CURSOS polotic misiones

PROGRAMACIÓN WEB

CON PYTHON Y JAVASCRIPT

CLASE 3

INTRODUCCIÓN AL LENGUAJE PYTHON

CURSOS polotic misiones

python™

CLASE 3

INGRESO DE DATOS EN PYTHON POR CONSOLA

- Ahora, trabajaremos en la redacción de un programa más interesante que pueda recibir información del usuario y saludar a ese usuario.
- Para hacer esto, usaremos otra función incorporada llamada entrada que muestra un mensaje al usuario y devuelve lo que el usuario proporciona como entrada. Por ejemplo, podemos escribir lo siguiente en un archivo llamado miAplicacion.py:

```
nombre = input("Nombre: ")
print("Hola, " + nombre)
```

OPERACIONES CON CARACTERES

- Si bien podemos usar el operador `+` para combinar cadenas como hicimos anteriormente, en las últimas versiones de Python, hay formas aún más fáciles de trabajar con cadenas, conocidas como cadenas formateadas (formatted strings) o **f-strings** para abreviar.

- Para indicar que estamos usando cadenas formateadas, simplemente agregamos una `f` antes de las comillas.

  Por ejemplo, en lugar de usar `"Hola," + nombre` como hicimos anteriormente, podríamos escribir `f "Hola, {nombre}"` para obtener el mismo resultado. Incluso podemos conectar una función en esta cadena si queremos, y convertir nuestro programa anterior en una sola línea:

```
print(f"Hello, {input(`Name: `)}")
```

OTRA MANERA DE FORMATEAR DATOS DE SALIDA

CURSOS polotic misiones

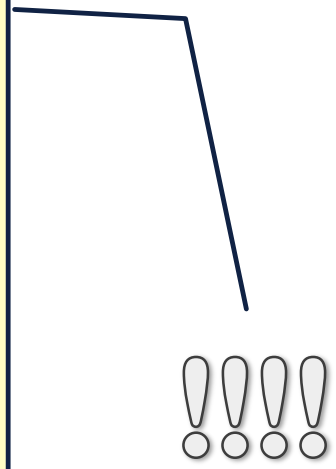
```
nombre = input("Nombre: ")
print(f"Hola, {nombre}")
```


Más info en https://realpython.com/python-f-strings/

CONDICIONES

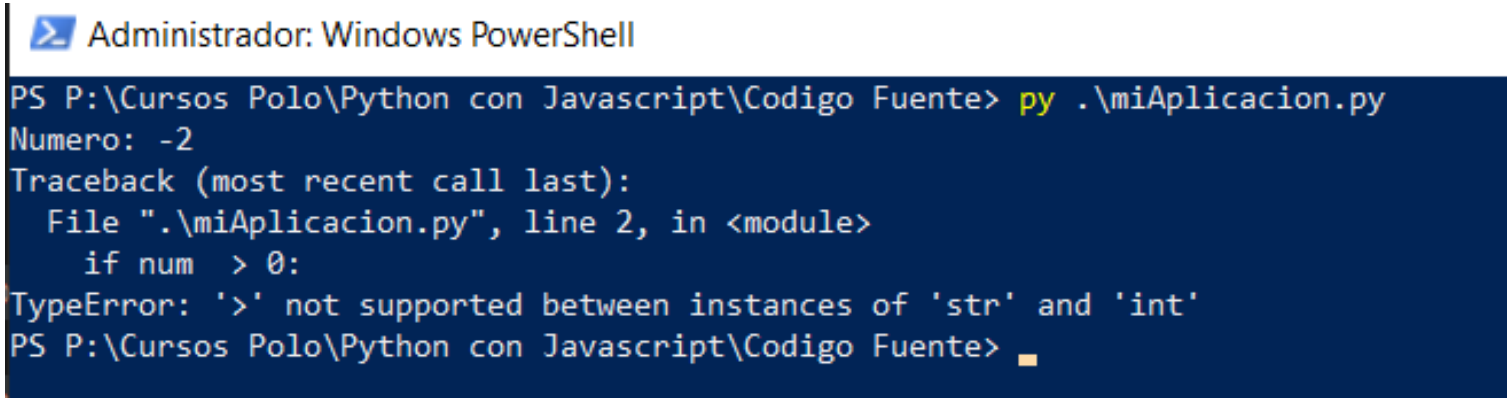
Al igual que en otros lenguajes de programación, Python nos brinda la capacidad de ejecutar diferentes segmentos de código en función de diferentes condiciones.
Por ejemplo, en el programa a continuación, cambiaremos nuestra salida según el número que ingrese un usuario:

```
num = input("Numero:")
if num > 0:
  print("Num es positivo")
elif num < 0:
  print("Num es negativo")
else:
  print("Num es 0")
```



EXCEPCIONES

- Una excepción es lo que sucede cuando ocurre un error mientras estamos ejecutando nuestro código Python y, con el tiempo, mejorarás cada vez más en la interpretación de estos errores, lo cual es una habilidad muy valiosa.
- Veamos un poco más de cerca esta excepción específica: si miramos la parte inferior, veremos que nos encontramos con un **`TypeError`**, lo que generalmente significa que Python *esperaba que una determinada variable fuera de un tipo*, pero descubrió que era de otro. tipo. En este caso, la excepción nos dice que no podemos usar el símbolo **`>`** para comparar **`str`** e **`int`**, y luego arriba podemos ver que esta comparación ocurre en la línea 2.
- En este caso, es obvio que -2 es un número entero negativo, por lo que debe ser el caso de que nuestra variable **`num`** sea un **`string`**. Esto sucede porque resulta que la función **`input`** siempre devuelve un **`string`**, y tenemos que especificar que se debe convertir (o transformar) en un número entero usando la función **`int`**.



TRANSFORMACION DE DATOS

- Esto significa que nuestra primera línea ahora se vería así:

```
num  = int(input("Numero: "))
```

SECUENCIAS DE DATOS

Una de las partes más poderosas del lenguaje Python es su capacidad para trabajar con secuencias de datos además de variables individuales.

- `list` – Lista, o secuencia de valores cambiantes
- `tuple` – Tupla, o secuencia de valores imutable
- `set` – Colección de valores unicos
- `dict` – Diccionario, o colección de pares de clave-valor

CADENAS

- Ordenable: Sí
- Mutable: No

- Ya hemos analizado un poco los strings, pero en lugar de solo variables, podemos pensar en un string como una secuencia de caracteres. ¡Esto significa que podemos acceder a elementos individuales dentro de la cadena! Por ejemplo:

```
nombre = "Harry"
print(nombre[0])
print(nombre[1])
```

- imprime el primer carácter (o índice-0) de la cadena, que en este caso resulta ser **H**, y luego imprime el segundo carácter (o índice-1), que es una **a**.

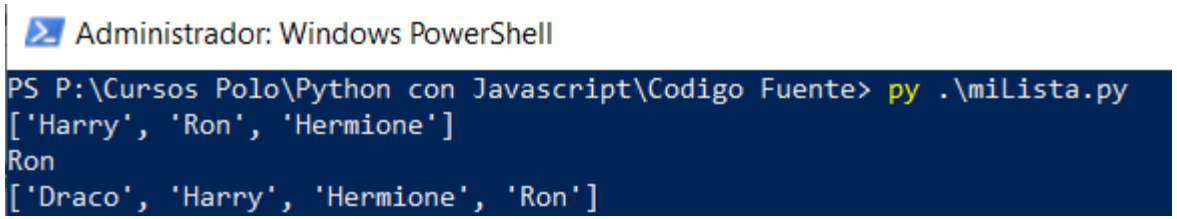
LISTAS

- Ordenable: Sí
- Mutable: Si

- Una lista de Python le permite almacenar cualquier tipo de variable. Creamos una lista usando corchetes y comas, como se muestra a continuación.
- De manera similar a los strings, podemos imprimir una lista completa o algunos elementos individuales. También podemos agregar elementos a una lista usando **append** y ordenar una lista usando **`sort`**

LISTAS

```
# Este es un comentario
nombres = ["Harry", "Ron", "Hermione"]
# Imprimir toda la lista:
print(nombres)
# Imprimir el segundo elemento de la lista:
print(nombres[1])
# Agregar un nombre a la lista:
nombres.append("Draco")
# Ordenar la lista:
nombres.sort()
# Imprimir la nueva lista:
print(nombres)
```



TUPLAS

- Ordenable: Sí
- Mutable: Si
- Las tuplas se utilizan generalmente cuando necesita almacenar solo dos o tres valores juntos, como los valores x e y para un punto en un plano.
- En el código Python, usamos paréntesis:

```
punto = (12.5, 10.6)
```

SETS

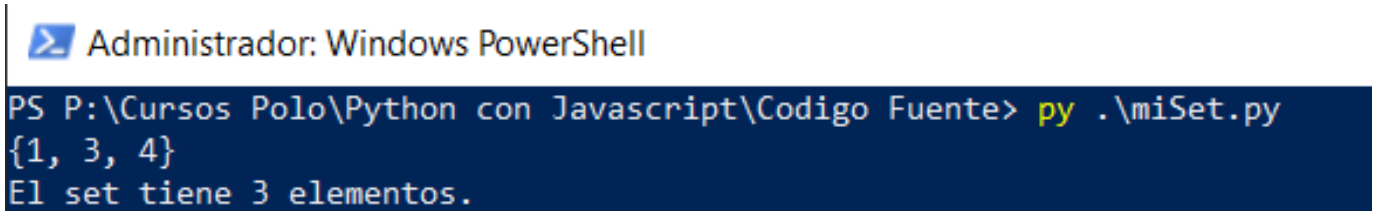
- Ordenable: No
- Mutable: No Aplica

- Los conjuntos o sets se diferencian de las listas y tuplas en que no están ordenados.
- También son diferentes porque, si bien puedes tener dos o más elementos iguales dentro de una lista / tupla, un set solo almacenará cada valor una vez.
- Podemos definir un set vacío usando la función `set`. Luego podemos usar agregar y quitar para agregar y quitar elementos de ese conjunto, y la función `len` para encontrar el tamaño del conjunto.
- Ten en cuenta que la función `len` funciona en todas las secuencias en Python. También ten en cuenta que a pesar de agregar 4 y 3 al conjunto dos veces, cada elemento solo puede aparecer una vez en un conjunto.

SETS (CONJUNTOS)

```
# Crear un conjunto (set) vacío:
conjunto = set()

# Agregar elementos:
conjunto.add(1)
conjunto.add(2)
conjunto.add(3)
conjunto.add(4)
conjunto.add(3)
conjunto.add(1)

# Remover 2 elementos del set
conjunto.remove(2)

# Imprimir el set s:
print(conjunto)
```

```
# Encontrar el tamaño del set
print(f"El set tiene {len(s)} elementos.")


""" Este es un comentario multilínea de Python:
La salida será:
{1, 3, 4}
Este set tiene 3 elementos.
"""
```



CLASE 3

DICCIONARIOS

- Ordenable: No
- Mutable: Si

- Los diccionarios o `dict` de Python, serán especialmente útiles en este curso.
- Un diccionario es un conjunto de pares clave-valor (key-value), donde cada clave tiene un valor correspondiente, al igual que en un diccionario, cada palabra (la clave) tiene una definición correspondiente (el valor).
- En Python, usamos llaves para contener un diccionario y dos puntos para indicar claves y valores. Por ejemplo:

```
# Definir un diccionario
casas = {"Harry": "Gryffindor", "Draco": "Slytherin"}
# Imprimir la casa de Harry
print(casas["Harry"])
# Agregar valores a un diccionario:
casas["Hermione"] = "Gryffindor"
# Imprimir la casa de Hermione:
print(casas["Hermione"])

""" Salida esperada:
Gryffindor
Gryffindor
"""
```

LOOPS (BUCLES)

- Los bucles son una parte increíblemente importante de cualquier lenguaje de programación, y en Python, vienen en dos formas principales: bucles `for` y bucles `while`. Por ahora, nos centraremos en los bucles `for`.
- Los bucles `for` se utilizan para iterar sobre una secuencia de elementos, realizando una tabulación o aumento del nivel de sangría en el bloque de código para cada elemento en una secuencia.
- Por ejemplo, el siguiente código imprimirá los números del 0 al 5:

```
for i in [0, 1, 2, 3, 4, 5]:
    print(i)
```

LOOPS (BUCLES)

- Podemos condensar este código usando la función de rango de Python, que nos permite obtener fácilmente una secuencia de números.
- El siguiente código da exactamente el mismo resultado que nuestro código anterior:

```
for i in range(6):
    print(i)
```

LOOPS (BUCLES)

- ¡Este tipo de bucle puede funcionar para cualquier secuencia! Por ejemplo, si deseamos imprimir cada nombre en una lista, podríamos escribir el siguiente código:

```
# Crear una lista
nombres = ["Harry", "Ron", "Hermione"]

# Imprimir cada nombre
for unNombre in nombres:
    print(unNombre)
```

LOOPS (BUCLES)

- ¡Podemos ser aún más específicos si queremos, y recorrer cada carácter en un solo nombre!

```
nombre = "Harry"
for caracter in nombre:
    print(caracter)
```

```
PS P:\Cursos Polo\Python con Javascript\Codigo Fuente> py .\misLoops.py
H
a
r
r
y
```

FUNCIONES

- Ya hemos visto algunas funciones de Python como `print` e `input`, pero ahora vamos a sumergirnos en escribir nuestras propias funciones.
- Para comenzar, escribiremos una función que toma un número y lo eleva al cuadrado:

```
def cuadrado(numero):
    return numero * numero
```

- Observe cómo usamos la palabra clave `def` para indicar que estamos definiendo una función, que estamos tomando una única entrada llamada `numero` y que usamos la palabra clave `return` para indicar cuál debería ser la salida de la función.
- Entonces podemos "llamar" a esta función tal como hemos llamado a otras: usando paréntesis:

```
for i in range(10):
    print(f“El cuadrado de {i} es {cuadrado(i)}")
```

MODULOS

- Si bien ya tengo definido mi archivo `misFunciones.py`, voy a poner parte del codigo en otro archivo llamado `miScript.py` de manera que ambos queden así:

```python
def cuadrado(numero):
    return numero * numero
```

misFunciones.py

```python
for i in range(10):
    print(f“El cuadrado de {i} es {cuadrado(i)}")
```

miScript.py

- Si intentamos ejecutar `miScript.py` nos saldrá el siguiente mensaje:

```
Traceback (most recent call last):
  File ".\miScript.py", line 2, in <module>
    print(f"El cuadrado de {i} es {cuadrado(i)}")
NameError: name 'cuadrado' is not defined
```

MODULOS

- Nos encontramos con este problema porque, de forma predeterminada, los archivos de Python no se conocen entre sí, por lo que tenemos que importar explícitamente la función `cuadrado` del módulo de funciones que acabamos de escribir en `miScript.py`

```
def cuadrado(numero):
    return numero * numero
```

misFunciones.py

```
from misFunciones import cuadrado

for i in range(10):
    print(f“El cuadrado de {i} es {cuadrado(i)}")
```

miScript.py

- Alternativamente, podemos optar por importar todo el módulo de funciones y luego usar la notación de puntos para acceder a la función cuadrada:

```
import misFunciones

for i in range(10):
    print(f"El cuadrado de {i} es {misFunciones.cuadrado(i)}")
```

MODULOS

- Hay muchos módulos de Python integrados que podemos importar, como
  - `math`
  - `csv`
- que nos dan acceso a aún más funciones. Además, podemos descargar incluso más módulos para acceder a más funciones. Dedicaremos mucho tiempo a usar el módulo Django, del que hablaremos en la próxima clase.

PROXIMAMENTE…

CURSOS polotic misiones

- Programación Orientada a Objetos con Python
- Programación Funcional
- Tratamiento de Excepciones

PROGRAMACIÓN WEB

CON PYTHON Y JAVASCRIPT

CURSOS polotic MISIONES

GRACIAS

programacionpolotic@gmail.com