Clase 3.4

DuocUC®

Funciones

- Permiten modularizar el código
- Al ser definidas el interprete la procesa, sin embargo sólo se ejecuta cuando es llamada.
- Suelen tener inputs (parámetros de entrada) y output (resultado o salida), pero pueden haber funciones que carezcan de una o ambas partes.

```
# usamos def para definir la funcion
def mi_funcion(entrada1, entrada2 ...):
    .... codigo ....
    return <aqui va salida, puede ser una variable>
           <si no sale nada, usar None>
```

DuocUC

Ejemplo - I/O

```
def saludo(nombre):
    return f"Hola, {nombre}! Bienvenido"

# Uso de la funcion:
print(saludo("Juan"))
```

DuocUC

Ejemplo - Sin I/O

```
from datetime import datetime

def get_fecha_hora_actual():
    # Obten la fecha y hora actual
    ahora = datetime.now()
    # Formatea a dd/mm/yy, H:M:S
    fecha_hora = ahora.strftime("%d/%m/%Y, %H:%M:%S")
    return fecha_hora

# Uso de la funcion:

print(get_fecha_hora_actual())
```

Ejemplo - I/ Sin O

```
def saludo(nombre):
    print(f"Hola, {nombre}! Bienvenido")
    return None

# Uso de la funcion:

saludo("Carlos")
```

DuocUC®

Ejemplo - Sin I/ Sin O

```
def imprime_mensaje_bienvenida():
    print(" Bienvenido !")
    return None

# Uso de la funcion:

imprime_mensaje_bienvenida()
```

DuocUC

Ejercicios

- Escribir una función que, dada una lista de números enteros ingresada por usuario, identifica y muestra los números pares e impares de manera clara y organizada y muestra el resultado en pantalla.
- Crear una función que reciba como input una frase y cuente cuantas palabras tiene, mostrando esto último como resultado.
- Crear una función de producto punto: es decir que reciba como input dos vectores (tuplas) $\vec{x} = (x_1, \ldots, x_n)$ e $\vec{y} = (y_1, \ldots, y_n)$ y entregue el resultado del producto punto $\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i$

DuocUC®