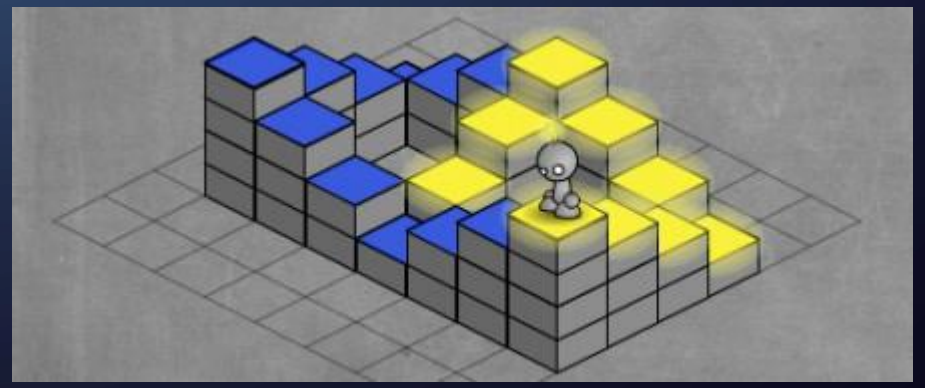
PYTHON - FUNCIONES

LIC. MARIELA ASENSIO

Son fragmentos de código que podemos ejecutar múltiples veces. Pueden recibir y devolver información para comunicarse con el programa principal.

Las funciones pueden utilizarse desde los programas o integradas en un módulo especial a manera de librería para organizar un proyecto de programación.

Esta estrategia de dividir la resolución de un problema en módulos y funciones es parte de la metodología denominada **Programación Modular** que es importante para resolver problemas grandes o complejos.



Declaración de una función

```
def  nombre(parámetros):
    instrucciones
```

Nombre: es la identificación de la función
Parámetros: son variables que reciben los datos que entran a la función
Instrucciones: se incluyen en la función ara producir resultados

Las instrucciones incluidas en la función deben estar encolumnadas como en las otras estructuras de control.

Si la función entrega resultados, debe usarse una instrucción especial para el retorno de resultados:

```
return variable

return [variable, variable…, variable]
```

```
>>> def saludo():
    print('hola, bienvenidos!!')

    >>> saludo()
hola, bienvenidos!!
```

Dentro de una función podemos utilizar variables y sentencias de control

```
>>> def dibujar_tabla_del_5():
    for i in range(10):
        print("5 * {} =
{}".format(i,i*5))
```

```
>>> dibujar_tabla_del_5()
5 * 0 = 0
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
```

Ámbito de las funciones

Una variable declarada en una función sólo existe dentro de esa función. Se dice que es una variable local.

```
>>> def test():
    o=5
    print(o)


>>> test()
5
>>> o
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    o
NameError: name 'o' is not defined
```

Otro ejemplo

```
>>> def test():
    o=5
    print(o)


>>> o=10
>>> test()
5
>>> o
10
```

Sin embargo, una variable declarada fuera de la función (al mismo nivel), sí que es accesible desde la función y desde el programa principal. Se dice que es una variable global

```
>>> m=10
>>> def test():
    print(m)


>>> test()
10
>>> m
10
```

La instrucción global

Podemos utilizar esta instrucción para definir una variable de tipo global y poder referenciarla tanto dentro de una función como desde el programa principal.

```
>>> def test():
    global o
    o = 5
    print(o)


>>> test()
5
>>> o
5
```

Podemos escribir una función en la ventana dinámica del IDLE

```
def f(x):
    y=2*x**2+1
    return y

>>> f(3)
19
```

Las funciones pueden escribirse junto al programa que las usan

```
def f(x):
    y=2*x**2+1
    return y
#Programa que usa la función
valor = int(input('Ingrese un valor: '))
resultado = f(valor)
print(resultado)
```

La función puede crearse y almacenarse separadamente en un archivo. Este objeto constituye un módulo, el nombre del módulo puede coincidir con el nombre de la función o tener un nombre diferente.

```
def f(x):
    y=2*x**2+1
    return y
```

módulo

Para usarla debe importarse el módulo

```
from funcion import f
valor = int(input('Ingrese un valor: '))
resultado = f(valor)
print(resultado)
```

Argumentos y Parámetros

Al definir una función los valores los cuales se reciben se denominan parámetros, pero durante la llamada los valores que se envían se denominan argumentos.

Función sin parámetros

```
>>> def cls():print('\n'*40)

>>> cls()
```

Despeja la pantalla

Argumentos por posición

Cuando enviamos argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición:

```
def resta(a,b):
    r = a-b
    return r
#Programa que usa la función
x = resta(35,13)
print(x)

22
```

El argumento a toma el valor 35 y el argumento b el valor 13

Argumentos por nombre

Es posible evadir el orden de los parámetros si indica durante la llamada que valor tiene cada parámetro a partir de su nombre:

```
def resta(a,b):
    r = a-b
    return r
#Programa que usa la función
x = resta(b=35,a=13)
print(x)

-22
```

Parámetros por valor y por referencia

Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

- **Paso por valor**: Se crea una copia local de la variable dentro de la función.

- **Paso por referencia**: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Generalmente:

- **Los tipos simples se pasan por valor**: Enteros, flotantes, cadenas, lógicos...
- **Los tipos compuestos se pasan por referencia**: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

```
def resta(a,b):
    x=a-b
    return x
#programa que llama a la función
v1 = int(input('Ingrese un valor: '))
v2 = int(input('Ingrese otro valor: '))
r = resta(v1,v2)
print(r)
```

Ejemplo de paso por referencia

En este caso, el parámetro de la función utiliza la misma dirección de la variable con que se llama a la función. Por lo tanto si la función modifica componentes del parámetro, estos cambios afectan a la variable en el sitio de la llamada a la función.

```
def funp (t):
    r=max(t)
    t[0]=-1
    return r
```

```
from funp import*
s=[8,3,9,4,7]
r = funp(s)
print(r)
print('la lista modificada')
print(s)
```

Argumentos indeterminados

En alguna ocasión si no se conoce previamente cuantos elementos se necesitan enviar a una función. En estos casos se puede utilizar los parámetros indeterminados

```
def indefinida (*args):
    for arg in args:
        print(arg)

#programa que usa la función
indefinida (1,'Mariela',2.5,-5)
```

```
1
Mariela
2.5
-5
```

Funciones Recursivas

Una función puede llamarse a sí misma. Estas funciones se denominan recursivas.
El uso de la recursión es una técnica útil en programación para resolver algunos problemas en sustitución de los métodos iterativos con for o while.

```
num=int(input('Ingrese un valor
para calcular el factorial: '))
f=1
for i in range(num,1,-1):
    f=f*num
    num=num-1
print(f)
```

Cálculo del factorial de forma tradicional

Ahora resolvemos el mismo problema de forma recursiva

```
def factorial(num):
    if num>1:
        num = num * factorial(num-1)
    return num
#programa que llama a la función
x = int(input('Ingrese el valor para calcular
el factorial: '))
f = factorial(x)
print(f)
```