FUNCIONES EN PYTHON
Programación funcional

1. **Subprograma:**
   Es un conjunto de instrucciones que realizan una tarea especifica. Un programa principal llama al subprograma cuando lo necesite.

   Un subprograma (función) se declara dentro de un programa con el fin de ejecutar una tarea específica y puede ser llamado o invocado desde el programa principal o desde otro subprograma cuando sea necesario.

   Un subprograma es código reutilizable

2. Subprograma tipo **Función:**
   Una función es un conjunto de instrucciones que recibe un valor a través de sus parámetros; realizan un cálculo (lógico o aritmético) y retornan el valor resultante.

   Cuando una función termina su tarea devuelve un valor al programa o subprograma que la invocó, entonces en el llamador el identificador de la función se convierte en el valor retornado.

   El valor retornado pertenece a algún tipo de dato conocido, el identificador de la función toma este valor retornado y puede ser utilizado por el llamador.

   Toda función recibe un valor en sus parámetros y retorna un valor.

3. **Divide y vencerás:**
   El descomponer un programa en subprogramas independientes más simples, se conoce también como el método de "Divide y vencerás". Los subprogramas son útiles en dos casos:

   1. Cuando existe un grupo de instrucciones o una tarea específica que deba ejecutarse en más de una ocasión.

   2. Cuando un problema es complejo o extenso, la solución se divide o segmenta en subprogramas que resuelven partes de un problema más grande.

4. **Sintaxis y declaración de una función:**

```
# Declaración de una función que retorna valores
def nomb_func(parámetro1, parámetro2):
    instrucción1
    instrucción2
    return resultado


# Declaración de una función sin retorno de valores
def nomb_proc(parámetro1, parámetro2):
    instrucción1
    instrucción2
```

A continuación, se utilizará un ejemplo que calcula el cubo de un número entero positivo:

```
# Función que calcula y retorna el cubo de un número
def cubo_f(num):
    return num ** 3

# Función que calcula EL cubo de un número
# esta función NO RETORNA VALORES.
def cubo_p(num):
    pot= num ** 3
    print(num, 'elevado a la 3 es ', pot)
```

Toda función retorna un valor

No retorna valores, solo los muestra en pantalla

Las funciones necesitan ser llamadas, y así poder activarlas, a continuación se muestra un programa completo donde se hace la declaración de una función y se ilustra la llamada usando el ejemplo anterior:

```
def cubo_f(num): # Función
    return num ** 3

# Programa principal
print('Calcular el cubo de un número entero')
para_el_cubo = int(input('Ingrese el número deseado: '))
print(para_el_cubo, 'elevado a la 3 es ', cubo_f(para_el_cubo))
```

Declaración de subprogramas

Llamada a la función

5. Llamada a una función:

Es posible que una función no retorne valores. También puede retornar valores, estos valores deben ser utilizados en el programa (o función) llamador, de no ser utilizado, el valor retornado se pierde.

Distintas formas de llamar a una función:

- Llamada a una función que no retorna valores

```
nomb_f(x)
```

- Llamado a una función que retorna valores bool: usar el valor retornado como condición.

```
if nomb_f(x):
```

- Asignando el valor retornado por la función a una variable:

```
res = nomb_f(x)
```

- Utilizando el valor retornado por la función en una expresión aritmética:

```
x = (a * b) + nomb_f(x)
```

- Mostrando en pantalla el valor retornado por la función:

```
print( nomb_f(x) )
```

- Una función siempre retorna valores, estos valores retornados también pertenecen a alguno de los tipos de dato conocidos (int, float, str, bool), en aquellos casos donde la función retorna un valor tipo bool, este valor puede ser usado como una condición, vea el siguiente programa de ejemplo donde se verifica si un número es par o impar:

```
# Declaración de la función
def es_par(numero_p):
    si_es = False
    if numero_p % 2 == 0:
        si_es = True
    return si_es

# Programa principal
print('Revisa si el número es par o impar')
num_g = int(input('Número a revisar: '))

if es_par(num_g):
    print(num_g, 'Es número par')
else:
    print(num_g, 'Es número impar')
```

Llamada a la función usada como condición

De ser necesario para la solución de un problema la llamada a la función puede ser usada en la condición de un ciclo de repetición while.

6. Tipos de parámetros en subprogramas:

6.1. De acuerdo con la ubicación en el código

Parámetros formales	**Parámetros actuales o argumentos**
Son los parámetros que escribimos en la declaración del subprograma, aquí se escriben los identificadores donde vamos a recibir los valores enviados en los parámetros actuales	Son los parámetros que escribimos en la llamada al subprograma, aquí se escriben las variables que contienen los valores que le vamos a enviar al subprograma

Parámetro formal en la declaración

Parámetro actual en la llamada

```
def es_par(numero_p):
    si_es = False
    if numero_p % 2 == 0:
        si_es = True
    return si_es

# Programa principal
num_g = int(input('Número: '))

if es_par(num_g):
    print(num_g, 'Es número par')
else:
    print(num_g, 'Es número impar')
```

6.2. **De acuerdo a la forma de recibir valores:** Esta clasificación tiene lugar al momento del intercambio de valores entre parámetros actuales y parámetros formales

Parámetros por valor	**Parámetros por referencia**
En Python este pase de parámetros solo es aplicable a los datos inmutables. • El valor que contiene el parámetro actual se copia en el parámetro formal. • El parámetro actual es independiente del parámetro formal. • Al finalizar la función no hay transformación del parámetro actual.	En Python solo los datos mutables son pasados por referencia. • El parámetro actual y el parámetro formal comparten el mismo espacio de memoria. • El parámetro actual y el parámetro formal están estrechamente relacionados. • Al finalizar la función el parámetro actual conserva todos los cambios hechos al parámetro formal.

7. Funciones CON retorno de valores vs. Funciones SIN retorno de valores

7.1. Diferencias

Funciones CON retorno de valores	**Funciones SIN retorno de valores**
• Retorna valores • Se utilizan para realizar algún tipo de cálculo lógico, aritmético o de otro tipo y retornar el resultado al programa o subprograma llamador. • Es necesario utilizar el valor retornado en la llamada.	• No retorna valores • Se utilizan para realizar acciones específicas dentro de un programa siempre que no implique retornar valores. • Su llamada se convierte en una instrucción más del llamador.

7.2. Similitudes

Funciones CON retorno de valores	**Funciones SIN retorno de valores**
• Reciben ninguno o varios parámetros • Pueden contener sus propias variables • En Python se utiliza la palabra reservada "**`def`**" para declararlas • La llamada se hace a través del identificador de la función.	• Reciben ninguno o varios parámetros • Pueden contener sus propias variables • En Python se utiliza la palabra reservada "**`def`**" para declararlas. • La llamada se hace a través del identificador de la función.