

05

UNIDAD

P1

Programación 1

FUNCIONES (PRIMERA PARTE)



- | Programación estructurada
- | Definición de funciones
- | Parámetros y retorno de datos
- | Función Return



- Al final de esta clase podrás
desenvolverte fluidamente con:
- | La programación estructurada
en funciones.
 - | El desarrollo de funciones.

INTRODUCCIÓN A LA CLASE

Hasta hace un tiempo, cuando hablábamos de “**Programación**” nos imaginábamos a una lejana persona trabajando solitaria en su computadora. Hoy nos encontramos con una extraordinaria **capacidad de crear programas** y aplicaciones **que solucionan** un sinnúmero de **problemas cotidianos** en nuestras manos.

En esta nueva clase aprenderemos acerca de la programación estructurada en funciones, logrando que nuestros programas empiecen a poder solucionar problemas más complejos y extensos sin perder el orden ni la prolijidad

La clase de hoy se estructura en tres temas:

| En el primero será la **programación estructurada**.

| En el segundo, trabajaremos sobre **los parámetros** de las funciones y su uso.

| Y finalmente, trabajaremos sobre el **retorno de datos en las funciones**.

¿Comenzamos?

Antes de comenzar con la clase, leé el siguiente artículo:

DRONES AL RESCATE... EN LA PLAYA

Nuevas tecnologías. Por primera vez se los puso en práctica en Francia. En 30 segundos llegan hasta el bañista en peligro y le sueltan un salvavidas.

En menos de 30 segundos despegando del puesto de socorro, se posiciona sobre el bañista en peligro y le suelta un salvavidas: en las playas de Biscarrosse (suroeste de Francia) se está probando el drone Helper para rescates en el mar.

Esta innovación, probada por primera vez en Europa a partir de este miércoles, es el resultado del encuentro entre Fabien Farge, médico de emergencia de esa ciudad balnearia y de Gérald Dumartin, al frente de Terra Drone, una empresa local especializada en cartografía con drones.

"Desde hace 15 años vi trabajar a los socorristas. Son verdaderos atletas, pero no se puede ir más allá de lo que permite la máquina del cuerpo humano. Hace dos años que estoy pensando en eso, cómo asociar un dron a un salvavidas", explica a la AFP Fabien Farge.

Primero realizaron pruebas con un drone común pero la experiencia fue poco concluyente. "Necesitábamos un drone capaz de resistir grandes variaciones de viento y con gran estabilidad", explica Farge. Luego decidieron concebir un drone específico y se asociaron con la empresa Mywebteam, especializada en informática y objetos conectados. El resultado es un aparato de 3,9 kg con una cámara de alta definición para visualizar la persona en peligro, un salvavidas capaz de inflarse automáticamente al entrar en contacto con el agua y un sistema para soltarlo.

Para los que lo concibieron, las ventajas del drone Helper (*Human Environment and Life Protection Emergency Response*) son numerosos. Tiene una velocidad de desplazamiento de entre 55 y 80 km/hora e indica rápidamente a la víctima que ha sido tomada en cuenta, lo que evita que entre en pánico. Una cámara permite ver en qué estado se encuentra para decidir qué medidas hay que adoptar. Además el uso del drone como guía visual permite orientar a los socorristas que salen a buscarlo.

"El drone también permite despejar dudas sobre una presunta víctima y evitar poner en peligro la

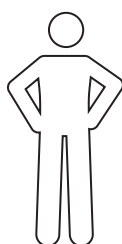
vida del socorrista", destaca Fabien Farge. Los inventores destacan que el drone se puede usar con condiciones meteorológicas poco favorables, vientos de hasta 50 km/hora y un oleaje que justifique bandera amarilla para el baño.

Durante una demostración, el aparato llegó en apenas 22 segundos a la altura de uno de sus colegas que hacía el papel de bañista en dificultades, a 100 metros de la orilla.

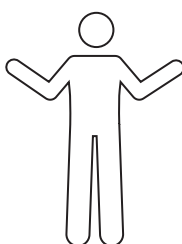
Salido a nado desde la playa en el mismo instante, un socorrista aún se encuentra a unos cincuenta metros del "ahogado" cuando el drone larga el salvavidas. En cuanto al jet-ski, herramienta indispensable para los socorristas, tiene que ser llevado hasta el agua, domar las olas y abrirse paso con precaución entre los bañistas.

Clarín – Julio de 2016

Los robots son dispositivos capaces de actuar de manera autónoma o semiautónoma, con la capacidad de adquirir datos del ambiente a partir de sensores y de tomar decisiones en consecuencia, de acuerdo a las instrucciones previstas por un programador.



Uno de los dispositivos que más terreno ha ganado en los últimos años, es el dron. Estos aparatos han crecido en masividad y consecuentemente, disminuido su costo, convirtiéndose de esta manera en asequibles para el gran público.



No es extraño leer noticias como la precedente, en las que se describen situaciones de nuevos usos o aplicaciones para esta tecnología.



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**.

FUNCIONES

Se define a una Función o Método, como una fracción de programa que cumple una misión específica. Generalmente el término Función es utilizado en la Programación Estructurada y el término Método, es utilizado en la Programación Orientada a Objetos (POO).

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; pero 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.**

EJEMPLO 49

Introducción a la programación estructurada

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.

| 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("*****")
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()
```

Como habrás observado, 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()
```

Recordá que el nombre de la función no puede tener espacios en blanco, ni comenzar con un número y el único carácter especial permitido es “_”

La sintaxis para declarar una función es mediante la palabra clave **def** seguida por el **nombre de la función**

| Luego del nombre de la función deben ir los datos que llegan entre paréntesis; si no llegan datos como es el caso de nuestras tres funciones, solo se disponen paréntesis abierto y cerrado. "()"

| Al final, dispondremos los dos puntos para concluir la sentencia. ":"

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()
```

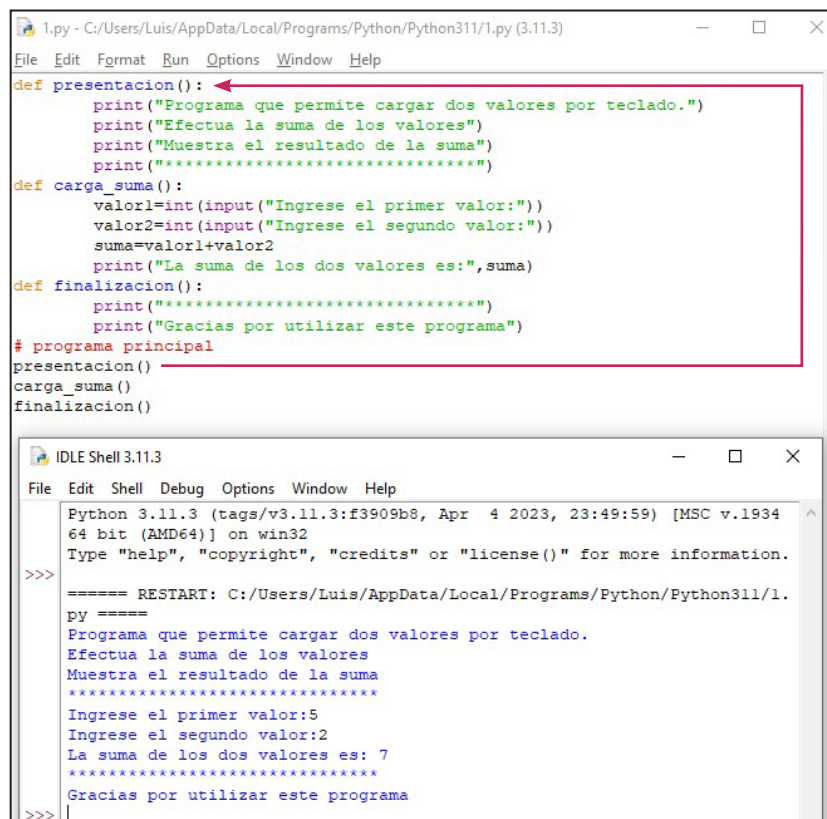
Si no hacemos las llamadas a las funciones los algoritmos que implementan las funciones nunca se ejecutarán.

| Cuando en el bloque del programa principal se llama una función hasta que no finalice no continúa 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.
```

Vizualizá la devolución del código en pantalla. [1]

1



The screenshot shows a Python IDE window titled '1.py - C:/Users/Luis/AppData/Local/Programs/Python/Python311/1.py (3.11.3)'. The code defines three functions: `presentacion()`, `carga_suma()`, and `finalizacion()`. The `presentacion()` function prints a menu, `carga_suma()` takes two inputs and prints their sum, and `finalizacion()` prints a thank you message. Below the code, the 'IDLE Shell 3.11.3' window shows the execution output, which matches the code's print statements, confirming the functions were called in the order they were defined.

```
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()
```

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Luis/AppData/Local/Programs/Python/Python311/1.
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
>>>
```

IMPORTANTE

El orden en que llamamos a las funciones es muy importante.

Spongamos, por ejemplo, que en nuestro bloque del programa principal llamamos las funciones en el siguiente orden:

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

Si ejecutáramos el programa anterior, 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 se que en principio al ser un problema tan sencillo y que venimos de resolver muchos programas con la metodología de programación lineal nos parecer mas fácil implementar este programa con la sintaxis:

```
print("Programa que permite cargar dos valores por teclado.")
print("Efectua 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")

```

Al principio, nos saldría más fácil programarlo como lo hicimos anteriormente, pero este método nos ayudará en la limpieza posterior del código. Así que *tengamos paciencia para aprender la nueva sintaxis de dividir un programa en funciones.*

EJEMPLO 50

5 sumas de enteros

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. [2]

| Programa:

```

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()

```

| 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:

2

```

ejercicio 48.py - F:\01 - Corrección de módulos\AED1\AED1) 03 - Correc. Empaquetados\08_...
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("*****")

IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help

Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: F:\01 - Corrección de módulos\AED1\AED1) 03 - Correc. Empaquetad
os\08_ AED1_ ASC_VersiónDigital\ejercicio 48.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
*****

```

```

# 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)**



Desempeño 62

- Desarrollar un programa con dos funciones.

La primera solicite el ingreso de un entero y muestre el cuadrado de dicho valor. La segunda que solicite la carga de dos valores y muestre el producto de los mismos. Llamar desde el bloque del programa principal a ambas funciones.



Desempeño 63

- Desarrollar un programa que solicite la carga de tres valores y muestre el menor. Desde el bloque principal del programa llamar 2 veces a dicha función (sin utilizar una estructura repetitiva)
- programa principal a ambas funciones.

FUNCIONES: PARÁMETROS

Vimos en el concepto anterior que **una función resuelve una parte de nuestro algoritmo.**

| Tenemos por un lado la declaración de una 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.**

EJEMPLO 51

Suma de enteros con despedida

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.

| Programa:

```
def mostrar_mensaje(mensaje):
    prit("*****")
    print(mensaje)
    prit("*****")
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 ingre-
sados por teclado.")
carga_suma()
mostrar_mensaje("Gracias por utilizar este programa")
```

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):
    prit("*****")
    print(mensaje)
    prit("*****")
```

Podemos imaginar a un parámetro, como una variable que solo se puede utilizar dentro de la función y el valor se carga cuando se la llama.

| 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.

EJEMPLO 52

Identificador de enteros

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

| Programa:

```
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("Ingresa el primer valor:"))
    valor2=int(input("Ingresa el segundo valor:"))
    valor3=int(input("Ingresa 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 variables 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.

EJEMPLO 53

Calculador de perímetro y superficie para un cuadrado

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

| Programa:

```
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 perímetro es",per)
def mostrar_superficie(lado):
    sup=lado*lado
    print("La superficie es",sup)
```

| La tercer 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 ingreso la variable respuesta procedemos a llamar a la función que efectúa el calculo 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.



Desempeño 64

- Desarrollar una función que reciba un string como parámetro y nos muestre la cantidad de vocales. Llamar a la función desde el bloque principal del programa 3 veces con string distintos.



Desempeño 65

- Confeccionar una función que reciba tres enteros y los muestre ordenados de menor a mayor. En otra función solicitar la carga de 3 enteros por teclado y proceder a llamar a la primer función definida.

RETORNO DE DATOS EN FUNCIONES

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

También observamos que podemos definir una función mediante un nombre y que ésta puede recibir datos por medio de sus parámetros.

Los parámetros son la forma para que una función reciba datos para ser procesados.

A continuación 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). Esto quedará mas claro a través del siguiente ejemplo.

EJEMPLO 54

Calculador de superficie de cuadrado

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

| Programa:

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

Aquí aparece una nueva palabra clave en Python para indicar el valor devuelto: **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
```

Tené 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*.

| Estaríamos cometiendo un **error lógico** al llamar a la función **retornar_superficie** y no asignar el valor a una variable:

```
retornar_superficie(va)
```

| En nuestro caso, el dato devuelto (la superficie del cuadrado) no se almacena. Pero sí 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))
```

| Por ultimo, la función **retornar_superficie** devuelve un entero y se lo pasamos a la función **print** para que lo muestre.

EJEMPLO 55

Identificador de número mayor

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

| Programa:

```
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*.

EJEMPLO 56

Contador de caracteres y muestra del mas largo

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 cual de las dos palabras tiene más caracteres.

| Programa:

```
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)

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)
```

| Calcularemos el largo del segundo nombre de forma similar:

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

| Solo nos queda analizar cual 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")
```

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.



Desempeño 66

- Elaborar una función que reciba tres enteros y nos retorne el valor promedio de los mismos



Desempeño 67

- Elaborar una función que nos retorne el perímetro de un cuadrado pasando como parámetros el valor de un lado.



Desempeño 68

- Confeccionar una función que calcule la superficie de un rectángulo y la retorne, la función recibe como parámetros los valores de dos lados distintos:

```
def retornar_superficie(lado1,lado2):
```

En el bloque principal del programa cargar los lados de dos rectángulos y luego mostrar cuál de los dos tiene una mayor superficie.



Desempeño 69

- Plantear una función que reciba un string en mayúsculas o minúsculas y retorne la cantidad de letras 'a' o 'A'

Esta clase hemos podido adentrarnos en la programación estructurada por medio de las funciones, logrando un código mucho más limpio y prolijo.

En la próxima clase vamos a unificar el uso de la programación estructurada y el uso de estructuras de datos de tipo lista por medio de funciones que faciliten nuestro flujo de trabajo.

¡Nos vemos en la novena clase!

Si detectás un error del tipo que fuere (falta un punto, un acento, una palabra mal escrita, un error en código, etc.), por favor comunicate con nosotros a correcciones@issd.edu.ar e indicanos por cada error que detectes la página y el párrafo.

MUCHAS GRACIAS POR TU APOORTE.

SOLUCIONES A LOS DESEMPEÑOS DEL MÓDULO

Te recomendamos utilizar esta sección luego de haber intentado por un largo tiempo la resolución y también para verificar tus soluciones.

Solución al desempeño 62

| Programa:

```
def calcular_cuadrado():
    valor=int(input("Ingrese un entero:"))
    cuadrado=valor*valor
    print("El cuadrado es",cuadrado)
def calcular_producto():
    valor1=int(input("Ingrese primer valor:"))
    valor2=int(input("Ingrese segundo valor:"))
    producto=valor1*valor2
    print("El producto de los valores es:",producto)
# bloque principal
calcular_cuadrado()
calcular_producto()
```

Solución al desempeño 63

| Programa:

```
def menor_valor():
    valor1=int(input("Ingrese primer valor:"))
    valor2=int(input("Ingrese segundo valor:"))
    valor3=int(input("Ingrese tercer valor:"))
    print("Menor de los tres")
    if valor1<valor2 and valor1<valor3:
        print(valor1)
    else:
        if valor2<valor3:
            print(valor2)
        else:
            print(valor3)
# bloque principal
menor_valor()
menor_valor()
```

Solución al desempeño 64

| Programa:

```
def cantidad_vocales(cadena):
    cant=0
    for x in range(len(cadena)):
        if cadena[x]=="a" or cadena[x]=="e" or cadena[x]=="i" or
        cadena[x]=="o" or cadena[x]=="u":
            cant=cant+1
    print("Cantidad de vocales de la palabra",cadena,"es",cant)
# bloque principal
cantidad_vocales("hola")
cantidad_vocales("administracion")
cantidad_vocales("correr")
```

Solución al desempeño 65

| Programa:

```
def ordenar_imprimir(v1,v2,v3):
    if v1<v2 and v1<v3:
        if (v2<v3):
            print(v1,v2,v3)
        else:
            print(v1,v3,v2)
    else:
        if (v2<v3):
            if (v1<v3):
                print(v2,v1,v3)
            else:
                print(v2,v3,v1)
        else:
            if (v1<v2):
                print(v3,v1,v2)
            else:
                vprint(v3,v2,v1)
```

```
def cargar():
    num1=int(input("Ingrese primer valor:"))
    num2=int(input("Ingrese segundo valor:"))
    num3=int(input("Ingrese tercer valor:"))
    ordenar_imprimir(num1,num2,num3)
# bloque principal
cargar()
```

Solución al desempeño 66

| Programa:

```
def retornar_promedio(v1,v2,v3):
    promedio=(v1+v2+v3)//3
    return promedio
# bloque principal
valor1=int(input("Ingrese primer valor:"))
valor2=int(input("Ingrese segundo valor:"))
valor3=int(input("Ingrese tercer valor:"))
print("Valor promedio de los tres numeros",
retornar_promedio(valor1,valor2,valor3))
```

Solución al desempeño 67

| Programa:

```
def retornar_perimetro(lado):
    perimetro=lado*4
    return perimetro
# bloque principal
lado=int(input("Lado del cuadrado:"))
print("El perimetro es:",retornar_perimetro(lado))
```

Solución al desempeño 68

| Programa:

```
def retornar_superficie(lado1,lado2):
    superficie=lado1*lado2
    return superficie
# bloque principal
print("Primer rectangulo")
lado1=int(input("Ingrese lado menor del rectangulo:"))
lado2=int(input("Ingrese lado mayor del rectangulo:"))
print("Segundo rectangulo")
lado3=int(input("Ingrese lado menor del rectangulo:"))
lado4=int(input("Ingrese lado mayor del rectangulo:"))
if retornar_superficie(lado1,lado2)==retornar_superficie(lado3,lado4):
    print("Los dos rectangulos tiene la misma superficie")
else:
    if retornar_superficie(lado1,lado2)>retornar_superficie(lado3,lado4):
        print("El primer rectangulo tiene una superficie mayor")
    else:
        print("El segundo rectangulo tiene una superficie mayor")
```

Solución al desempeño 69

| Programa:

```
def cantidad_vocal_a(palabra):  
    cant=0  
    for x in range(len(palabra)):  
        if palabra[x]=='a' or palabra[x]=='A':  
            cant=cant+1  
    return cant  
# bloque principal  
palabra=input("Ingrese una palabra:")  
print("La palabra",palabra,"tiene",cantidad_vocal_a(palabra),"a")
```