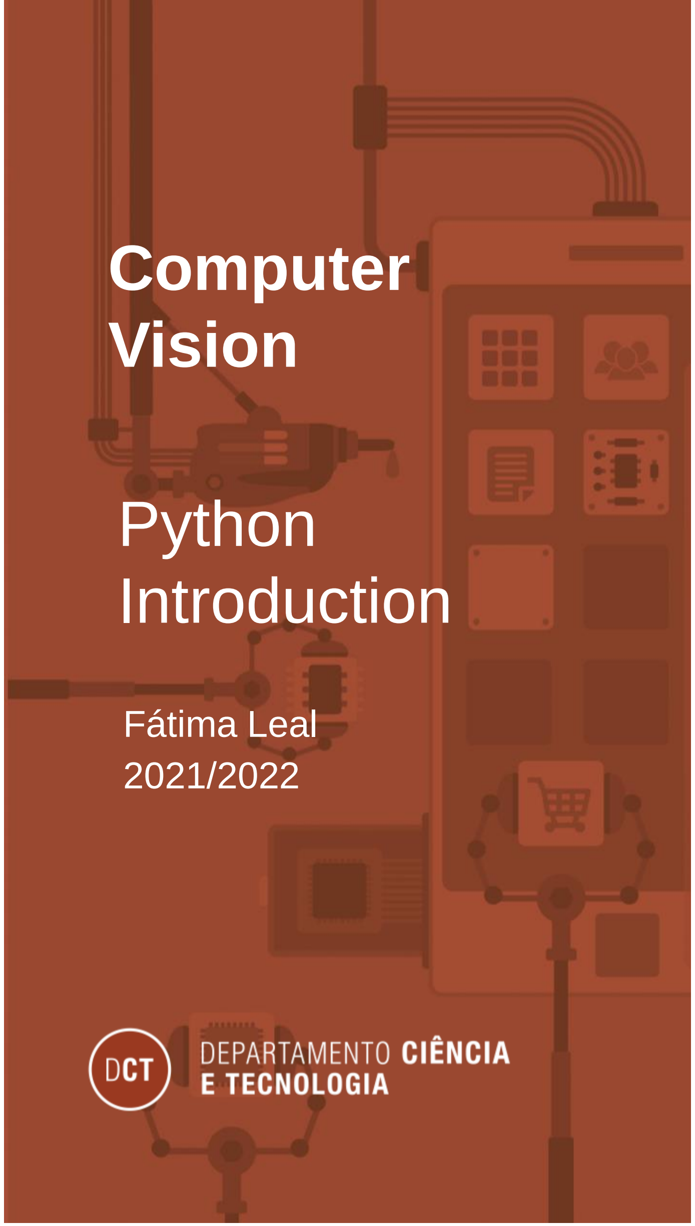


Computer Vision

Python Introduction

Fátima Leal

2021/2022

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UNIVERSIDADE PORTUCALENSE

Content

- Installation of Python Environment
- Basic Python Concepts
- Practical Examples

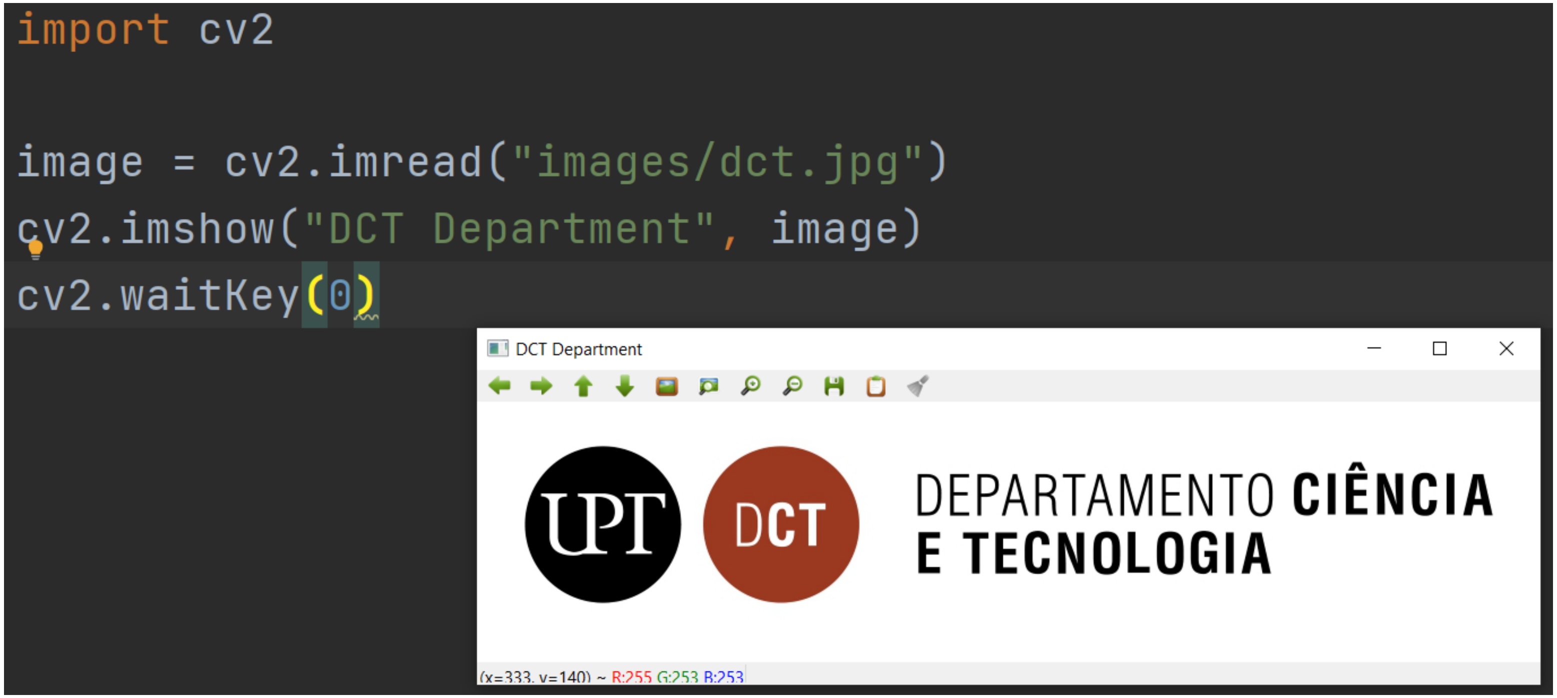
Installation Environment

- **Install** Anaconda with Python 3.8
- **Install** Pycharm for Community
- **Open Anaconda** Prompt
  - Create a **virtual environment**:
    - `conda create -n computervision@upt python=3.8`
  - **Activate** your virtual environment:
    - `conda activate computervision@upt`
  - **Verify** that your virtual environment was installed correctly:
    - `conda list`
  - **Install** the following modules:
    - `conda install -c anaconda numpy scipy scikit-image scikit-learn matplotlib nb_conda_kernels`
  - **Install** OpenCV
  - **Install** Mahotas

Installation Environment

- **Deactivate** your virtual environment:
  - `conda deactivate`
- **Close** Anaconda Prompt
- **Setup PyCharm** with an anaconda virtual environment:
  - Create **new project**
  - **Select the existing interpreter**
    - **Select conda** environment
    - **Select make available for all projects**
  - Create
  - **See** if everything is **correctly installed**:
    - file -> settings
    - Project: name of your project -> Project interpreter

Introduction to Python



```
import cv2

image = cv2.imread("images/dct.jpg")
cv2.imshow("DCT Department", image)
cv2.waitKey(0)
```

```
(x=333, y=140) ~ R:255 G:253 B:253
```

Python: variables and data types

- **Variables**

```
>>>b = 2 # b is integer type

>>>b = b*2.0 # now b is float type
```

- **Types**: int, float, string, Boolean, etc.

- **Strings**

```
>>>s = “3 9 81”
>>>print(s.split())
>>>s[0]='p' # error
```

Python: Strings

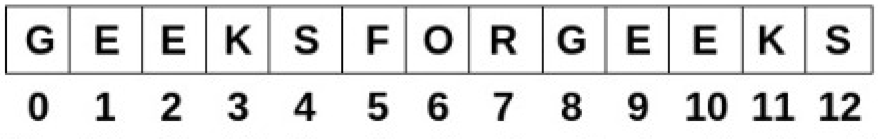
- **Strings** in Python are enclosed in either **single quotes** ’ or **double quotes** "

```
>>> type(‘This is a string.’)
<class ’str’>
>>> type(“And so is this.”)
<class ’str’>
```

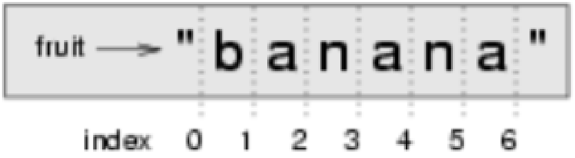
- Operations with strings:
  - `len(str) -` length of a string.
  - `str[i]` the subscript operation extracts the i’th character of the string, as a new string.
  - `str[i:j]` the slice operation extracts a substring out of a string.
  - `str.find(target)` returns the index where target occurs within the string, or -1 if it is not found.

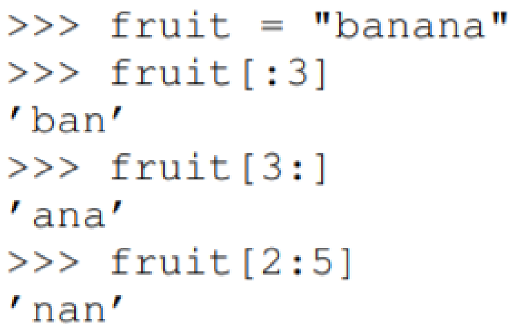
Python: Strings

- Examples



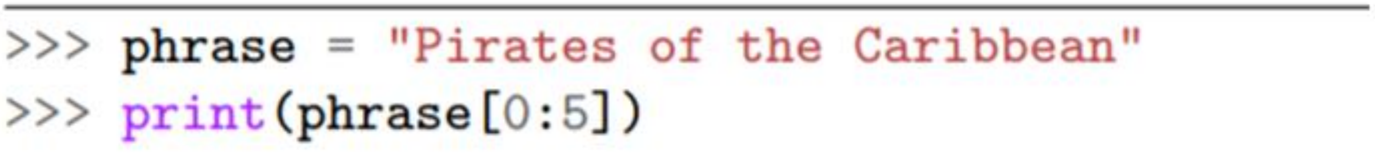
G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12





```
>>> fruit = "banana"
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
>>> fruit[2:5]
'nan'
```

Python: Strings



```
>>> phrase = "Pirates of the Caribbean"
>>> print(phrase[0:5])
```

What is the answer?

Python: Lists and tuples

- **Lists** and **Tuples** are **similar** data types. Both are used for **grouping data**.
- **Lists** are like **arrays** being **mutable**.
- **Tuples** are **immutable**, *i.e.,* the content cannot be changed.
- A **list** uses **[ ]** while **tuples** use **( ).**
- Like lists, tuples may contain any data type including other tuples.

Python: Lists and tuples

Method	Sintax	Description
`append`	`mylist.append(item)`	Adds na element in the end
`insert`	`mylist.insert(pos, item)`	Inserts an element in a given position
`pop`	`mylist.pop()`	Removes and return the last element
`pop`	`mylist.pop(pos)`	Removes and return the element in a given position
`sort`	`mylist.sort()`	Orders the list (increasing or alphabetically)
`reverse`	`mylist.reverse()`	Reverse ordering
`index`	`mylist.index(item)`	Returns the first position
`count`	`mylist.count(item)`	Counts the item in the list
`remove`	`mylist.remove(item)`	Removes the first occurrence of the item

Python: Lists and tuples

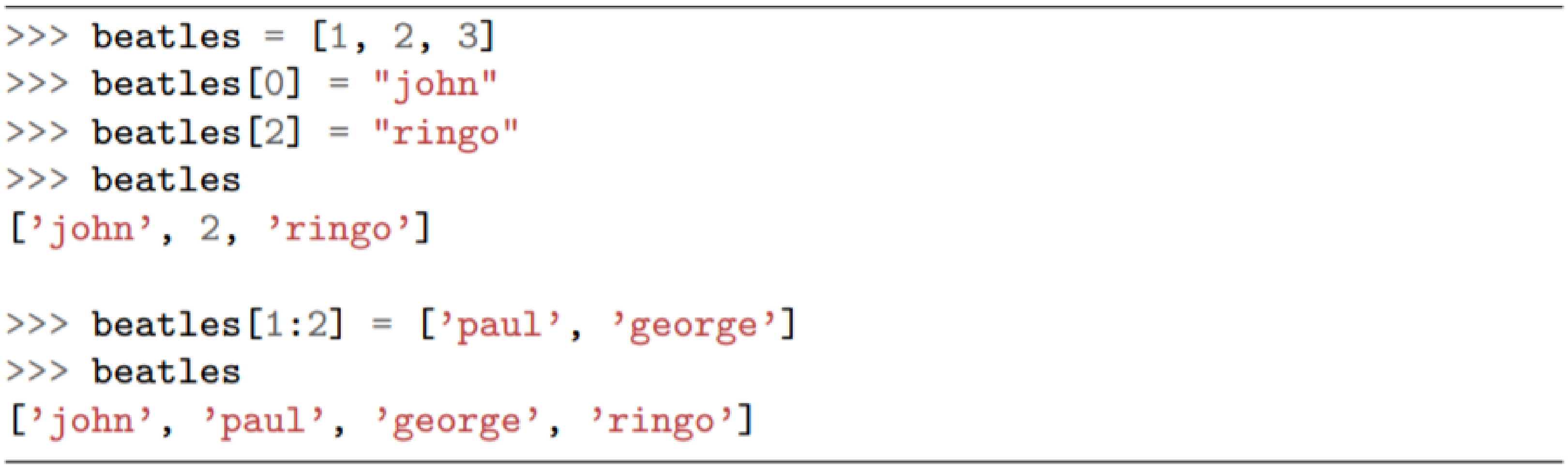
- **Tuples - Examples**

```
>>>x = ('smith','john',(6,23,68)) # this is a tuple
>>>lastname,firstname,birthname = x # unpacking the tuple
>>>print(firstname)
>>>x=(2,) # this is a tuple with a single object
>>>x[0]='p' # error. Tuples are immutable
```

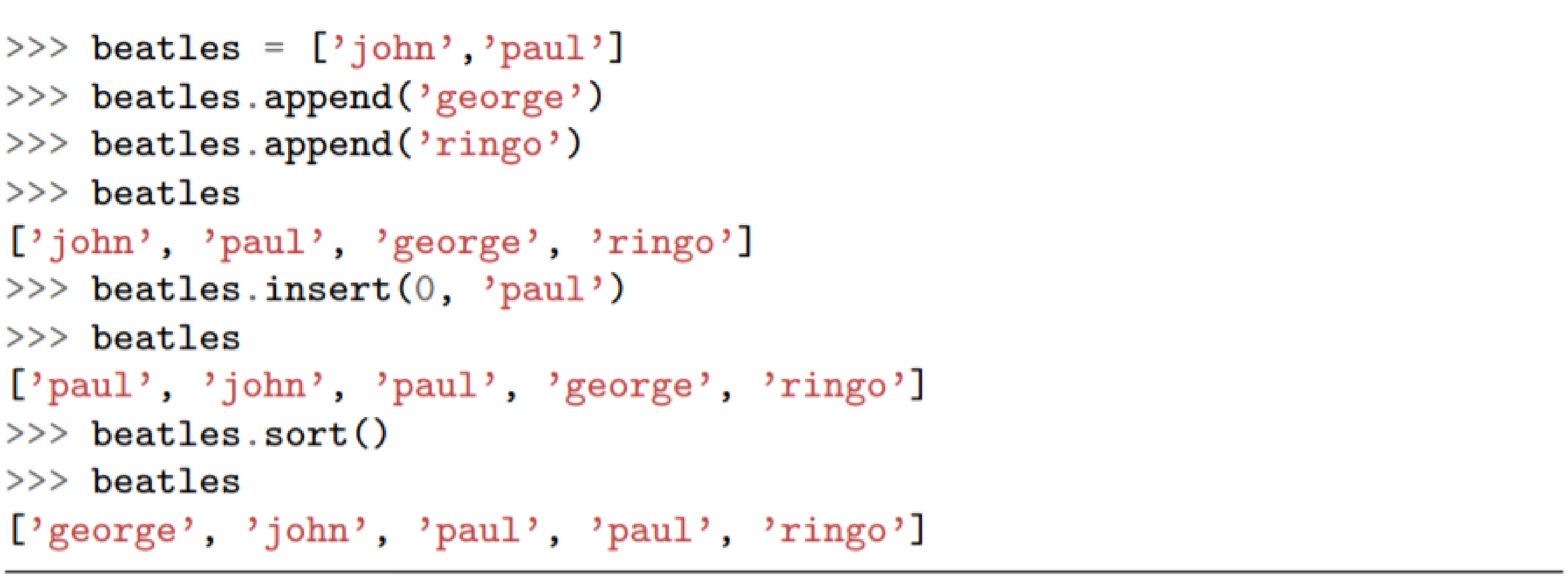
- **Lists - Examples**

```
>>>a = [1.0, 2.0, 3.0] # this is a list
>>>a.append(4.0) # adding a new element to the list
>>>a[2:4] = [1.0, 1.0, 1.0] # modify selected elements
>>>b=a # create a reference to a
>>>c=a[:] # create an independent copy of a
>>>a=[[1,2,3],[4,5,6]] # create a matrix
```

Python: Lists and tuples



```
>>> beatles = [1, 2, 3]
>>> beatles[0] = "john"
>>> beatles[2] = "ringo"
>>> beatles
[’john’, 2, ’ringo’]

>>> beatles[1:2] = [’paul’, ’george’]
>>> beatles
[’john’, ’paul’, ’george’, ’ringo’]
```



```
>>> beatles = [’john’,’paul’]
>>> beatles.append(’george’)
>>> beatles.append(’ringo’)
>>> beatles
[’john’, ’paul’, ’george’, ’ringo’]
>>> beatles.insert(0, ’paul’)
>>> beatles
[’paul’, ’john’, ’paul’, ’george’, ’ringo’]
>>> beatles.sort()
>>> beatles
[’george’, ’john’, ’paul’, ’paul’, ’ringo’]
```

Python: Dictionaries

- Dictionaries are Python's built-in mapping type.
- It maps keys (any immutable type) to values.
- One way to create a dictionary is to start with the empty dictionary and add `{key:value pairs}`.

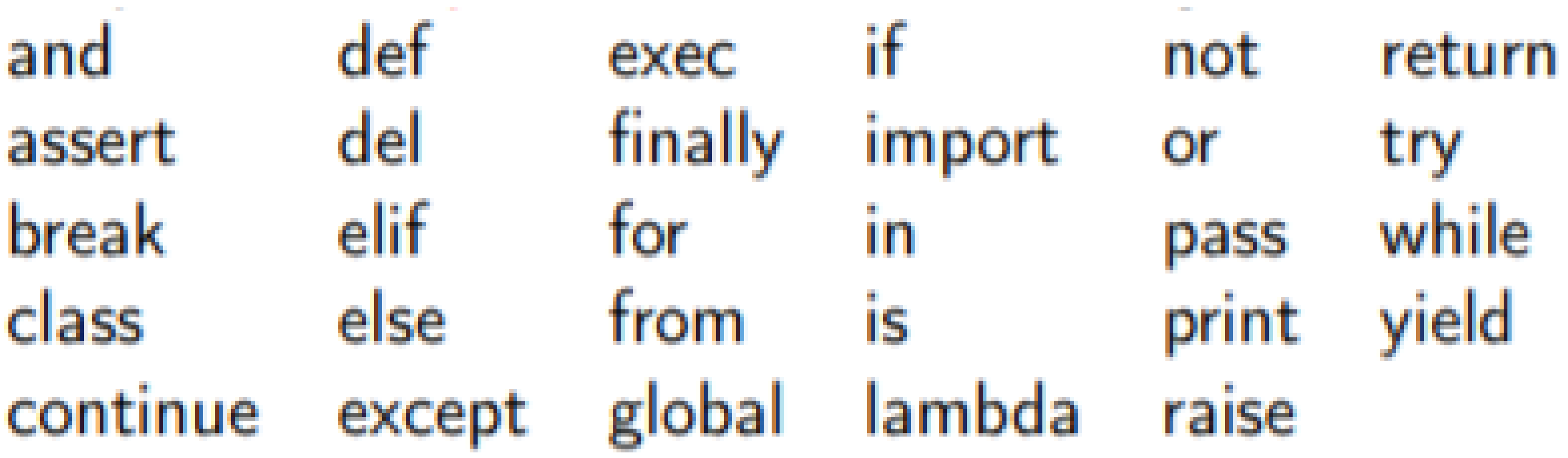
```
>>> d = dict() # empty dictionary
>>> d = { } # empty dictionary
>>> pairs = [("cow", 5), ("dog", 98), ("cat", 1)] #dict of a list of pairs
>>> d = dict(pairs)
>>> d = { "cow":5, "dog":98, "cat":1 } # dict statically allocated
```

Python: Dictionaries

Method	Sintax	Description
keys	`mydict.keys()`	Returns the dictionary keys as a list
values	`mydict.values()`	Returns the dictionary values as a list
items	`mydict.items()`	Returns the dictionary elements as a list of tuples
get	`mydict.get(key)`	Returns the value of the indicated key

Python

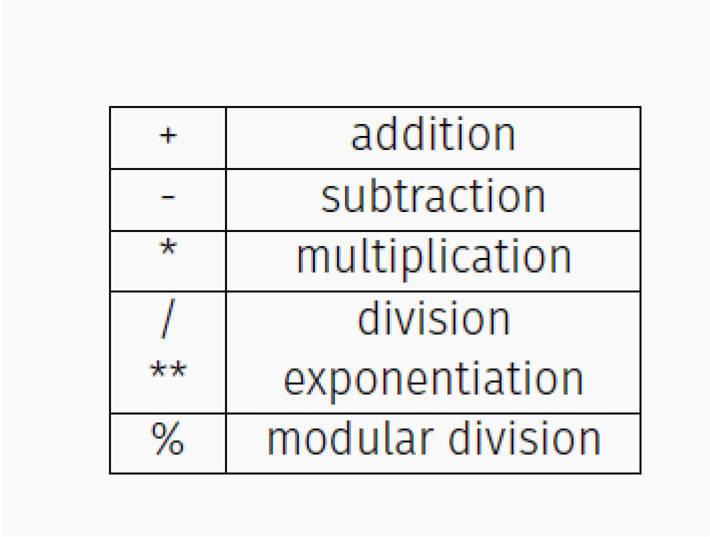
- Reserved words



and
assert
break
class
continue
def
del
elif
else
except
exec
finally
for
from
global
if
import
in
is
lambda
not
or
pass
print
raise
return
try
while
yield

Python: arithmetic operations

```
>>>s = “hello”

>>>t = “to you”

>>>print(s+t)

>>>a = [1,2,3]

>>>print(3*s)
```



+	addition
-	subtraction
*	multiplication
/ **	division exponentiation
%	modular division

Python: arithmetic operations

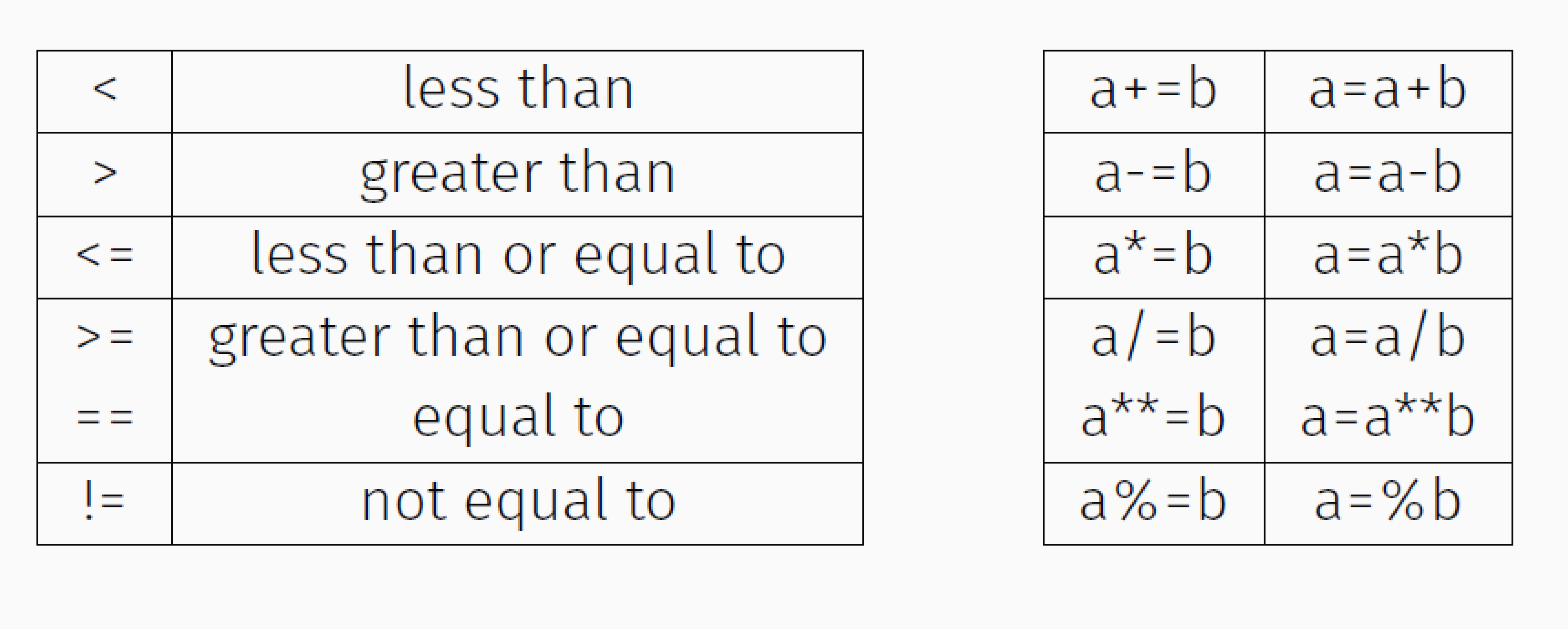
- Example: Python as calculator

- Indicate the result of the following expression using $a = 1$; $b = 1$ and $c = -\frac{1}{3}$

$$\sqrt{b^2 - 4ac}$$

```
>>>a=1
>>>b=1
>>>c=-1/3
>>>math.sqrt(b**2-4*a*c)
1.5275252316519465
```

Python: other operations



<	less than
>	greater than
<=	less than or equal to
>= ==	greater than or equal to equal to
!=	not equal to

a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b

a/=b
a**=b
	a=a/b
	a=a**b

a%=b	a=%b

Python: Conditionals

- Execute a block of statements:

```
if <condition>:
    block
```

- If the condition is False the block is skipped to:

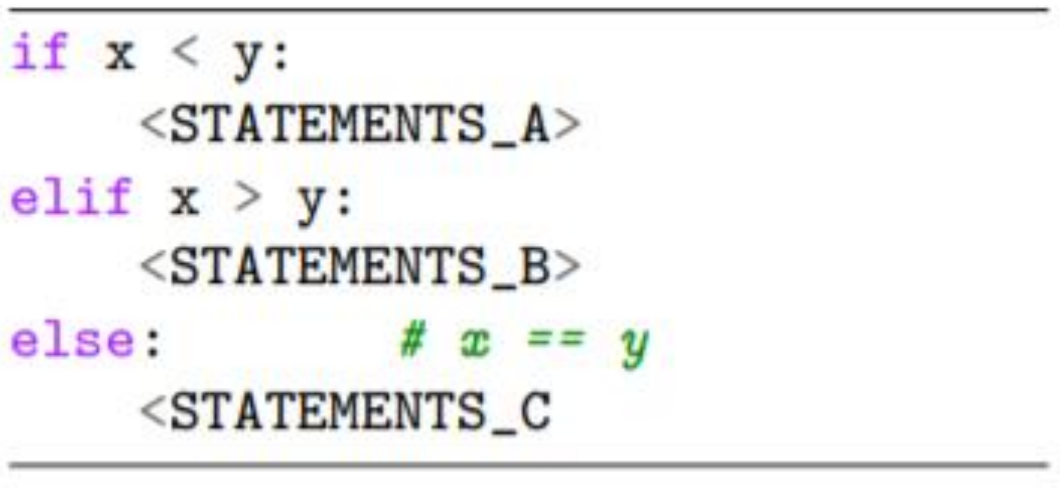
```
elif <condition>:
    block
```

- If none of previous statements are true then:

```
else:
    block
```

Python: Conditionals

- Example: Write a Python program which identifies the higher number of two values. The program should print an appropriate message.



```
if x < y:
    <STATEMENTS_A>
elif x > y:
    <STATEMENTS_B>
else:           # x == y
    <STATEMENTS_C
```

Python: Loops or cycles

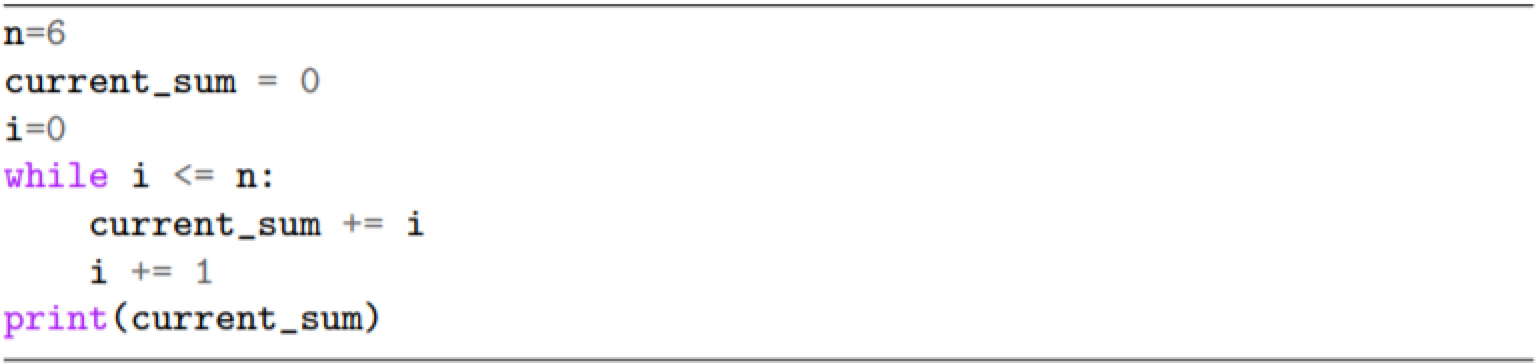
- **While**: executes a block of statements **if the condition is true**.

```
while <condition>:
    block
```

- After the execution of the block the condition is evaluated again.

- If it is still true, the block is executed again.

- This process is continued until the condition becomes false.

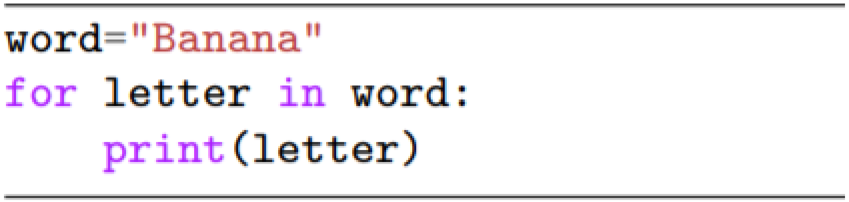


```
n=6
current_sum = 0
i=0
while i <= n:
    current_sum += i
    i += 1
print(current_sum)
```

Python: Loops or cycles

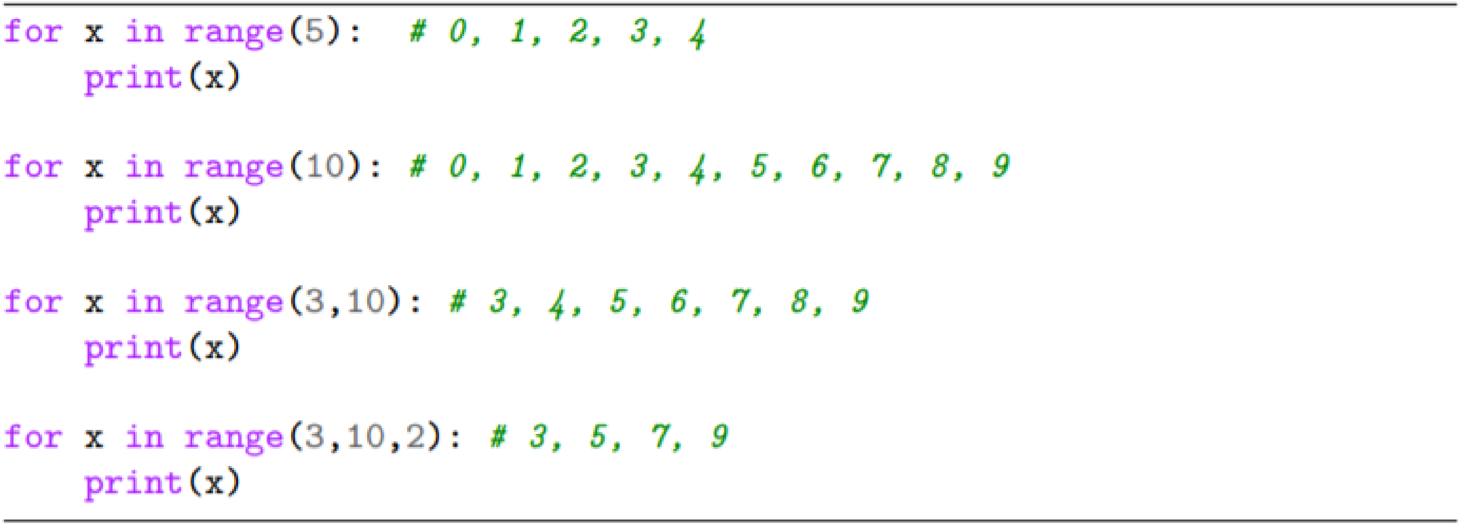
- For cycle:

```
for <target> in <sequence>:
        block
```



```
word="Banana"
for letter in word:
    print(letter)
```

- This statement requires a target and a sequence over which the target loops.



```
for x in range(5):   # 0, 1, 2, 3, 4
    print(x)

for x in range(10): # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    print(x)

for x in range(3,10): # 3, 4, 5, 6, 7, 8, 9
    print(x)

for x in range(3,10,2): # 3, 5, 7, 9
    print(x)
```

Python: Loops or cycles

- Example: Consider the following list:

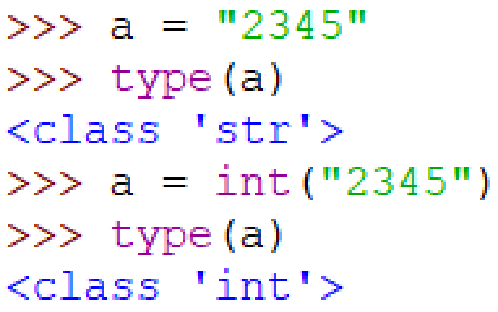
```
xs = [12, 10, 32, 3, 66, 17, 42, 99, 20]
```

Implement a python program which calculates the average of the list and identifies the highest number.

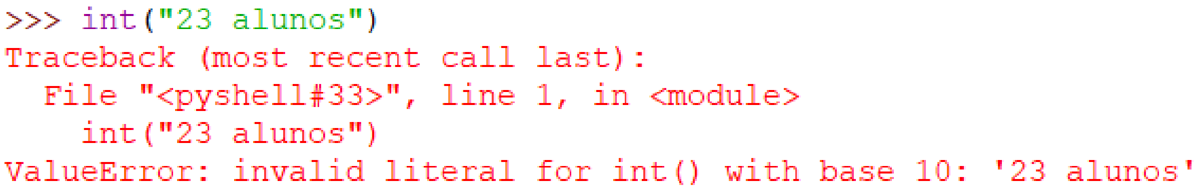
```
xs = [12, 10, 32, 3, 66, 17, 42, 99, 20]
sum=0
higher=0
for x in xs:
    sum += x
    if x>higher:
        higher=x
average = sum /len(xs)

print("The average of the list is: ", average)
print("The highest number is: ", higher)
```

Python: Type Conversion

```
>>>a=5
>>>b=-3.6
>>>c=“4”

>>>print(a+b)
>>>print(int(b))

>>>d = a + float(d)
>>>print(d)
```



```
>>> a = "2345"
>>> type(a)
<class 'str'>
>>> a = int("2345")
>>> type(a)
<class 'int'>
```



```
>>> int("23 alunos")
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    int("23 alunos")
ValueError: invalid literal for int() with base 10: '23 alunos'
```

Python: Reading input and printing output

```
>>>age = input ("Introduce your age: ")
>>>age = int(age)
>>>print("Your age is:",age)
```

Note that all **data provided by user** comes in **string format**.

Python: Reading input and printing output

- Format method:
- The **format method** substitutes its arguments into the **place holders**.
- The numbers in the **place holders** are **indexes**
- It is possible **to define** the **data type**

```
>>> phrase = "His name is {0}! ". format("Arthur")
>>> print(phrase)
His name is Arthur!
```

Python: Reading input and printing output

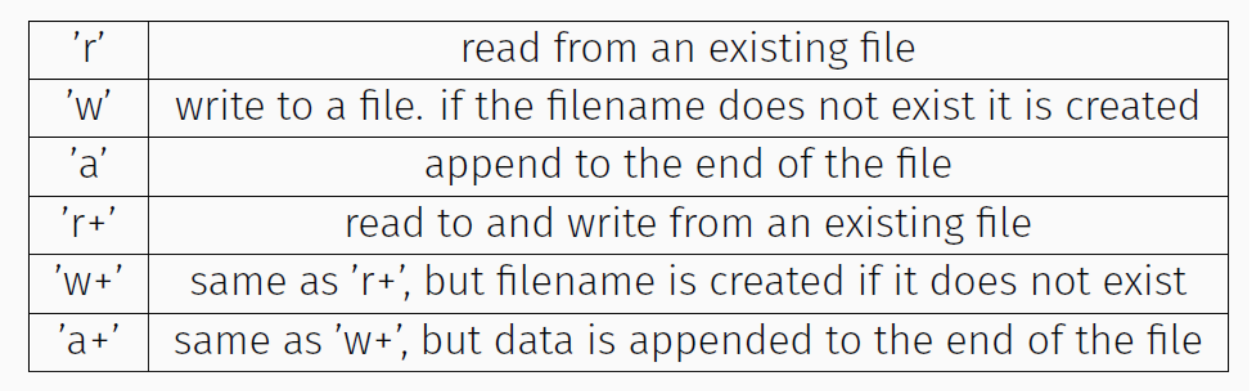
```
>>> name = "Alice"
>>> age = 10
>>> phrase = "I am {1} and I am {0} years old. ". format(age, name)
>>> print(phrase)
I am Alice and I am 10 years old.
```

```
>>> x = 4
>>> y = 5
>>> phrase = "2**10 = {0} and {1} * {2} = {3:f}". format(2**10, x, y, x * y)
>>> print(phrase)
2**10 = 1024 and 4 * 5 = 20.000000
```

Python: Opening and Closing a File

```
file = open(filename,action) #open a file
```

where action is one of the following strings:



’r’	read from an existing file
’w’	write to a file. if the filename does not exist it is created
’a’	append to the end of the file
’r+’	read to and write from an existing file
’w+’	same as ’r+’, but filename is created if it does not exist
’a+’	same as ’w+’, but data is appended to the end of the file

```
file.close() # close a file
```

Python: Functions

```
def func_name(param1,param2,...):
       statements
       return return_values
```

- A **function** is a piece of code which **executes specific tasks**
- It has a **name** and may have **parameters**
- A **parameter** can be **any python object**, including another function.
- A **parameter can have default** values.
- If the **return statement is omitted** the function returns the **null object**.

Python: Functions

- **Exercise**: Define a function which calculates the circumference area. The function should receive as parameter the circumference radius and return the corresponding area.

```
import math
def area(radius)
        res = math.pi*radius**2
        return res
```

Note that `res` is a local variable.

Python: Modules

- **A module** is a file containing **Python definitions and statements** intended for use in **other Python programs**.

- There are many Python modules that come with Python as part of the standard library.

```
import <module>
```

- math, numpy, cv2, matplotlib

Python: Matplotlib module

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python

```
import matplotlib.pyplot as plt
import numpy as np

# Example
x=np.arange(0.0,6.2,0.2)
# plot with specified line and marker style
plt.plot(x, np.sin(x),'o-’,x, np.cos(x),'^-')
plt.show()
```

Python: Numpy module

- The standard Python data types are not very suited for mathematical operations.

- Image that you need calculate the double of each position of a list. You need a for cycle for that. How to do that with numpy?

```
xs = [12, 10, 32, 3, 66, 17, 42, 99, 20]
```

```
import numpy as np
xs = [12, 10, 32, 3, 66, 17, 42, 99, 20]
xs = np.array(xs)

print(2*xs)
```

Python: Numpy module

- One of the most important properties an array is its **shape**

```
>>> import numpy as np
>>> a = np.array([2, 3, 8])
>>> a.shape
(3,)


>>> b = np.array([
[2, 3, 8],
[4, 5, 6],
])
>>> b.shape
(2, 3)
```

Python: Numpy module

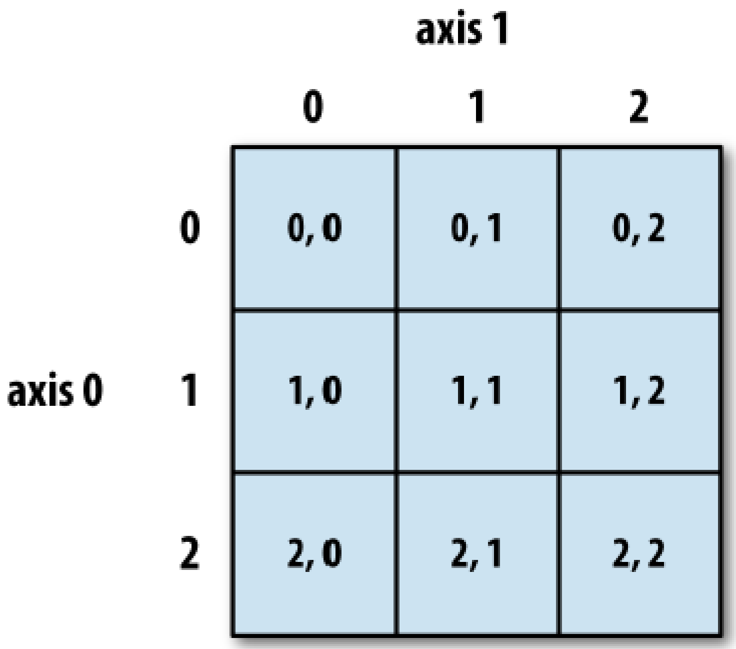
- To select certain values from an array, for 1D arrays it works just like for normal python lists:

```
>>> a = np.array([2, 3, 8])
>>> a[2]
8
>>> a[1:]
np.array([3, 8])
```

- With higher dimensional arrays:

```
>>> b = np.array([
[2, 3, 8],
[4, 5, 6],
])
>>> b[1][2] # select individual items
6
```

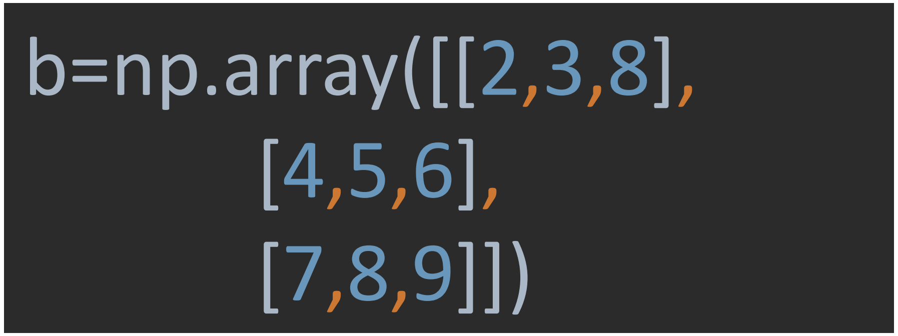
Python: Numpy module



		axis 1		
		0	1	2
	0	0, 0	0, 1	0, 2
axis 0	1	1, 0	1, 1	1, 2
	2	2, 0	2, 1	2, 2

Python: Numpy module slicing

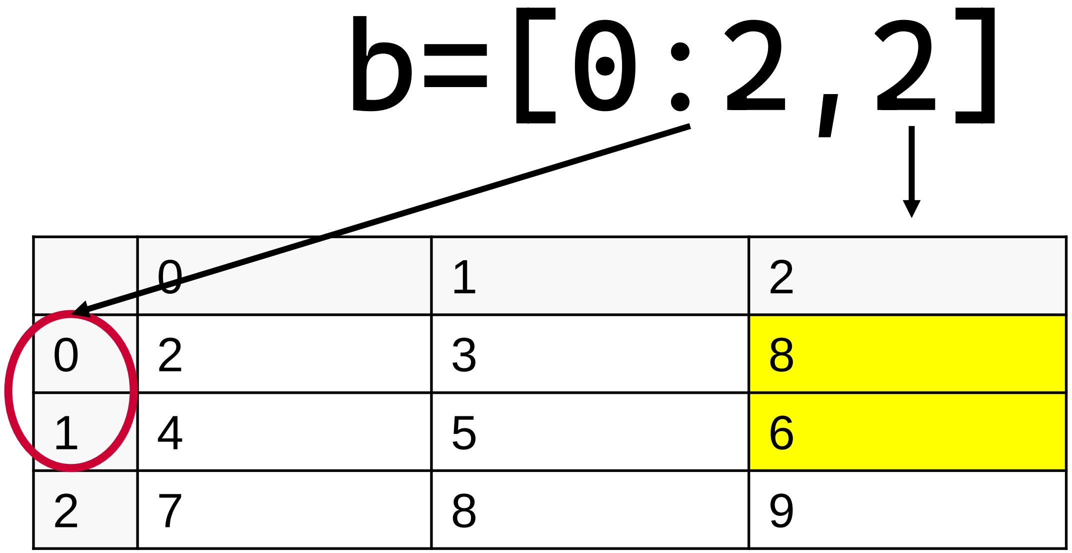
- Consider the following matrix:



```
b=np.array([[2,3,8],
            [4,5,6],
            [7,8,9]])
```

- How to access the element in row 1 and column 1?
- How to get all the elements of column 2 in rows 0 and 1?

Python: Numpy module slicing



```
b=[0:2,2]
```

	0	1	2
0	2	3	8
1	4	5	6
2	7	8	9

Python: Numpy module

```
>>>np.arange(3)
array([0, 1, 2])

>>>np.arange(3.0)
array([ 0.,  1.,  2.])

>>>np.arange(3,7)
array([3, 4, 5, 6])

>>>np.arange(3,7,2)
array([3, 5])
```

Values are generated within
the half-open interval [start, stop, step]

Python: Numpy module

shape

value

```
>>>np.full((2, 2), 10)
array([[10, 10],
       [10, 10]])
```

```
>>>np.full((2, 2), [1, 2])
array([[1, 2],
       [1, 2]])
```

Return a new array of given shape and type, filled with *fill_value*.

```
>>> np.zeros((3, 6))
array([[ 0., 0., 0., 0., 0., 0.],
[ 0., 0., 0., 0., 0., 0.],
[ 0., 0., 0., 0., 0., 0.]])
```

Return a new array of given shape and type, filled with *0*.

```
>>>np.ones((3, 6))
```

Python: Numpy module

```
>>>shape=(1,9)
>>>b=np.ones(shape)
[[1. 1. 1. 1. 1. 1. 1. 1. 1.]]

>>>b=np.reshape(b,(3,3))
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Gives a new shape to an array without changing its data.

Python: Numpy module

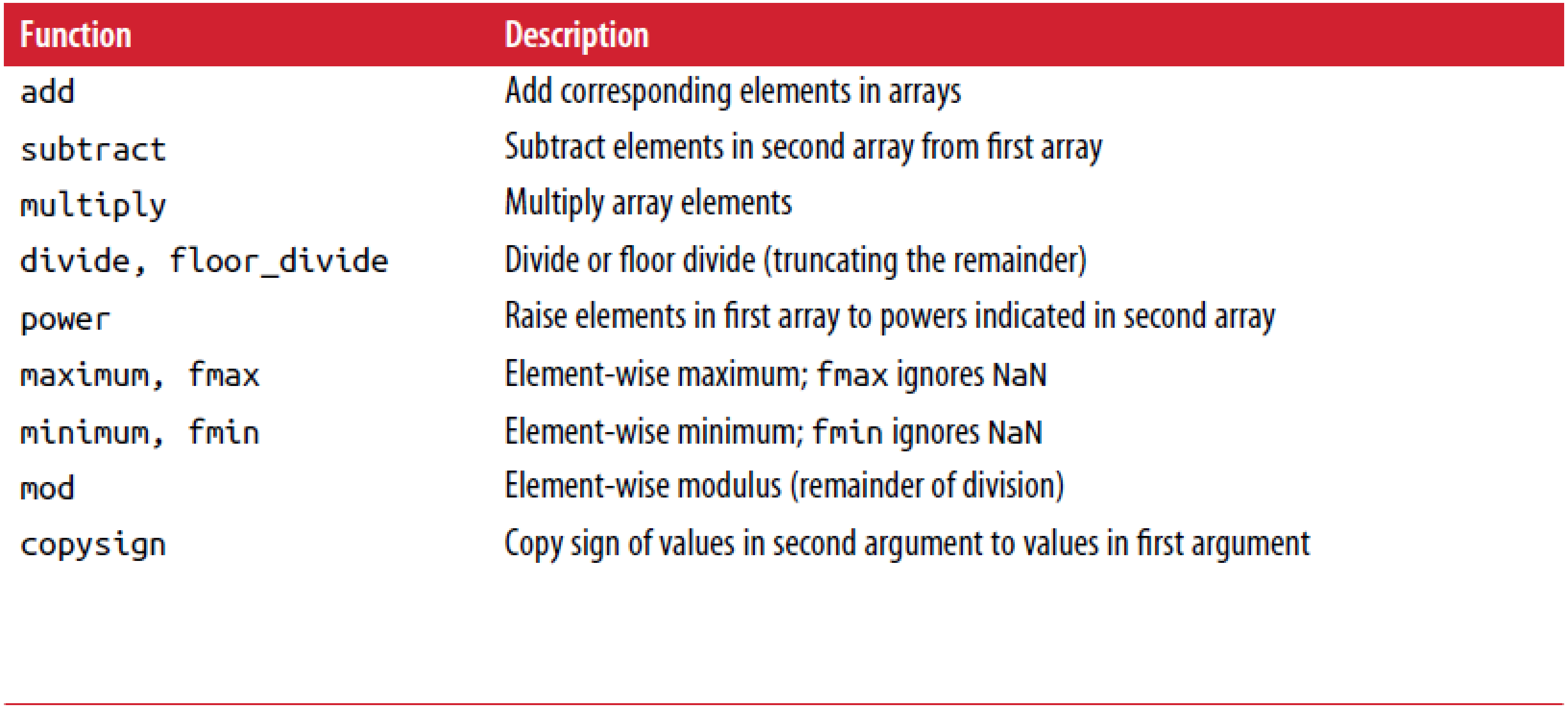
- **Masking** is one of the most powerful features of numpy
- Suppose, we have an array, and we want to throw away all values above a certain cut-off:

```
>>> a = np.array([230, 10, 284, 39, 76])
>>> cutoff = 200
>>> a > cutoff
np.array([True, False, True, False, False])
```

Exercise: Substitute by 0 all values in the matrix higher than 200

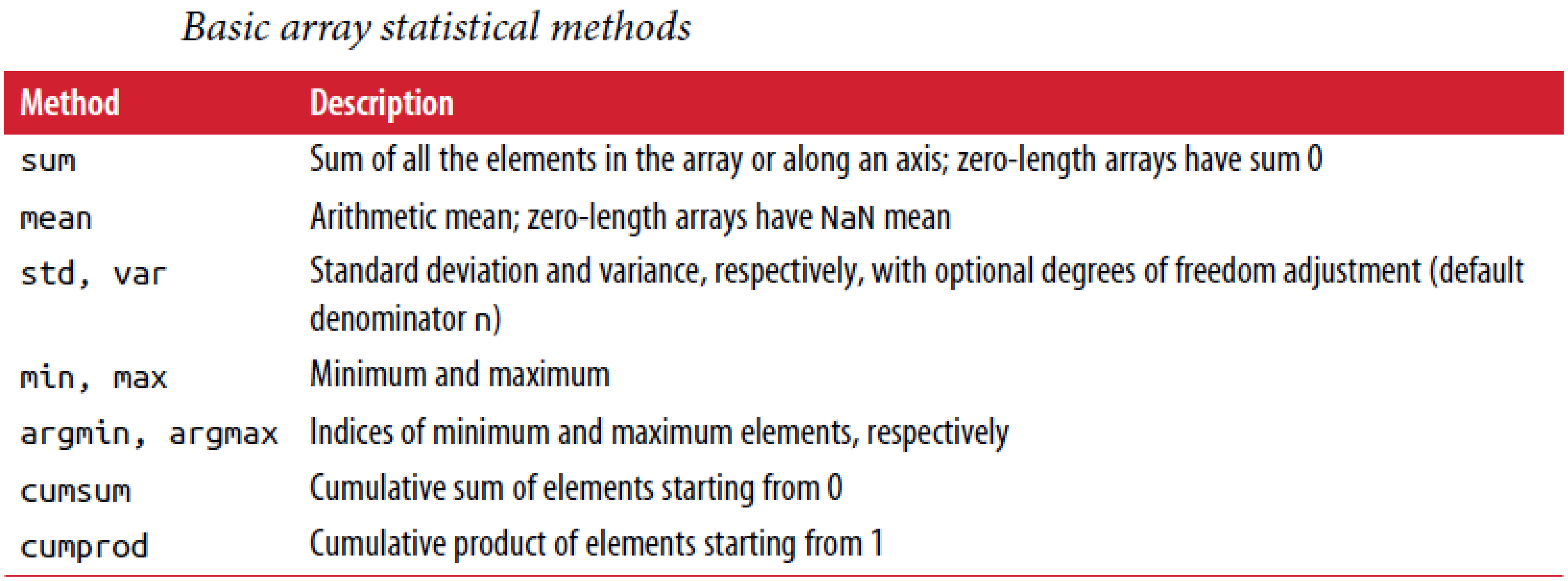
```
>>> a[a > cutoff] = 0
>>> a
np.array([0, 10, 0, 39, 76])
```

Python: Numpy module



Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum; `fmax` ignores `NaN`
`minimum, fmin`	Element-wise minimum; `fmin` ignores `NaN`
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument

Python: Numpy module

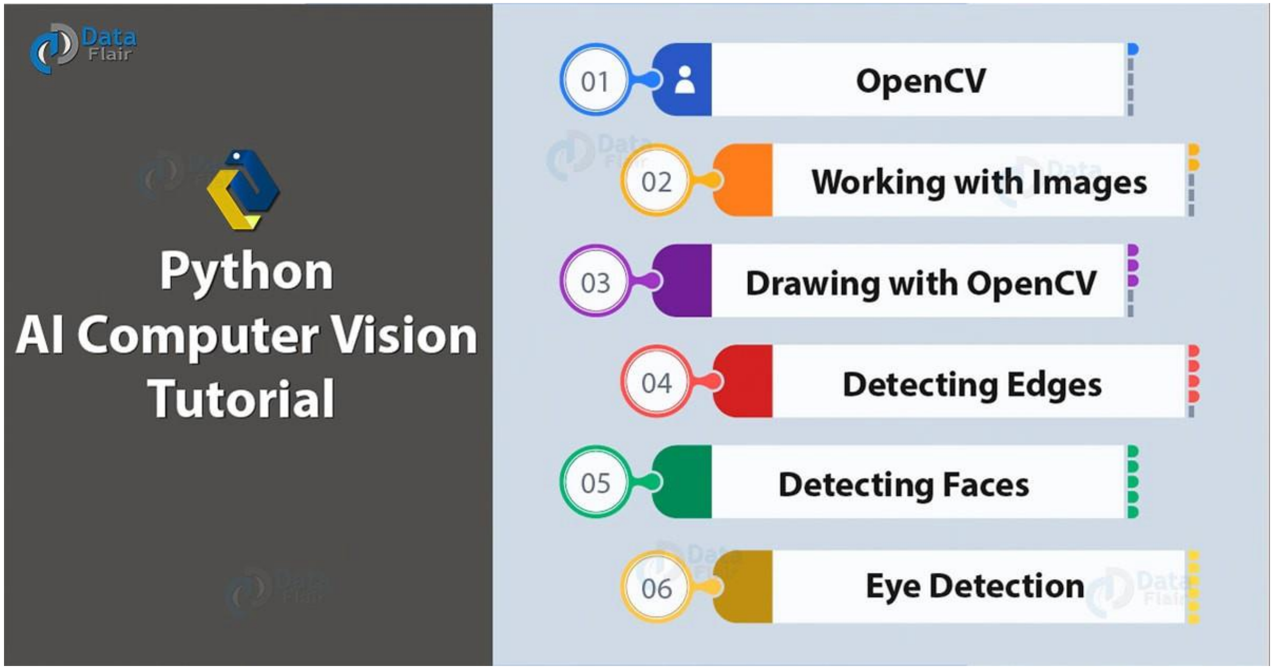


Basic array statistical methods

Method	Description
`sum`	Sum of all the elements in the array or along an axis; zero-length arrays have sum 0
`mean`	Arithmetic mean; zero-length arrays have `NaN` mean
`std, var`	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator `n`)
`min, max`	Minimum and maximum
`argmin, argmax`	Indices of minimum and maximum elements, respectively
`cumsum`	Cumulative sum of elements starting from 0
`cumprod`	Cumulative product of elements starting from 1

Python: OpenCV Module

- OpenCV Python is a library designed to solve computer vision problems
- During the semester we will learn to use it.



Python: Exercise

- With the beginning of the year, you decide to get in shape and lose some weight. You record your weight every day for five weeks starting on a Monday.

```
import numpy as np
dailywts = 100 - np.arange(5*7)/5
print(dailywts)
```

- Given these daily weights, build an array with your average weight per week. How much weight you have loose per week?

Python: Exercise

- From 2 numpy arrays, extract the indexes in which the elements in the 2 arrays match

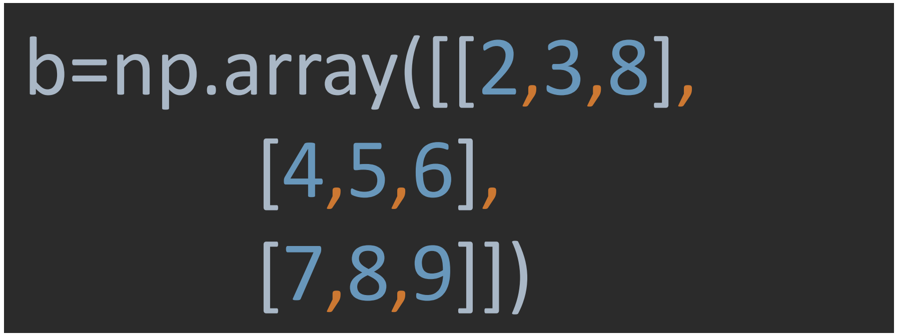
```
a = np.array([1,2,3,4,5])
b = np.array([1,3,2,4,5])
```

- Using the previous arrays, find out the purpose of the following numpy operations
  - vstack
  - hstack

Let’s code!

Python: Exercise

- Using the following array, find out the purpose of the following numpy operations:
  - ndim
  - size
  - Shape
  - Sum()
  - max()
  - mean()
  - std()
  - np.where(b>6)



```
b=np.array([[2,3,8],
            [4,5,6],
            [7,8,9]])
```

Consider the next array and print the total missing values in an array

```
p = np.array([5,10, np.nan, 3, 2, 5, 6, np.nan])
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.