Introduction to Python 1/2

Text Analytics - Andrea Esuli
What is Python?

Python is a programming language created by Guido van Rossum in 1991. It is a high level, interpreted, dynamic typed, strong typed, garbage collected language, supporting imperative, functional, and object oriented programming styles.

The name comes after the Monty Python Flying Circus.
Why Python?

Python has a simple, clean syntax. It’s easy to learn.

The type system doesn’t get in the way of coding, if not required.

Python has rich standard libraries and a large amount of powerful additional packages:

- Built-in data types for numbers, strings, lists, tuples, sets, dictionaries
- Strong numeric processing capabilities, with support to fast CPU/GPU parallel processing.
- Large collections of data processing, machine learning tools.
- Good amount of NLP tools.

It’s the fastest growing language among the common used ones.
Which Python version?

3

.5 .6 ...
Which Python version?

Python 3 has been first released in 2008 (3.4 in 2014), it is not a recent novelty.

Python 2 had its last release, 2.7, in 2010, since then it is on end-of-life support.

“Python 2.x is legacy, Python 3.x is the present and future of the language”

Even if you currently use Python 2.x, do a favor to your future self, move to Python 3.
Which Python version?

“But... I use tools that are written in Python 2”

All the major libraries and tools now support Python 3, any Python 2-only package can be considered as not up to date.

Google released its first version of TensorFlow as a Python 2.7 package.

Now TensorFlow on Windows works only with Python 3.5.
“But... my code is written in Python 2”

Although Python 3 is not fully backward compatible, there are only a few relevant aspects that differ:

- Strings, encodings support (it’s better at supporting non-trivial characters)
- `print`, `except` syntax (it’s more intuitive)
- `irange` → `range` (it’s more efficient)
- Integer division (it’s more intuitive)

Porting Py2 code to Py3 is simple and supported by dedicated tools.

If Instagram has moved its 400M users platform to Py3 you can TRY to move your scripts too.
Installation
Installation

The open source reference implementation of python is available from the python foundation.

However, I strongly suggest you to install the Anaconda distribution.

Anaconda can be installed without super user privileges, and it does not conflicts with existing python installations.

The conda management tool for environments and packages is simple to use, and it provides precompiled packages for many platforms.
Installation

Once anaconda python is installed, start the ‘Anaconda prompt’ and issue the command:

>conda install nb_conda
Fetching package metadata ..........  
Solving package specifications: .  

Package plan for installation in environment Anaconda3:

The following NEW packages will be INSTALLED:

[..]

Proceed ([y]/n)? y
Environments allow to have multiple, distinct, independent installations of Python, each one with its selection of installed packages:

>conda create -n py2 python=2 ipykernel
>conda create -n py3 python=3 ipykernel

In this way you can manage a dedicated setup for each of your projects. Messing up one environment does not affects the others.

When you want to use an environment you activate it:

mac/linux>source activate py3
windows>activate py3
Installation

The conda command can be used to install/remove packages:

>conda install nltk scikit-learn matplotlib gensim feedparser dill

When a package is not available from the anaconda repository, it can be installed using the pip tool, the standard package manager for python:

>pip install tweepy
Installation

Packages and environments can be managed also from jupyter:

From the Anaconda prompt:

> jupyter notebook
Notebooks

A notebook is an interactive computational environment, in which pieces of code are organized in “code cells” whose output is shown “output cells” the notebook itself.

Notebooks can contain many types of cells, such as rich text, plots, animations.

Notebook are useful for exploration, experimentation, and reporting results.
A script is a Python source file, i.e., text file, with .py extension, that defines a directly executable program and/or a module declaring functions and classes.

Content of a hello.py file:

def hello():
    print('Hello world!')

hello()

Execution:

$ python hello.py
Hello world!
$
Which editor?

I strongly suggest PyCharm.
Basics of Python
Statements

Newlines separates statements.

```
a = 1
b = 2
c = a + b
```

Ending “;” à la C, Java... is OPTIONAL, and almost always omitted.

```
a = 1;
b = 2;c = a + b
```
Variables

A variable is a **reference** to an object in memory.

```
a = 1
```

The object of type int “1” is created in some location in memory, `a` points to that location.

```
b = a
```

`b` points to the same location of `a`, thus returns the object “1”

```
b = 2
```

Now `b` points to the location of a new object int “2”, `a` still points to “1”
Variables

A variable name is composed of letters, numbers, and the underscore character ‘_’

- Can’t start with a number
  
  ```python
  1a = 1
  SyntaxError: Invalid syntax
  ```

- Can use non-ASCII letters in Py3
  
  ```python
  è_una_variabile = 1
  ```

- **Cannot** be one of these **reserved words**
  
  False class finally is return None continue for lambda try
  True def from nonlocal while and del global not with as elif
  if or yield assert else import pass break except in raise
Variables

A variable is created when it is first assigned to.

```python
a = 1
b = a
```

A variable can be deleted.
Deletion removes the name, not the referenced object.

```python
del a
b
```
Out: 1
Variables

When an object loses all references it can be garbage collected.

```python
a = 'hello'  # A string object with value ‘hello’
            # is created in memory.
            # Variable ‘a’ points to it.
a = 'world'  # Now variable ‘a’ points to this new object.
            # The ‘hello’ object has no references
            # and it is inaccessible, it can be deleted
            # from memory.
```

Python, Java, C#, Javascript use garbage collection. A dedicated process tracks references and deletes inaccessible objects. C, C++ require the user to explicitly perform memory management.
Variables

The type of a variable changes with the type of the object it references.

```python
a = 1
type(a)
Out: int
a = 'ciao'
type(a)
Out: str
a = [1,2,3]
type(a)
Out: list
```

Drawback: type errors are caught only at runtime (yet static typing is possible)
The help(x) function prints the available documentation and a description for any variable, type, function or class.

help(), without arguments, starts an interactive help session.

The dir() function lists the names defined in the current scope.

The dir(x) function lists the names defined under the scope identified by x.

help and dir are useful exploration tools.
Types
Types

Python interpreter has a number of built-in types:

● NoneType
● Boolean
● Numeric
● Sequences
● Strings
● Sets
● Dictionaries
● Functions
● Classes and methods
None is the single existing object of type NoneType and it is the equivalent of *null* for many other programming languages.

It is used to indicate the absence of a referred value, yet the variable exists.

A common use is to reset a variable, to signal ‘soft’ failures, and in the evaluation of boolean expressions.

Any function that does not explicitly return a value, returns None.
Booleans

Truth values are represented by the bool type:

type(True)
Out: bool

Constants: True, False

Equivalent to False: the None value, the value zero in any numeric type, empty sequences, strings, and collections.

Boolean operators: and, or, not

Boolean tests: ==, !=, >, <, in, is
Booleans

Boolean tests: ==, !=, >, <=, >=, <, in, is

== checks for equivalence of value, is checks for equivalence of identity

The id() function returns a unique numeric “identity” of any object.

```python
a = 'hello '  
b = 'world'

c = a + b

d = a + b

c==d, c is d, id(c), id(d)

Out: (True, False, 2287672582000, 2287672579120)
```
Numbers

Three numeric types: `int`, `float`, `complex`.

Integers have unlimited precision (try $9^{*1000}$)

`bool` ⊂ `int` ⊂ `float` ⊂ `complex`

Any operation is done using the “wider” type of the two arguments, if it is defined for the wider type.

Integers can be converted to float only when they are less than:

```python
import sys
sys.float_info.max
Out: 1.7976931348623157e+308
```
Numbers

\[ x + y \] # sum of \( x \) and \( y \)
\[ x - y \] # difference of \( x \) and \( y \)
\[ x \times y \] # product of \( x \) and \( y \)
\[ x / y \] # quotient of \( x \) and \( y \)
\[ x // y \] # floored quotient of \( x \) and \( y \)
\[ x \% y \] # remainder of \( x \) / \( y \)
\[ -x \] # \( x \) negated
\[ \text{abs}(x) \] # absolute value or magnitude of \( x \)
\[ \text{pow}(x, y) \] # \( x \) to the power \( y \)
\[ x ** y \] # \( x \) to the power \( y \)
\[ \text{int}(x), \text{float}(x), \text{complex}(x) \] # explicit number/string conversion

**Big difference w.r.t. Py2:** the quotient operation ‘/’ produces a float even when applied on two integers. Integer quotient operation is ‘//’.
Numbers

The **math module**, defines many other mathematical functions.

```python
import math
math.factorial(10)
Out: 3628800

math.gcd(234, 224)
Out: 1

math.pi
Out: 3.141592653589793
```

For scientific computing **numpy** is the reference package to use.
Sequence types
Lists

A list is a **mutable** ordered sequence of items of **possibly varied type**.

Note: Ordered ≠ Sorted

Mutable means that the element of the list can be changed *in place*. The identity of the list does not change when it is modified.

A list is defined using square brackets, with comma separating items.

```
a = []
a = [ 1, 2, 3]
a = [ 1, 2, 3, 'ciao', [], ['a', 4, None]]
```
Tuples

A tuple is an **immutable** ordered sequence of items of **possibly varied type**.

Immutable means that once defined an object **cannot be modified**. Operations that “modify” an immutable object **create** in fact **a new object**.

Round brackets define the empty tuple, otherwise **commas define a tuple**, brackets are just a visual clue. Trailing comma is needed only for one-element tuples.

```
a = ()
a = 1,
a = 1, 2, 3
```

Tuples are often used as return value in functions.
Strings

A string is an **immutable** ordered sequence of **Unicode characters**.

A string is defined by using single quotes or double quotes.

```
a = 'this is a test' # or "this is a test" it is the same
```

Triple double quotes define multiline strings.

```
a = """This is a
multiline string""
```

Out: 'This is a\nmultiline string'

**Escape sequences** allows to put quotes, newlines, tabs, or other non-trivial chars in strings.
Accessing elements

Elements of a sequence (it is the same for lists, tuples and strings) can be accessed using square bracket index notation.

**NOTE: the first element of a sequence has index 0!**

```python
a = [10, 20, 30]
a[0]
Out: 10
a[1]
Out: 20
len(a) # returns the number of elements in the sequence
Out: 3
len([ 1, 2, 3, 'ciao', [], ['a', 4, None]]) # ???
```
Slicing

Slicing is a powerful tool that allows to **copy** subsequences of a sequence.

```
a[start_idx:end_idx] # copy from start_idx, stop before end_idx
a[start_idx:] # copy from start_idx through the end of sequence
a[:end_idx] # copy from beginning, stop before end_idx
a[: ] # copy all the sequence, different from b=a!!!
a[start_idx:end_idx:step] # only pick items every step
```

A **negative index** means it is relative to the end of the sequence.

```
a = [1, 2, 3, 4, 5]
a[: -2]
Out: [1, 2, 3]
```
The `in` operator checks for the presence of an item in a sequence.

```
a = [2, 5, 6, 6, 5]
1 in a
Out: False
5 in a
Out: True
```

On strings it matches *substrings*.

```
a = 'this is a test'
'a test' in a
Out: True
```
The + operator creates a new sequence by concatenating its arguments.

[1, 2, 3] + [4, 5, 6]
Out: [1, 2, 3, 4, 5, 6]

(1, 2, 3) + (4, 5, 6)
Out: (1, 2, 3, 4, 5, 6)

"Hello" + " " + "World"
Out: 'Hello World'
The * operator creates **a new sequence** by concatenating as many times the sequence argument as expressed by the integer argument.

```
[1] * 9
Out: [1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, [2, 3]] * 3
Out: [1, [2, 3], 1, [2, 3], 1, [2, 3]]

4 * "Hello " + "world" + "!" * 3
Out: 'Hello Hello Hello Hello Hello world!!!'
```
List operations

```python
a = [1, 2, 3]
a.append('a')
a
Out: [1, 2, 3, 'a']

**append** adds an element at the end of the list

a.insert(2, 'b')
a
Out: [1, 2, 'b', 3, 'a']

**insert** puts the elements in the position specified by the first argument
List operations

a = [1, 2, 3]
a.append([4,5,6])
a
Out: [1, 2, 3, [4,5,6]]

Use `extend` to copy values for a sequence into another.

a.extend([4,5,6])
a
Out: [1, 2, 3, 4, 5, 6]

Note: + creates a new list, does not modify input lists.
List operations

b = a.pop()
b,a
Out: 'a',[1, 2, 3]

pop returns and removes the element at the end of the list. Use del to remove an element given its position:
del a[1]
a
Out: [1, 3]

remove removes the first instance of the given value:
a.remove(3)
a
Out: [1]
List operations

```
a = ['t','e','s','t']
a.index('t')
Out: 0
```

**index** returns position of the first instance of the given value.

```
a.count('t')
Out: 2
```

**count** returns the number of instances equivalent (==) to the given value.
List operations

```
a = [2, 1, 5, 4, 3]
a.reverse()
a
Out: [3, 4, 5, 1, 2]
```

**reverse** the list **in place**.

```
a.sort()
a
Out: [1, 2, 3, 4, 5]
```

**sort** the list **in place**.

Custom sort can be defined by passing a **sorting key function**.
Tuple operations

Being immutable, tuples lack most of the functionalities of lists.

Tuples only have **count** and **index** operations.

```python
a = (1, 2, 3, 3, 4, 3, 5, 3, 6, 7)
a.count(3)
Out: 4

a.index(3)
Out: 2

a[2]
Out: 3
```
String operations

Strings can be seen as text-specialized tuples.

They offer a number of text-oriented operations.

capitalize, encode, format, isalpha, islower, istitle, lower, replace, rpartition, splitlines, title, casefold, endswith, format_map, isdecimal, isnumeric, isupper, lstrip, rfind, rsplit, startswith, translate, center, expandtabs, index, isdigit, isprintable, join, maketrans, rindex, rstrip, strip, upper, count, find, isalnum, isidentifier, isspace, ljust, partition, rjust, split, swapcase, zfill
The **print** function prints values on the screen (or in a file).

Many options of **string formatting** allow to combine text and values.

```python
print('My name is %s and I\'m %d years old' % ('Andrea', 39))
Out: My name is Andrea and I'm 39 years old
```

In Python 2, `print` is a statement that does not require parentheses.

Use the `str()` function to get a string representation of any value.

```python
'the list is ' + str([1, 2, 3, 4])
Out: 'the list is [1, 2, 3, 4]'
```
Regular expressions

A regular expression is a search pattern.

Regular expressions are used to find matching patterns in text and to extract relevant substrings from text.

The `re` module defines objects and methods to apply regular expressions to strings.

Regular expressions are defined as strings that follow a specific syntax.

'\[A-Z][a-z]\{3\}' = match a sequence of any capital letter followed by exactly three lower-case letters, e.g., 'Pisa'
Regular expressions

Basic matching

'a' = character a
'abc' = string abc
'a | b' = match a or b
'a*' = zero or more a
'a+' = one or more a
'a{3}' = exactly 3 a
'a{2,5}' = from 2 to 5 a (the more the better)
'a{2,5}?=' = from 2 to 5 a (the less the better)
'a{4,}' = at least 4 a
'a{,3}' = at least 3 a
Regular expressions

Groups

'(abc)' = group, sequence of characters abc
'(abc)+' = one or more time the sequence
'(?P<name>...)' = group named "name"
'(?P=name)' = match the group name
'(?::)' = non capturing (just to define a sequence)
Regular expressions

Characters classes

\[ \text{[abc]} = \text{one in set a,b,c} \]
\[ \text{[a-z0-9]} = \text{one in set of character from a to z and from 0 to 9} \]
\[ \text{[^a-z]} = \text{one character but not those from a to z} \]
\[ \text{\d} = \text{one digit character} \]
\[ \text{\D} = \text{one non-digit character} \]
\[ \text{\s} = \text{one white space character} \]
\[ \text{\S} = \text{one non white space character} \]
\[ \text{\w} = \text{one word character (e.g. a-z A-Z 0-9 _ )} \]
\[ \text{\W} = \text{one non-word character} \]
Regular expressions

Other matches

^ = start of string
$ = end of string
\n = newline
\r = carriage return
\t = tab

Start exploring regular expressions here and here.
Regular expressions

**Compilation** allows efficient reuse of regular expressions, and a clean separation between their definition and their use.

```python
tagre = re.compile('(?P<tag>.+)(?P<text>.*)(?P=tag>')
tagged = tagre.match('<pre>Ciao</pre>')
tagged['tag']
Out: 'pre'
tagged['text']
Out: 'Ciao'
```
Sets and Dictionaries
Sets

A set is an **unordered** collection of **distinct** (i.e., no duplicates) objects.

```python
a = set()
a.add(1)
a.add(2)
a
Out: a = {1, 2}
```

```python
b = set()
b.add('b')
b.add(3)
a.union(b)
a.union(b)
Out: {1, 2, 'b', 3}
```
Dictionaries define a **mapping** between a set of **keys** and a set of **values**.

Keys must have an immutable type.

Values can have any type.

In a dictionary both keys and values can have mixed types.

```python
code
population = dict()  # or population = {}
population['pisa'] = 91104
population['livorno'] = 160027
population
Out: {'pisa': 91104, 'livorno': 160027}
```
Dictionaries

population['pisa']
Out: 91104

'firenze' in population # check if key exists
Out: False

population['firenze'] # exception is raised if key does not exist
KeyError: 'firenze'

Keys are unique, reassigning replaces value:

population['pisa'] = 10000000
population['pisa']
Out: 10000000
Dictionaries

del population['livorno']
population['livorno']
KeyError: 'livorno'

Empty a dictionary with **clear** (also works on sets and lists):

population.clear()
population
Out: {"}
Dictionaries

ages = {'Andrea': 39, 'Giuseppe': 67, 'Paolo': 58}
ages.keys()
Out: dict_keys(['Andrea', 'Giuseppe', 'Paolo'])

ages.values()
Out: dict_values([39, 67, 58])

ages.items()
Out: dict_items([('Andrea', 39), ('Giuseppe', 67), ('Paolo', 58)])

dict_keys, dict_values, dict_items are mutable views of the dictionary, i.e., they change as the dictionary changes. They are iterables.