introduccion.py clase04.py

```
Programming ‘Language’ = {
    Introducción = [Python, Micropython]

}
```

```
Clase 04 = {
    Presentación = [Les damos la bienvenida al curso]
}
```

Clases = {

- **Clase 01** — Breve historia de Python y su Filosofía. Principios de diseño de Python (PEP 20). Instalación y Configuración de Python y entornos de desarrollo (IDE).
- **Clase 02** — Sintaxis Básica y Estructuras de Control. Variables, tipos de datos y operadores. Estructuras de control (if, for, while).
- **Clase 03** — Estructuras de Datos. Listas, tuplas, diccionarios y conjuntos. Manipulación y métodos asociados.
- 👉 **Clase 04** — **Funciones y Módulos. Definición y uso de funciones. Importación y creación de módulos.**

Funciones {

```
# En Python, una función es un bloque de código que constituyen una unidad lógica dentro del programa. Resuelve un problema específico, y permiten la modularidad del código.

# Una función puede definir opcionalmente parámetros de entrada, que permiten pasar argumentos a la función en el momento de su llamada.

# Además, una función también puede devolver un valor como salida. Las funciones nos permiten dividir el trabajo que hace un programa en tareas más pequeñas, separadas del código principal. Ese es el concepto de función en programación.

# En Python, las funciones se definen utilizando la palabra clave def, seguida del nombre de la función, paréntesis () y dos puntos :. El cuerpo de la función se escribe con sangría.
```

}

Funciones integradas {

```
# Python trae integradas una serie de funciones listas para usar, algunas ya las conocemos.
```

Función	Significado	Ejemplo
print	Muestra información en la consola.	`print("Hola, mundo!“)`
input	Recibe la entrada del usuario desde la consola.	`input("Ingrese su nombre:")`
len	Devuelve el número de elementos de un objeto, como una cadena de texto.	`longitud=len("Python")`
type	Devuelve el tipo de un objeto.	`tipo_numero=type(42)`

}

Beneficios de usar funciones {

Encapsulamiento o modularidad. Se divide y organiza el código en partes más sencillas que se pueden encapsular en funciones. Esto permite además que se trabaje en equipo mas fácil.

Simplifica la lectura. El código estructurado en funciones tiene un cuerpo principal reducido y funciones bien delimitadas.

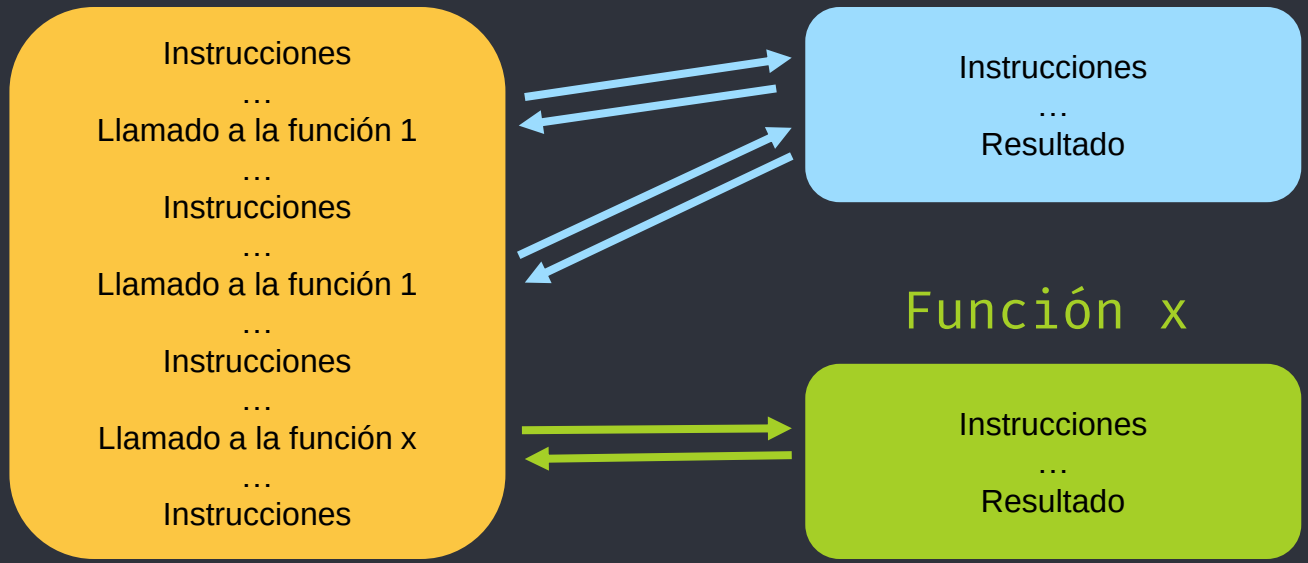
Reutilización. El código encapsulado en una función puede utilizarse en diferentes proyectos.

Mantenimiento. El software que utiliza funciones es más fácil de mantener

Abstracción. Ocultan los detalles de implementación y proporcionan una interfaz simple para realizar tareas complejas.

}

Funciones {


Programa principal
Instrucciones
…
Llamado a la función 1
…
Instrucciones
…
Llamado a la función 1
…
Instrucciones
…
Llamado a la función x
…
Instrucciones
Función 1
Instrucciones
…
Resultado
Función x
Instrucciones
…
Resultado

}

Definición {

```python
# Definición de la función
def suma(a, b):
    s = a + b
    print("Suma:", s)

# Programa principal
suma(5, 10) # Resultado: 15
```

}

Parámetros

Nombre

Cuerpo de la función

Invocación

Nombres {

```python
# Los nombres de las funciones siguen las mismas pautas vistas para
nombrar variables, aunque utilizando verbos en infinitivo.

        def saludar():
            print("Introducción a Python y MicroPython")
            print("UTN-FRT")

        saludar()
        # Introducción a Python y MicroPython
        # UTN-FRT

# La función se debe definir antes de ser invocada por primera vez. Desde
el programa principal se invoca a la función escribiendo su nombre.
```

}

Nombres {

```
    # Las reglas para el nombre de una función son las mismas que para la de una variable.

    # El nombre debe comenzar con una letra

    # Puede contener letras, números y guiones bajos

    # No puede llamarse igual que una palabra reservada del lenguaje

    # No debe contener caracteres especiales como letras acentuadas (á, é, í, ó,ú) ni caracteres especiales como la virgulilla (ñ).

    # No se puede utilizar dentro de una función una variable que tenga el mismo nombre que la función.
```

}

Ejemplo 1 {

Programa principal

```python
def dibujar_bandera():
    print(f"***********")
    print(f"*    o    *")
    print(f"***********")

dibujar_bandera()
print()
print()
dibujar_bandera()
```

Resultado

```
***********
*    o    *
***********


***********
*    o    *
***********
```

}

Parámetros y argumentos {

```
# Un parámetro es una variable que se utiliza en la definición de la función para representar un dato que la ésta espera recibir cuando es llamada.

# Los parámetros permiten que una función acepte valores externos y los utilice dentro de su bloque de código. Estos valores, que son proporcionados al llamar a la función, se denominan "argumentos".

# Los argumentos son obligatorios, si definimos n cantidad de parámetros debemos llamar la función con n cantidad de argumentos. Si no se pasa los parámetros con nombre, entonces Python los toma como posicionales.
```

}

Parámetros y argumentos {

```
    # Los parámetros son las variables que ponemos cuando se define una
    función. En la siguiente función tenemos un parámetro:

    # Los argumentos son los valores que se pasan a la función cuando ésta es
    invocada.

    def multiplicar_por_5(numero):
        print(numero * 5)

    multiplicar_por_5(8)
```

Definición

Parámetro

Invocación

Argumento

}

Ejemplo 2 {

```python
def saludar(nombre):
    print()
    print(f'Hola {nombre}')
    print('¿Cómo estás?')
    print()

# Programa principal
print()
nom = input("Ingrese su nombre: ")
saludar(nom)
```

}

Ejemplo 3 {

```python
def calcularCuadrado(base):
    print()
    print(f'El cuadrado de {base} es {base**2}')
    print()

# Programa principal
print()
num = int(input("Ingrese un número: "))
calcularCuadrado(num)
```

}

Argumentos múltiples {

```python
# En una función se pueden colocar uno o más argumentos, cuidando que en
la invocación, los parámetros se coloquen en el mismo orden.

def generarMail(nombre, apellido, anio):
    print()
    mail = f“Email: {nombre[0].lower()}{apellido.lower()}{anio}@gmail.com"
    print(mail)
    print()

print()
a = input("Ingrese su nombre: ")
b = input("Ingrese su apellido: ")
c = int(input("Ingrese su año de nacimiento: "))
generarMail(a, b, c)
```

}

Ejemplo 4 {

```python
def obtenerDescuento(precio, porcentaje):
    descuento = precio * porcentaje/100
    precioNuevo = precio - descuento
    print()
    print(f'Precio original: ${precio:.2f}')
    print(f'Descuento: $ {descuento:.2f}')
    print(f'Precio nuevo: $ {precioNuevo:.2f}')
    print()

print()
precio_original = int(input("Ingrese el valor original del producto: "))
descuento = int(input("Ingrese el descuento que quiere aplicar: "))
obtenerDescuento(precio_original, descuento)
```

}

Parámetros opcionales {

```
# En una función Python se pueden indicar una serie de parámetros
opcionales con el operador =. Son parámetros que, si no se incluyen al
invocar a la función, toman ese valor por defecto.

        def sumar(a = 0, b = 15):
            print("La suma de a + b es: ", a + b)

        sumar(2,6) # 8
        sumar(5) # 20
        sumar() # 15

# Importante: En una función se pueden especificar tantos parámetros
opcionales como se quiera. Sin embargo, una vez que se indica uno, todos
los parámetros a su derecha también deben ser opcionales.
```

}

Ejemplo 5 {

```python
# En este ejemplo se utilizan argumentos opcionales para calcular la raíz de un número:

def fnRaiz(num, raiz=2):
    print(num**(1/raiz))

fnRaiz(9)          # 3
fnRaiz(8)          # 2,8284
fnRaiz(8,3)        # 2

# La función posee dos argumentos. El segundo es opcional. Si no se incluye el parámetro correspondiente en la llamada, se asume que es 2, y se calcula la raíz cuadrada.
```

}

Llamada de parámetros {

```
# Al invocar una función con diferentes argumentos, los valores se asignan
a los parámetros en el mismo orden en que se indican, esto quiere decir
que son posicionales. Sin embargo, el orden se puede cambiar si llamamos a
la función indicando el nombre de los parámetros.

def potencia(base, exponente = 2):
    print(base**exponente)

# Programa principal
potencia(6,1)
potencia(exponente = 5, base = 2)
potencia(8)
potencia(base = 2)
```

}

Devolución de valores {

```
# La devolución de valores en una función se refiere al proceso mediante el cual una función regresa un resultado al lugar desde el cual fue llamada (por ejemplo: el programa principal). En Python, esto se logra mediante la declaración return dentro del cuerpo de la función.

# Cuando una función tiene un return, se ejecuta hasta que se alcanza esa instrucción, la ejecución de la función se detiene y el valor especificado en return se devuelve a la ubicación donde se llamó a la función. Si existe código después del return, no se ejecuta.

# Si no se coloca el return o no se especifica ningún valor en el mismo, la función devuelve None por defecto.
```

}

Ejemplo 6 {

```python
# La función multiplica dos valores numéricos.
def multiplicar(num1, num2):
    multip = num1*num2
    return multip

# Programa principal
resultado = multiplicar(10,3)
print("El resultado de la multiplicacion es:", resultado)
```

}

Devolución de valores {

```python
# La sentencia return es opcional, y puede devolver o no un valor. Es posible que aparezca más de una vez dentro de una misma función.

def cuadradoDePar(numero):
    if not numero % 2 == 0:
        return
    else:
        return numero ** 2

# Programa principal
print(cuadradoDePar(8)) # 64
print(cuadradoDePar(3)) # None, porque no es par
```

}

Devolución de valores {

```python
# La siguiente función muestra por pantalla si el número es par o no,
utilizando dos instrucciones return.

def esPar(numero):
    if numero % 2 == 0:
        return True
    else:
        return False

# Programa principal
print(esPar(2)) # True
print(esPar(5)) # False
```

}

Devolución de valores {

```python
# Es posible devolver más de un valor con una sentencia return.

def operacionesBasicas(num1, num2):
    suma = num1 + num2
    resta = num1 - num2
    multiplicacion = num1 * num2
    division = num1 / num2
    return suma, resta, multiplicacion, division

a, b, c, d = operacionesBasicas(10, 2)
print(f"Suma: {a}")
print(f"Resta: {b}")
print(f"Multiplicación: {c}")
print(f"División: {d}")
```

}

Función en función {

```python
# Es posible devolver más de un valor con una sentencia return.


def sumar(num1, num2):
    return num1 + num2
def restar(num1, num2):
    return num1 - num2
def multiplicar(num1, num2):
    return num1 * num2
def dividir(num1, num2):
    if num2 != 0:
        return num1 / num2
    else:
        return None
```

}

```python
def calcular(num1, num2, op):
    if op == 1:
        return sumar(num1, num2)
    elif op == 2:
        return restar(num1, num2)
    elif op == 3:
        return multiplicar(num1, num2)
    elif op == 4:
        return dividir(num1, num2)
    else:
        print("Opción incorrecta")
```


Módulos {

```
# En Python los módulos son archivos que contienen definiciones que se pueden importar en otros scripts para reutilizar sus funcionalidades.

# Un módulo es un archivo de Python cuyos objetos (funciones, clases, excepciones, etc.) pueden ser accedidos desde otro script. Constituye una muy buena herramienta para organizar el código en proyectos grandes o complejos, lo cual, nos brinda muchas ventajas.

# Organización del código: Los módulos permiten dividir el código en partes más pequeñas y manejables.

# Reutilización: Las funciones, clases y variables definidas en un módulo pueden ser reutilizadas en otros programas.

# Mantenimiento: Facilitan la actualización y el mantenimiento del código.
```

}

Ejemplo {

```python
# Supongamos que tenemos un archivo llamado mimodulo.py con el siguiente contenido:

        def saludar(nombre):
            return f"¡Hola, {nombre}!"

        def despedirse(nombre):
            return f"Adiós, {nombre}."


# Para poder utilizarlo en otro script, necesitamos importarlo. Hay distintas maneras de importar un modulo.
```

}

Ejemplo {

```python
# Importar todo el modulo

import mimodulo

print(mimodulo.saludar("Juan"))  # ¡Hola, Juan!
print(mimodulo.despedirse("Juan"))  # Adiós, Juan.

# Importar funciones especificas del modulo

from mimodulo import saludar

print(saludar("Juan"))  # ¡Hola, Juan!
print(despedirse("Juan"))  # ImportError
```

}

Ejemplo {

```python
# Importar todo el contenido

from mimodulo import *

print(saludar("Juan"))  # ¡Hola, Juan!
print(despedirse("Juan"))  # Adiós, Juan.

# Importar un módulo con un alias

import mimodulo as mm

print(mm.saludar("Juan"))  # ¡Hola, Juan!
print(mm.despedirse("Juan"))  # Adiós, Juan.
```

}

```
1
2
3  Aprender a programar
4  es aprender a pensar.
5
6
7
8
9
10
11 { Steve Jobs; }
12
13
14
```


{ Nos vemos en la proxima clase }