Módulo I – Técnicas de Programación

Python. Funciones.

Añadiendo funciones nuevas

Hasta ahora, solo hemos estado usando las funciones que vienen incorporadas en Python, pero es posible añadir también funciones nuevas.

Una definición de función especifica el nombre de una función nueva y la secuencia de sentencias que se ejecutan cuando esa función es llamada o invocada en nuestro programa.

Una finalizada la ejecución de esas sentencias, la función retorna un resultado.

Una vez definida una función, se puede reutilizar una y otra vez a lo largo de todo el programa u otros programas.

Añadiendo funciones nuevas

Ejemplo:

```
def saludar ():
    print ("Hola Bienvenido al Curso de Programador")

saludar ()
Hola Bienvenido al Curso de Programador
```

Añadiendo funciones nuevas

def es una palabra clave que indica que se trata de una definición de función. El nombre de la función es ***saludar()***.

Las reglas para los nombres de las funciones son los mismos que para las variables: se pueden usar letras, números y algunos signos de puntuación, pero el primer carácter no puede ser un numero.

Los paréntesis vacíos después del nombre indican que esta función no toma ningún argumento.

Añadiendo funciones nuevas

La primera línea de la definición de la función es llamada la cabecera; el resto se llama el cuerpo. La cabecera debe terminar con dos puntos (:), y el cuerpo debe ir indentado.

La sintaxis para llamar a nuestra nueva función es la misma que usamos para las funciones internas:

```
saludar ()
Hola Bienvenido al Curso de Programador
```

Flujo de ejecución.

La ejecución siempre comienza en la primera sentencia del programa. Las sentencias son ejecutadas una por una, en orden de arriba hacia abajo.

Las *definiciones* de funciones no alteran el flujo de la ejecución del programa, pero recuerda que las sentencias dentro de una función no son ejecutadas hasta que se llama a esa función.

Flujo de ejecución.

Una llamada a una función es como un desvío en el flujo de la ejecución. En vez de pasar a la siguiente sentencia, el flujo salta al cuerpo de la función, ejecuta todas las sentencias que hay allí, y después vuelve al punto donde lo dejó.

Todo esto parece bastante sencillo, hasta que uno recuerda que una función puede llamar a otra.

Parámetros y Argumentos.

Algunas de las funciones internas que hemos visto necesitan argumentos. Por ejemplo, cuando se llama a ***print ()***, la misma recibe un argumento de tipo String.

Algunas funciones necesitan mas de un argumento: por ejemplo, ***pow (base, exponente)*** toma dos argumentos, la base y el exponente para calcular la potencia de un número.

Parámetros y Argumentos.

Dentro de las funciones, los argumentos son asignados a variables llamadas parámetros.

Un parámetro no es mas que una variable utilizada para recibir valores de entrada Ejemplo:

```
def obtenerPorcentaje (numero, porcentaje):
    return numero * porcentaje/100
```

La función porcentaje recibe dos parámetros ***numero*** y un ***porcentaje***. Luego la función retornará un resultado, el ***x porciento*** de un número.

Parámetros y Argumentos.

Ejemplo:

```
from porcentaje import*

print (obtenerPorcentaje (500, 50))
```

Resultado:

```
250.0
```

La sentencia ***from porcentaje import**** indica que queremos importar todas las funciones contenidas en la librería porcentaje creada por el programador.

Las librerías de Python son archivos que contienen un conjunto de funciones que permiten resolver cuestiones específicas y comunes a varios programas.

Parámetros y Argumentos.

Ejemplo:

```
from porcentaje import*
numero = float (input ("Ingresar Valor: "))
porcentaje = float (input ("Ingresar Porcentaje: "))
print (obtenerPorcentaje (numero, porcentaje))
```

Importante: los argumentos de la función numero y porcentaje nada tienen que ver con los parámetros del mismo nombre definidos en la función.

Parámetros y Argumentos.

```
from porcentaje import*
numero = float (input ("Ingresar Valor: "))
porcentaje = float (input ("Ingresar Porcentaje: "))
print (obtenerPorcentaje (numero, porcentaje))
```

```
def obtenerPorcentaje (numero, porcentaje):
    return numero * porcentaje/100
```

Ambos, parámetros y argumentos pueden llamarse iguales o no puesto que la función solo conoce los parámetros definidos en su ámbito y el programa las variables definidos en el suyo.

Parámetros y Argumentos.

```
from porcentaje import*
numero = float (input ("Ingresar Valor: "))
porcentaje = float (input ("Ingresar Porcentaje: "))
print (obtenerPorcentaje (numero, porcentaje))
```

```
def obtenerPorcentaje (numero, porcentaje):
    return numero * porcentaje/100
```

Es como tener dos programas distintos con iguales nombres en sus variables, pero que se desconocnen, son programas independientes.

Parámetros y Argumentos.

```
from porcentaje import*
numero = float (input ("Ingresar Valor: "))
porcentaje = float (input ("Ingresar Porcentaje: "))
print (obtenerPorcentaje (numero, porcentaje))
```

```
def obtenerPorcentaje (numero, porcentaje):
    return numero * porcentaje/100
```

Es como tener dos programas distintos con iguales nombres en sus variables, pero que se desconocnen, son programas independientes.

¿Por que funciones?

- ✓El crear una función nos permite dar nombre a un grupo de sentencias, lo cual hace que el programa sea mas fácil de leer, entender, y depurar.
- ✓Las funciones pueden hacer un programa mas pequeño, al eliminar código repetido. Además, si querés realizar cualquier cambio en el futuro, sólo tenés que hacerlo en un único lugar.
- ✓Dividir un programa largo en funciones te permite depurar las partes de una en una y luego ensamblarlas juntas en una sola pieza.
- ✓Las funciones bien diseñadas a menudo resultan útiles para otros programas. Una vez que escribiste y depuraste una, podés **reutilizarla.**

Bibliografía

- Python para informáticos, Explorando la información, Charles Severance, Agosto 2015.