Programación usando Python

Cecilia Manzino

Arreglos en Python

- Los arreglos no son una estructura de datos nativa de Python.
- NumPy es una Librería de Python library que permite trabajar con arreglos de forma eficiente (http://www.numpy.org/).
- Otra alternativa es usar el módulo `array`.

```
>>> import array
```

Módulo Array

- El módulo Array permite representar arreglos de valores de tipos básicos: enteros, caracteres o puntos flotantes.
- El tipo de los elementos es especificado con un *type code* (código de tipo), algunos de ellos son:

Type Code	Python Type
`'c'`	`character`
`'i'`	`int`
`'I'`	`long`
`'f', 'd'`	`float`

Creando arreglos

Un arreglo se inicializa dando el tipo de los elementos del mismo y posiblemente una secuencia inicial de datos. Sintaxis:

```
array.array(typecode, initializer)
```

Ejemplos:

```
1  import array
2
3  # arreglo vacio de elementos enteros
4  b = array.array ('i')
5  c = array.array('i', [0, 1, 2, 3, 4])
6
7  # otra forma de definir c
8  d = array.array('i', xrange(5))
9  print 'El arreglo b: ', b
10 print 'El arreglo c: ', c
11 print 'El arreglo d: ', d
```

Creando arreglos

```
1 e = array.array('c', 'hola mundo')
2
3 # un pedazo de e
4 f = array.array('c', e[0:4])
5
6 print 'El arreglo e: ', e
7 print 'El arreglo f: ', f
```

Operaciones sobre arreglos

- Para agregar elementos al final usar `append` o `insert`.

  nombre_arreglo.append(elemento)

  nombre_arreglo.insert(indice, elemento)

- Para agregar más de un elemento usar `extend`, `fromlist` o `fromstring`.

  nombre_arreglo.extend(arreglo_a_agregar)

  nombre_arreglo.fromlist(lista_a_agregar)

  nombre_arreglo.fromstring(string_a_agregar)

Más operaciones

- Para acceder a los elementos del arreglo:

  nombre_arreglo[indice]

- Para eliminar un elemento del arreglo usar `remove` o `pop` que elimina el último elemento:

  nombre_arreglo.remove(indice)

  nombre_arreglo.pop()

Ejemplos

```
1 print c[0], c[1], c[2], c[3], c[4]
```

```
0 1 2 3 4
```

```
1 c.append(6)
2 print c
```

```
array('i', [0, 1, 2, 3, 4, 6])
```

```
1 c.insert(1, 10)
2 print c
```

```
array('i', [0, 10, 1, 2, 3, 4, 6])
```

Ejemplos

```
1 c.fromlist([5,6,7])
2 print c
```

```
array('i', [0, 10, 1, 2, 3, 4, 6, 5, 6, 7])
```

```
1 c.pop()
2 print c
```

```
array('i', [0, 10, 1, 2, 3, 4, 6, 5, 6])
```

```
1 c.remove(0)
2 print c
```

```
array('i', [10, 1, 2, 3, 4, 6, 5, 6])
```

Operaciones sobre arreglos

- *nombre_arreglo.buffer_info()*

  retorna una tupla con la dirección en memoria del arreglo y el tamaño del mismo (cantidad de elementos).

- *nombre_arreglo.count(x)*

  devuelve la cantidad de apariciones de x en el arreglo.

- *nombre_arreglo.reverse ()*

  invierte el orden de los elementos en el arreglo

La sentencia **for**

Esta sentencia itera sobre los ítems de cualquier secuencia (una lista o una cadena de texto), en el orden que aparecen en la misma. Ejemplos:

```
1 # mide cadenas de texto
2 palabras = ['gato', 'ventana', 'perro']
3
4 for p in palabras:
5   print(p, len(p))
```

```
('gato', 4)
('ventana', 7)
('perro', 5)
```

Sentencia **for**

```
1 # para iterar sobre una secuencia de numeros
2 for i in range(5,10):
3   print(i)
```

```
5
6
7
8
9
```

```
1 # especificamos tambien el crecimiento
2 for i in range(5,10,2):
3   print(i)
```

```
5
7
9
```

Definición de funciones

- La palabra reservada def se usa para definir funciones.
- Debe seguirle el nombre de la función y la lista de parámetros entre paréntesis.
- Las sentencias que forman el cuerpo de la función deben empezar en la línea siguiente, y estar con sangría.
- La sentencia return devuelve un valor en una función. return sin un argumento no retorna nada. Si se alcanza el final de una función, tampoco se retorna nada.

Ejemplo

```
1 def signo (a):
2     if a >0: result = 1
3     elif a <0 : result = -1
4     else : result = 0
5     return result
```

```
>>> f (3)
1
>>> f (-2)
-1
```