

Diseño e implementación de metaheurísticas con Python

Sergio Pérez Peló

Jesús Sánchez-Oro



Universidad
Rey Juan Carlos

Listas

- Una lista es una secuencia de valores de cualquier tipo.
- Se encierran entre corchetes y se separa cada valor por comas.
 - $a = [1, 2, 3]$
- Las listas son punteros, por lo que la asignación solo copia la referencia.
- Se puede definir una lista vacía con `[]`.

Listas

- La función `len()` devuelve la longitud de una lista.
- El operador `+` concatena listas.
- El operador `*` repite una lista varias veces.
- También se puede acceder a una posición con `lista[i]` y cortar con el operador `:` como con las cadenas.
- Se puede iterar por una lista con la estructura `for ... in`

Listas

- Para añadir un elemento a una lista, se puede utilizar la concatenación o la función `append()`:

```
a = [1,2,3]  
a += 4  
a.append(4)
```



- La segunda opción es más eficiente, porque no crea listas intermedias.

Listas

- Para eliminar de una lista se utiliza la sentencia del:

```
a = [1,2,3]  
del a[1]
```

- Elimina el elemento que ocupa la posición indicada, sin crear copias intermedias.

Listas

- Para saber si un elemento está en la lista, podemos utilizar la sentencia `in`:

`a = [1,4,7,9]`

`9 in a`

`2 in a`

- También funciona con cadenas y subcadenas.
- OJO! $\rightarrow O(n)$

Matrices

- Para guardar la siguiente matriz:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- En Python haremos:

```
M = [[1,2,3],[4,5,6],[7,8,9]]
```



Matrices

- El elemento que ocupa la fila i -ésima y la columna j -ésima se accede con $M[i][j]$.
- Si accedemos con un único índice, recuperaremos la fila:
 - $M[i] \rightarrow [1, 2, 3]$
- Y el segundo índice accede a la posición en esa fila:
 - $M[i][j] \rightarrow 1$

Matrices

- Si queremos acceder al número de filas, utilizamos `len(M)`.
- Si queremos acceder al número de columnas, utilizamos `len(M[0])`.

Diccionarios

- Correspondencia entre claves y valores.
- En Python, se crea así:

```
d = {}
```

- Ahora, podemos ir añadiendo pares clave valor:

```
d['python'] = 'bueno'
```

```
d['java'] = 'mejor'
```

```
d['c'] = 'malo'
```

Diccionarios

- Para consultar un valor, basta con acceder como una lista, consultando la clave.
- Si la clave no está, se lanzará un error.
- Para saber si una clave está o no en el diccionario, utilizamos el operador `in`.
- Podemos recorrer un diccionario con `for...in`.
- Para borrar, utilizamos `del()`.

Ejercicio

- Implementa un programa que lea una lista de enteros y después un entero, y cuente cuántas veces aparece ese entero en la lista.

- Módulo principal para cómputo científico de alto rendimiento.
- Para utilizarlo, debemos instalarlo e importarlo con:

```
import numpy as np
```

- ¿Cómo podemos crear arrays en numpy?
 - `np.array([1,2,3])`: `[1,2,3]`
 - `np.zeros((3,4))`: matriz de ceros de 3×4
 - `np.ones((2,3,4))`: matriz de unos de $2 \times 3 \times 4$
 - `np.arange(0, 13, 3)`: array de valores similar a range
 - ...
- Existen muchas formas de crear arrays de diferentes tipos con numpy.

- Propiedades de los arrays de numpy:
 - `a.shape`: Dimensiones del array
 - `len(a)`: Longitud del array
 - `b.ndim`: Número de dimensiones del array
 - `e.size`: Número de elementos del array
 - `b.dtype`: Tipo de datos de los elementos del array
 - `b.dtype.name`: Denominación del tipo de datos
 - `b.astype(int)`: Convertir un array a un tipo diferente

- Operaciones aritméticas:
 - Suma: `np.add(a,b)`, $a + b$
 - Resta: `np.subtract(a,b)`, $a - b$
 - Multiplicación: `np.multiply(a,b)`, $a * b$
 - Producto matricial: `np.dot(a,b)`
 - División: `np.divide(a,b)`, a / b
 - Otras operaciones habituales elemento a elemento:
`np.exp(a)`, `np.sqrt(a)`, `np.sin(a)`, `np.cos(a)`,
`np.log(a)`

Ejercicio

- Crea una lista en Python y otra con Numpy de muchos elementos, y luego ordénalas, ¿hay diferencia en tiempo?

Depuración



Diseño e implementación de metaheurísticas con Python

Sergio Pérez Peló

Jesús Sánchez-Oro



Universidad
Rey Juan Carlos