Funciones

Fundamentos de Programación
FIEC04341
Sesión 01

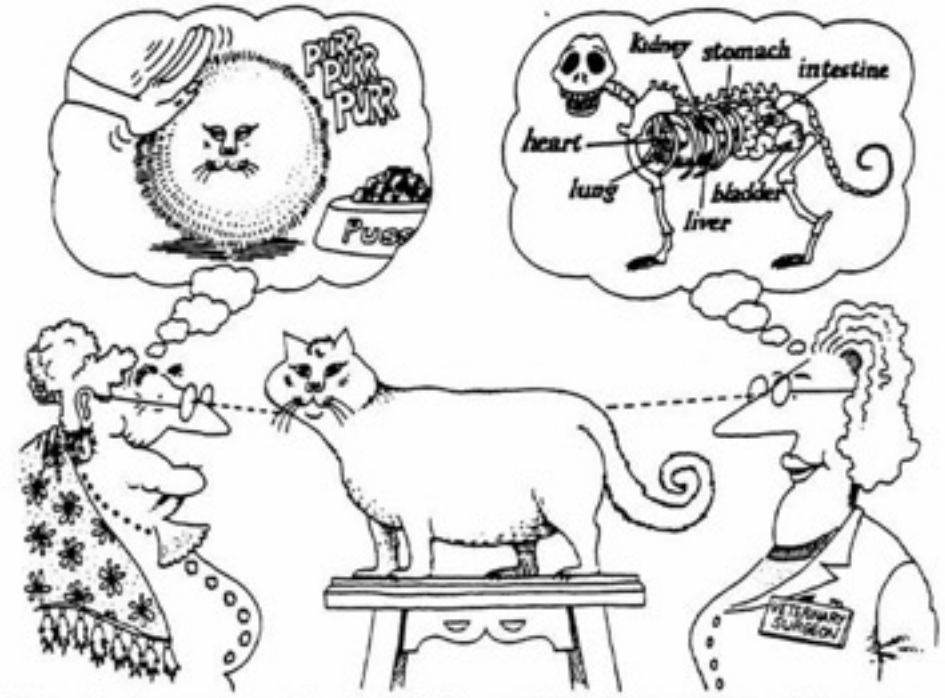
Agenda

- Definición, implementación y uso de funciones
- Paso de parámetros y retorno de valores
- Tipos de funciones
- Alcance de variables
- Módulos

Terminología

Terminología:

- **Abstraer**: Formar una idea mental de un objeto extrayendo sus rasgos esenciales desde una perspectiva en particular.



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Fuente: https://mondeca.wordpress.com

Terminología

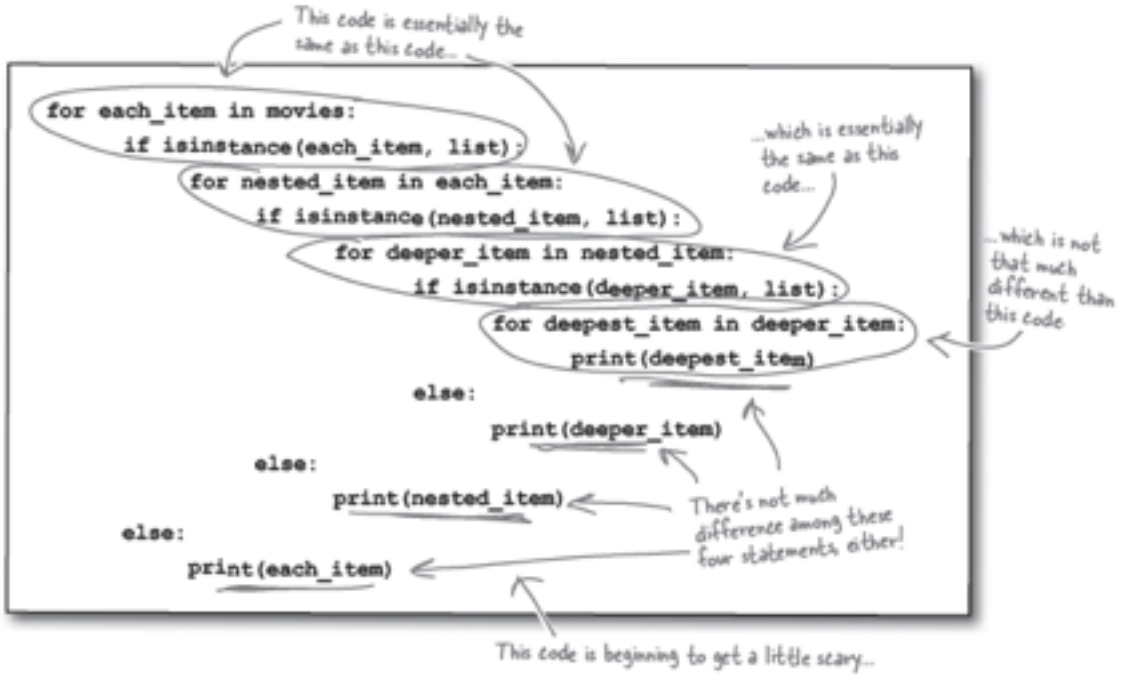
- **Encapsular**: meter en cápsula o cápsulas

  - Fuente: http://www.capsugel.es/

Funciones

Funciones

- Repetir código fuente en un programa, hace a éste difícil de leer y de mantener.



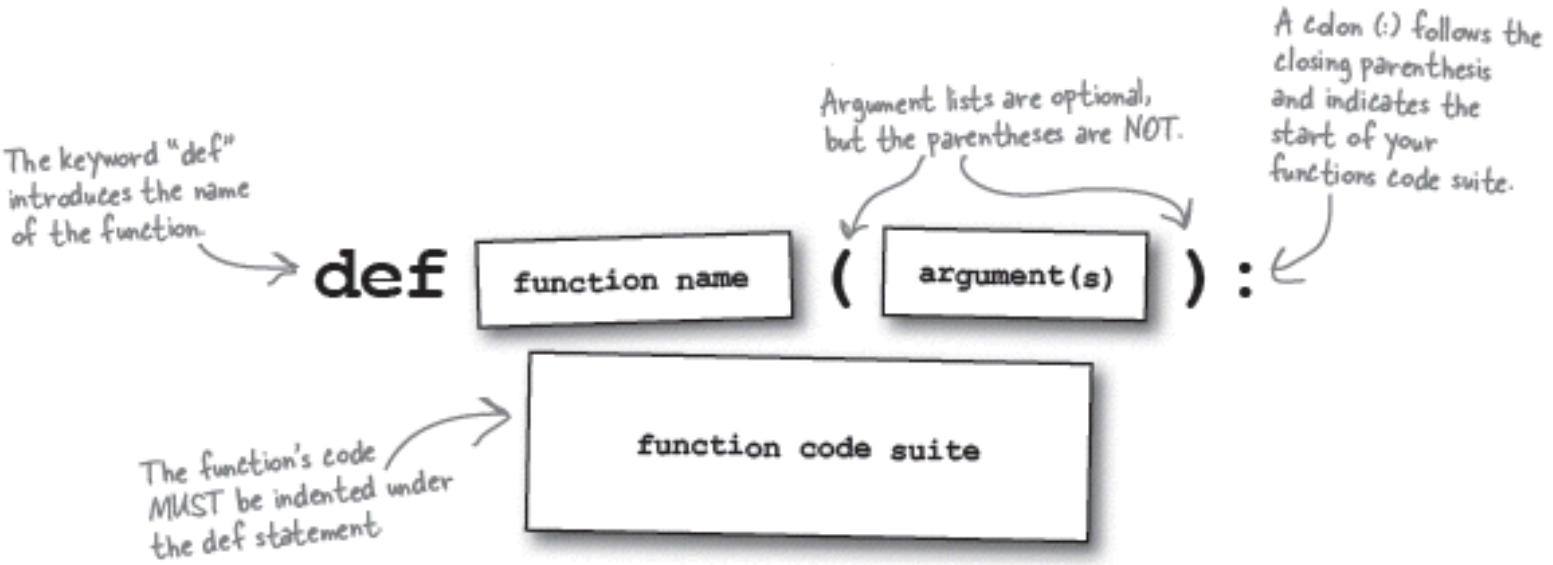
- Dividir un programa en funciones favorece la reutilización de código.

Definición, implementación y uso de funciones

Definición de funciones

- Python tiene incorporadas funciones como por ejemplo, len() y range()
- Pero también se puede agregar funciones personalizadas con la finalidad de dividir el código fuente y hacerlo más manejable.
- De esta manera, un programa se constituye de una serie de funciones, en donde cada una de ellas es más fácil de crear y entender.

Definición de una función



Abstracción

- La abstracción nos permite concentrarnos en el gran problema sin preocuparnos de los detalles. Así, se puede usar una función sin preocuparse en los detalles de como esta lleva a cabo su tarea.
- Por ejemplo, cuando se realiza el pedido de un combo de hamburguesa en un local de comidas rápidas, el comprador no necesita preocuparse de servir la bebida, preparar la hamburguesa, freír las papas, lavar los implementos, etc.

Implementación de una función

```
>>> def print_lol(the_list):
        for each_item in the_list:
            if isinstance(each_item, list):
                print_lol(each_item)
            else:
                print(each_item)
```

Encapsulamiento

- Ninguna variable que se crea en una función, incluyendo sus parámetros, pueden ser accedidos directamente fuera de la función. A esto se conoce como encapsulamiento.
- El encapsulamiento ayuda a mantener el código independiente y realmente separado por medio del ocultamiento o encapsulamiento de los detalles.
- Los parámetros y los valores de retornos comunican la única información que debe ser intercambiada.

Uso de una función

- Llamar a una función creada por el programador es casi como llamar a una función embebida. Use el nombre de la función seguida de los respectivos paréntesis. Por ejemplo:
  - instructions()

Paso de parámetros y retorno de valores

Paso de parámetros

- Las funciones pueden recibir valores y devolver valores de retorno.
- Por ejemplo, la función len(), a la cual se le provee una secuencia y retorna su longitud.

```
def display(message):
    print(message)

def give_me_five():
    five = 5
    return five

def ask_yes_no(question):
    """Ask a yes or no question."""
    response = None
    while response not in ("y", "n"):
        response = input(question).lower()
    return response
```

```
# main
display("Here's a message for you.\n")

number = give_me_five()
print("Here's what I got from give_me_five():", number)

answer = ask_yes_no("\nPlease enter 'y' or 'n': ")
print("Thanks for entering:", answer)

input("\n\nPress the enter key to exit.")
```

Retorno de valores

- Desde una función se retorna valores por medio de la sentencia *return*. Como en nuestro ejemplo:
  - *return* five
- Cuando esta línea se ejecuta, la función pasará el valor de five de retorno a la parte del programa que la llamó.

```
number = give_me_five()
print("Here's what I got from give_me_five():", number)
```

- Una función siempre termina luego que alcanza la sentencia *return*.
- Se puede retornar más de un valor listando todos los valores a retornar separados por coma.

Parámetros predeterminados y argumentos de palabra clave

```
# Birthday Wishes
# Demonstrates keyword arguments and default parameter values

# positional parameters
def birthday1(name, age):
    print("Happy birthday,", name, "!", " I hear you're", age, "today.\n")

# parameters with default values
def birthday2(name = "Jackson", age = 1):
    print("Happy birthday,", name, "!", " I hear you're", age, "today.\n")


birthday1("Jackson", 1)
birthday1(1, "Jackson")
birthday1(name = "Jackson", age = 1)
birthday1(age = 1, name = "Jackson")

birthday2()
birthday2(name = "Katherine")
birthday2(age = 12)
birthday2(name = "Katherine", age = 12)
birthday2("Katherine", 12)

input("\n\nPress the enter key to exit.")
```

Parámetros predeterminados y argumentos de palabra clave

- Los parámetros posicionales obtienen los valores enviados en orden, a menos que se le especifique los contrario a la función.
- Se le puede indicar a la función que asigne ciertos valores a parámetros específicos, sin importar el orden con los parámetros claves.

```
birthday1(age = 1, name = "Jackson")
```

- Usando valores predeterminados para parámetros

```
# parameters with default values
def birthday2(name = "Jackson", age = 1):
    print("Happy birthday,", name, "!", " I hear you're", age, "today.\n")
```

Tipos de funciones

Tipos de funciones

- Hay tres tipos de funciones:
  - Funciones que no retornan ningún valor se conocen como procedimientos.

```
def display(message):
    print(message)
```

  - Funciones que retornan valores lógicos (True o False) se denominan como funciones predicado.
  - Funciones que reciben argumentos y parámetros.

```
def ask_yes_no(question):
    """Ask a yes or no question."""
    response = None
    while response not in ("y", "n"):
        response = input(question).lower()
    return response
```

Alcance de variables

Alcance de variables

- El alcance representa los diferentes partes de un programa que están separadas entre sí.
- Cada función define su propio alcance.

Alcance de variables

```
# Global Reach
# Demonstrates global variables

def read_global():
    print("From inside the local scope of read_global(), value is:", value)

def shadow_global():
    value = -10
    print("From inside the local scope of shadow_global(), value is:", value)

def change_global():
    global value
    value = -10
    print("From inside the local scope of change_global(), value is:", value)

# main
# value is a global variable because we're in the global scope here
value = 10
print("In the global scope, value has been set to:", value, "\n")

read_global()
print("Back in the global scope, value is still:", value, "\n")

shadow_global()
print("Back in the global scope, value is still:", value, "\n")

change_global()
print("Back in the global scope, value has now changed to:", value)

input("\n\nPress the enter key to exit.")
```

Alcance de variables

- En el ejemplo anterior analice:
  - La lectura de una variable global desde dentro de una función.
  - El ocultamiento de una variable global desde dentro de una función.
  - La modificación de una variable global desde dentro de una función.

Alcance de variables

- Las variables globales lo hacen confuso a un programa. Se debe minimizar su uso.
- Por el contrario, las globales constantes (`variables globales tratadas como` constantes), hacen al programa menos confuso.