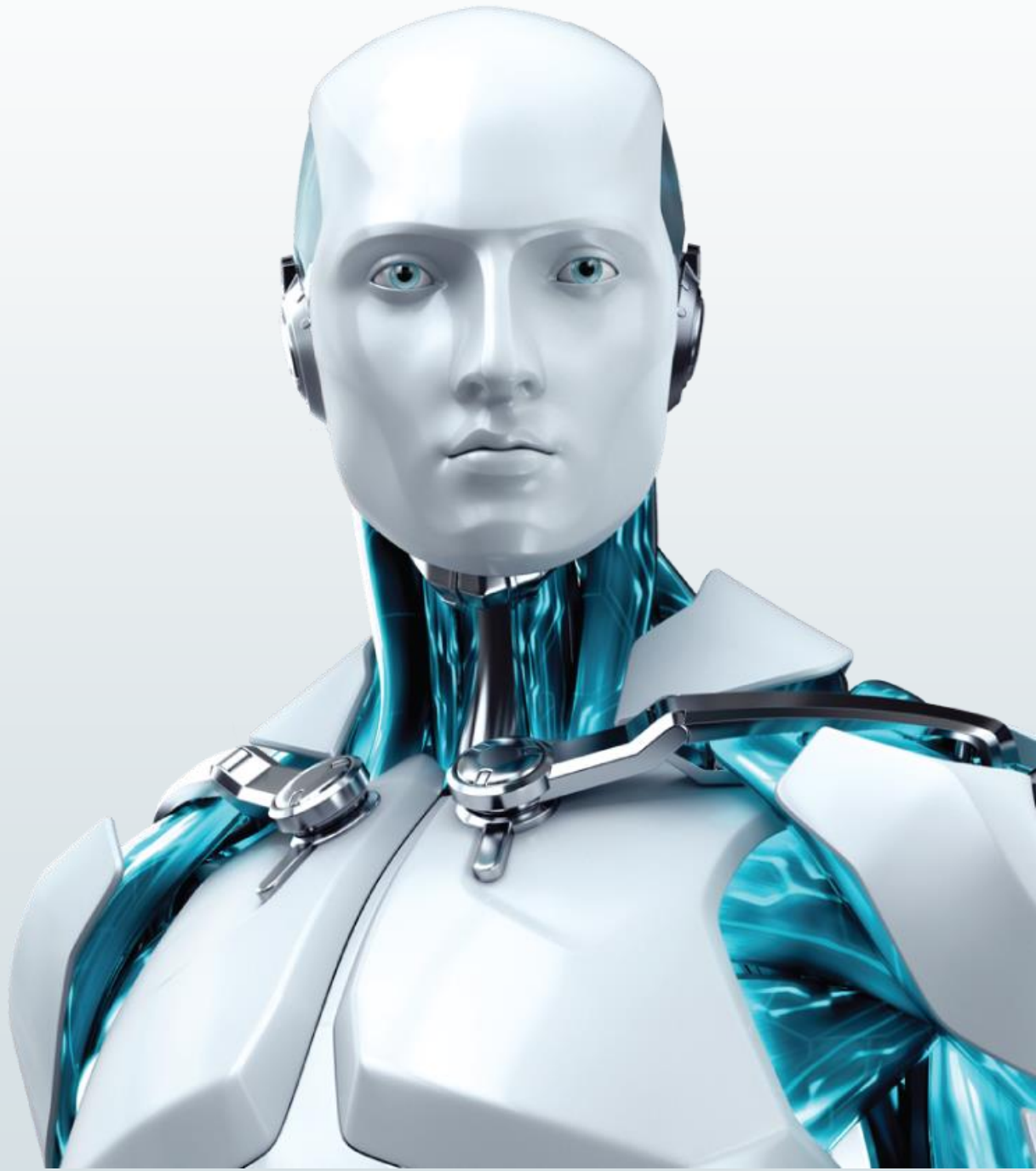


Funciones, Clases y Objetos



Contenido

Funciones	3
¿Qué es una función?	3
Sintaxis.....	3
Argumentos y valores de retorno.....	4
Clases y Objetos.....	4
¿Qué es un objeto?.....	5
¿Qué es una clase?	5
Clases, objetos y métodos	5
Módulos.....	6
¿Qué es un módulo?.....	6
Reutilizando código	7
Conclusión	8

Introducción

Hasta ahora, vimos parte por parte todos los elementos de Python, sus variables, algunas funciones básicas y las estructuras condicionales de código. Al hacerlo, fuimos repasando los conceptos más básicos de la programación y logramos agrupar distintas líneas de código en scripts para desarrollar tareas específicas. Sin embargo, no hemos visto gran parte del potencial de este lenguaje y por sobre todo, de qué manera podemos optimizar su uso para que nuestros códigos y programas sean más y más eficientes.

En el módulo 3 del curso de **Introducción a Python de la Plataforma Educativa de ESET** veremos qué son y para qué sirven las funciones, como así también algunas maneras de agrupar diferentes sentencias de código para que tengan comportamientos específicos según sea el caso. Nos adentraremos en el mundo de los objetos, y con ellos veremos cómo construir programas más eficientes, más robustos y mucho más claros de entender.

Funciones

Al comenzar a programar, nuestro código puede no ser muy extenso, no contar con las grandes funcionalidades o incluso no tener una complejidad muy elevada. A medida que comenzamos a desarrollar y vamos ganando conocimiento y experiencia, nos damos cuenta de que existen una serie de tareas que se repiten una y otra vez en diferentes lugares de nuestros programas.

Cuando necesitamos ejecutar las mismas líneas de código una y otra vez en diferentes lugares de nuestro programa lo mejor que podemos hacer es agruparlas en una función. De esta manera podremos ejecutar esa sección del programa cuando lo necesitemos y en caso de tener que cambiar algo, solo hay que hacerlo en un lugar y no recordar todos los lugares donde escribimos lo mismo.

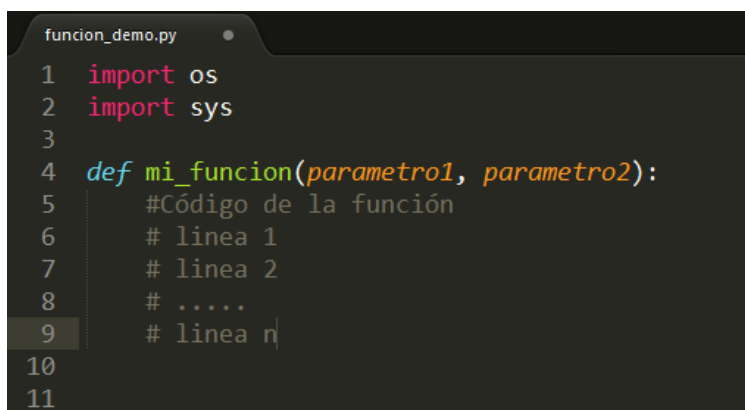
¿Qué es una función?

En pocas palabras una función es un conjunto de instrucciones que se utilizan para un fin específico, como por ejemplo para crear una conexión a un sitio web, escribir en un archivo, realizar una suma o analizar una cadena de texto.

A lo largo de los módulos anteriores, fuimos escribiendo nuestras líneas de código en la consola o en un script, y utilizamos funciones ya existentes, ahora vamos a ver cómo crear nuestras propias funciones.

Sintaxis

La sintaxis para definir una función es la siguiente:



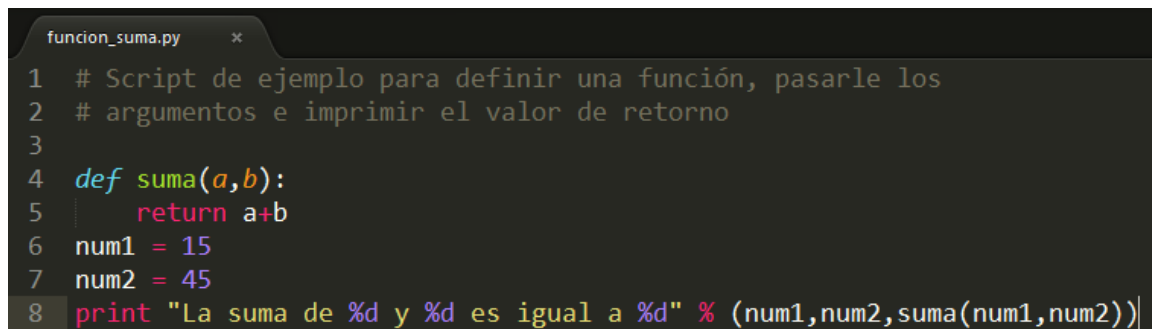
```
funcion_demo.py
1  import os
2  import sys
3
4  def mi_funcion(parametro1, parametro2):
5      #Código de la función
6      # línea 1
7      # línea 2
8      # .....
9      # línea n
10
11
```

Imagen 1 - Ejemplo de una función en Python

Una de las palabras reservadas de Python que se utiliza para especificar el comienzo de una función es **def**, seguido por el nombre de la función más los argumentos que recibe entre paréntesis y separados por comas. Finalmente hay que poner un ":" y en la línea siguiente se escribe el código que corresponde a la función dejando los espacios correspondientes a la indentación, tal como lo hicimos con el **if**, el **for** y el **while**.

Argumentos y valores de retorno

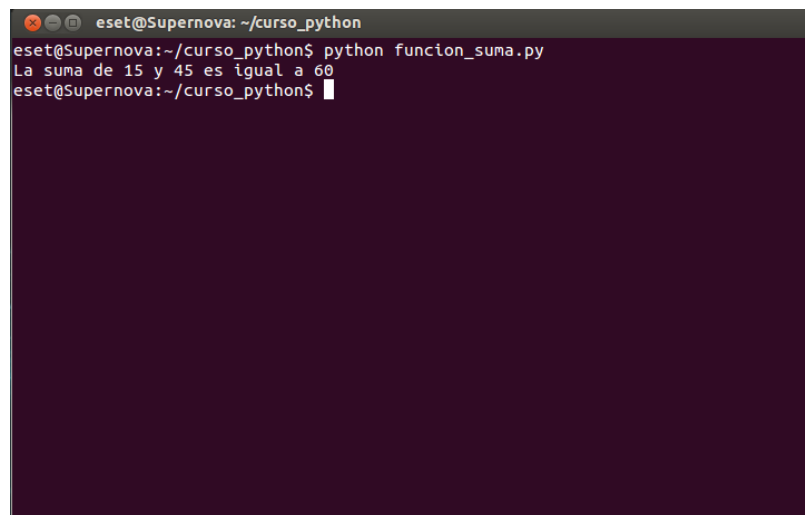
Como vimos en el ejemplo anterior, una función puede recibir valores a través de los argumentos. Además, una función puede retornar un valor u objeto (un poco más sobre esto más adelante en este documento), para ello deberá incluir la palabra **return** como podemos ver en el siguiente ejemplo:



```
funcion_suma.py x
1 # Script de ejemplo para definir una función, pasarle los
2 # argumentos e imprimir el valor de retorno
3
4 def suma(a,b):
5     return a+b
6
7 num1 = 15
8 num2 = 45
9 print "La suma de %d y %d es igual a %d" % (num1,num2,suma(num1,num2))
```

Imagen 2 - Función con dos argumentos y un valor de retorno

Cuando ejecutemos este script desde la consola, la salida será la siguiente:



```
eset@Supernova: ~/curso_python
eset@Supernova:~/curso_python$ python funcion_suma.py
La suma de 15 y 45 es igual a 60
eset@Supernova:~/curso_python$
```

Imagen 3 - Ejecución de funcion_suma.py

Los invito a probar cómo sería escribir una función similar a la que vimos pero para la resta, la multiplicación y la división. En todos los casos que devuelva como valor de retorno el resultado de la operación. Todas las funciones las pueden escribir dentro del mismo archivo .py.

Clases y Objetos

En el mundo de la programación existen una gran cantidad de paradigmas, cada uno de ellos refiere a una manera de pensar sobre cómo resolver un problema. Uno de los paradigmas que más ha crecido durante los últimos años es el de la Programación Orientada a Objetos (POO)¹.

A lo largo del curso hemos interactuado con objetos. Por ejemplo, cuando nosotros hablamos de una cadena de texto (**string**) y llamamos a métodos como `split()`, `upper()`, `lower()` estamos interactuando con objetos. Un objeto tiene un **comportamiento**, un **estado** y una **identidad**. Más adelante definiremos, objetos, clases y métodos más formalmente.

Otro concepto muy importante en relación a la Programación Orientada a Objetos es el concepto de **herencia**². La herencia nos permite crear subclases de una más grande agrupando algunas características generales. En los diferentes lenguajes de

¹ Programación Orientada a Objetos: http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

² Herencia: [http://es.wikipedia.org/wiki/Herencia_\(inform%C3%A1tica\)](http://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica))

programación orientados a objetos existen aquellos que soportan herencia simple o herencia múltiple. En el caso de Python, este lenguaje trabaja con *herencia múltiple*³.

Cuando trabajamos con herencia, lo que sucede es que la clase “hija” herede todos los métodos y atributos de la clase “padre”. Un ejemplo para ver este concepto podría ser en la relación entre los diferentes códigos maliciosos que existen. Digamos que tenemos una Clase en Python que se llama **malware** que tendrá atributos y métodos similares en todos los tipos de amenazas que existen. Dos subclases que podríamos definir serían **troyano** y **gusano**. Los troyanos y gusanos por ejemplo se diferencian por las técnicas de propagación que utilizan y es por ello que cada una podría tener su propia implementación del método propagarse. Para más información sobre la clasificación de los códigos maliciosos y sus tipos les recomendamos visitar [Curso básico sobre códigos maliciosos de [ACADEMIA ESET](#)]

¿Qué es un objeto?

Se conoce como objeto a la instanciación de una clase. Dicho de otra manera, nosotros no programamos objetos, sino que creamos clases con métodos y atributos. Luego lo que hacemos en nuestro código es instanciar los objetos que tendrán el comportamiento de su clase.

¿Qué es una clase?

Entonces, una **clase** es un *template* en donde se definen los atributos y el comportamiento. Todos los objetos que pertenezcan a una misma clase realizarán acciones similares y se diferenciarán según el valor de sus atributos.

Clases, objetos y métodos

En nuestro código definimos las clases, sus métodos y atributos para luego instanciar los objetos que llevaran a cabo las acciones. A modo de representar este concepto, interpreten qué es lo que pasa en el siguiente script:

```

ejemplo_clases_y_objetos.py
1 # Ejemplo para definir una clase, las subclases y probar su comportamiento
2 class malware():
3     def __init__(self,nombre):
4         self.nombre = nombre
5
6     def get_nombre(self):
7         return self.nombre
8
9     def infectar():
10        # Código para infectar un sistema
11        print "Sistema infectado"
12
13 class gusano(malware):
14     def propagarse(self):
15         # Código para propagar la amenaza
16         print "Infectando USB"
17
18 class troyano(malware):
19     def propagarse(self):
20         # Código para propagar la amenaza
21         print "Enviando mail con adjunto malicioso a los contactos"
22
23 ##### Código #####
24
25 amenazas = [gusano("Win32/Dorkbot"), troyano("Win32/Waledac")]
26
27 for amenaza in amenazas:
28     print "Sistema infectado por %s." % amenaza.get_nombre()
29     amenaza.propagarse()

```

Imagen 4 - Clases y objetos

Para entender todos estos conceptos necesitarán un poco de tiempo, y sobretodo mucha práctica. Para programar bien hay que dedicarle tiempo y estudio ya que muchos de los conceptos que vemos aquí son introductorios. Existen otros conceptos como

³ Herencia Múltiple: http://es.wikipedia.org/wiki/Herencia_multiple

polimorfismo⁴ y encapsulamiento⁵ que no trataremos en detalle durante el resto del curso, pero les dejamos enlaces para que puedan profundizar si les interesa.

Módulos

Cuando un programa crece en su tamaño y funcionalidades, tener todo en un solo archivo puede ser más que complicado. Para simplificar esta problemática, los lenguajes de programación como Python nos permiten dividir nuestro código en módulos y paquetes. De esta manera podremos agrupar mejor nuestro código según la relación que existan entre nuestras clases y funciones.

¿Qué es un módulo?

Un módulo es un archivo con extensión `.py` que nosotros creamos y guardamos en nuestra computadora. Los módulos se pueden invocar desde diferentes partes del programa cuando sea necesario y tener una mejor organización de nuestro código.

En el código que se puede observar en la imagen 4, definimos tres clases **malware**, **gusano** y **troyano**, todas dentro del mismo archivo. Ahora, nosotros podemos importar desde otro lugar las clases sin la necesidad de volver a escribir ese mismo código. En otras palabras, agrupar las funciones y clases en diferentes módulos también nos permite reutilizar código de una manera bastante sencilla.

Agrupando funciones, clases y objetos

En esta sección veremos cómo hacer para importar clases y funciones desde otro módulo y lograr que se ejecuten sin ningún tipo de problemas. Para ello vamos a trabajar con dos archivos diferentes, primero el archivo **codigos_maliciosos.py** que contiene las definiciones de las clases y sus métodos, importaremos las clases desde el archivo **ejemplo.py** y veremos cómo se pueden ejecutar.

Para importar un módulo desde otro archivo tenemos dos maneras diferentes de hacerlo. La primera opción la utilizamos anteriormente y es a través del uso de la palabra **import** `<módulo>`, esta opción importará todo el módulo. La segunda es utilizando las palabras **from** `<módulo> import <clases o funciones>`, que a diferencia de la anterior solo importará los elementos que mencionemos.

Malware.py:

⁴ Polimorfismo en la programación orientada a objetos:

[http://es.wikipedia.org/wiki/Polimorfismo_\(programaci%C3%B3n_orientada_a_objetos\)](http://es.wikipedia.org/wiki/Polimorfismo_(programaci%C3%B3n_orientada_a_objetos))

⁵ Encapsulamiento: [http://es.wikipedia.org/wiki/Encapsulamiento_\(inform%C3%A1tica\)](http://es.wikipedia.org/wiki/Encapsulamiento_(inform%C3%A1tica))

```
codigos_maliciosos.py x
1 # Ejemplo para definir una clase, las subclases y probar su comportamiento
2 class malware():
3     def __init__(self,nombre):
4         self.nombre = nombre
5
6     def get_nombre(self):
7         return self.nombre
8
9     def infectar():
10        # Codigo para infectar un sistema
11        print "Sistema infectado"
12
13 class gusano(malware):
14     def propagarse(self):
15         # Codigo para propagar la amenaza
16         print "Infectando USB"
17
18 class troyano(malware):
19     def propagarse(self):
20         # Codigo para propagar la amenaza
21         print "Enviando mail con adjunto malicioso a los contactos"
22
23
24
```

Imagen 5 - codigos_maliciosos.py

Ejemplo.py:

```
codigos_maliciosos.py x  ejemplo.py x
1 from codigos_maliciosos import troyano, gusano
2
3 amenazas = [gusano("Win32/Dorkbot"), troyano("Win32/Waledac")]
4
5 for amenaza in amenazas:
6     print "Sistema infectado por %s." % amenaza.get_nombre()
7     amenaza.propagarse()
```

Imagen 6 - ejemplo.py

Al ejecutar el archivo *ejemplos.py*, Python sabrá que debe ir a buscar las clases **troyano** y **gusano** al archivo **codigos_maliciosos.py** y así podremos ejecutar los métodos de estas clases sin problemas al instanciar los objetos. En este caso, para que funcione nuestro script, ambos archivos deberán estar en el mismo directorio.

Reutilizando código

La reutilización de código es muy importante para evitar tener que duplicar las mismas líneas de código en diferentes lugares de nuestro programa, haciendo que sea más prolijo y uniforme todo lo que hagamos. Además, nos evita problemas al cambiar cualquier sentencia, ya que lo tendremos que hacer una sola vez en lugar de modificar las mismas líneas en todos los lugares en los que aparezcan.

Conclusión

A lo largo del presente módulo introdujimos todos los conceptos y definiciones relacionados a las clases y objetos, como así también aprendimos a reutilizar nuestro código sin tener que escribir una y otra vez las mismas líneas. Lleva un tiempo acostumbrarse a estos conceptos y manejarlos correctamente, sin embargo con un poco de práctica lograrán aplicarlos y lograr sus programas sean más eficientes, estén mejor organizados y no se vean forzados a escribir una y otra vez las mismas sentencias.

La programación Orientada a Objetos es uno de los paradigmas más utilizados en la actualidad. Si bien se ajusta a un montón de situaciones que en el día a día nos podemos encontrar, en Python lo podemos combinar con otros paradigmas para sacar lo mejor del lenguaje y aumentar nuestra productividad manteniendo un diseño óptimo del código.