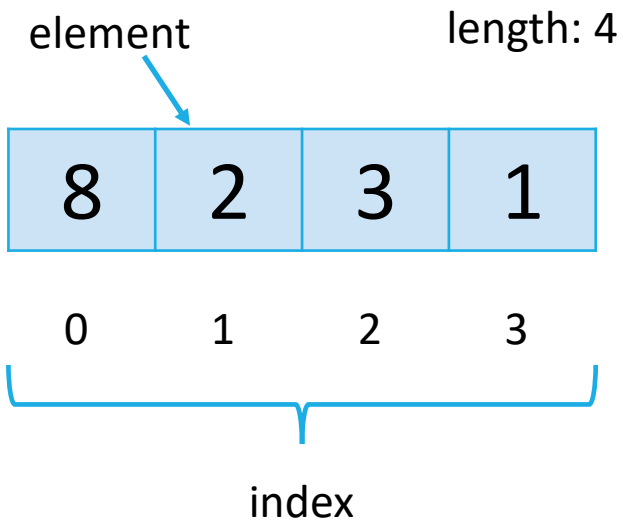
The basics of NumPy arrays

Fundamentals

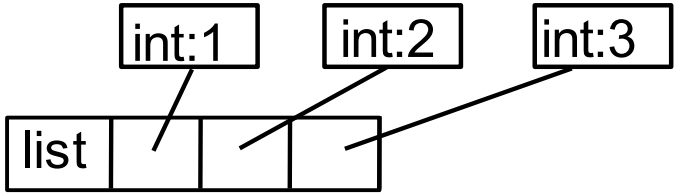
Vetors

- A vector is an elementar data structure, capable of storing an ordered collection of numeric values
- The access to the elements of a vector is made by means of an index



How to represent vectors?

- Using a tuple
  - imutable object
- Using a list
  - Mutable object, but not efficient



- Using the **numpy library** (numeric python)
  - Allows to define and manipulate vectors and matrices efficiently
  - You need to import the **numpy library**

```
import numpy
```

numpy: Creating vectors

- It is possible to create a vector by explicitly providing its elements

```
v1 = numpy.array([3, 8, 6])
```

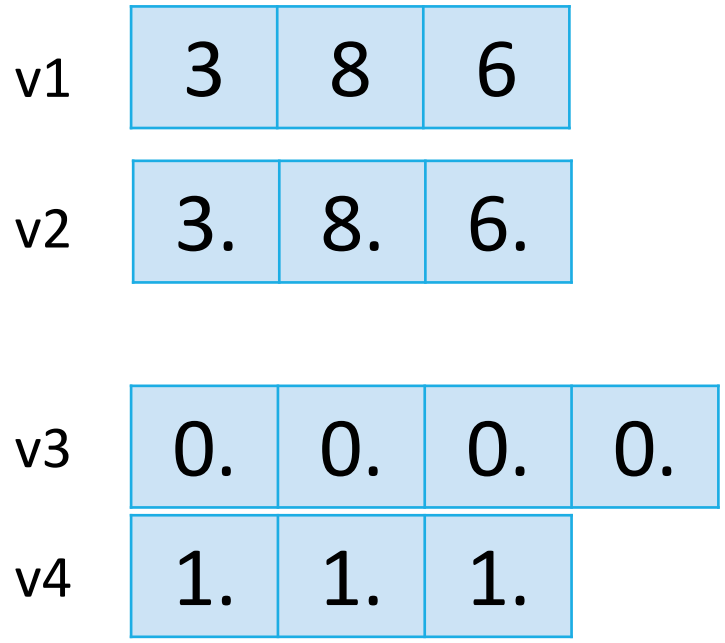
- We can explicitly indicate the type of the vector

```
v2 = numpy.array([3, 8, 6], float)
```

- Or by initiating the vector with zeros or ones

```
v3 = numpy.zeros(4)
v4 = numpy.ones(3)
```

  - The size of the vector is determined by the integer that appears in brackets



Vector Manipulation

- The access or modification of the vector elements is made through the index

```
v = numpy.zeros(4, int)

v[0] = 2

v[1:] = 4

v[3] = 6

last = v[-1]
size = v.size # len(v) also works
```

v	0	0	0	0
v	2	0	0	0
v	2	4	4	4
v	2	4	4	6

last	6
size	4

Vector Manipulation: operations

```
v = numpy.array([1, 3, 7, 2, 4])

print(v[1:4])
```

```
[3 7 2]
```

```
print(v * 2)
```

2	6	14	4	8

```
print(v + v2)
```

5	5	10	3	11

```
gt2 = v > 2

v[v>2] += 1     # v[v>2]=v[v>2]+1
```

v

1	3	7	2	4

v2

4	2	3	1	7

gt2

False	True	True	False	True

v

1	4	8	2	5

Vector Manipulation: functions (methods)

```
v = numpy.array([1, 3, 7, 2, 4])

v.sum()          17

v.max()          7

v.argmax()       2

v.mean()         3.4

v.shape          (5,)

v.sort()
```

v

1	3	7	2	4

v

1	2	3	4	7

Matrices

- The notion of matrix in programming is related to the mathematical notion of matrix
- The access to the elements of a matrix are made through a pair of indexes

Creating Matrices

- You can create a matrix by explicitly providing its elements

```
m = numpy.array([[2, 3, 6], [7, 12, 2]])
```

m

	0	1	2
0	2	3	6
1	7	12	2

- Or starting the matrix with zeros or ones

```
m = numpy.zeros([2,3], int)
```

m

0	0	0
0	0	0

```
m = numpy.ones([2,3], int)
```

m

1	1	1
1	1	1

Matrices manipulation : modification

- Again, the modification of the elements is done by means of an index

```
m = numpy.zeros([2, 3])
m[0, 2] = 2.1
m[1, 0] = 5.3
```

m

0	0.	0.	2.1
1	5.3	0.	0.
	0	1	2

Matrices manipulation: access

- The dimensions of a matrix can be obtained through the shape attribute

```
number_of_rows = m.shape[0]
number_of_columns = m.shape[1]
number_of_elements = m.size
single_element = m[0, 0]
```

```
m = numpy.array([[1, 2, 3], [4, 5, 6]])
```

```
m.shape: (2, 4)
m.size: 24
```

Matrices manipulation: operations

```
m = numpy.array([[2, 3, 6], [3, 7, 5]])
```

```
m[1]
```

```
[3 7 5]
```

```
m[1,1:]
```

```
[7 5]
```

```
m[:,1:]
```

3	6
7	5

```
m * 2
```

4	6	12
6	14	10

```
m > 3
```

False	False	True
False	True	True

m

2	3	6
3	7	5

Matrices manipulation: methods (functions)

```
m = numpy.array([[2, 3, 6], [3, 7, 5]])
```

```
m.sum()
```
```
26
```

```
m[0].sum()
```
```
11
```

```
m.max()
```
```
7
```

```
m.argmax()
```
```
4
```

```
m.mean()
```
```
4.3333
```

```
m.shape
```
```
(2,3)
```

```
m2 = m.T
```

m

2	3	6
3	7	5

m2

2	3
3	7
6	5

Summary

- Fast & Memory Efficient: Optimized for numerical operations.
- Arrays: Homogeneous, multi-dimensional arrays, faster and more efficient than Python lists.
- Indexing & Slicing: Access elements using integer indexes, slices, boolean masks.
- Broadcasting: Perform operations on arrays of different shapes without explicit looping.