Practical sessions tools

- Python: a high level interpreted language
- Numpy: a scientific computing libray
- Keras: a high level deep learning library
- Jupyter notebook: A web application allowing to run code in a user-friendly environment

Contents

1. Python in 5 minutes
   - Expressions, variables and types
   - Functions
   - Classes
   - Loops and tests
   - Packages
2. Numpy
3. Jupyter notebook
4. Conclusion

Python

Python

- High-level language
- Object-oriented with dynamic typing
- Easy to interface with C++
- Easy to learn (?)

We will use Python 3

Contents

1. Python in 5 minutes
   - Expressions, variables and types
   - Functions
   - Classes
   - Loops and tests
   - Packages
2. Numpy
3. Jupyter notebook
4. Conclusion

Expressions

A session in the interactive interpreter:

Examples of expressions

```
>>> 2
2
>>> 2+2
4
>>> "hello"
'hello'
>>> # this is a comment
```

Variables

Variables are not declared:

```
>>> a = 2
>>> a
2
>>> b = 3
>>> print(b)
3
>>> b = a
>>> a = 1
>>> print(a,b)
1 2
```

Compound types: tuples and lists

Tuples

```
>>> t1 = ( 1, "zz")
>>> type(t1)
<type ’tuple’>
>>> print(t1, t1[0], t1[1])
(1,"zz") 1 "zz"
>>> a,b = t1
>>> print(a)
1
>>> print(b)
’zz’
```

Tuples are inmutable

Once they are created, the can only be erased. They cannot be modified.

Compound types: tuples and lists

Lists

```
>>> l1=[ "abc" ]
>>> l1.append(3.14159) # lists can be heterogeneous
>>> l1 [ 'abc', 3.14159 ]
>>> l1.append( [ 1, t1 ] ) # lists can contain lists
>>> l1 ['abc', 3.14159, [1, (1, 2)]]
>>> print(l1[1]) # indexing starts at 0
3.14159
```

Lists are mutable

New elements can be added; existing elements can be modified of deleted.

Contents

1. Python in 5 minutes
    - Expressions, variables and types
    - Functions
    - Classes
    - Loops and tests
    - Packages

2. Numpy

3. Jupyter notebook

4. Conclusion

Functions (1)

Declaring a function

```
def f(a): # note the ’:’ (colon)
     return a*2 # there are 4 spaces or 1 tab
                # before the "return"
```

Indentation is *critical*: it’s part of the syntax.

Execution

```
>>> f("abcd")
’abcdabcd’
```

Functions (2)

Return value

```
def theAddition(a, b):
     return a + b
```

Contents

1. Python in 5 minutes
   - Expressions, variables and types
   - Functions
   - Classes
   - Loops and tests
   - Packages

2. Numpy

3. Jupyter notebook

4. Conclusion

A Python class

A class in Python (as in many other object-oriented languages) is a container that can hold:

- Attributes (internal variables) and
- Methods (internal functions).

A class is defined using the *class* keyword.

Example

```
class Bird:
    def set_name(self, given_name): # class method
        self.name = given_name # instance attribute
```

Class instance

The execution of the above code creates a class. It can be instantiated by calling it as a function:

```
>>> robin = Bird()
>>> robin.set_name(‘‘European robin’’)
```

The result is an object or instance built according to the class model. The keyword *self* represents the class instance.
Instance attributes should in fact be initialized in a special method, that is called when the class is instantiated:

```
class Bird:
    def __init__(self, given_name):
        self.name = given_name
```

```
>>> robin = Bird(‘‘European robin’’)
>>> robin.name
European robin
```

Contents

1. Python in 5 minutes
    - Expressions, variables and types
    - Functions
    - Classes
    - Loops and tests
    - Packages

2. Numpy

3. Jupyter notebook

4. Conclusion

Tests

Tests and conditions

```
if 1 == 2 : # note the ’:’ !
     print("Wow, Problem!") # <- 4 spaces or 1 tab

if myGlass.isFull(): # note ’:’
     myGlass.doEmpty()
elif myGlass.isEmpty():
     myGlass.doFill()
else:
     pass # "empty" action
```

Loops (1)

Simple loops

```
for i in range( 5 ):
     print(i)
```

Will display:

```
0
1
2
3
4
```

Loops (2)

Loops on lists

To loop over the items in a list, 2 possibilities:

```
L = [ 1 , 2 , "abc" ]
for i in range( len(L) ):
     print(L[i])
```

which is equivalent to:

```
for x in L:
     print(x)
```

Python style

The second construct is more python-ish that the first, and should be preferred.

Loops (3)

“while” loops

Infinite “while” loop:

```
a = 0
while True:
     a += 1
     if a == 10:
          break
```

“while” loop with a test:

```
while a < 10:
     a+=1
```

Contents

1. Python in 5 minutes
    - Expressions, variables and types
    - Functions
    - Classes
    - Loops and tests
    - Packages
2. Numpy
3. Jupyter notebook
4. Conclusion

Packages

A Python package is a module that brings new functions and classes when *imported*.
The Python ecosystem offers thousands of packages.

Importing packages

```
>>> import numpy
>>> numpy.cos(0)
1.0
```

You can also use a short name for the package:

```
>>> import numpy as np
>>> np.cos(0)
1.0
```

For the lazy ones (bad practice, just for quick tests!):

```
>>> from numpy import *
>>> cos(0)
1.0
```

Contents

1. Python in 5 minutes
2. Numpy
3. Jupyter notebook
4. Conclusion

Numpy

- A python module for scientific computing
- Based on a powerful multi-dimensional array object
- Efficient core coded in C++

Numpy Arrays

Vectors, matrices and tensors can be represented with numpy arrays.

```
>>> import numpy as np
>>> a = np.array([2, 0, 3])
>>> print(a)
[2 0 3]
>>> type(a)
<class ’numpy.ndarray’>
>>> a.dtype # data type
dtype(’int64’)
>>> a.ndim # number of dimensions
1
>>> a.shape
(3, )
```

2D arrays

```
>>> b = np.array([[2, 0, 4], [1, 3, 3]])
>>> b
array([[2, 0, 4], [1, 3, 3]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> b[1, 0]
1
```

Vector and matrix operations

By default, standard operators like *, -, + , / , are applied element-wise to arrays and vectors.

```
>>> a = np.array([3, 0, 2])
>>> b = np.array([1, 2, 3], [0, 1, 0])
>>> a*b
array([[3, 0, 6], [0, 0, 0]])
>>> b*b
array([[1, 4, 9], [0, 1, 0]])
>>> b* b.transpose()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together
with shapes (2,3) (3,2)
```

Matrix and vector operations

```
>>> a = np.array([3, 0, 2])
>>> b = np.array([1, 2, 3], [0, 1, 0])
>>> np.matmul(b,a)
array([9, 0])
>>> np.matmul(b,b.transpose())
array([[14, 2], [ 2, 1]])
```

Choosing elements within an array

```
>>> b = np.array([1, 2, 3], [0, 1, 0])
>>> b[1,1]
1
>>> b[1,:]
array([0, 1, 0])
>>> b[:,1]
array([2, 1])
>>> b[:,0:2]
array([[1, 2], [0, 1]]
```

Contents

1. Python in 5 minutes
2. Numpy
3. Jupyter notebook
4. Conclusion

Contents

1. Python in 5 minutes
2. Numpy
3. Jupyter notebook
4. Conclusion

The end

Good Luck !