Introducción a la Programación Segura

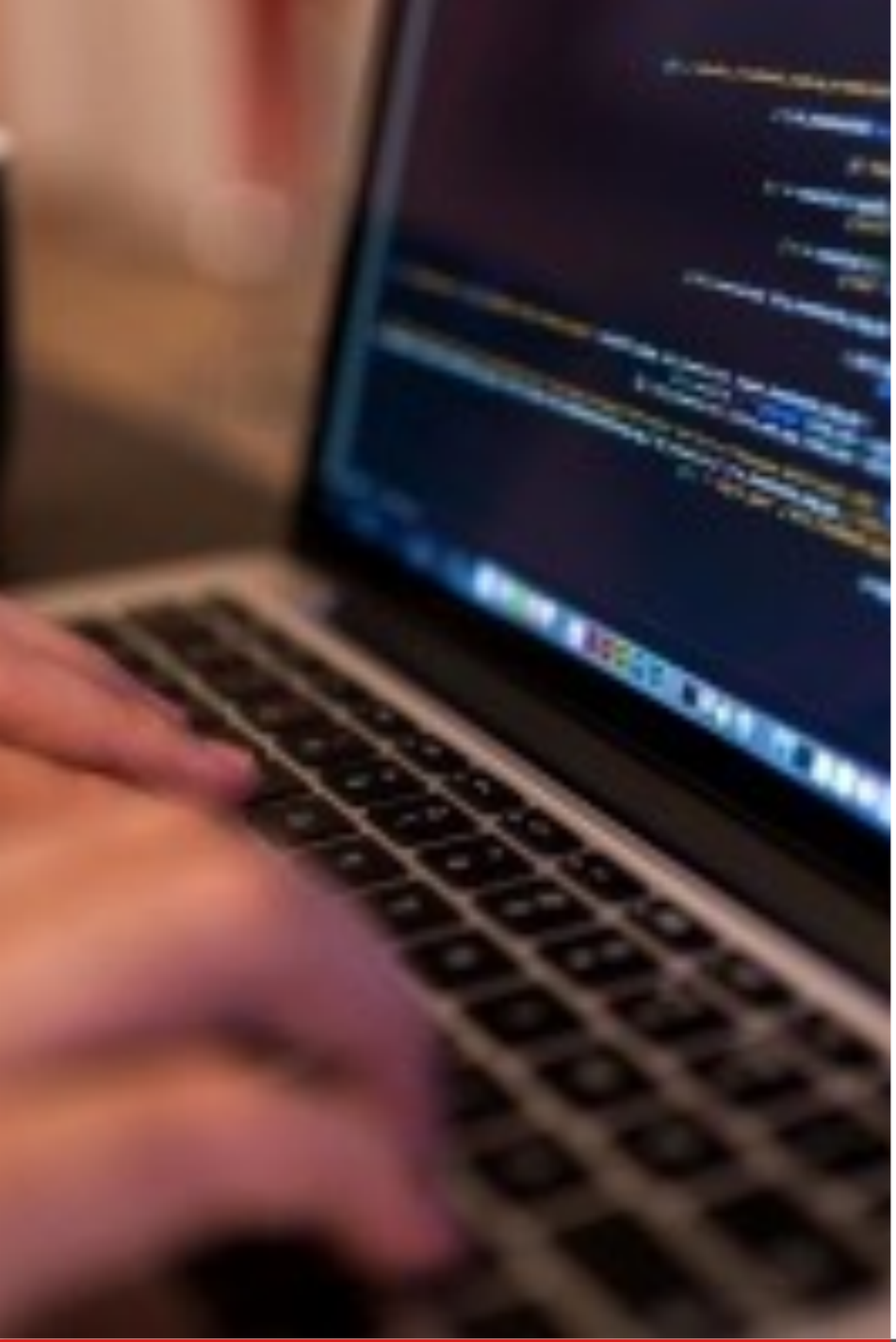
Estructuras de decisión, bucles y funciones del lenguaje Python.

INTRODUCCIÓN A LA PROGRAMACIÓN SEGURA

UNIDAD 2: ESTRUCTURAS DE DECISIÓN, BUCLES Y FUNCIONES DEL LENGUAJE PYTHON.

Contenidos

1. Operadores relacionales.
2. Condiciones y ejecución condicional.
3. Declaración if, if-else, elif.
4. Estructuras de control: Bucles y condicionales.
5. Operadores lógicos.
6. Funciones con retorno, envío de valores, parámetros y argumentos.



A practicar...

Ejercicios:

De acuerdo con el contenido visto en clases, desarrolle un script en lenguaje Python:

Escribe un programa en el que el usuario intente adivinar una contraseña predefinida. El programa debe hacer lo siguiente:

- Definir una contraseña predefinida en el código del programa.
- Solicitar al usuario que ingrese una contraseña.
- Comparar la contraseña ingresada por el usuario con la contraseña predefinida.
- Mostrar un mensaje indicando si la contraseña ingresada es correcta o incorrecta.
- El programa debe permitir al usuario intentar adivinar la contraseña varias veces.
- Después de cada intento, el programa debe mostrar un mensaje indicando si la contraseña es incorrecta y cuántos intentos le quedan al usuario.
- Si el usuario adivina la contraseña, el programa debe mostrar un mensaje de felicitaciones y terminar.

Diseño y escritura de funciones:

¿Qué es una función en Python?

En Python, una función es un bloque de código que realiza una tarea específica y se puede invocar desde cualquier parte del programa para realizar esa tarea. Las funciones permiten reutilizar el código, modularizar el programa y hacerlo más legible y mantenible.

Si no hubiésemos utilizado funciones, en este caso, para ingresar 3 nombres y apellidos y mostrar su respectivo saludo, el código dentro de la función tendría que estar escrito 3 veces.

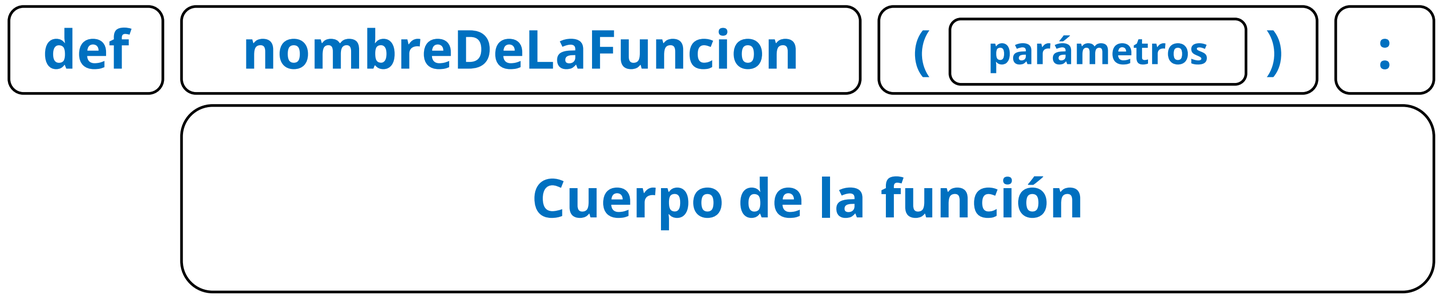
```python
def Saludar():
    print('-----------------------------------')
    nombre = input('Ingresa tu nombre: ')
    apellido = input('Ingresa tu apellido: ')
    print('¡Buenas tardes',nombre,apellido,'!')
    print('-----------------------------------')

Saludar()
Saludar()
Saludar()
```

```
-----------------------------------
Ingresa tu nombre: Teresa
Ingresa tu apellido: Araya
¡Buenas tardes Teresa Araya !
-----------------------------------
-----------------------------------
Ingresa tu nombre: Marcos
Ingresa tu apellido: Veliz
¡Buenas tardes Marcos Veliz !
-----------------------------------
-----------------------------------
Ingresa tu nombre: Franco
Ingresa tu apellido: Tapia
¡Buenas tardes Franco Tapia !
-----------------------------------
```

Diseño y escritura de funciones:

Declaración de una función:

En Python, una función se declara utilizando la palabra clave def, seguida del nombre de la función, paréntesis que pueden contener los argumentos (parámetros) de la función y dos puntos. Después, se escribe el cuerpo de la función, que contiene el código que se ejecutará cuando la función sea llamada.



Diseño y escritura de funciones:

Declaración de una función:

En Python, una función se declara utilizando la palabra clave def, seguida del nombre de la función, paréntesis que pueden contener los argumentos (parámetros) de la función y dos puntos. Después, se escribe el cuerpo de la función, que contiene el código que se ejecutará cuando la función sea llamada.

```
def Saludar():
    print('-----------------------------------')
    nombre = input('Ingresa tu nombre: ')
    apellido = input('Ingresa tu apellido: ')
    print('¡Buenas tardes',nombre,apellido,'!')
    print('-----------------------------------')
```

Diseño y escritura de funciones:

Llamada a una función:

Para invocar o llamar a una función, simplemente escribimos el nombre de la función seguido de paréntesis y los valores de los argumentos, si los tiene. Si la función devuelve un valor, podemos almacenarlo en una variable o usarlo directamente en nuestro programa.

```
def nombreDeLaFuncion ( parámetros ) :
    Cuerpo de la función
```

```
nombreDeLaFuncion()
```

Al llamar a la función esta ejecutara el código dentro del cuerpo de la función.

Diseño y escritura de funciones:

Llamada a una función:

Para invocar o llamar a una función, simplemente escribimos el nombre de la función seguido de paréntesis y los valores de los argumentos, si los tiene. Si la función devuelve un valor, podemos almacenarlo en una variable o usarlo directamente en nuestro programa.

```
Saludar()
Saludar()
Saludar()
```

```
----------------------------------
Ingresa tu nombre: Teresa
Ingresa tu apellido: Araya
¡Buenas tardes Teresa Araya !
----------------------------------
----------------------------------
Ingresa tu nombre: Marcos
Ingresa tu apellido: Veliz
¡Buenas tardes Marcos Veliz !
----------------------------------
----------------------------------
Ingresa tu nombre: Franco
Ingresa tu apellido: Tapia
¡Buenas tardes Franco Tapia !
----------------------------------
```

En este caso se llamó a la función 3 veces, por tanto, el cuerpo de la función se ejecutará 3 veces.

Las Funciones:

Funciones parametrizadas:

Las funciones parametrizadas en Python son funciones que aceptan uno o más parámetros (argumentos) como entrada. Estos parámetros son valores que se pasan a la función cuando es llamada y permiten que la función realice alguna tarea basada en esos valores. Las funciones parametrizadas son una forma de generalizar la lógica de la función y hacerla más flexible y reutilizable, ya que pueden procesar diferentes datos en diferentes llamadas.

Cuando definimos una función parametrizada, podemos declarar los nombres de los parámetros dentro de los paréntesis de la función. Estos nombres son como variables locales dentro de la función y se utilizan para recibir los valores proporcionados cuando la función es llamada

Las Funciones:

Funciones parametrizadas:

En este caso la función recibe como parámetros el nombre y el apellido y al recibirlos el cuerpo de la función hace uso de ellos.

La llamada a la función envía estos datos como argumento.

```python
# Declaración de la función
def Saludar(nombre,apellido):
    print('-----------------------------------')
    print('¡Buenas tardes',nombre,apellido,'!')
    print('-----------------------------------')

# Llamada a la función
nombre = input('Ingresa tu nombre: ')
apellido = input('Ingresa tu apellido: ')
Saludar(nombre,apellido)
```

```
Ingresa tu nombre: Alex
Ingresa tu apellido: Araos
-----------------------------------
¡Buenas tardes Alex Araos !
-----------------------------------
```

Parámetros y argumentos:
En la definición de una función, los valores que se reciben se denominan parámetros, y en la llamada se denominan argumentos.

Las Funciones:

Los Parámetros de una función:

Los parámetros de funciones son variables que se utilizan para recibir datos cuando la función es llamada. Estos parámetros permiten que una función sea más flexible y genérica, ya que pueden recibir diferentes valores cada vez que son invocados. Los parámetros actúan como placeholders (espacios reservados) para los valores que se pasan a la función.

La definición de parámetros en una función se realiza dentro de los paréntesis de la declaración de la función. Puedes definir cualquier cantidad de parámetros separándolos por comas. Cuando la función es llamada, los argumentos que se le pasan corresponden a los valores que se asignarán a los parámetros.

Las Funciones:

Los Parámetros de una función:

```python
# Declaración de la función
def AgregarProducto(nombre,marca,precio):
    # Preparamos una cadena con los parámetros recibidos
    producto = f"{nombre} ({marca}) ==> ${precio}"
    print(f"Se ha agregado: {producto} al inventario.")

# Llamamos a la función y pasamos los argumentos
AgregarProducto("Teclado","Microsoft",22500)

# Llamamos a la función y pasamos los argumentos
AgregarProducto("Mouse","HP",15900)

# Llamamos a la función y pasamos los argumentos
AgregarProducto("Monitor","Acer",185000)
```

En este caso no es necesario programar el ingreso 3 veces, sino una sola vez, pasando los datos de cada producto como argumentos en cada llamada.

```
Se ha agregado: Teclado (Microsoft) ==> $22500 al inventario.
Se ha agregado: Mouse (HP) ==> $15900 al inventario.
Se ha agregado: Monitor (Acer) ==> $185000 al inventario.
```

Las Funciones:

Argumentos posicionales:

Los argumentos posicionales son aquellos argumentos que se pasan a una función en el mismo orden en que se definen en la declaración de la función. Cuando definimos una función con parámetros, los valores que se pasan cuando se llama a la función se asignan a los parámetros en el mismo orden en que aparecen en la declaración de la función.

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, cantidad, precio, formaPago):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)

# Llamada a la función con argumentos posicionales
Pagar('Coca-Cola 3 lts.',2,2300,'Efectivo')
```

Si cambiamos el orden de los argumentos, la operación entregara un resultado no esperado o incluso errores ValueError.

Las Funciones:

Argumentos posicionales:

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, cantidad, precio, formaPago):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)
```

```python
# Llamada a la función con argumentos posicionales
Pagar('Coca-Cola 3 lts.',2,2300,'Efectivo')
```

```
Coca-Cola 3 lts. ....$ 2300 x 2 = $ 4600
forma de Pago: Efectivo
```

```python
# Llamada a la función con argumentos posicionales
Pagar('Coca-Cola 3 lts.','Efectivo',2,2300)
```

```
  line 5, in Pagar
    total = int(precio) * int(cantidad)
                          ^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'Efectivo'
```

Las Funciones:

Argumentos de palabra clave:
Python también admite los argumentos por palabra clave (keyword arguments) en los cuales podemos especificar los nombres de los parámetros al llamar a la función. Esto permite pasar los argumentos en cualquier orden siempre y cuando indiquemos los nombres de los parámetros al hacer la llamada.

```
# Declaramos la función con 4 parámetros
def Pagar(producto, cantidad, precio, formaPago):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)

# Llamada a la función con argumentos de palabra clave
Pagar(cantidad=2,producto='Coca-Cola 3 lts.',precio=2300,formaPago='Efectivo')
```

```
Coca-Cola 3 lts. ....$ 2300 x 2 = $ 4600
forma de Pago: Efectivo
```

Las Funciones:

Argumentos mixtos:

Los argumentos mixtos en una función hacen referencia a una combinación de argumentos posicionales y argumentos por palabra clave. Esto significa que puedes tener parámetros sin un valor predeterminado (que deben ser proporcionados en el orden definido) y parámetros con valores predeterminados (que pueden ser omitidos si se proporciona un valor por palabra clave).

Los parámetros en este caso son 2 posicionales y 2 con palabra clave.

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, precio, formaPago='Efectivo', cantidad=1):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)

# Llamada a la función con argumentos de palabra clave
Pagar('Coca-Cola 3 lts.',2300,'Débito',5)
```

Los argumentos mantienen el orden definido.

Las Funciones:

Argumentos mixtos:

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, precio, formaPago='Efectivo', cantidad=1):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)
```

Todas las llamadas tienen distinto orden en sus argumentos, sin embargo, esto no afecta el resultado debido a que todos ellos proporcionan su valor a través de las palabras clave.

```python
# Llamada a la función con argumentos mixtos
Pagar(cantidad=5,formaPago='Débito',producto='Coca-Cola 3 lts.',precio=2300)
```

```
Coca-Cola 3 lts. ....$ 2300 x 5 = $ 11500
forma de Pago: Débito
```

```python
# Llamada a la función con argumentos mixtos
Pagar(producto='Coca-Cola 3 lts.',precio=2300,formaPago='Débito',cantidad=5)
```

```
Coca-Cola 3 lts. ....$ 2300 x 5 = $ 11500
forma de Pago: Débito
```

```python
# Llamada a la función con argumentos mixtos
Pagar(precio=2300,producto='Coca-Cola 3 lts.',cantidad=5,formaPago='Débito')
```

```
Coca-Cola 3 lts. ....$ 2300 x 5 = $ 11500
forma de Pago: Débito
```

```python
# Llamada a la función con argumentos mixtos
Pagar(formaPago='Débito',producto='Coca-Cola 3 lts.',precio=2300,cantidad=5)
```

```
Coca-Cola 3 lts. ....$ 2300 x 5 = $ 11500
forma de Pago: Débito
```

Las Funciones:

Valores predeterminados de Parámetros:

Los valores predeterminados en los parámetros de una función se utilizan para proporcionar un valor predeterminado en caso de que el argumento no sea pasado al llamar a la función. Esto hace que el parámetro sea opcional. Los valores predeterminados se definen al declarar los parámetros en la función.

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, precio, formaPago='Efectivo', cantidad=1):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)
```

En estos 3 casos, las llamadas fueron generadas de con distinta cantidad de argumentos, es decir, se omitieron algunos, pero solo aquellos que en la declaración indicaban valores por defecto.

```python
# Llamada a la función con argumentos mixtos
Pagar(producto='Coca-Cola 3 lts.',precio=2300)
```

```
Coca-Cola 3 lts. ....$ 2300 x 1 = $ 2300
forma de Pago: Efectivo
```

```python
# Llamada a la función con argumentos mixtos
Pagar(formaPago='Débito',producto='Coca-Cola 3 lts.',precio=2300)
```

```
Coca-Cola 3 lts. ....$ 2300 x 1 = $ 2300
forma de Pago: Débito
```

```python
# Llamada a la función con argumentos mixtos
Pagar(producto='Coca-Cola 3 lts.',precio=2300,formaPago='Débito',cantidad=5)
```

```
Coca-Cola 3 lts. ....$ 2300 x 5 = $ 11500
forma de Pago: Débito
```

python™

Inacap

Las Funciones:

Devolver el resultado de una función (Declaración de retorno):
En Python, la declaración de retorno se utiliza para devolver un valor desde una función. Cuando una función alcanza una instrucción de retorno, su ejecución se detiene y se devuelve el valor especificado (si hay alguno) al lugar desde donde fue llamada. La sintaxis para devolver un valor es la palabra clave return seguida del valor que se desea retornar.

```python
# Declaramos la función con 4 parámetros
def CalcularPrecio(precio, formaPago='Efectivo', cantidad=1):
    totalPagar = int(precio) * int(cantidad)
    if formaPago == 'Crédito':
        descuento = 0.1 * totalPagar
    else:
        descuento = 0
    totalPagar -= descuento
    # Retornamos la variable totalPagar
    return totalPagar

# Llamada a la función con argumentos mixtos
pagar = CalcularPrecio(25000,formaPago='Crédito',cantidad=2)

# Utilizamos el valor retornado por la función
print('Total a pagar $',pagar)
```

```
Total a pagar $ 45000.0
```

En este caso la función retorna el resultado del proceso realizado dentro de la función y la llamada lo recoge para guardarlo en una variable y luego imprimirlo.

python™

Inacap

Las Funciones:

Devolver el resultado de una función (el valor None):
En Python, None es un valor especial que representa la ausencia de un valor. Si una función no tiene una declaración de retorno o tiene una declaración de retorno sin valor, automáticamente se devuelve None.

```python
# Declaramos una función sin retorno
def funcionSinRetorno():
    print("Esta función no devuelve nada.")

# Llamada a la función y guardamos lo que retorna
resultado = funcionSinRetorno()

# Mostramos por pantalla
print('Valor del retorno:',resultado)
```

```
Esta función no devuelve nada.
Valor del retorno: None
```

Las Funciones:

Devolver el resultado de una función (retornando listas):

En Python, una función puede tener múltiples declaraciones de retorno a lo largo de su ejecución, y el valor retornado puede ser de cualquier tipo de dato válido en Python. Además, puedes devolver múltiples valores como tuplas, listas u otros objetos, lo que permite una gran flexibilidad en el diseño y uso de funciones.

```python
# Declaramos la función con 4 parámetros
def CalcularPrecio(precio, formaPago='Efectivo', cantidad=1):
    total = int(precio) * int(cantidad)
    if formaPago == 'Crédito':
        descuento = 0.1 * total
    else:
        descuento = 0
    totalPagar = total - descuento
    # Retornamos una lista con toda la información necesaria

    return [total,descuento,totalPagar]

# Llamada a la función con argumentos mixtos que retornara una lista
pagar = CalcularPrecio(25000,formaPago='Crédito',cantidad=2)

# Utilizamos el valor retornado por la función
print('Total sin descuento $',pagar[0],' - $',pagar[1],'(descuento) = $',pagar[2])
```

```
Total sin descuento $ 50000  - $ 5000.0 (descuento) = $ 45000.0
```

Las Funciones:

Funciones y ámbitos:

En Python, un ámbito (scope) se refiere al contexto en el cual una variable es accesible. Las funciones en Python tienen su propio ámbito local, lo que significa que las variables definidas dentro de una función solo son accesibles dentro de esa función. Estas variables se denominan variables locales. Fuera de la función, esas variables no existen y, si intentas acceder a ellas, generará un error.

```python
def calcularDescuento():
    descuento = 0.15   # Esta es una variable local dentro de la función
    print(descuento)

calcularDescuento()   # Salida: 0.15
# Esto generará un NameError, ya que descuento no está definido fuera de la función.
print(descuento)
```

```
    print(descuento)
          ^^^^^^^^^
NameError: name 'descuento' is not defined
```

Las Funciones:

Funciones y ámbitos:

En Python, un ámbito (scope) se refiere al contexto en el cual una variable es accesible. Las funciones en Python tienen su propio ámbito local, lo que significa que las variables definidas dentro de una función solo son accesibles dentro de esa función. Estas variables se denominan variables locales. Fuera de la función, esas variables no existen y, si intentas acceder a ellas, generará un error.

```python
def calcularDescuento():
    descuento = 0.15   # Esta es una variable local dentro de la función
    print('Descuento desde la función:',descuento)

# Llamamos a la función
calcularDescuento()   # Salida: 0.15

# Declaramos descuento fuera de la función
descuento = 0

# Verificamos si la variable sufrio cambios.
print('Descuento fuera de la función:',descuento)
```

```
Descuento desde la función: 0.15
Descuento fuera de la función: 0
```

Al declararla dentro y fuera de la función es accedida, pero con distintos resultados.

Las Funciones:

Variables Globales:

Las variables definidas fuera de cualquier función en Python tienen un alcance global, lo que significa que son accesibles desde cualquier parte del código, incluso dentro de las funciones. Sin embargo, si deseas modificar una variable global dentro de una función, debes usar la declaración global para indicar que la variable que estás usando es la variable global y no una variable local.

```python
def calcularDescuento():
    # Indicamos que estamos utilizando la variable global descuento
    global descuento
    # Esto modifica la variable global dentro de la función
    descuento = 0.15
    print('Descuento desde la función:',descuento)

# Declaramos descuento fuera de la función
descuento = 0
print('Descuento antes de llamar a la función:',descuento)

# Llamamos a la función
calcularDescuento()   # Salida: 0.15

# Verificamos si la variable sufrio cambios.
print('Descuento después de llamar a la función:',descuento)
```

```
Descuento antes de llamar a la función: 0
Descuento desde la función: 0.15
Descuento después de llamar a la función: 0.15
```

Las Funciones:

Interacción de Parámetros con sus Argumentos:
Cuando una función es llamada, los argumentos que se pasan se asignan a los parámetros de la función de acuerdo con la coincidencia de parámetros mencionados anteriormente (posicional, por palabra clave o mixtos).
Los parámetros actúan como variables locales dentro de la función y toman el valor de los argumentos que se les pasan.

```python
# Declaramos la función con 4 parámetros
def Pagar(producto, precio, formaPago='Efectivo', cantidad=1):
    total = int(precio) * int(cantidad)
    print(producto,'....$',precio,'x',cantidad,'= $',total)
    print('forma de Pago:',formaPago)

# Llamada a la función con argumentos de palabra clave
Pagar('Coca-Cola 3 lts.',2300,'Débito',5)
```

Las Funciones:

La Recursividad:

Es una técnica que permite a una función llamarse a sí misma. En otras palabras, una función recursiva es aquella que se define en términos de sí misma, y esto puede ser útil para resolver problemas que se pueden dividir en subproblemas más pequeños y similares al problema original.

Función Recursiva

Caso base:

Es el caso más simple que se puede resolver directamente sin llamar a la función nuevamente. Un caso base detiene la recursión y evita que la función se llame a sí misma indefinidamente.

Caso recursivo:

Es el caso en el cual la función se llama a sí misma para resolver un problema más pequeño que es similar al problema original, pero más simple. La función se sigue llamando recursivamente hasta alcanzar el caso base.

Las Funciones:

La Recursividad:
Al llamar a esta función como factorial(5) ocurre lo siguiente:

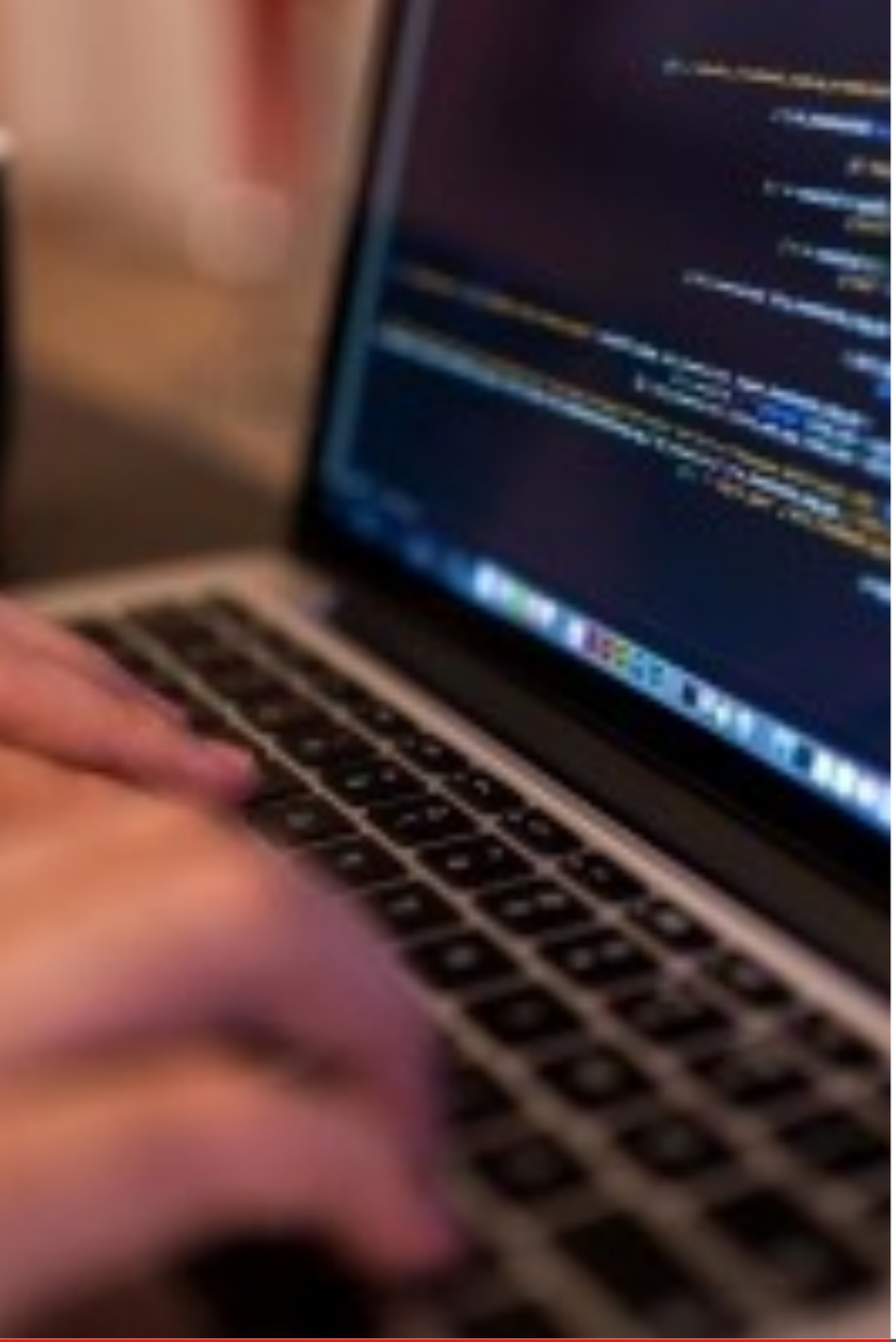
1. factorial(5) llama a factorial(4) ...para calcular 4!
2. factorial(4) llama a factorial(3) ...para calcular 3!
3. factorial(3) llama a factorial(2) ...para calcular 2!
4. factorial(2) llama a factorial(1) ...para calcular 1!
5. factorial(1) *retorna 1, ya que es el caso base*.

```python
def factorial(n):
    if n == 0 or n == 1:
        # Caso base: el factorial de 0 y 1 es 1
        return 1
    else:
        # Caso recursivo: n! = n * (n-1)!
        return n * factorial(n - 1)
```

- factorial(2) multiplica 2 por el resultado de factorial(1), que es 1, y retorna 2.
- factorial(3) multiplica 3 por el resultado de factorial(2), que es 2, y retorna 6.
- factorial(4) multiplica 4 por el resultado de factorial(3), que es 6, y retorna 24.
- factorial(5) multiplica 5 por el resultado de factorial(4), que es 24, y retorna 120.

En el caso recursivo la función se llamará a sí misma hasta entrar en el caso base.

El resultado final es 120, que es el factorial de 5..



A practicar...

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

1. Desarrolle un script que a través de una función una cadena de caracteres y muestre como resultado dos cadenas, una que contenga solo las vocales y otra solo las consonantes:

```python
def separarVocalesConsonantes(cadena):
    vocales = ""
    consonantes = ""

    for letra in cadena:
        if letra in 'aeiou':
            vocales += letra
        else:
            consonantes += letra

    return vocales, consonantes


palabra = input('Ingrese una palabra y luego presione ENTER: ')
vocalesResultado, consonantesResultado = separarVocalesConsonantes(palabra)
print("Vocales:", vocalesResultado)
print("Consonantes:", consonantesResultado)
```

```
Ingrese una palabra y luego presione ENTER: Buenas tardes
Vocales: ueaae
Consonantes: Bns trds
```

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

2. Desarrolle un script que a través de una función tome una cadena de texto como argumento y cuente la cantidad de veces que aparece cada palabra en la cadena. Debe retornar un diccionario donde las claves son las palabras y los valores son las cantidades de veces que conto a la palabra:

```python
def contarPalabras(cadena):
    palabras = cadena.split()
    contador = {}

    for palabra in palabras:
        palabra = palabra.lower()
        if palabra in contador:
            contador[palabra] += 1
        else:
            contador[palabra] = 1

    return contador


texto = "A Cuesta le cuesta subir la cuesta y en medio de la cuesta va y se acuesta"
resultado = contarPalabras(texto)
print(resultado)
```

```
{'a': 1, 'cuesta': 4, 'le': 1, 'subir': 1, 'la': 2, 'y': 2, 'en': 1, 'medio': 1, 'de': 1, 'va': 1, 'se': 1, 'acuesta': 1}
```

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

3. Desarrolle un script que a través de una función que recibe una palabra, retorne como respuesta si la palabra ingresada por teclado es o no Palíndromo:

```python
def esPalindromo(palabra):
    palabra = palabra.lower().replace(" ", "")
    return palabra == palabra[::-1]


palabra_ingresada = input("Ingrese una palabra: ")
if esPalindromo(palabra_ingresada):
    print("La palabra es un palíndromo.")
else:
    print("La palabra no es un palíndromo.")
```

```
Ingrese una palabra: ana
La palabra es un palíndromo.
```

```
Ingrese una palabra: casa
La palabra no es un palíndromo.
```

inacap.cl