

Guía 4

Funciones

Concepto de funciones - Programación estructurada

Hasta ahora hemos trabajado con una metodología de programación lineal. Todas las instrucciones de nuestro archivo *.py se ejecutan en forma secuencial de principio a fin. Esta forma de organizar un programa solo puede ser llevado a cabo si el mismo es muy pequeño (decenas de líneas)

Cuando los problemas a resolver tienden a ser más grandes la metodología de programación lineal se vuelve ineficiente y compleja. El segundo paradigma de programación que veremos es la programación estructurada.

La programación estructurada busca dividir o descomponer un problema complejo en pequeños problemas. La solución de cada uno de esos pequeños problemas nos trae la solución del problema complejo.

En Python el planteo de esas pequeñas soluciones al problema complejo se hace dividiendo el programa en funciones. Una función es un conjunto de instrucciones en Python que resuelven un problema específico.

El lenguaje Python ya tiene incorporada algunas funciones básicas. Algunas de ellas ya las utilizamos en conceptos anteriores como son las funciones: print, len y range. Veamos ahora como crear nuestras propias funciones. El tema de funciones en un principio puede presentar dificultades para entenderlo y ver sus ventajas ante la metodología de programación lineal que veníamos trabajando en conceptos anteriores.

Los primeros problemas que presentaremos nos puede parecer que sea más conveniente utilizar programación lineal en vez de programación estructurada por funciones. A medida que avancemos veremos que si un programa empieza a ser más complejo (cientos de líneas, miles de líneas o más) la división en pequeñas funciones nos permitirá tener un programa más ordenado y fácil de entender y por lo tanto en mantener.

Problema 1:

Confeccionar una aplicación que muestre una presentación en pantalla del programa. Solicite la carga de dos valores y nos muestre la suma. Mostrar finalmente un mensaje de despedida del programa. Implementar estas actividades en tres funciones.

```
def presentacion():  
    print("Programa que permite cargar dos valores por teclado.")  
    print("Efectua la suma de los valores")
```

```
print("Muestra el resultado de la suma")
print("*****")

def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

def finalizacion():
    print("*****")
    print("Gracias por utilizar este programa")

# programa principal

presentacion()
carga_suma()
finalizacion()
```

La forma de organizar nuestro programa cambia en forma radical.

El programa ahora no empieza a ejecutarse en la línea 1.

El programa principal comienza a ejecutarse luego del comentario "programa principal":

```
# programa principal

presentacion()
carga_suma()
finalizacion()
```

Primero declaramos las tres funciones llamadas presentacion, carga_suma y finalizacion.

La sintaxis para declarar una función es mediante la palabra clave def seguida por el nombre de la función (el nombre de la función no puede tener espacios en blanco, comenzar con un número y el único carácter especial permitido es el _)

Luego del nombre de la función deben ir entre paréntesis los datos que llegan, si no llegan datos como es el caso de nuestras tres funciones solo se disponen paréntesis abierto y cerrado. Al final disponemos los :

Todo el bloque de la función se indenta cuatro espacios como venimos trabajando cuando definimos estructuras condicionales o repetitivas.

Dentro de una función implementamos el algoritmo que pretendemos que resuelva esa función, por ejemplo la función presentacion tiene por objetivo mostrar en pantalla el objetivo del programa:

```
def presentacion():  
    print("Programa que permite cargar dos valores por teclado.")  
    print("Efectua la suma de los valores")  
    print("Muestra el resultado de la suma")  
    print("*****")
```

La función `carga_suma()` permite ingresar dos enteros por teclado, sumarlos y mostrarlos en pantalla:

```
def carga_suma():  
    valor1=int(input("Ingrese el primer valor:"))  
    valor2=int(input("Ingrese el segundo valor:"))  
    suma=valor1+valor2  
    print("La suma de los dos valores es:",suma)
```

La función `finalizacion()` tiene por objetivo mostrar un mensaje que muestre al operador que el programa finalizó:

```
def finalizacion():  
    print("*****")  
    print("Gracias por utilizar este programa")
```

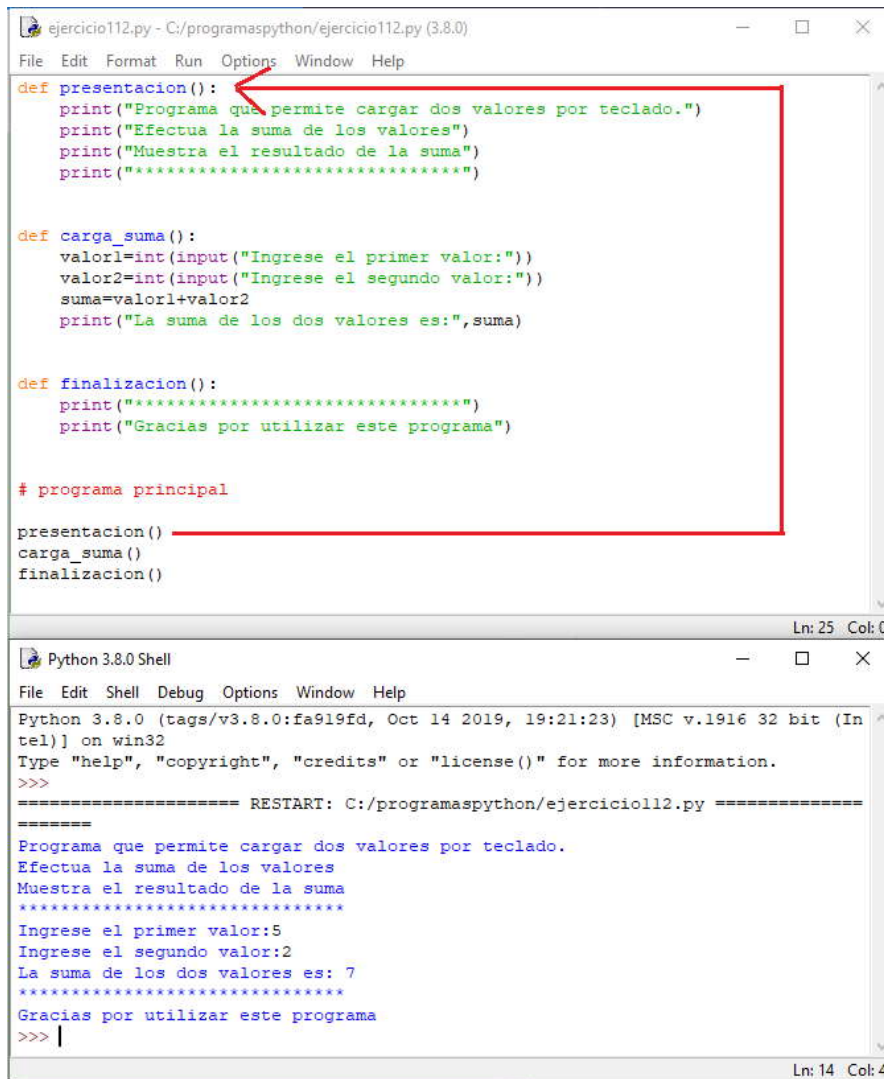
Luego de definir las funciones tenemos al final de nuestro archivo `*.py` las llamadas de las funciones:

```
# programa principal  
  
presentacion()  
carga_suma()  
finalizacion()
```

Si no hacemos las llamadas a las funciones los algoritmos que implementan las funciones nunca se ejecutarán.

Cuando en el bloque del programa principal se llama una función hasta que no finalice no continua con la llamada a la siguiente función:

```
# programa principal  
  
presentacion() # se ejecutan las cuatro líneas que contiene la función presentacion y  
recién continúa con la próxima línea.
```



```
ejercicio112.py - C:/programaspython/ejercicio112.py (3.8.0)
File Edit Format Run Options Window Help

def presentacion():
    print("Programa que permite cargar dos valores por teclado.")
    print("Efectua la suma de los valores")
    print("Muestra el resultado de la suma")
    print("*****")

def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

def finalizacion():
    print("*****")
    print("Gracias por utilizar este programa")

# programa principal

presentacion()
carga_suma()
finalizacion()

Ln: 25 Col: 0

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/programaspython/ejercicio112.py =====
=====
Programa que permite cargar dos valores por teclado.
Efectua la suma de los valores
Muestra el resultado de la suma
*****
Ingrese el primer valor:5
Ingrese el segundo valor:2
La suma de los dos valores es: 7
*****
Gracias por utilizar este programa
>>> |

Ln: 14 Col: 4
```

El orden en que llamamos a las funciones es muy importante. Supongamos que nuestro bloque del programa principal llamamos las funciones en este orden:

```
# programa principal
```

```
finalizacion()
carga_suma()
presentacion()
```

Veremos que primero muestra el mensaje de finalización, luego la carga y suma de datos y finalmente aparece por pantalla los mensajes de presentación de nuestro programa.

Yo sé que en principio al ser un problema tan sencillo y que venimos de resolver muchos programas con la metodología de programación lineal nos parecerá más fácil implementar este programa con la sintaxis:

```
print("Programa que permite cargar dos valores por teclado.")
print("Efectúa la suma de los valores")
print("Muestra el resultado de la suma")
```

```
print("*****")

valor1=int(input("Ingrese el primer valor:"))
valor2=int(input("Ingrese el segundo valor:"))
suma=valor1+valor2
print("La suma de los dos valores es:",suma)

print("*****")
print("Gracias por utilizar este programa")
```

Tengamos un poco de paciencia para aprender la nueva sintaxis de dividir un programa en funciones, imagina que las soluciones a problemas complejos pueden requerir miles de líneas y un planteo secuencial será muy difícil.

Problema 2:

Confeccionar una aplicación que solicite la carga de dos valores enteros y muestre su suma.

Repetir la carga e impresión de la suma 5 veces.

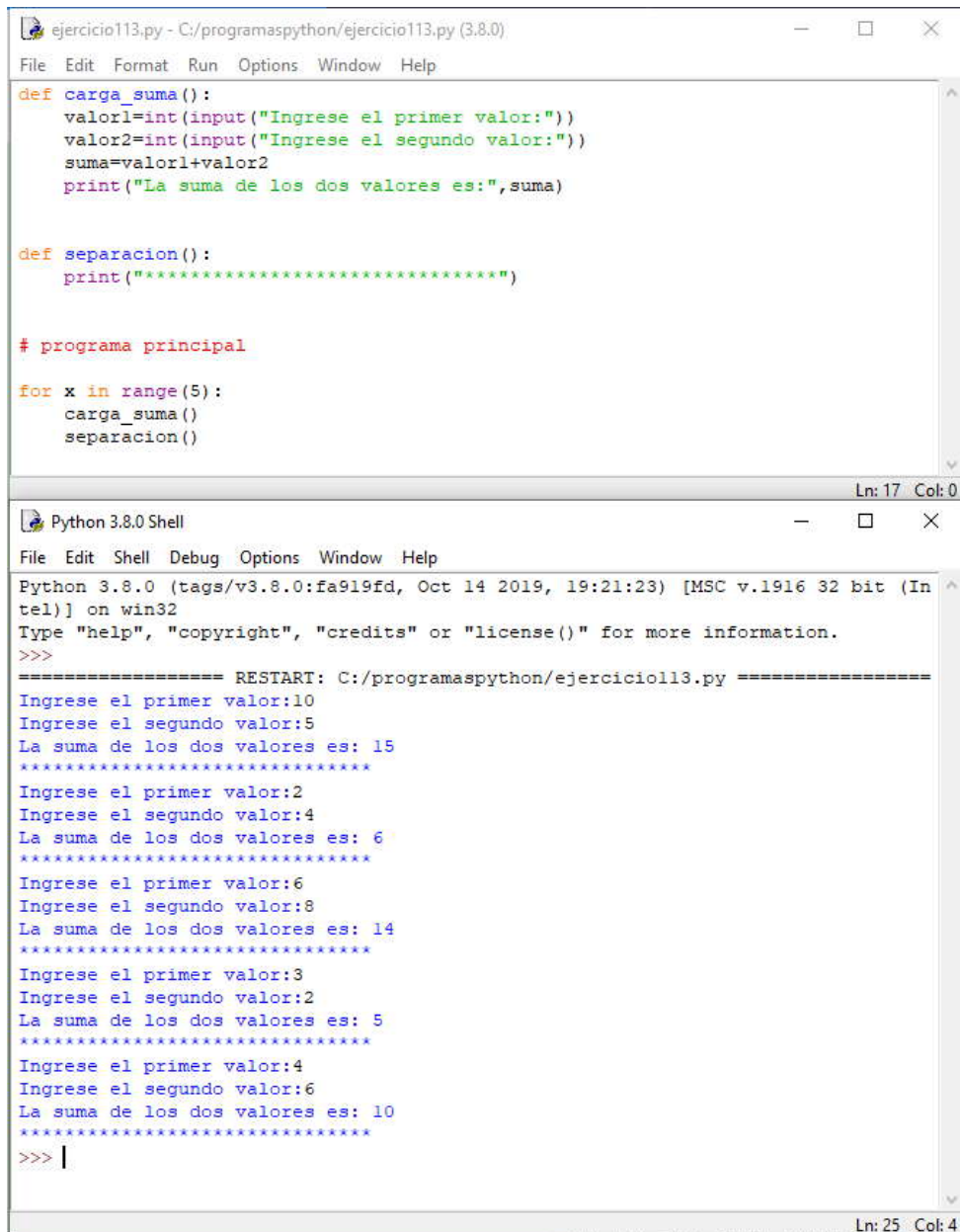
Mostrar una línea separadora después de cada vez que cargamos dos valores y su suma.

```
def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

def separacion():
    print("*****")

# programa principal

for x in range(5):
    carga_suma()
    separacion()
```



```
ejercicio113.py - C:/programaspython/ejercicio113.py (3.8.0)
File Edit Format Run Options Window Help

def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

def separacion():
    print("*****")

# programa principal

for x in range(5):
    carga_suma()
    separacion()

Ln: 17 Col: 0

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/programaspython/ejercicio113.py =====
Ingrese el primer valor:10
Ingrese el segundo valor:5
La suma de los dos valores es: 15
*****
Ingrese el primer valor:2
Ingrese el segundo valor:4
La suma de los dos valores es: 6
*****
Ingrese el primer valor:6
Ingrese el segundo valor:8
La suma de los dos valores es: 14
*****
Ingrese el primer valor:3
Ingrese el segundo valor:2
La suma de los dos valores es: 5
*****
Ingrese el primer valor:4
Ingrese el segundo valor:6
La suma de los dos valores es: 10
*****
>>> |

Ln: 25 Col: 4
```

Hemos declarado dos funciones, una que permite cargar dos enteros sumarlos y mostrar el resultado:

```
def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)
```

Y otra función que tiene por objetivo mostrar una línea separadora con asteriscos:

```
def separacion():
    print("*****")
```

Ahora nuestro bloque principal del programa, recordemos que estas líneas son las primeras que se ejecutarán cuando iniciemos el programa:

```
# programa principal
```

```
for x in range(5):  
    carga_suma()  
    separacion()
```

Como vemos podemos llamar a la función `carga_suma()` y `separación()` muchas veces en nuestro caso en particular 5 veces.

Lo nuevo que debe quedar claro es que la llamada a las funciones desde el bloque principal de nuestro programa puede hacerse múltiples veces (esto es lógico, recordemos que `print` es una función ya creada en Python y la llamamos múltiples veces dentro de nuestro algoritmo)

Funciones con parámetros

Vimos en el concepto anterior que una función resuelve una parte de nuestro algoritmo. Tenemos por un lado la declaración de la función por medio de un nombre y el algoritmo de la función seguidamente. Luego para que se ejecute la función la llamamos desde el bloque principal de nuestro programa.

Ahora veremos que una función puede tener parámetros para recibir datos. Los parámetros nos permiten comunicarle algo a la función y la hace más flexible.

Problema 1:

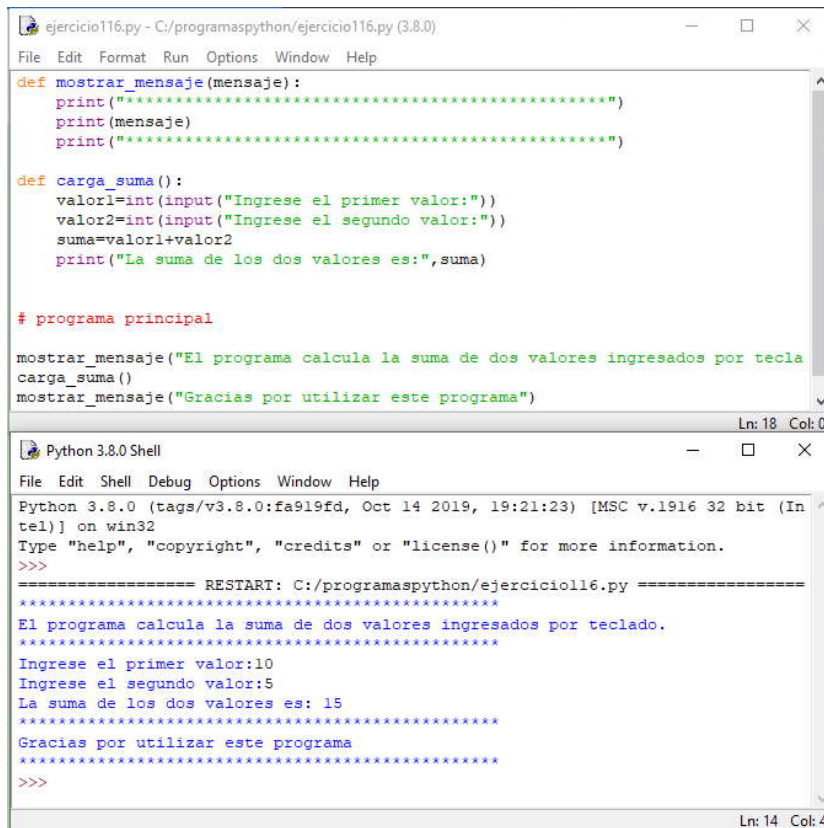
Confeccionar una aplicación que muestre una presentación en pantalla del programa. Solicite la carga de dos valores y nos muestre la suma. Mostrar finalmente un mensaje de despedida del programa.

```
def mostrar_mensaje(mensaje):
    print("*****")
    print(mensaje)
    print("*****")

def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

# programa principal

mostrar_mensaje("El programa calcula la suma de dos valores ingresados por teclado.")
carga_suma()
mostrar_mensaje("Gracias por utilizar este programa")
```

```
ejercicio116.py - C:/programasp/python/ejercicio116.py (3.8.0)
File Edit Format Run Options Window Help

def mostrar_mensaje(mensaje):
    print("*****")
    print(mensaje)
    print("*****")

def carga_suma():
    valor1=int(input("Ingrese el primer valor:"))
    valor2=int(input("Ingrese el segundo valor:"))
    suma=valor1+valor2
    print("La suma de los dos valores es:",suma)

# programa principal

mostrar_mensaje("El programa calcula la suma de dos valores ingresados por teclado")
carga_suma()
mostrar_mensaje("Gracias por utilizar este programa")

Ln: 18 Col: 0

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/programasp/python/ejercicio116.py =====
*****
El programa calcula la suma de dos valores ingresados por teclado.
*****
Ingrese el primer valor:10
Ingrese el segundo valor:5
La suma de los dos valores es: 15
*****
Gracias por utilizar este programa
*****
>>>

Ln: 14 Col: 4
```

Ahora para resolver este pequeño problema hemos planteado una función llamada `mostrar_mensaje` que recibe como parámetro un string (cadena de caracteres) y lo muestra en pantalla.

Los parámetros van seguidos del nombre de la función encerrados entre paréntesis (y en el caso de tener más de un parámetro los mismos deben ir separados por coma):

```
def mostrar_mensaje(mensaje):
    print("*****")
    print(mensaje)
    print("*****")
```

Un parámetro podemos imaginarlo como una variable que solo se puede utilizar dentro de la función.

Ahora cuando llamamos a la función `mostrar_mensaje` desde el bloque principal de nuestro programa debemos pasar una variable string o un valor de tipo string:

```
mostrar_mensaje("El programa calcula la suma de dos valores ingresados por teclado.")
```

El string que le pasamos: "El programa calcula la suma de dos valores ingresados por teclado." lo recibe el parámetro de la función.

Una función con parámetros nos hace más flexible la misma para utilizarla en distintas circunstancias. En nuestro problema la función `mostrar_mensaje` la utilizamos tanto para la presentación inicial de nuestro programa como para mostrar el mensaje de

despedida. Si no existieran los parámetros estaríamos obligados a implementar dos funciones como el concepto anterior.

Problema 2:

Confeccionar una función que reciba tres enteros y nos muestre el mayor de ellos. La carga de los valores hacerlo por teclado.

```
def mostrar_mayor(v1,v2,v3):  
    print("El valor mayor de los tres numeros es")  
    if v1>v2 and v1>v3:  
        print(v1)  
    else:  
        if v2>v3:  
            print(v2)  
        else:  
            print(v3)  
  
def cargar():  
    valor1=int(input("Ingrese el primer valor:"))  
    valor2=int(input("Ingrese el segundo valor:"))  
    valor3=int(input("Ingrese el tercer valor:"))  
    mostrar_mayor(valor1,valor2,valor3)  
  
# programa principal  
  
cargar()
```

Es importante notar que un programa en Python no ejecuta en forma lineal las funciones definidas en el archivo *.py sino que arranca en la zona del bloque principal. En nuestro ejemplo se llama primero a la función "cargar()", esta función no tiene parámetros. La función cargar solicita el ingreso de tres enteros por teclado y llama a la función mostrar_mayor y le pasa a sus parámetros las tres variable enteras valor1, valor2 y valor3.

La función mostrar_mayor recibe en sus parámetros v1, v2 y v3 los valores cargados en las variables valor1, valor2 y valor3.

Los parámetros son la forma que nos permite comunicar la función cargar con la función mostrar_mayor.

Dentro de la función mostrar_mayor no podemos acceder a las variables valor1, valor2 y valor3 ya que son variables locales de la función cargar.

La definición mostrar_mayor debe estar escrita antes de la definición de la función cargar que es donde la llamamos.

Otra cosa importante notar que en la sección del programa principal solo llamamos a la función cargar, es decir que en esta zona no es obligatorio llamar a cada una de las funciones que definimos.

Problema 3:

Desarrollar un programa que permita ingresar el lado de un cuadrado. Luego preguntar si quiere calcular y mostrar su perímetro o su superficie.

```
def mostrar_perimetro(lado):  
    per=lado*4  
    print("El perimetro es",per)  
  
def mostrar_superficie(lado):  
    sup=lado*lado  
    print("La superficie es",sup)  
  
def cargar_dato():  
    la=int(input("Ingrese el valor del lado de un cuadrado:"))  
    respuesta=input("Quiere calcular el perimetro o la superficie[ingresar texto:  
perimetro/superficie]?")  
    if respuesta=="perimetro":  
        mostrar_perimetro(la)  
    if respuesta=="superficie":  
        mostrar_superficie(la)  
  
# programa principal  
  
cargar_dato()
```

Definimos dos funciones que calculan y muestran el perímetro por un lado y por otro la superficie:

```
def mostrar_perimetro(lado):  
    per=lado*4  
    print("El perimetro es",per)  
  
def mostrar_superficie(lado):  
    sup=lado*lado  
    print("La superficie es",sup)
```

La tercera función permite cargar el lado del cuadrado e ingresar un string que indica que cálculo deseamos realizar si obtener el perímetro o la superficie. Una vez que se ingresó la variable respuesta procedemos a llamar a la función que efectúa el cálculo respectivo pasando como dato la variable local "la" que almacena el valor del lado del cuadrado.

Los parámetros son la herramienta fundamental para pasar datos cuando hacemos la llamada a una función.

Funciones: retorno de datos

Hemos comenzado a pensar con la metodología de programación estructurada. Buscamos dividir un problema en subproblemas y plantear algoritmos en Python que los resuelvan.

Vimos que una función la definimos mediante un nombre y que puede recibir datos por medio de sus parámetros.

Los parámetros son la forma para que una función reciba datos para ser procesados. Ahora veremos otra característica de las funciones que es la de devolver un dato a quien invocó la función (recordemos que una función la podemos llamar desde el bloque principal de nuestro programa o desde otra función que desarrollemos)

Problema 1:

Confeccionar una función que le enviemos como parámetro el valor del lado de un cuadrado y nos retorne su superficie.

```
def retornar_superficie(lado):  
    sup=lado*lado  
    return sup  
  
# bloque principal del programa  
  
va=int(input("Ingrese el valor del lado del cuafrado:"))  
superficie=retornar_superficie(va)  
print("La superficie del cuadrado es",superficie)
```

Aparece una nueva palabra clave en Python para indicar el valor devuelto por la función: return

La función retornar_superficie recibe un parámetro llamado lado, definimos una variable local llamada sup donde almacenamos el producto del parámetro lado por sí mismo.

La variable local sup es la que retorna la función mediante la palabra clave return:

```
def retornar_superficie(lado):  
    sup=lado*lado  
    return sup
```

Hay que tener en cuenta que las variables locales (en este caso sup) solo se pueden consultar y modificar dentro de la función donde se las define, no se tienen acceso a las mismas en el bloque principal del programa o dentro de otra función.

Hay un cambio importante cuando llamamos o invocamos a una función que devuelve un dato:

```
superficie=retornar_superficie(va)
```

Es decir, el valor devuelto por la función retornar_superficie se almacena en la variable superficie.

Es un error lógico llamar a la función `retornar_superficie` y no asignar el valor a una variable:

```
retornar_superficie(va)
```

El dato devuelto (en nuestro caso la superficie del cuadrado) no se almacena. Si podemos utilizar el valor devuelto para pasarlo a otra función:

```
va=int(input("Ingrese el valor del lado del cuadrado:"))  
print("La superficie del cuadrado es",retornar_superficie(va))
```

La función `retornar_superficie` devuelve un entero y se lo pasamos a la función `print` para que lo muestre.

Problema 2:

Confeccionar una función que le enviemos como parámetros dos enteros y nos retorne el mayor.

```
def retornar_mayor(v1,v2):  
    if v1>v2:  
        mayor=v1  
    else:  
        mayor=v2  
    return mayor  
  
# bloque principal  
valor1=int(input("Ingrese el primer valor:"))  
valor2=int(input("Ingrese el segundo valor:"))  
print(retornar_mayor(valor1,valor2))
```

Nuevamente tenemos una función que recibe dos parámetros y retorna el mayor de ellos:

```
def retornar_mayor(v1,v2):  
    if v1>v2:  
        mayor=v1  
    else:  
        mayor=v2  
    return mayor
```

Si queremos podemos hacerla más sintética a esta función sin tener que guardar en una variable local el valor a retornar:

```
def retornar_mayor(v1,v2):  
    if v1>v2:  
        return v1  
    else:  
        return v2
```

Cuando una función encuentra la palabra `return` no sigue ejecutando el resto de la función, sino que sale a la línea del programa desde donde llamamos a dicha función.

Problema 3:

Confeccionar una función que le enviemos como parámetro un string y nos retorne la cantidad de caracteres que tiene. En el bloque principal solicitar la carga de dos nombres por teclado y llamar a la función dos veces. Imprimir en el bloque principal cuál de las dos palabras tiene más caracteres.

```
def largo(cadena):  
    return len(cadena)  
  
# bloque principal  
  
nombre1=input("Ingrese primer nombre:")  
nombre2=input("Ingrese segundo nombre:")  
la1=largo(nombre1)  
la2=largo(nombre2)  
if la1==la2:  
    print("Los nombres:",nombre1,nombre2,"tienen la misma cantidad de caracteres")  
else:  
    if la1>la2:  
        print(nombre1,"es mas largo")  
    else:  
        print(nombre2,"es mas largo")
```

Hemos definido una función llamada `largo` que recibe un parámetro llamado `cadena` y retorna la cantidad de caracteres que tiene dicha cadena (utilizamos la función `len` para obtener la cantidad de caracteres)

El lenguaje Python ya tiene una función que retorna la cantidad de caracteres de un string: `len`, nosotros hemos creado una que hace lo mismo, pero tiene un nombre en castellano: `largo`.

Desde el bloque principal de nuestro programa llamamos a la función `largo` pasando primero un string y guardando en una variable el entero devuelto:

```
la1=largo(nombre1)
```

En forma similar calculamos el largo del segundo nombre:

```
la2=largo(nombre2)
```

Solo nos queda analizar cuál de las dos variables tiene un valor mayor para indicar cual tiene más caracteres:

```
if la1==la2:  
    print("Los nombres:",nombre1,nombre2,"tienen la misma cantidad de caracteres")  
else:  
    if la1>la2:  
        print(nombre1,"es mas largo")  
    else:  
        print(nombre2,"es mas largo")
```