



Universidad
Tecnológica
del Perú

**FACULTAD DE
INGENIERÍA**

**Domina
Python y
Conquista el
Mundo de los
Datos.**



PYTHON Y EL MUNDO DE LOS DATOS



SESIÓN 02

Introducción a Python

Logros de Aprendizaje

General

Desarrollar competencias en programación con Python, desde fundamentos básicos hasta el manejo avanzado de datos con NumPy y Pandas. Los participantes aprenderán a implementar estructuras, funciones, conceptos de POO (herencia y encapsulamiento) y librerías de análisis, integrando estos conocimientos en un caso práctico para resolver problemas reales y presentar resultados de manera efectiva.

Sesión

Al finalizar la sesión, el estudiante desarrolla programas básicos con Python haciendo uso de funciones y el paradigma de la programación orientada a objetos (POO).

PYTHON Y EL MUNDO DE LOS DATOS



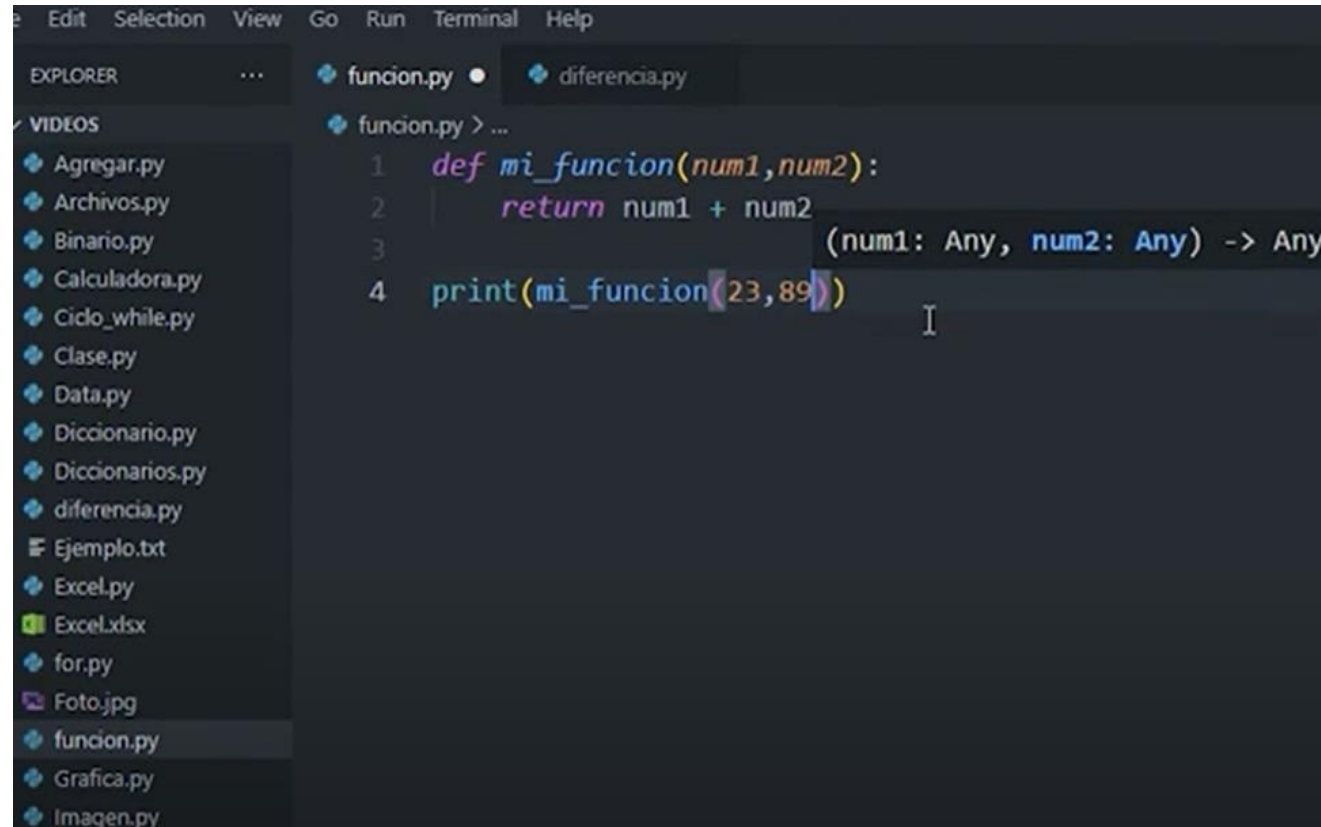
SESIÓN 02

Funciones en Python y POO.

- Definición de funciones: parámetros y valores de retorno.
- Introducción a POO: Clases, objetos, instancias y métodos básicos.

INICIO - Conocimientos Previos

¿Cómo se crea Funciones en Python?

A screenshot of a Python IDE (likely VS Code) with a dark theme. The Explorer sidebar on the left shows a list of files including 'Agregar.py', 'Archivos.py', 'Binario.py', 'Calculadora.py', 'Ciclo_while.py', 'Clase.py', 'Data.py', 'Diccionario.py', 'Diccionarios.py', 'diferencia.py', 'Ejemplo.txt', 'Excel.py', 'Excel.xlsx', 'for.py', 'Foto.jpg', 'funcion.py' (which is selected), 'Grafica.py', and 'Imagen.py'. The main editor window shows the code in 'funcion.py':

```
1 def mi_funcion(num1,num2):  
2     return num1 + num2  
3  
4 print(mi_funcion(23,89))
```

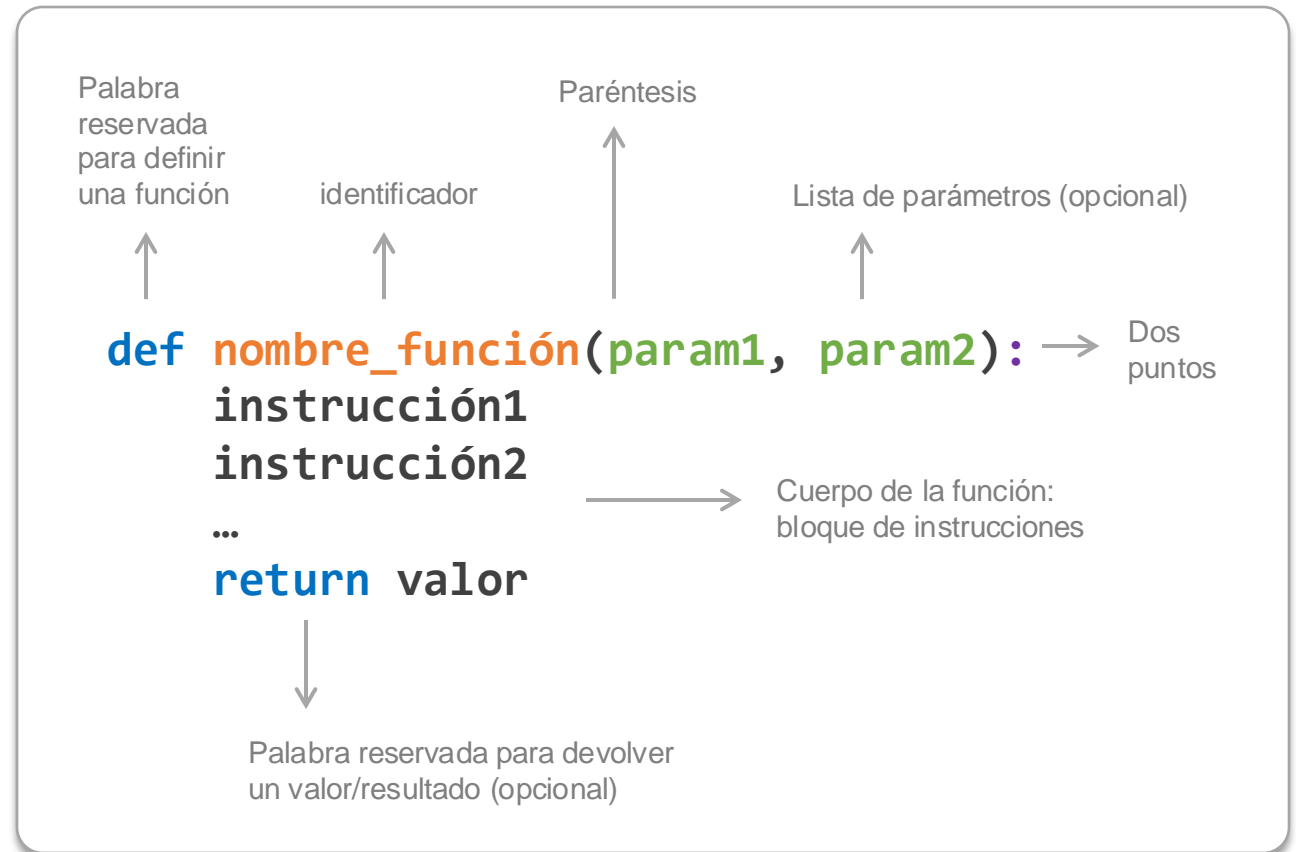
A tooltip is visible over the function call, showing the type signature: `(num1: Any, num2: Any) -> Any`. The cursor is positioned at the end of the fourth line of code.

<https://www.youtube.com/watch?v=1U7zmcrj-QM&t=1s>

¿Cuál es la utilidad de las Funciones usando Python?

Utilidad:

Las funciones en Python son herramientas poderosas que mejoran la organización, legibilidad y eficiencia de los programas, permitiendo escribir código más limpio, mantenible y reutilizable.

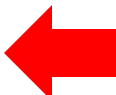


Definición de funciones: parámetros y valores de retorno.

- Escribir las mismas instrucciones varias veces puede resultar tedioso. Una solución es juntar las instrucciones en una función que se comporta como las funciones en matemáticas: dando argumentos de entrada se obtiene una respuesta o salida.



```
def nombre_función(parámetros):  
    instrucciones
```





4 espacios




Definición de funciones: parámetros y valores de retorno.

- Definimos una función **hola1** que, dado un argumento, **imprime** «Hola» seguido del argumento.

Por ejemplo, queremos que `hola1('Mateo')` imprima «Hola Mateo».



```
def hola1(nombre):  
    print('Hola', nombre)
```



Definición de funciones: parámetros y valores de retorno.

Nombre de la función

Parámetros

Código de la función

```
def suma(a, b):  
    resultado = a + b  
    return resultado
```


Valor de retorno

```
print(suma(5, 6))
```

Invocación

Definición de funciones: parámetros y valores de retorno.

- Una función de Python está formada por **líneas de texto**.
- Cada línea debe contener una única instrucción, aunque puede haber varias instrucciones en una línea, separadas por “;”.
- Por motivos de legibilidad, se recomienda que las líneas no superen los **79 caracteres**.
- Si una instrucción supera esa longitud, se puede dividir en varias líneas usando el caracter **contrabarra** “\”.





```
1 def procesa_pares_lista(L):
2     suma = 0; conteo = 0
3     for num in L:
4         if (num % 2 == 0):
5             suma += num; conteo += 1
6     print("Suma de los valores pares es: ", \
7         suma)
```

Definición de funciones: parámetros y valores de retorno.

- Definimos una función **hola2** que pregunte su nombre al usuario de e imprima «Encantado de conocerte» seguido de su nombre.
- En cálculo numérico **no solemos abusar de esta vía** para solicitar los datos de forma interactiva. De esta forma, si no se indica lo contrario, los parámetros serán introducidos directamente como argumentos de la función **input**


```
def hola2():  
    """  
    Ejemplo de función sin argumento.  
    Imprime el dato ingresado.  
    """  
    print('Hola, soy la compu')  
    nombre = input('¿Cuál es tu nombre? ')  
    print('Encantada de conocerte,', nombre)
```



```
In [10]: hola2()  
Hola, soy la compu  
  
¿Cuál es tu nombre?  
Julio  
Encantada de conocerte, Julio
```

Definición de funciones: parámetros y valores de retorno.

```
def hola2():  
    """  
    Ejemplo de función sin argumento.  
    Imprime el dato ingresado.  
    """  
    print('Hola, soy la compu')  
    nombre = input('¿Cuál es tu nombre? ')  
    print('Encantada de conocerte,', nombre)
```



Uso de la función `help()` para obtener el *docstring* definido en el cuerpo de la función.

```
In [11]: help(hola2)  
Help on function hola2 in module __main__:  
  
hola2()  
    Ejemplo de función sin argumento.  
    Imprime el dato ingresado.
```

Definición de funciones: parámetros y valores de retorno.

- Definimos una función **sumar2** que, dados dos argumentos, **devuelva** su suma.
- Observa el uso de la instrucción **return**.

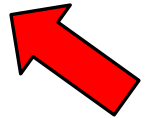
```
def sumar2(a, b):  
    """Suma los argumentos."""  
    return a + b # resultado de la función
```



A diferencia de **print**, la orden **return** permite asignar el valor resultante a un nuevo objeto

```
In [2]: a = sumar2(1,2)
```

```
In [3]: a  
Out[3]: 3
```



Definición de funciones: parámetros y valores de retorno.

- Obviamente, a fin de obtener un resultado satisfactorio, los objetos introducidos como argumentos deben poder “sumarse”. Es decir, las operaciones que se realicen con los objetos deben tener sentido.
- Por ejemplo, no podemos sumar un número y una cadena de caracteres.
- Observa cómo se suman dos cadenas de caracteres.

```
In [2]: sumar2("a",2)
```

```
Traceback (most recent call last):
```

```
File "<ipython-input-2-be30d13a4a78>", line 1, in <module>  
    sumar2("a",2)
```

```
File "<ipython-input-1-dc5ab59ec9d5>", line 3, in sumar2  
    return a + b # resultado de la función
```

```
TypeError: can only concatenate str (not "int") to str
```

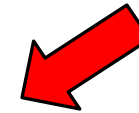
```
In [3]:
```

```
In [3]: sumar2("a","b")
```

```
Out[3]: 'ab'
```

```
In [4]: sumar2(1,2)
```

```
Out[4]: 3
```



Definición de funciones: parámetros y valores de retorno.

- Las funciones también son **objetos**, y cuando definimos una función se fabrica un objeto de tipo ***function*** (función), con su propio contexto, y se construye una variable que tiene por identificador el de la función y hace referencia a ella.

```
In [1]: def f(x):  
...:     """Multiplicar por 2."""  
...:     return 2*x
```

```
In [2]: def g(x):  
...:     """Multiplicar por 4."""  
...:     return f(f(x))
```

```
In [3]: g(3)  
Out[3]: 12
```

```
In [4]: type(g)  
Out[4]: function
```

Funciones Matemáticas en Python

- Las funciones se pueden escribir de muchas formas distintas. De hecho, la programación es un ejercicio muy creativo.

```
def f(x):  
    resultado = 2*x  
    return resultado
```

```
def f(x):  
    return 2*x
```

```
f = lambda x: 2*x
```


Introducción a POO:

Clases, objetos, instancias y métodos básicos.

- Los lenguajes de programación orientados a Objetos agrupan objetos de distintas clases que interactúan entre sí, y que, en conjunto, consiguen que un programa cumpla su propósito.
- Este paradigma de programación intenta emular el funcionamiento de los objetos del mundo real.
- En Python, cualquier elemento del lenguaje pertenece a una clase, todas las clases tienen el mismo rango y se utilizan del mismo modo.
- Es importante tener en cuenta que, además de las clases de objetos que hemos visto (int, floats, tuples, lists, o functions), Python permite definir nuevas clases con nuevas funcionalidades.

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

```
In [1]: lenguaje = "Python"
...: type(lenguaje)
Out[1]: str
```

```
In [2]: valor = 2.25
...: type(valor)
Out[2]: float
```

```
In [3]: import os
...: type(os)
Out[3]: module
```

```
In [4]: lista = [1, 2, 3, 4, 5]
...: type(lista)
Out[4]: list
```

- Una clase es una especie de plantilla que se utiliza para crear instancias individuales del mismo tipo de objeto.
- Las **clases** permiten definir los atributos y el comportamiento de los objetos de un programa.
- Los **atributos** definen las características propias del objeto y modifican su estado.
- Los **métodos** definen el comportamiento de la clase, es decir, lo que la clase puede hacer.

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

Definiendo clases:

En Python definimos una clase haciendo uso de la palabra reservada **class**, seguido del identificador de la clase.

Ejemplo:

```
# Creando una clase vacía  
class Perro:  
    pass
```

Se trata de una clase vacía y sin mucha utilidad práctica, pero es la mínima clase que podemos crear. Nótese el uso del “*pass*” que no hace realmente nada, pero daría un error si después de los “:” no tenemos contenido.

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

Creando Clases en Python:

- Ahora que tenemos la **clase**, podemos crear un **objeto** de la misma.
- Podemos hacerlo como si de una variable normal se tratase: nombre de la variable igual a la clase, seguido de “()”.
- Dentro de los paréntesis irían los parámetros de entrada, si los hubiera.

```
# Creamos un objeto de la clase Perro  
mi_perro = Perro()
```

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

Definiendo atributos

Existen dos tipos de atributos:

- Atributos de **instancia**: pertenecen a la instancia de la clase o al objeto. Son atributos particulares de cada instancia.
- Atributos de **clase**: se trata de atributos que pertenecen a la clase, por lo tanto, serán compartidos por todos los objetos.

Empecemos creando un par de **atributos de instancia** para nuestra clase Perro: el “nombre” y la “raza”. Para ello creamos el método “`__init__`” (constructor) que será llamado automáticamente cuando creamos un objeto.

```
class Perro:
    # El método __init__ es llamado al crear el objeto
    def __init__(self, nombre, raza):
        print(f"Creando perro {nombre}, {raza}")

        # Atributos de instancia
        self.nombre = nombre
        self.raza = raza
```

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

- Podemos ahora crear el objeto pasando el valor de los atributos. Usando “type ()” podemos ver como efectivamente el objeto es de la clase “Perro”.
- El uso de “__init__” y el doble “__” no es una coincidencia. Cuando veas un método con esa forma, significa que está reservado para un uso especial del lenguaje.

```
mi_perro = Perro("Toby", "Bulldog")
print(type(mi_perro))
# Creando perro Toby, Bulldog
# <class '__main__.Perro'>
```

```
print(mi_perro.nombre) # Toby
print(mi_perro.raza)   # Bulldog
```

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

- Hasta ahora hemos definido atributos de instancia, ya que son atributos que pertenecen a cada perro concreto.
- Ahora vamos a definir un **atributo de clase**, que será común para todos los perros.
- Por ejemplo, la especie de los perros es algo común para todos los objetos Perro.

```
class Perro:
    # Atributo de clase
    especie = 'mamífero'

    # El método __init__ es llamado al crear el objeto
    def __init__(self, nombre, raza):
        print(f"Creando perro {nombre}, {raza}")

    # Atributos de instancia
    self.nombre = nombre
    self.raza = raza
```


Introducción a POO:

Clases, objetos, instancias y métodos básicos.

- Dado que es un atributo de clase, no es necesario crear un objeto para acceder a los atributos. Podemos hacer lo siguiente:
- Se puede acceder también al atributo de clase desde el objeto.
- De esta manera, todos los objetos que se creen de la clase perro compartirán ese atributo de clase, ya que pertenecen a la misma.

```
print(Perro.especie)  
# mamífero
```

```
mi_perro = Perro("Toby", "Bulldog")  
mi_perro.especie  
# 'mamífero'
```

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

Definiendo métodos

- A continuación, vamos a ver como definir métodos que le den alguna funcionalidad a nuestra clase, siguiendo con el ejemplo de Perro.
- Vamos a codificar dos métodos **ladra()** y **camina()**. El primero no recibirá ningún parámetro y el segundo recibirá el número de pasos que queremos andar.
- Seguramente te hayas fijado en el “self” que se pasa como parámetro de entrada del método. Es una variable que representa la instancia de la clase, y deberá estar siempre ahí.

```
class Perro:
    # Atributo de clase
    especie = 'mamífero'

    # El método __init__ es llamado al crear el objeto
    def __init__(self, nombre, raza):
        print(f"Creando perro {nombre}, {raza}")

        # Atributos de instancia
        self.nombre = nombre
        self.raza = raza

    def ladra(self):
        print("Guau")

    def camina(self, pasos):
        print(f"Caminando {pasos} pasos")
```

Introducción a POO:

Clases, objetos, instancias y métodos básicos.

- Por lo tanto, si creamos un objeto “mi_perro”, podremos hacer uso de sus métodos llamándolos con “.” y el nombre del método. Como si de una función se tratase, pueden recibir y devolver argumentos.

```
mi_perro = Perro("Toby", "Bulldog")
mi_perro.ladra()
mi_perro.camina(10)

# Creando perro Toby, Bulldog
# Guau
# Caminando 10 pasos
```

PRÁCTICA – Sesión 2

- **Ejercicio 01:** Escribir una función que devuelva los números pares de una lista.

[3, 15, 4, 16, 8, 21, 9, 44, 23, 28, 16]



[4, 16, 8, 44, 28, 16]

PRÁCTICA – Sesión 2

- Solución:

`def` permite definir un método que puede tener cero, uno o varios parámetros

recorremos la lista con un **for** de colección y verificamos cada valor. Si el valor es par, se agrega a la lista **pares**. Finalmente, retornamos la lista con los números pares encontrados.

```
def obtener_pares(lista):  
    """  
    Devuelve una lista con los números pares obtenidos  
    a partir de la lista proporcionada  
    :param lista: lista de entrada  
    :return: lista conteniendo números pares  
    """  
    pares = []  
    for numero in lista:  
        if numero % 2 == 0:  
            pares.append(numero)  
    return pares
```



docstring para documentar el código desarrollado

```
lista_numeros = [3, 15, 4, 16, 8, 21, 9, 44, 23, 28, 16]  
print(obtener_pares(lista_numeros))
```

Salida: [4, 16, 8, 44, 28, 16]

PRÁCTICA – Sesión 2

- Solución optimizada usando “*list comprehension*”:

```
def obtener_pares_v2(lista):  
    """  
    Devuelve una lista con los números pares obtenidos  
    a partir de la lista proporcionada  
    :param lista: lista de entrada  
    :return: lista conteniendo números pares  
    """  
    return [numero for numero in lista if numero % 2 == 0]
```



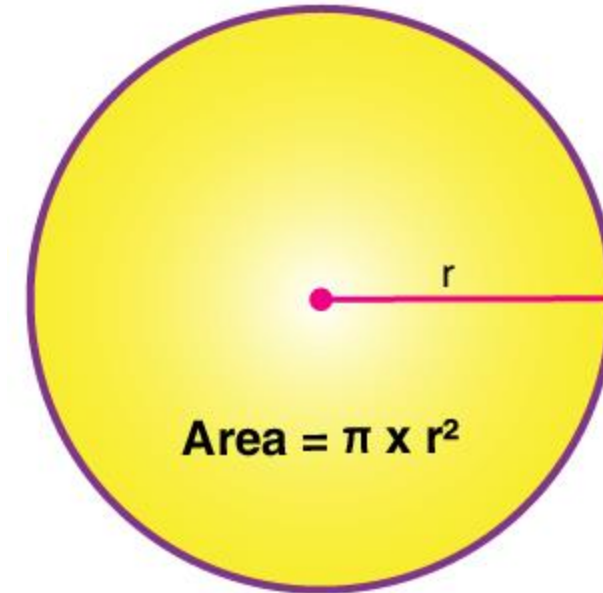
Una comprensión de lista (o *list comprehension*) es una forma compacta y eficiente de crear listas en Python. Permite construir una nueva lista aplicando una expresión a cada elemento de una secuencia (como una lista, rango, etc.), opcionalmente filtrando esos elementos mediante una condición.

```
lista_numeros = [3, 15, 4, 16, 8, 21, 9, 44, 23, 28, 16]  
print(obtener_pares_v2(lista_numeros))
```

Salida: [4, 16, 8, 44, 28, 16]

PRÁCTICA – Sesión 2

- **Ejercicio 02:** Crear una función que calcule el área de un círculo dado su radio.




PRÁCTICA – Sesión 2

- Solución versión 1: validando radio y retornando área o mensaje de error.

importamos la biblioteca
`math`

si el radio es positivo
devolvemos el área, si
no, un mensaje de error.



```
import math

def area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio
    :param radio: valor del radio
    :return: área del círculo
    """
    if radio > 0:
        return math.pi * pow(radio, 2)
    else:
        return "radio no puede ser cero o negativo"
```

```
print(area_circulo(12))
print(area_circulo(543.65))
print(area_circulo(-55))
```

```
452.3893421169302
928514.4298953619
radio no puede ser cero o negativo
```

PRÁCTICA – Sesión 2

- Solución versión 2: validando radio y retornando área o una excepción.

```
import math

def area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio
    :param radio: valor del radio
    :return: área del círculo
    """
    if radio > 0:
        return math.pi * pow(radio, 2)
    else:
        raise ValueError("radio no puede ser cero o negativo")
```



si el radio es positivo
devolvemos el área, si
no, una excepción

```
try:
    print(area_circulo_v2(18.3))
    print(area_circulo_v2(-24))
except ValueError as e:
    print(e)
```

Salida:

```
1052.0879637606859
radio no puede ser cero o negativo
```

PRÁCTICA – Sesión 2

- **Ejercicio 03:** Definir una clase Persona con atributos (nombre y edad) y un método para mostrar la información. Crear instancias de la clase y llamar al método.

Persona
nombre: string edad: int
<code>__init__(nombre, edad)</code> <code>print()</code> <code>__str__()</code>



PRÁCTICA – Sesión 2

definimos la clase con **class**

```
class Persona:  
    nombre = ""  
    edad = 0
```

constructor de la clase

```
def __init__(self, nombre, edad):  
    self.nombre = nombre  
    self.edad = edad
```

método para imprimir atributos

```
def print(self):  
    print(f"{self.nombre} ({self.edad})")
```

`__str__` que se usa para representar un objeto como una cadena de texto cuando se le aplica `str()` o cuando el objeto es impreso con `print()`

```
def __str__(self):  
    return f"{self.nombre} ({self.edad})"
```

```
persona1 = Persona("Andrés Rojas", 35)  
print(persona1.nombre)  
print(persona1.edad)  
  
persona2 = Persona("Marizta Vega", 28)  
print(persona2) # se invoca __str__ implícitamente
```



Un “*método mágico*” en Python es un método que inicia y termina con “`__`”. Son definidos como clases preconstruidas y usadas comúnmente para sobrecarga de operadores. Son llamados métodos “dunder”, que significa Double Under (*underscore*)

ESPACIO PRÁCTICO (Tarea) – Sesión 2

¡Ahora inténtalo tú!

Ejercicio propuesto:

- Crear una clase que administre la información de un dispositivo de almacenamiento, considerando los atributos: número de serie, una etiqueta, capacidad total y espacio ocupado.
- Incluir métodos para ocupar y desocupar espacio y un método que muestre el espacio disponible.



ESPACIO PRÁCTICO (Tarea) – Sesión 2

- Solución:

```
class DispositivoAlmacenamiento:
    etiqueta = ''
    numero_serie = ''
    capacidad_total = 0
    espacio_ocupado = 0

    def __init__(self, etiqueta, numero_serie, capacidad_total):
        self.etiqueta = etiqueta
        self.numero_serie = numero_serie
        self.capacidad_total = capacidad_total

    def asignar_espacio(self, total_bytes):
        if self.espacio_ocupado + total_bytes <= self.capacidad_total:
            self.espacio_ocupado += total_bytes
            return True

        return False
```



ESPACIO PRÁCTICO (Tarea) – Sesión 2

```
def liberar_espacio(self, total_bytes):  
    if self.espacio_ocupado > 0:  
        if total_bytes >= self.espacio_ocupado:  
            self.espacio_ocupado = 0  
        else:  
            self.espacio_ocupado -= total_bytes  
  
    return True  
else:  
    return False
```



```
disco1 = DispositivoAlmacenamiento("C:\\", "HD00987", 30000)  
disco1.asignar_espacio(2000)  
disco1.asignar_espacio(1500)  
disco1.liberar_espacio(500)  
disco1.liberar_espacio(250)  
  
print(f'Dispositivo "{disco1.etiqueta}", ' \\  
      f'capacidad: {disco1.capacidad_total} bytes')  
print('Espacio disponible: ' + \\  
      str(disco1.capacidad_total - disco1.espacio_ocupado) + " bytes")
```



puedes usar el carácter de barra invertida (\) al final de una línea para indicar que la sentencia continúa en la siguiente línea.

CIERRE - Sesión 02

Describe la
sintaxis de
una función
en Python

¿Cómo
creamos
una clase
en Python?

Describe
cómo
creamos y
usamos
objetos



Gracias por su atención