

# Scientific Programming in Python

---

Francesco Mannella

Institute of Cognitive Sciences and technologies - CNR

# Outline

- |                      |                       |
|----------------------|-----------------------|
| ① Python basics      | ⑥ Functions and Lists |
| ② Numeric types      | ⑦ Tuples              |
| ③ Operators          | ⑧ References          |
| ④ Strings            | ⑨ Dictionaries        |
| ⑤ Control Statements | ⑩ Custom objects      |

# Outline

- |                      |                       |
|----------------------|-----------------------|
| 1 Python basics      | 6 Functions and Lists |
| 2 Numeric types      | 7 Tuples              |
| 3 Operators          | 8 References          |
| 4 Strings            | 9 Dictionaries        |
| 5 Control Statements | 10 Custom objects     |

# Python basics - Python's scientific ecosystem



**Python**



**NumPy**

Base N-dimensional array



**SymPy**

Symbolic mathematics

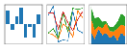


**Matplotlib**

Comprehensive plotting library

**pandas**

$\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n = y$



**SciPy**

Library for scientific computing



**scikit-learn**

Machine Learning in Python

**nest ::**

**PyNEST**

python interface for NEST:  
Neural Simulation Tool

# Python basics - the interpreter

Just type "python" in the command line to open:

```
$ python
Python 2.7.12 (default, Nov 19 2016,
    06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "
    license"
for more information.
>>>
```

It can be used as a calculator:

```
>>> 4 + 5
9
>>> _
```

You can execute single lines of code

```
>>> a = 5
>>> b = 3
>>> a + b
8
>>>
```

You can also run a control statement:

```
>>> for i in range(10):
...     print i**2
d..
0
1
4
9
16
25
36
49
64
81
>>>
```

- |                        |                       |
|------------------------|-----------------------|
| 1 Python basics        | 6 Functions and Lists |
| <b>2 Numeric types</b> | 7 Tuples              |
| 3 Operators            | 8 References          |
| 4 Strings              | 9 Dictionaries        |
| 5 Control Statements   | 10 Custom objects     |

# Numeric types - hands-on code

Code: find if a number is a prime number

```
1 # Python program to check if the input number
2 # is prime or not
3
4 from __future__ import print_function
5 from __future__ import division
6
7 num = 17
8
9 # take input from the user
10 # num = int(input("Enter a number: "))
11
12 # prime numbers are greater than 1
13 if num > 1:
14     # check for factors
15     for i in range(2,num):
16         if (num % i) == 0:
17             print(num,"is not a prime number")
18             print(i,"times",num//i,"is",num)
19             break
20     else:
21         print(num,"is a prime number")
22
23 # if input number is less than
24 # or equal to 1, it is not prime
25 else:
26     print(num,"is not a prime number")
```

```
$ python prime.py
407 is not a prime number
11 times 37 is 407
```

# Numeric types

## # Boolean

```
a = True  
b = False
```

## # Integer

```
a = 34  
b = 45//3  
c = int(34/2)  
d = int(3.14)  
e = a + b//d
```

## # Float

```
a = 2.3  
b = 45/3  
c = float(34)  
d = b/a
```

## # Complex

```
a = complex(1, 2)  
b = 3 - 4.5j  
c = a**2 + b  
d = c.real() # float  
e = c.imag() # float
```



- ① Python basics
- ② Numeric types
- ③ **Operators**
- ④ Strings
- ⑤ Control Statements
- ⑥ Functions and Lists
- ⑦ Tuples
- ⑧ References
- ⑨ Dictionaries
- ⑩ Custom objects

# Operators - arithmetic operators

**a** = 10; **b** = 20

**a** + **b** == 30

**Addition**

Adds values on either side of the operator.

**b** - **a** == 10

**Subtraction**

Subtracts right hand operand from left hand operand.

**a** \* **b** == 200

**Multiplication**

Multiplies values on either side of the operator.

**b** / **a** == 2

**Division**

Divides left hand operand by right hand operand.

**a** / 4.0 == 2.5

**b** % **a** == 0

**Modulus**

Divides left hand operand by right hand operand and returns remainder.

**a** / 4.0 == 2.5

**a** \*\* **b** == 10

**Exponent**

Divides left hand operand by right hand operand.

**a** // 4.0 == 2

**Floor Division**

The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

**Assignment operators:** = += -= \*= /= //= %= \*\*=

# Operators - comparison operators

<b>a == b</b>	<b>False</b>	Is <i>a</i> equal to <i>b</i> ?
<b>a != b</b> <b>a &lt;&gt; b</b>	<b>True</b>	Is <i>a</i> different from <i>b</i> ?
<b>a &lt; b</b>	<b>True</b>	Is <i>a</i> less than <i>b</i> ?
<b>a &gt; b</b>	<b>False</b>	Is <i>a</i> greater than <i>b</i> ?
<b>a &lt;= b</b>	<b>True</b>	Is <i>a</i> less than or equal to <i>b</i> ?
<b>a &gt;= b</b>	<b>False</b>	Is <i>a</i> greater than or equal to <i>b</i> ?

# Operators - logical operators

```
a = True  
b = False
```

a <b>and</b> b	False	are both <i>a</i> and <i>b</i> true?
----------------	-------	--------------------------------------

a <b>or</b> b	True	is <i>a</i> or <i>b</i> true?
---------------	------	-------------------------------

<b>not</b> a	False	is not <i>a</i> true?
--------------	-------	-----------------------

- |                      |                       |
|----------------------|-----------------------|
| 1 Python basics      | 6 Functions and Lists |
| 2 Numeric types      | 7 Tuples              |
| 3 Operators          | 8 References          |
| <b>4 Strings</b>     | 9 Dictionaries        |
| 5 Control Statements | 10 Custom objects     |

# Strings

```
name = "Federico"  
surname = "Rossi"  
street = 'via' + 'A. De Sanctis'  
number = 6  
zipcode = 100  
city = "Roma"
```

```
address = ""  
Federico Rossi  
via A. De Sanctis, 6  
00100 Roma  
""
```

```
address0 = "%s %s\n%s, %d\n%05d %s" % (name,  
    surname, street, number, zipcode, city)
```

```
address1 = "{} {} \n {}, {} \n{:05d} {}".format(  
    name, surname, street, number, zipcode,  
    city)
```

```
address2 = ""  
{ } { }  
{ } { }  
{:05d} { }  
"".format(name, surname, street, number,  
    zipcode, city )
```

```
address3 = r "{} {} \n {}, {} \n{:05d} {} \n".format(  
    name, surname, street, number, zipcode,  
    city)
```

```
>>> print(address0)  
Federico Rossi  
via A. De Sanctis, 6  
00100 Roma  
>>>  
>>> print(address1)  
Federico Rossi  
via A. De Sanctis, 6  
00100 Roma  
>>>
```

```
>>> print(address2)  
Federico Rossi  
via A. De Sanctis, 6  
00100 Roma  
>>> print(address3)  
Federico Rossi\nvia A. De Sanctis, 6\n00100  
Roma  
>>> _
```

# Outline

- |                      |                       |
|----------------------|-----------------------|
| ① Python basics      | ⑥ Functions and Lists |
| ② Numeric types      | ⑦ Tuples              |
| ③ Operators          | ⑧ References          |
| ④ Strings            | ⑨ Dictionaries        |
| ⑤ Control Statements | ⑩ Custom objects      |

# Control statements

## IF statement

```
1 print_on = True
2 if print_on :
3     print('print this!')
4
5 if print_on is True:
6     print('print this!')
```

```
1 if a > 0.5:
2     b += a
```

```
1 a = 7
2 if a < 5:
3     print('less than 5')
4 elif 5 <= a < 10:
5     print('between 5 and 10')
6 elif 10 <= a <= 15:
7     print('between 10 and 15')
8 else:
9     print('greater than 15')
```

## FOR loops

```
1 for i in range(10):
2     print(i)
```

```
1 for i in range(10):
2     if i == 8:
3         continue
4     print(i)
```

```
1 a = []
2 for i in range(5):
3     a += i
4
5 a == [0, 1, 2, 3, 4]
```

## WHILE loops

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
```

```
1 i = 1
2 while i < 6:
3     print(i)
4     if i == 3:
5         break
6     i += 1
```



- |                      |                       |
|----------------------|-----------------------|
| ① Python basics      | ⑥ Functions and Lists |
| ② Numeric types      | ⑦ Tuples              |
| ③ Operators          | ⑧ References          |
| ④ Strings            | ⑨ Dictionaries        |
| ⑤ Control Statements | ⑩ Custom objects      |

# Functions and Lists - hands-on code

The quicksort algorithm:

```
1 from __future__ import print_function
2 from __future__ import division
3
4 def quickSort(arr):
5     """
6     The QuickSort algorithm is an
7     efficient sorting algorithm,
8     serving as a systematic method for
9     placing the elements of an array
10    in order.
11
12    Args:
13        :param arr: a list containing the
14                    elements to sort
15        :return: the sorted list of arguments
16    """
17    less = []
18    pivotList = []
19    more = []
20    if len(arr) <= 1:
21        return arr
22    else:
```

```
23         pivot = arr[0]
24         for i in arr:
25             if i < pivot:
26                 less.append(i)
27             elif i > pivot:
28                 more.append(i)
29             else:
30                 pivotList.append(i)
31         less = quickSort(less)
32         more = quickSort(more)
33         return less + pivotList + more
34
35 if __name__ == "__main__":
36     a = [4, 65, 2, -31, 0, 99, 83, 782, 1]
37     a = quickSort(a)
38     print(a)
```

```
$ python quicksort.py
[-31, 0, 1, 2, 4, 65, 83, 99, 782]
```

# Lists

Lists:

- are ordered
- can be filled with objects of all types
- can be nested
- are mutable
- are dynamic

```
a = []  
b = [1, 2, 3, 4, 5]  
c = ['one', 2.0, 3, True, [1, 2]]
```

You can append new items:

```
a.append(3)  
a == [1, 2, 3, 4, 5, 3]
```

Recall elements and slices:

```
b[0] == 1  
c[:2] == ['one', 2.0]  
c[1:3] == [2.0, 3]  
c[-1] == [1, 2]  
c[-3:] == [3, True, [1, 2]]
```

```
d = range(6)  
d == [0, 1, 2, 3, 4, 5]
```

```
e = range(2, 7)  
e == [2, 3, 4, 5, 6]
```

```
f = range(2, 9, 2)  
f == [2, 4, 6, 8]
```

```
f[::-1] == [8, 6, 4, 2]  
d[::-2] == [5, 3, 1]
```

```
for i in c:  
    print(i)
```

Strings are a special kind of list:

```
g = 'This is a string'  
g[-5:] == 'tring'
```

You can split a string into a list of strings:

```
h = f.split(' ')  
h == ['This', 'is', 'a', 'string']
```

- |                      |                       |
|----------------------|-----------------------|
| 1 Python basics      | 6 Functions and Lists |
| 2 Numeric types      | <b>7 Tuples</b>       |
| 3 Operators          | 8 References          |
| 4 Strings            | 9 Dictionaries        |
| 5 Control Statements | 10 Custom objects     |

# Tuples - hands-on code

## Code: A simple perceptron

```
1 data = [[0.44, 0.83], 0],
2         [0.83, 0.66], 1],
3         [0.52, 0.83], 0],
4         [0.84, 0.55], 1],
5         [0.71, 0.92], 0],
6         [0.51, 0.15], 1],
7         [0.24, 0.35], 0],
8         [0.34, 0.43], 0],
9         [0.29, 0.81], 0],
10        [0.66, 0.3 ], 1]]
11
12 epochs = 2
13 eta = 1.0
14
15 weights = [0, 0]
16
17 def step_fun(x):
18
19     if x > 0:
20         return 1
21     else:
22         return 0
23
24
25 for item in data:
26     inp, lab = item
27
28     # potential
29     pot = 0
30     for i, x in enumerate(inp):
31         pot += x * weights[i]
32
33     # activation
34     act = step_fun(pot)
35
36     # learn
37     for i, x in enumerate(inp):
38         weights[i] += eta * x * (lab - act)
39
40
41 print(lab, act, weights)
42 print("")
```

# Tuples

Tuples are used to initialize many objects at once

```
1  a, b = (2, 3)
2  c, _, d = 4, 3, 1
```

A function can return more than one element by packing objects in a tuple

```
1  def division(numerator, denominator):
2      res = numerator // denominator
3      remainder = numerator % denominator
4      return res, remainder
5
6  n, _ = division(23, 4)
7  print(n) # n == 5
8
9  t = division(14, 5)
10 print(t[0])    # t[0] == 2
11 print(t[1])    # t[1] == 4
```

Tuples are immutable:

```
1  a = ['one', 2, 3.0, 'four']
2  del a[1]    # Correct. a becomes
3              # ['one', 3.0, 'four']
4  a[2] = 45 # Correct, a becomes
5              # ['one', 3.0, 45]
6
7  b = (0, 45, 'giallo', 6.0)
8  del b[0]    # Error!! cannot delete
9              # elements
10 b[1] = 'new' # Error!! cannot change values
```

# Zip

Both **lists** and **tuples** (and **strings**) are **containers**. A container is an object that contains **references** to other objects. Containers can be iterated upon (they are also called **iterables**), meaning that you can traverse through all the values.

The **zip()** function takes iterables (can be zero or more), makes an iterator that aggregates elements based on the iterables passed, and returns an iterable of tuples.

```
1 days = [27, 28, 30]
2 weekdays = ('Monday', 'Tuesday', 'Thursday')
```

```
>>>zip(index, weekdays)
[(27, 'Monday'), (28, 'Tuesday'), (30, 'Thursday')]
```

**zip()** is often used in for loops:

```
1 for i, wd in zip(index, weekdays):
2     print i, wd, weekdays[i]
```

# Enumerate

When you iterate over a list or tuple you often need to have both the value of each element and its position in the iterable.

```
1 sentence = 'This is a string'
2 for i, ch in zip(range(len(sentence)),
3                 sentence):
4     if ch == ' ':
5         print 'character %d is a space' % i
```

In those case the `enumerate` function simplifies the code:

```
7 for i, ch in enumerate(sentence):
8     if ch == ' ':
9         print 'character %d is a space' % i
```



# More on functions, abstraction - hands-on code

Code: you can separate code parts in functions

```
12 epochs = 2
13 eta = 1.0
14
15 weights = [0, 0]
16
17 def step_fun(x):
18
19     if x > 0:
20         return 1
21     else:
22         return 0
23
24 for epoch in range(epochs):
25     for item in data:
26         inp, lab = item
27
28         # potential
29         pot = 0
30         for i, x in enumerate(inp):
31             pot += x * weights[i]
32
33         # activation
34         act = step_fun(pot)
35
36         # learn
37         for i, x in enumerate(inp):
38             weights[i] += eta * x * (lab - act)
39
40
41         print(lab, act, weights)
42         print("")
```

# More on functions, abstraction - hands-on code

Code: you can separate code parts in functions

```
12 epochs = 2
13 eta = 0.01
14
15 weights = [0, 0]
16
17 def step_fun(x):
18
19     if x > 0:
20         return 1
21     else:
22         return 0
23
24 def wsum(vec, weights):
25     res = 0
26     for i, x in enumerate(vec):
27         res += x*weights[i]
28     return res
29
30 def learn(eta, inp, out, teach, weights):
31     for i, x in enumerate(inp):
32         weights[i] += eta * x * (teach - out)
33
34
35 for epoch in range(epochs):
36     for item in data:
37         inp, lab = item
38
39         # potential
40         pot = wsum(inp, weights)
41
42         # activation
43         act = step_fun(pot)
44
45         # learn
46         learn(eta, inp, act, lab, weights)
47
48         print(lab, act, weights)
49         print("")
```

- |                      |                       |
|----------------------|-----------------------|
| 1 Python basics      | 6 Functions and Lists |
| 2 Numeric types      | 7 Tuples              |
| 3 Operators          | <b>8 References</b>   |
| 4 Strings            | 9 Dictionaries        |
| 5 Control Statements | 10 Custom objects     |

# Value vs. reference assignment

- Immutable types (bool, int, float, complex, tuple) can be only passed by value.
- Mutable types (list, dictionary, custom objects) are passed by reference.

```
7 a = 1
8 b = a
9
10 a = 34444
```

```
>>>print(a)
1
>>>print(b)
3444
```

```
7 a = [1, 2, 3]
8 b = a
9
10 a[2] = 34444
```

```
>>>print(a)
[1, 2, 3444]
>>>print(b)
[1, 2, 3444]
```

```
7 def foo(arg):
8     arg += 99
9
10 a = 1
11 foo(a)
```

```
>>>print(a)
1
```

```
7 def foo(cont):
8     for i in range(len(cont)):
9         cont[i] += 99
10
11 a = [1, 2, 3]
12 foo(a)
```

```
>>>print(a)
[100, 101, 102]
```

- |                      |                       |
|----------------------|-----------------------|
| ① Python basics      | ⑥ Functions and Lists |
| ② Numeric types      | ⑦ Tuples              |
| ③ Operators          | ⑧ References          |
| ④ Strings            | ⑨ <b>Dictionaries</b> |
| ⑤ Control Statements | ⑩ Custom objects      |

# Dictionaries - hands-on code

Code: building a function for creating histograms

```
1 from __future__ import print_function
2 from __future__ import division
3
4 def histogram(data, n_bins=10):
5     ''' Create an histogram
6         :param data: A list with all values
7         :n_bins: Classes of data
8     '''
9
10    # Find the minimum value
11    min_num = min(data)
12
13    # Find the maximum value
14    max_num = max(data)
15
16    # Compute the range of each bin
17    gap = (max_num - min_num) / n_bins
18
19    # Compute the limits of bins
20    bin_lims = []
21    for bin_el in range(n_bins):
22        bin_lims.append([
23            min_num + bin_el * gap,
24            min_num + (bin_el + 1) * gap])
25
26    # Compute the frequency for each bin
27    freqs = {}
28    for el in data:
29        for i, lims in enumerate(bin_lims):
30            if lims[0] <= el < lims[1]:
31                if i in freqs.keys():
32                    freqs[i] += 1
33                else:
34                    freqs[i] = 1
35
36    # Sum of frequencies
37    tot = sum(freqs.values())
38
39    # Plot the histogram
40    for idx, freq in freqs.items():
41        # Compute the proportion in each bin
42        prop = freq / tot
43        # Each star in the string is 1% of
44        # values
45        stars = ("*" * int(100*(prop)))
46        # Put together all params for printing
47        els = bin_lims[idx] + [freq, stars]
48        # It must be a tuple (not a list)
49        els = tuple(els)
50        # Fill the format string and print it
51        print("%5.2f <-> %5.2f: %#3d %s" % els)
52
53    return freqs, bin_lims
```

# Dictionaries - hands-on code

Code: building a function for creating histograms

```
54 if __name__ == "__main__":
55
56     # Load data from file
57     data = []
58     with open("hist_data.txt", "r") as datafile:
59         for line in datafile.readlines():
60             data.append(float(line))
61
62     # make histogram
63     histogram(data)
```

```
>>> python hist.py
11.34 <-> 12.00: 1
12.00 <-> 12.66: 10 *
12.66 <-> 13.32: 24 **
13.32 <-> 13.98: 117 *****
13.98 <-> 14.64: 205 *****
14.64 <-> 15.30: 260 *****
15.30 <-> 15.95: 237 *****
15.95 <-> 16.61: 95 *****
16.61 <-> 17.27: 41 ****
17.27 <-> 17.93: 9
```

# Dictionaries

## Dictionaries

- are **iterables**
- are maps between keys and values
- keys can be of any non-iterable type
- values can be of any non-iterable type
- Each key is unique

### Initializing:

```
1 a = {}  
2 b = {'one': 232, 'two': 2.3}
```

### Fill up:

```
1 a[1] = 3  
2 a['new'] = 0.04  
3 # a == {1: 3, 'new': 0.04}
```

### Get keys:

```
1 k = a.keys()  
2 # k == [1, 'new']
```

### Get values:

```
1 v = a.values()  
2 # v == [3, 0.04]
```

### Iterate through keys-value pairs:

```
1 for k, v in a.items():  
2     print('{}: {}'.format(k, v))
```



- 1 Python basics
- 2 Numeric types
- 3 Operators
- 4 Strings
- 5 Control Statements
- 6 Functions and Lists
- 7 Tuples
- 8 References
- 9 Dictionaries
- 10 Custom objects

# Custom objects - hands-on code

Code: creating new types of objects

```
12 epochs = 2
13 eta = 0.01
14
15 weights = [0, 0]
16
17 def step_fun(x):
18
19     if x > 0:
20         return 1
21     else:
22         return 0
23
24 def wsum(vec, weights):
25     res = 0
26     for i, x in enumerate(vec):
27         res += x*weights[i]
28     return res
29
30 def learn(eta, inp, out, teach, weights):
31     for i, x in enumerate(inp):
32         weights[i] += eta * x * (teach - out)
```

```
35 for epoch in range(epochs):
36     for item in data:
37         inp, lab = item
38
39         # potential
40         pot = wsum(inp, weights)
41
42         # activation
43         act = step_fun(pot)
44
45         # learn
46         learn(eta, inp, act, lab, weights)
47
48     print(lab, act, weights)
49     print("")
```

# Custom objects - hands-on code

Code: creating new types of objects

```
12 epochs = 2
13
14 def step_fun(x):
15     if x > 0:
16         return 1
17     else:
18         return 0
19
20 def wsum(vec, weights):
21     res = 0
22     for i, x in enumerate(vec):
23         res += x*weights[i]
24     return res
25
26 class Perceptron:
27
28     def __init__(self, eta, out_fun):
29
30         self.eta = eta
31         self.out_fun = out_fun
32         self.weights = [0, 0]
33
34     def activation(self, inp):
35
36         pot = wsum(inp, self.weights)
37         act = step_fun(pot)
38
39         return act, pot
40
41     def learn(self, inp, out, teach):
42         for i, x in enumerate(inp):
43             self.weights[i] += self.eta * x * (
44                 teach - out)
45
46 perc = Perceptron(eta=0.1, out_fun=step_fun)
47
48 for epoch in range(epochs):
49     for item in data:
50         inp, lab = item
51
52         act,_ = perc.activation(inp)
53         perc.learn(inp, act, lab)
54
55     print(lab, act, perc.weights)
56     print("")
```

# Custom objects

A class is a declaration of a custom type of objects

```
1 class NewType:
2     def __init__(self):
3         self.a_data_member = []
4     def add(self, x):
5         self.a_data_member.append(x)
6     def sum(self):
7         return sum(self.a_data_member)
```

An object is an element (or instance) of a class:

```
1 my_object = NewType()
2
3 my_object.add(3)
4 my_object.add(5)
5 my_object.add(7)
6 res = my_object.sum()    # res == 15
```