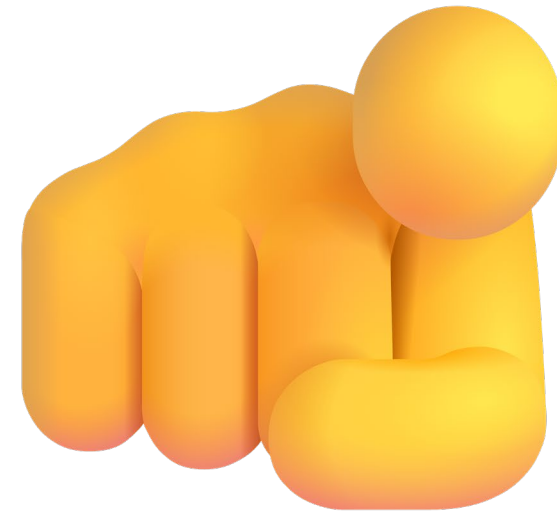
Permiten iterar sobre una secuencia y ejecutar un bloque de código para cada elemento

```
for variable in secuencia:  
    # código que se ejecuta  
    # en cada iteración
```


6. Estructuras de control

Bucle For

Ahora, sobre el ejercicio anterior, una vez tengamos el número, contemos hasta llegar a él. 🤖 : `range(numero)`



6. Estructuras de control

Bucle While

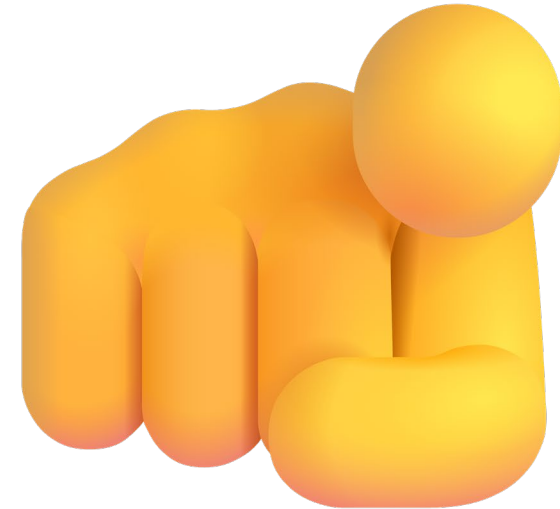
Ejecuta un bloque de código repetidamente mientras una condición sea True

```
while condicion:  
    # código que se  
    # ejecuta mientras la  
    # condición sea True
```

6. Estructuras de control

Bucle While

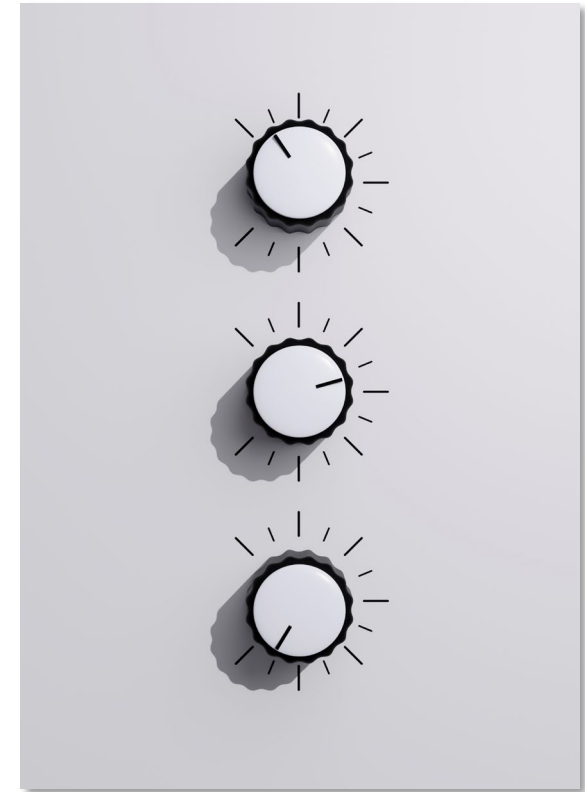
Ahora, sobre el ejercicio anterior, pero sin `range ()`



6. Estructuras de control

Modificadores

- **break**: sale del bucle antes de que realice todas las iteraciones
- **continue**: omite el resto del código de la iteración actual, pasa a la siguiente
- **pass**: no hace nada
- **else**: en bucles, si no se ha usado break

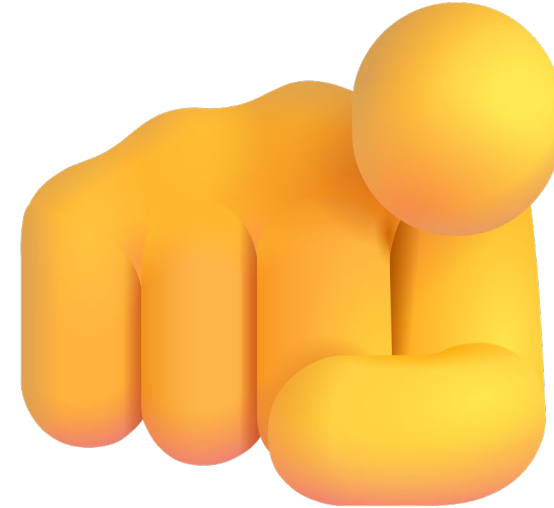


6. Estructuras de control

Modificadores

Escribe un programa que recorra una lista de números del 1 al número introducido por el usuario. El programa debe realizar lo siguiente:

- Si el número es divisible por 5, utiliza `continue` para saltar esa iteración (no imprimir el número)
- Si el número es divisible por 7, utiliza `break` para terminar el bucle completamente
- Usa `pass` para aquellos números pares (no debe hacer nada, simplemente seguir con el bucle)
- Al final del bucle, si ha terminado sin usar `break`, imprime un mensaje indicando que el bucle se completó exitosamente



7. Cadenas de texto

1. Introducción
2. Operaciones básicas
3. Acceso a caracteres y *slicing*
4. Métodos comunes
5. Formateo
6. División y unión
7. Iteración
8. Inmutabilidad
9. Caracteres especiales
10. Conversión

7. Cadenas de texto

1. Introducción

Secuencia de caracteres (letras, números, símbolos) encerrados entre comillas:

- simples (')
- dobles ("), o
- triples (' ' ' o " " ")

Uno de los tipos de datos más usados en Python



7. Cadenas de texto

1. Introducción

Comillas simples o dobles: no hay diferencia. Úsalas como prefieras... **¡pero sé consistente!**

Comillas triples: varias líneas. Recuerda: las podemos usar a modo de comentarios

```
cadena1 = 'Hola'
cadena2 = "Mundo"
cadena3 = '''Ésta es
una cadena
multilínea.'''
```


7. Cadenas de texto

1. Introducción

Comillas dentro de comillas:

- simples dentro de dobles
- dobles dentro de simples
- "escapadas"

```
frase = "Me dijo 'Hola'"
print(frase)
```

```
frase = 'Ella dijo "Adiós"'
print(frase)
```

```
frase = "Me dijo \"Hola\""
print(frase)
```

7. Cadenas de texto

1. Introducción

- Cadenas vacías
- No es lo mismo que None

```
cadena_vacia = ""  
print(cadena_vacia)  
# No se imprime nada  
  
if cadena_vacia is None:  
    print("Cadena vacía")
```

7. Cadenas de texto

2. Operaciones básicas

Concatenación: permite unir dos o más cadenas en una sola

Se realiza con el operador +

```
saludo = "Hola" + ", " + "mundo"  
print(saludo)  
# Salida: Hola, mundo
```

7. Cadenas de texto

2. Operaciones básicas

Repetición: un número específico de veces

Se realiza con el operador *

```
emocion = "Python! " * 3  
print(emocion)  
# Salida: Python! Python! Python!
```

7. Cadenas de texto

2. Operaciones básicas

Longitud: con la función `len ()`

Se tienen en cuenta todos los caracteres, incluso los no visibles

```
cadena = "Deep Learning"  
print(len(cadena))  
# Salida: 13
```

7. Cadenas de texto

3. Acceso a caracteres y slicing

- ¡Muy importante! No sólo para las cadenas, también para las listas, como veremos más adelante
- Acceso a caracteres específicos de una cadena usando su índice
- Comienza en 0 para el primer carácter
- Utiliza índices negativos para contar desde el final de la cadena

```
palabra = "Python"
print(palabra[0])
# Salida: 'P' (primer carácter)
print(palabra[1])
# Salida: 'y' (primer carácter)
print(palabra[-1])
# Salida: 'n' (último carácter)
print(palabra[-2])
# Salida: 'o' (penúltimo carácter)
```

7. Cadenas de texto

3. Acceso a caracteres y slicing

- El slicing permite obtener una subcadena especificando un rango de índices. La sintaxis es `cadena[inicio:fin:paso]`, donde:
- `inicio`: el índice donde comienza el slicing (incluido)
- `fin`: el índice donde termina (excluido)
- `paso`: el incremento entre cada índice (opcional)

```
palabra = "Programación"
```

```
print(palabra[0:6]) # 'Progra'
```

```
print(palabra[6:]) # 'mación'
```

```
print(palabra[:6]) # 'Progra'
```

```
print(palabra[::2]) # 'Pormcón'
```

```
print(palabra[::-1]) # 'nóicamargorP'
```

7. Cadenas de texto

4. Métodos comunes

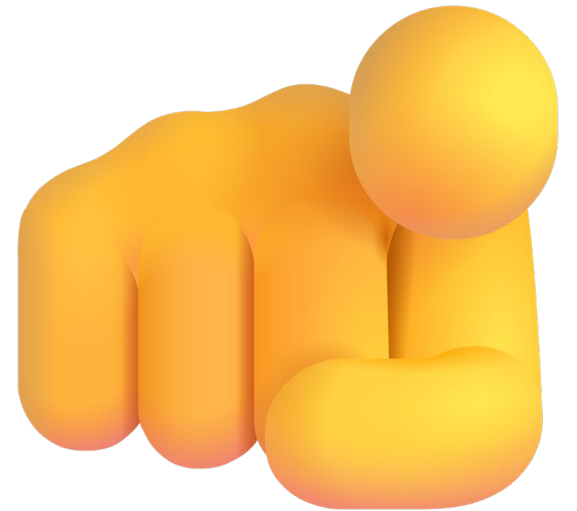
- `upper(cadena)`
- `lower(cadena)`
- `capitalize(cadena)`
- `title(cadena)`
- `strip(cadena)`
- `lstrip(cadena)`
- `rstrip(cadena)`
- `find(subcadena)`
- `index(subcadena)`
- `replace(viejo, nuevo)`
- `startswith(subcadena)`
- `endswith(subcadena)`
- `isalnum()`
- `isdigit()`
- `isalpha()`

7. Cadenas de texto

4. Métodos comunes

Escribe un programa que reciba una cadena del usuario y aplique los siguientes métodos:

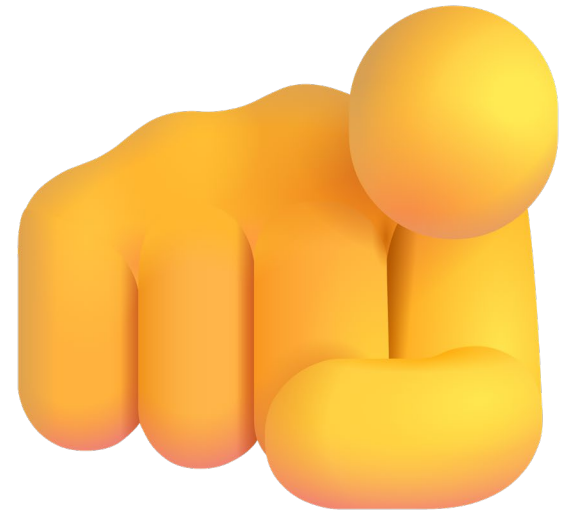
- convierta la cadena a mayúsculas
- convierta la cadena a minúsculas
- capitalice la primera letra de la cadena
- aplique el formato de título a la cadena



7. Cadenas de texto

4. Métodos comunes

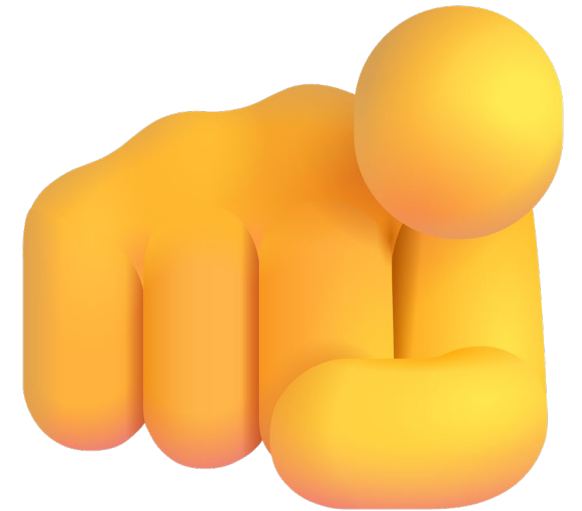
Crea una cadena con espacios en blanco al principio y al final. Luego, utiliza los métodos `strip()`, `lstrip()`, y `rstrip()` para eliminar los espacios y muestra el resultado



7. Cadenas de texto

4. Métodos comunes

Escribe un programa que busque la palabra "Java" dentro de una frase usando `find()` y `index()`. Si la encuentra, reemplázala por "Python" usando `replace()`

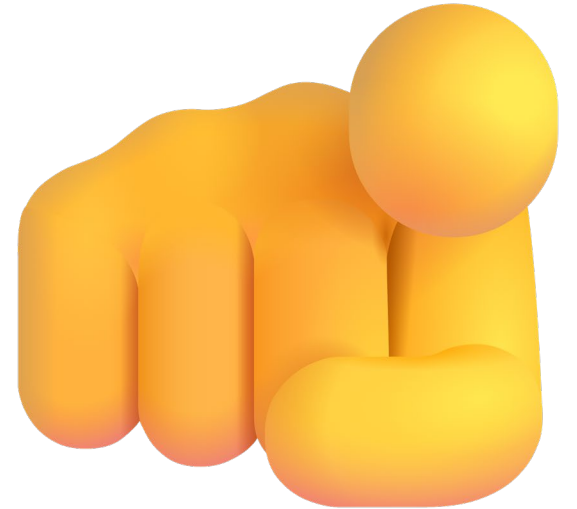


7. Cadenas de texto

4. Métodos comunes

Escribe un programa que solicite una cadena y verifique si:

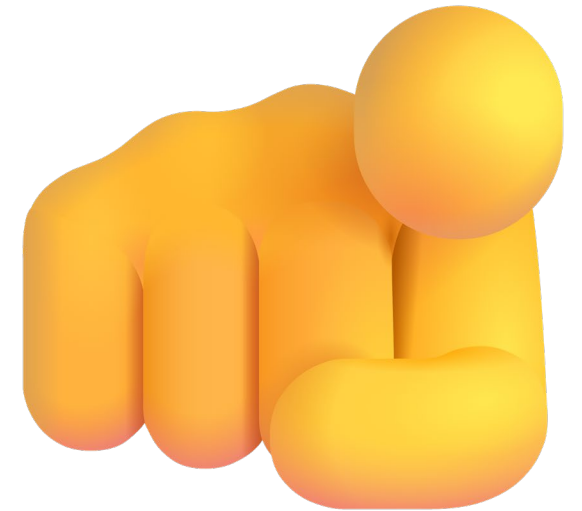
- la cadena comienza con "Hola" usando `startswith()`
- la cadena termina con "!" usando `endswith()`
- la cadena contiene solo letras (`isalpha()`), solo dígitos (`isdigit()`) o es alfanumérica (`isalnum()`)



7. Cadenas de texto

4. Métodos comunes

Crea un programa que reciba una frase y una palabra del usuario. Luego, busca cuántas veces aparece la palabra en la frase y su primera posición utilizando `find()` y `count()`



7. Cadenas de texto

5. Formateo

Concatenación simple

Utilizando los operadores que ya conocemos

Funciona, pero es tedioso:
cerrar comillas, abrirlas,
transformar los datos... Debe
existir una forma mejor

```
nombre = "Andy"
edad = 50
mensaje = "Me llamo " +
nombre + " y tengo " +
str(edad) + " años."
print(mensaje)
# Salida: Me llamo Andy y
tengo 50 años.
```

7. Cadenas de texto

5. Formateo

Formateo (f-strings, interpolación)

Manera moderna y recomendada de formatear cadenas en Python

Permiten insertar variables y expresiones directamente dentro de una cadena precedida por una f
f"texto {expresion}"

Podemos realizar operaciones en la expresión

```
nombre = "Andy"
edad = 50
mensaje = f"Me llamo {nombre}
y tengo {edad} años."
print(mensaje)
# Salida: Me llamo Andy y
tengo 50 años.
```

7. Cadenas de texto

5. Formateo

f-strings, interpolación

Podemos realizar operaciones
en la expresión

Podemos alterar el formato

```
valor = 5
mensaje = f"La mitad de
{valor} es {valor /
2:.2f}"
print(mensaje)
# Salida: La mitad de 5
es 2.50
```


7. Cadenas de texto

5. Formateo

Alternativas

Usando % (ya no soportado)

Usando `.format()`

Recomendación: f-strings



7. Cadenas de texto

6. División y unión

`split(separador, veces)`

- separador: lo que se busca
- veces: cuántas veces se hará la división

```
frase = "Python es increíble"
palabras = frase.split()
print(palabras)
# Salida: ['Python', 'es', 'increíble']
```

```
fecha = "2024-10-14"
partes = fecha.split("-")
print(partes)
# Salida: ['2024', '10', '14']
```

```
texto = "uno,dos,tres,cuatro"
partes = texto.split(",", 2)
print(partes)
# Salida: ['uno', 'dos', 'tres,cuatro']
```

7. Cadenas de texto

6. División y unión

`join(lista)`

Une la lista de cadenas usando el valor proporcionado como separador

```
palabras = ['Python', 'es',  
'increíble']
```

```
frase = " ".join(palabras)
```

```
print(frase)
```

```
# Salida: 'Python es  
increíble'
```

```
partes = ['2024', '10', '14']
```

```
fecha = "-".join(partes)
```

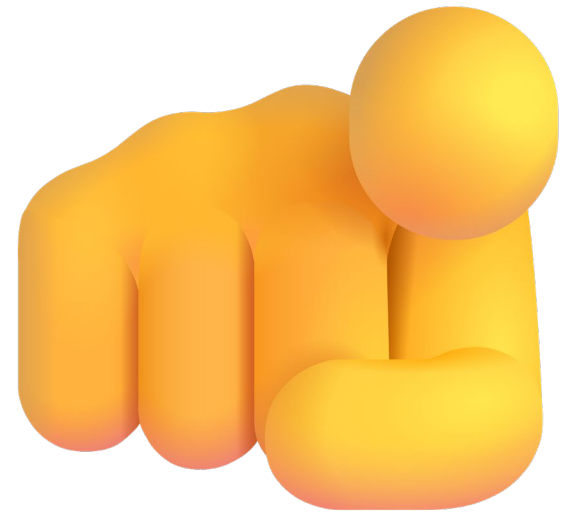
```
print(fecha)
```

```
# Salida: '2024-10-14'
```

7. Cadenas de texto

7. Iteración

- Utilizando for
- Con índices
- Sin índices



7. Cadenas de texto

8. Inmutabilidad

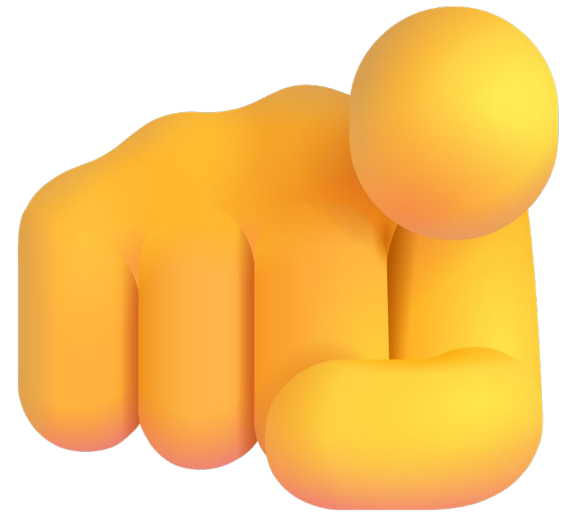
En Python, las **cadenas de texto son inmutables**. Esto significa que, una vez creada, una cadena **no se puede modificar directamente**. Cada vez que intentas cambiar el contenido de una cadena, en realidad estás creando una nueva cadena en lugar de modificar la existente



7. Cadenas de texto

8. Inmutabilidad

¡Comprobadlo!



7. Cadenas de texto

9. Caracteres especiales

Secuencias de escape que comienzan con una barra invertida (\)

- \n: salto de línea
- \t: tabulación
- \\: barra invertida (\).
- \': comilla simple
- \": comilla doble

Cadenas con r: sin secuencias de escape

```
texto = "Primera línea\nSegunda  
línea\tcon tabulación\\ y una barra  
invertida"
```

```
print(texto)
```

```
ruta = r"C:\nueva\carpeta"
```

```
print(ruta)
```


7. Cadenas de texto

10. Conversión

Valores a cadena con `str()`

Cadenas a valores con `int()` o `float()`

```
edad = 50
print("Tengo " + str(edad)
+ " años.")
# Salida: 'Tengo 25 años.'
```

```
texto = "42"
numero = int(texto)
print(numero) # Salida: 42
print(numero + 8) #
Salida: 50
```


Conclusiones

- Python es un lenguaje de tipado dinámico
- La comprensión de las estructuras de control es clave
- Las cadenas de texto son inmutables
- Existen numerosas formas de manipular cadenas
- La conversión de tipos es fundamental



Recursos y referencias

- [Clean Code](#), de [Robert Martin](#)
- [Clean Code in Python](#)
- [Clean Code in Python: Good vs. Bad Practices Examples](#)
- [Python Operators](#)
- [Python's F-String for String Interpolation and Formatting](#)

2. Conceptos básicos (1/2)

Introducción a Python para Deep Learning

JUNTA DE EXTREMADURA
Consejería de Economía, Empleo y Transformación Digital

ETD
EXTREMADURA
Estrategia de Transformación
Digital de Extremadura


INTIA
INSTITUTO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

uex

