Funciones
Estructuras de Datos
Dinamicos

Elaborado por: Inst. Leticia Mendieta

Funciones

- Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.
- El uso de funciones es un componente muy importante del paradigma de la programación llamada *estructurada*, y tiene varias ventajas:
- **modularización**: permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- **reutilización**: permite reutilizar una misma función en distintos programas.
- Python dispone de una serie de *funciones integradas* al lenguaje, y también permite crear funciones definidas por el usuario para ser usadas en su propios programas.

Sentencia def

- La sentencia def es una definición de función usada para crear objetos funciones definidas por el usuario.
- Una definición de función es una sentencia ejecutable. Su ejecución enlaza el nombre de la función en el namespace local actual a un objecto función (un envoltorio alrededor del código ejecutable para la función). Este objeto función contiene una referencia al namespace local global como el namespace global para ser usado cuando la función es llamada.
- La definición de función no ejecuta el cuerpo de la función; esto es ejecutado solamente cuando la función es llamada.

La sintaxis para una definición de función en Python es:

```
def NOMBRE(LISTA_DE_PARAMETROS):
    """DOCSTRING_DE_FUNCION"""
    SENTENCIAS
    RETURN [EXPRESION]
```

A continuación se detallan el significado de pseudocódigo fuente anterior:
NOMBRE, es el nombre de la función.
LISTA_DE_PARAMETROS, es la lista de parámetros que puede recibir una función.
DOCSTRING_DE_FUNCION, es la cadena de caracteres usada para documentar la función.
SENTENCIAS, es el bloque de sentencias en código fuente Python que realizar cierta operación dada.
RETURN, es la sentencia return en código Python.
EXPRESION, es la expresión o variable que devuelve la sentencia return.

Un ejemplo simple de función esta seguidamente:

```
>>> def hola(arg):
...     """El docstring de la función"""
...     print "Hola", arg, "!"
...
>>> hola("Plone")
Hola Plone !
```

Advertencia:
Los bloques de function deben estar indentado como otros bloques estructuras de control.

La palabra reservada def se usa para definir funciones. Debe seguirle el nombre de la función en el ejemplo anterior hola() y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar indentado.

Funciones: Parámetros

Es posible ingresar datos al ser invocadas a estos datos se les denomina argumentos y son ligados a nombres, los cuales se conocen como parámetros. El número de argumentos ingresados debe corresponder al número de parámetros que se definen. En caso de que no se ingresen los argumentos necesarios, se generará un error de tipo *TypeError*.

```
def suma(numero1,numero2):
    '''función la cual suma dos números'''
    print numero1 + numero2
    print "\n"
```

Sentencia pass

Es una operación nula — cuando es ejecutada, nada sucede. Eso es útil como un contenedor cuando una sentencia es requerida sintácticamente, pero no necesita código que ser ejecutado, por ejemplo:

```
>>> # una función que no hace nada (aun)
... def consultar_nombre_genero(letra_genero): pass
...
>>> type(consultar_nombre_genero)
<type 'function'>
>>> consultar_nombre_genero("M")
>>>
>>> # una clase sin ningún método (aun)
... class Persona: pass
...
>>> macagua = Persona
>>> type(macagua)
<type 'classobj'>
```

Sentencia return

Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la sentencia return. El proceso de comunicación con el exterior se hace devolviendo valores. A continuación, un ejemplo de función usando return:

```
def suma(numero1,numero2):
    '''función la cual suma dos números'''
    print numero1 + numero2
    print "\n"
```

```
>>> suma(23,74)
97
```

Retorno múltiple

Una característica interesante, es la posibilidad de devolver valores múltiples separados por comas:

```
>>> def prueba():
...     return "Plone CMS", 20, [1,2,3]
...
>>> prueba()
('Plone CMS', 20, [1, 2, 3])
```

En el código anterior los valores múltiples se tratan en conjunto como una *tupla* inmutable y se pueden reasignar a distintas variables:

```
>>> def prueba():
...     return "Plone CMS", 20, [1,2,3]
...
>>> prueba()
('Plone CMS', 20, [1, 2, 3])
>>> cadena, numero, lista = prueba()
>>> print cadena, type(cadena)
Plone CMS <type 'str'>
>>> print numero, type(numero)
20 <type 'int'>
>>> print lista, type(lista)
[1, 2, 3] <type 'list'>
```

Estructura de datos dinámicos: Listas

Las listas son estructuras de datos que se encargan de almacenar y ordenar elementos, las listas son del tipo mutable ¿Que quiere decir esto? que se pueden añadir, eliminar y editar datos en tiempo de ejecución a diferencia de las tuplas que son inmutables y una vez creadas ya no se pueden modificar.

Las listas en Python pueden manejar varios tipos de datos a la vez, es decir puede tener elementos numéricos y cadenas de texto mezclados sin ningún problema, recordemos que las variables en Python son dinámicas, entonces no tenemos problemas para realizar esta declaración:

```
>> lista = [True, 1, "hola", 42, False]
```

Podemos ver que la sintaxis es muy simple, corchetes que contienen los elementos que van separados por comas, podemos observar que primero hay un booleano, luego un número, una cadena, otro numero y finalmente otro booleano.

Mostrar elementos de una Lista

Para mostrar en pantalla todos los elementos podemos usar la función print() así:

>> print("\nEl contenido de la lista es:\n",lista, "\n")

El contenido de la lista es:

[True, 1, 'hola', 42, False]

Para el mismo propósito y de manera similar, podemos hacer uso de un for donde imprimimos los valores de la variable "i" que cambia a lo largo del recorrido de la lista:

```
>>for i in lista:
>>     print(i)
True
1
hola
42
False
```

Mostrar elementos de una lista

Si queremos acceder a un elemento específico de la lista lo podemos hacer por medio de su índice, como en los Arrays, el índice parte de 0, entonces si queremos el primer término colocaríamos:

>>lista[0]

True

>>lista[3]

hola

Si queremos acceder a los elementos de una lista por medio de un rango, digamos por ejemplo del elemento 1 al 3, usamos [1:4] y muestra los elementos cuyo índice sea 1,2 ó 3 (no incluye el 4), por ejemplo:

>>lista[1:4]

[1, 'hola', 42]

>>lista[0:2]

[True, 1]

Métodos de las Listas

Las listas en Python tienen muchos métodos que podemos utilizar, entre todos ellos vamos a nombrar los más importantes. Para esto utilizaremos esta lista de ejemplo.

```
my_list = [2, 5, , 'DEFG', 1.2, 5]
```

Append()

Este método nos permite agregar nuevos elementos a una lista.

```
my_list.append(10) # [2, 5, 'DEFG', 1.2, 5, 10]
```

```
my_list.append([2,5]) # [2, 5, 'DEFG' , 1.2, 5, [2, 5]]
```

Podemos agregar cualquier tipo de elemento a una lista, pero tengan en cuenta lo que pasa cuando agregamos una lista dentro de otra, esta lista se agrega como uno y solo un elemento.

Métodos de las Listas

Extend()

Extend también nos permite agregar elementos dentro de una lista, pero a diferencia de append al momento de agregar una lista, cada elemento de esta lista se agrega como un elemento más dentro de la otra lista.

```
my_list.extend([2,5]) # [2, 5, 'DEFG', 1.2, 5, 2, 5]
```

Remove()

El método remove va a remover un elemento que se le pase como parámentro de la lista a donde se le esté aplicando.

```
my_list.remove(2) # [5, 'DEFG', 1.2, 5]
```

En este ejemplo estamos removiendo el elemento 2, de la lista que tiene por nombre "my_list".

Pop()

Quita el ítem en la posición dada de la lista, y lo devuelve. Si no se especifica un índice, a.pop() quita y devuelve el último ítem de la lista.

```
my_list.pop() # [5, 'DEFG', 1.2] #elimino el 5
```

Métodos de las Listas

Index()

Index devuelve el número de indice del elemento que le pasemos por parámetro.

```
my_list.index('DEFG') # 2
```

Aquí estamos preguntando por el indice de la cadena 'DEFG' dentro de la lista "my_list", esto devuelve 2.

Count()

Para saber cuántas veces un elemento de una lista se repite podemos utilizar el metodo count().

```
my_list.count(5) # 2
```

Contamos cuantas veces se repite el número 5 dentro de la lista, y esto devuelve 2.

Reverse()

También podemos invertir los elementos de una lista.

```
my_list.reverse() # [5, 1.2, 'DEFG', 5, 2]
```

Estas son algunos de los métodos más útiles y más utilizados en las listas.

Fuentes Consultadas

http://docs.python.org.ar/tutorial/3/datastructures.html

https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/funciones.html

https://devcode.la/tutoriales/listas-python/