Introducción al Pensamiento Lógico
Programación 1

Unidad Nº 2

Contenido

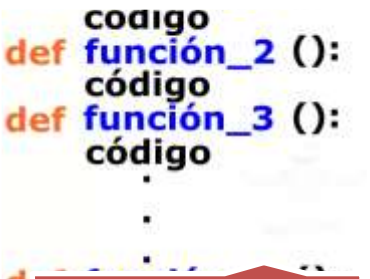
Objetivos de la Unidad

- Que el estudiante comprenda el concepto de funciones y las aplique.

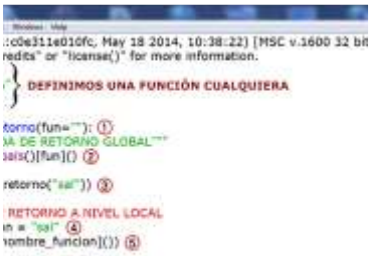
Contenidos de la unidad

Funciones: concepto

Usos



Sintaxis de una función



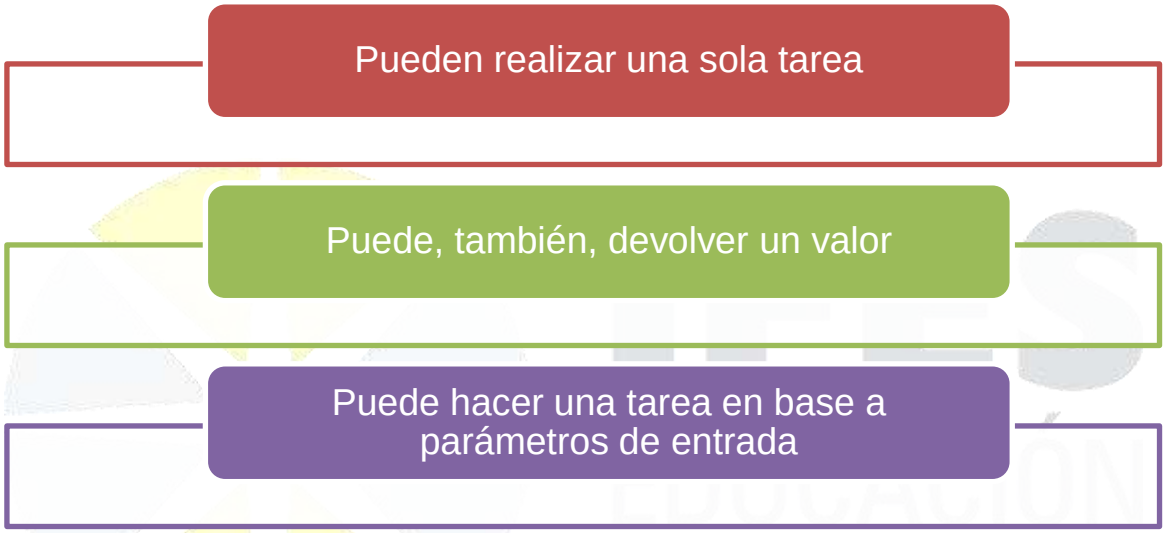
Argumentos, parámetros y retorno

Funciones

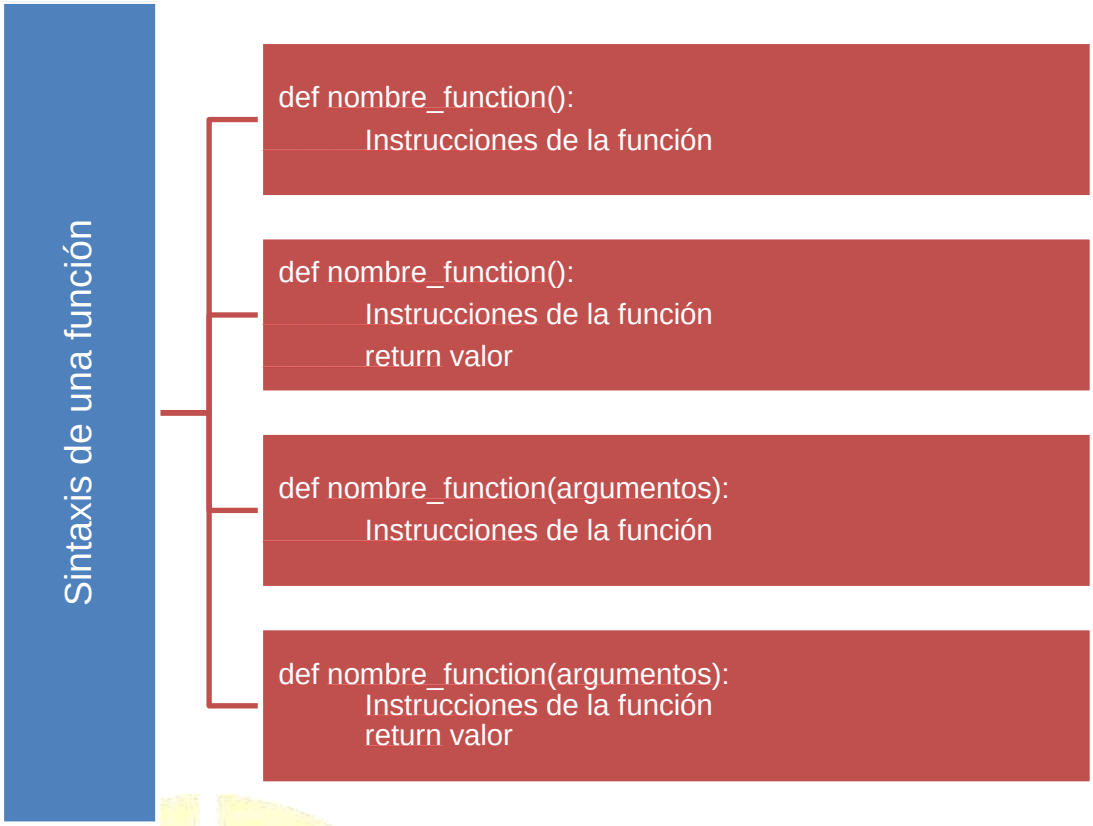
Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como salida

Podemos decir que una función es un conjunto de instrucciones agrupadas que funcionan como una unidad llevando a cabo una tarea especifica.

¿Para que sirven? Para reutilizar código de programación sin la necesidad de escribirlo cada vez que lo necesitamos. También para ordenar el código de programación cuando éste es muy extenso.



Anteriormente hemos usado funciones nativas que vienen con Python como print() e input() también podemos definir nuestras propias funciones. Para ello hacemos uso de def.



Funciones que ya conocemos

print()	**función sin argumento y sin retorno de valor**
print("Hola Mundo")	función con argumento y sin retorno de valor
Nombre=input("Ingrese un nombre: ")	función con argumento y con retorno de valor

```
def nombre_funcion(argumentos):
    código
    return retorno
```

Algo que diferencia en cierto modo las funciones en el mundo de la programación, es que no sólo realizan una operación con sus entradas, sino que también parten de los siguientes principios:

- El principio de reusabilidad, que nos dice que si por ejemplo tenemos un fragmento de código usado en muchos sitios, la mejor solución sería pasarlo a

una función. Esto nos evitaría tener código repetido, y que modificarlo fuera más fácil, ya que bastaría con cambiar la función una vez.

- Y el principio de modularidad, que defiende que en vez de escribir largos trozos de código, es mejor crear módulos o funciones que agrupen ciertos fragmentos de código en funcionalidades específicas, haciendo que el código resultante sea más fácil de leer.

Pasando argumentos de entrada

Empecemos por la función más sencilla de todas. Una función sin parámetros de entrada ni parámetros de salida.

```
def di_hola():
    print("Hola")
```

Acá hemos declarado o definido la función. El siguiente paso es llamarla con di_hola(). Si lo realizamos veremos que se imprime Hola.

```
di_hola() # Hola
```

Quedando el código así:

```
def di_hola():
    print("Hola")
di_hola() # aca la llamamos para que se ejecute
```

Vamos a complicar un poco las cosas pasando un argumento de entrada. Ahora si pasamos como entrada un nombre, se imprimirá Hola y el nombre.

```
def di_hola(nombre):
    print("Hola", nombre)

di_hola("Laura") #pasamos el parámetro Laura
# Hola Laura
```

Python permite pasar argumentos también de otras formas. A continuación las explicamos todas.

Argumentos por posición

Los argumentos por posición o **posicionales** son la forma más básica e intuitiva de pasar parámetros. Si tenemos una función `resta()` que acepta dos parámetros, se puede llamar como se muestra a continuación.

```
def resta(a, b):
    return a-b
resta(5, 3) # 2
```

Al tratarse de parámetros posicionales, se interpretará que el primer número es la a y el segundo la b. El número de parámetros es fijo, por lo que si intentamos llamar a la función con solo uno, dará error.

```
#resta(1) # Error! TypeError
```

Tampoco es posible usar mas argumentos de los tiene la función definidos, ya que no sabría que hacer con ellos. Por lo tanto si lo intentamos, Python nos dirá que toma 2 posicionales y estamos pasando 3, lo que no es posible.

```
#TypeError: resta() takes 2 positional arguments but 3 were given
#resta(5,4,3) # Error
```

Argumentos por nombre

Otra forma de llamar a una función, es usando el nombre del argumento con `=` y su valor. El siguiente código hace lo mismo que el código anterior, con la diferencia de que los argumentos no son posicionales.

```
resta(a=3, b=5) # -2
```

Al indicar en la llamada a la función el nombre de la variable y el valor, el orden ya no importa, y se podría llamar de la siguiente forma.

```
resta(b=5, a=3) # -2
```

Como es de esperar, si indicamos un argumento que no ha sido definido como parámetro de entrada, también se generará un error.

```
#resta() got an unexpected keyword argument 'c'
#resta(b=5, c=3) # Error!
```

Argumentos por defecto

¿Qué pasaría si se quiere tener una función con un parámetro opcional, que pueda ser usado o no dependiendo de diferentes circunstancias? Lo que se puede hacer es asignar un valor por defecto a la función. En el siguiente caso c valdría cero salvo que se indique lo contrario.

```
def suma(a, b, c=0):
    return a+b+c
suma(5,5,3) # 13
```

Dado que el parámetro `c` tiene un valor por defecto, la función puede ser llamada sin ese valor.

```
suma(4,3) # 7
```

Se puede incluso asignar un valor por defecto a todos los parámetros, por lo que se podría llamar a la función sin ningún argumento de entrada.

```
def suma(a=3, b=5, c=0):
    return a+b+c
suma() # 8
```

Las siguientes llamadas a la función también son válidas

```
suma(1)     # 1
suma(4,5)   # 9
suma(5,3,2) # 10
```

O haciendo uso de los nombres de los argumentos.

```
suma(a=5, b=3) #8
```

Sentencia return

El uso de la sentencia return permite realizar dos cosas:

- Salir de la función y transferir la ejecución de vuelta a donde se realizó la llamada.
- Devolver uno o varios parámetros, fruto de la ejecución de la función.

En lo relativo a lo primero, una vez se llama a return se para la ejecución de la función y se vuelve o retorna al punto donde fue llamada. Es por ello por lo que el código que va después del return no es ejecutado en el siguiente ejemplo.

```
def mi_funcion():
    print("Entra en mi_funcion")
    return
    print("No llega")
mi_funcion() # Entra en mi_funcion
```

Por ello, sólo llamamos a return una vez hemos acabado de hacer lo que teníamos que hacer en la función.

Por otro lado, se pueden devolver parámetros. Normalmente las funciones son llamadas para realizar unos cálculos en base a una entrada, por lo que es interesante poder devolver ese resultado a quien llamó a la función.

```
def di_hola():
    return "Hola"
di_hola()
# 'Hola'
```

También es posible devolver mas de una variable, separadas por , . En el siguiente ejemplo tenemos una función que calcula la suma y media de tres números, y devuelve su resultado.

```
def suma_y_media(a, b, c):
    suma = a+b+c
    media = suma/3
    return suma, media
suma, media = suma_y_media(9, 6, 3)
print(suma)  # 18
print(media) # 6.0
```

Documentación

Ahora que ya tenemos nuestras propias funciones creadas, tal vez alguien se interese en ellas y podamos compartírselas. Las funciones pueden ser muy complejas, y leer código ajeno no es tarea fácil. Es por ello por lo que es importante documentar las funciones. Es decir, añadir comentarios para indicar como deben ser usadas.

```
def mi_funcion_suma(a, b):
    """
    Descripción de la función. Como debe ser usada,
    que parámetros acepta y que devuelve
    """
    return a+b
```

Para ello debemos usar la triple comilla """ al principio de la función. Se trata de una especie de comentario que podemos usar para indicar como la función debe ser usada. No se trata de código, es un simple comentario un tanto especial, conocido como docstring.

Ahora cualquier persona que tenga nuestra función, podrá llamar a la función help() y obtener la ayuda de como debe ser usada.

```
help(mi_funcion_suma)
```

Otra forma de acceder a la documentación es la siguiente.

```
print(mi_funcion_suma.__doc__)
```

Códigos de Ejemplo

```
import os
os.system("cls")

def saludoAlumnos():
    print ("hola como estan")
    print ("espero que esten bien")
    print ("y con ganas de aprender")
    print ("este lenguaje brillante que se llama python")


saludoAlumnos()
saludoAlumnos()
saludoAlumnos()
```

```
import os
os.system("cls")

def promedioNotas():
    Nota1=9
    Nota2=9
    Nota3=9
    promedio=(Nota1+Nota2+Nota3)/3
    print (f"El promedio es: {promedio}")

promedioNotas()
```

```
import os

def promedioNotas(Nota1,Nota2,Nota3):
    promedio=(Nota1+Nota2+Nota3)/3
    print (f"El promedio es: {promedio}")

promedioNotas(10,9,10)
promedioNotas(7,7,7)
promedioNotas(9,7,10)
```

```
import os
os.system("cls")

def promedioNotas(Nota1,Nota2,Nota3):
    promedio=(Nota1+Nota2+Nota3)/3
    return promedio

promedio1=promedioNotas(9,7,8)
promedio2=promedioNotas(7,7,7)
promedio3=promedioNotas(9,7,10)

print (promedio1)
print (promedio2)
print (promedio3)
```

```
import os
os.system("cls")

def promedioNotas(Nota1,Nota2,Nota3):
    promedio=(Nota1+Nota2+Nota3)/3
    return promedio

print("Calculo de promedio de notas de alumnos")
print()
Nombre1=input("Ingrese el nombre del alumno: ")
Nota1=int(input("Ingrese la 1er nota del alumno: "))
Nota2=int(input("Ingrese la 2da nota del alumno: "))
Nota3=int(input("Ingrese la 3er nota del alumno: "))
Promedio1=promedioNotas(Nota1,Nota2,Nota3)
print()
Nombre2=input("Ingrese el nombre del alumno: ")
Nota1=int(input("Ingrese la 1er nota del alumno: "))
Nota2=int(input("Ingrese la 2da nota del alumno: "))
Nota3=int(input("Ingrese la 3er nota del alumno: "))
Promedio2=promedioNotas(Nota1,Nota2,Nota3)
print()
Nombre3=input("Ingrese el nombre del alumno: ")
Nota1=int(input("Ingrese la 1er nota del alumno: "))
Nota2=int(input("Ingrese la 2da nota del alumno: "))
Nota3=int(input("Ingrese la 3er nota del alumno: "))
Promedio3=promedioNotas(Nota1,Nota2,Nota3)
print()
print ("El promedio de "+Nombre1+f" es {Promedio1}")
print ("El promedio de "+Nombre2+f" es {Promedio2}")
print ("El promedio de "+Nombre3+f" es {Promedio3}")
```

```
import os
os.system("cls")

def areac(lado):
    ar=lado*lado
    return ar

def peri(lado):
    p=4*lado
    return p

lado=int(input("ingrese lado del cuadrado: "))
area=areac(lado)
perim=peri(lado)
print(f"El area es: {area}")
print(f"El perimetro es: {perim}")
```

```
import os
os.system("cls")
#Los lados de un triángulo equilátero miden 4 cm. Calcular su área y perímetro. Mostrar datos y resultados
from math import sqrt

Lados=4
Mitadtriangulo=2
Perimetro=(Lados*3)
H=sqrt(Lados**2-Mitadtriangulo**2)
Area=(Lados*H) / 2

print("La altura es:",H)
print("El perimetro es:",Perimetro)
print("El Area es:",Area)
```

Resumen de la unidad

Una función en Python (y en cualquier otro lenguaje de programación) es un bloque de líneas de código o un conjunto de instrucciones cuya finalidad es realizar una tarea específica. Puede reutilizarse a voluntad para repetir dicha tarea. Las funciones nos ayudan a que el código sea más fácil de leer y entender.

Una función de Python está compuesta por distintos elementos, como los argumentos de entrada, el código a ejecutar sobre esos argumentos, y los parámetros de salida.

Para llamar una función en Python es necesario realizar una llamada a la misma, aludiendo al nombre que se asignó en su definición (con el comando def)

En Python las funciones se definen usando la palabra reservada def y luego el nombre de la función con paréntesis y dos puntos que indican que lo que sigue son las sentencias, eventualmente una función debe retornar un valor, para esto se usa la palabra reservada return.

Códigos de funciones de ejemplo

```
# ejemplo de definición de una función
def suma(a, b):
  return a + b
result = suma(8,5)
# result = 13
```

```
def suma(a, b=8):
  return a + b
result = suma(5)
# result = 13
```

Bibliografía de la unidad

- Programación y algoritmos – Editorial MP Ediciones
- Sitios web.