Python for Data Science
Introduction to Python and Its Scientific Computing Libraries

Heru Praptono

heru.pra@cs.ui.ac.id

Salemba, December 2018

Outline

Introduction

- Programming Basics
- What does programming in Python look?
- Why Python?

Programming in Python

- What you need to prepare
- Python: How to do

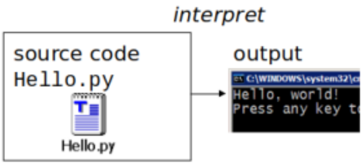
NumPy and Scikit-Learn

- NumPy
- Scikit Learn

Programming Basics

- **code or source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
- Some source code editors pop up the console as an external window, and others contain their own console window.

What does programming in Python look? Case: Python vs Java



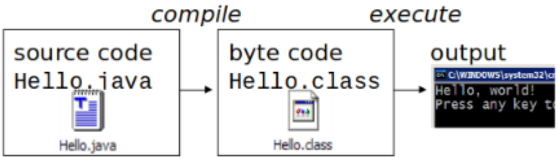
interpret

source code
Hello.py

Hello.py

output

```
C:\WINDOWS\system32\c
Hello, world!
Press any key t
```



compile

execute

source code
Hello.java

Hello.java

byte code
Hello.class

Hello.class

output

```
C:\WINDOWS\system32\c
Hello, world!
Press any key t
```

- See the above comparison, let us say, Python vs Java (or C, C#).
- Many languages require you to *compile (translate)* your program into a form that the machine understands.
- Python is instead directly *interpreted* into machine instructions. On the other hand, java require ones to compile first before execute.

Why Python?

- Similar to Java that handles your memory management – allocates memory and has garbage a collector to free up that memory once it is no longer needed (in use).
- Python is relatively easy to use, powerful, multi-platform and versatile. Hence, it...
- ...is a great choice for beginners and experts like!

Python in relation with data science

Advantages:

- Very rich scientific computing libraries (a bit less than Matlab, though)
- Well thought out language, allowing to write very readable and well structured code: we "code what we think".
- Many libraries for other tasks than scientific computing (web server management, serial port access, etc.)
- Free and open-source software, widely spread, with a vibrant community.

Drawbacks:

- less pleasant development environment than, for example, Matlab. (More geek-oriented)
- Not all the algorithms that can be found in more specialized software or toolboxes

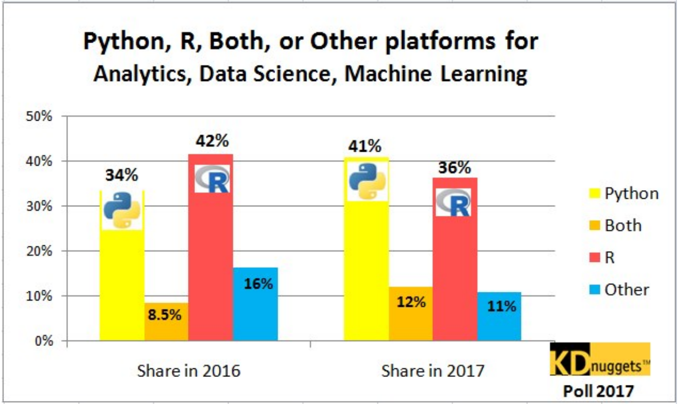
Python in relation with data science

Do not worry

But again, do not worry! A lot of libraries are available out there, including its tutorial! This introductory guidance helps you to keep focus on essential to start with Python.

Python in relation with data science

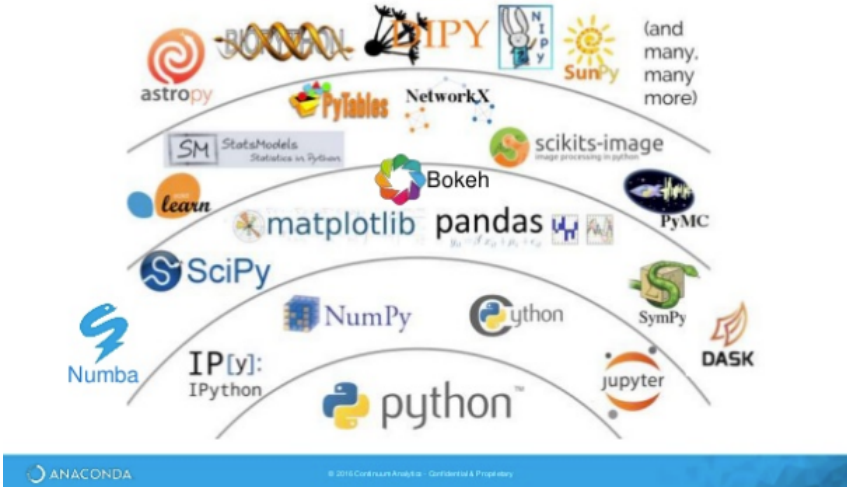
A comparison[1]



[1] source: https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html

Python in relation with data science

Supported Libraries[2]



[2] source: https://www.noobloops.com/2017/03/most-amazing-used-libraries-in-python.html

What you need to prepare

- Make sure you have had Python 3.xx installed in your computer.
- We may use any conda, since it is an isolated environment, so we can create virtual environment on our way.
- Can be accessed within terminal, or Jupyter Notebook.
- Now, let us start!

Expressions

- **expression**: A data value or set of operations to compute a value.
  examples: 1 + 4 * 3   #13
- Arithmetic operators we will use:
  + - * /   #addition, subtraction/negation, multiplication, division
  %   #modulus, a.k.a. remainder
  **   #exponentiation
- precedence: Order in which operations are computed.
  * / % ** have a higher precedence than + -
  1 + 3 * 4 is 13
- Parentheses can be used to force a certain order of evaluation.

Integer Division

- In python 3.x, integer division will give decimal
  - However, in Python 2.7 it will give an integer
- The % operator computes the remainder from a division of integers.
  - 14%4   #2
  - 218%5   #3

Real Numbers

- Python can also manipulate real numbers.
  - Examples:
    6.022
    -15.9997
    42.0
    2.143e17
- The operators + - * / % ** ( ) all work for real numbers.
  - The / produces an exact answer: 15.0 / 2.0 is 7.5
  - The same rules of precedence also apply to real numbers:
    Evaluate ( ) before * / % before + -
- When integers and reals are mixed, the result is a real number.
  - Example: 1 / 2.0 is 0.5

Variables

- **variable**: A named piece of memory that can store a value.
  - Usage:
    Compute an expression's result,
    store that result into a variable,
    and use that variable later in the program.
- **assignment statement**: Stores a value into a variable.
  - Syntax:
    `name = value`
  - Examples:
    - x = 5
    - gpa = 3.14
- A variable that has been given a value can be used in expressions.
  - x + 4 is 9
- Exercise: Evaluate the quadratic equation for a given a, b, and c!

Print

- `print` : Produces text output on the console.
- Syntax:
  - `print("Message")`
  - `print(Expression)`
- Prints the given text message or expression value on the console, and moves the cursor down to the next line.
  - print(Item1, Item2, ..., ItemN)
- Prints several messages and/or expressions on the same line.

Input

- `input` : Reads a number from user input.
- You can assign (store) the result of input into a variable.

  ```
  age = input("How old are you?  ")
  print("Your age is", age)
  print("You have", 65 - age, "years until retirement")
  ```

  Output:

  ```
  How old are you?  53
  Your age is 53
  You have 12 years until retirement
  ```

Loop: For

- `for` loop: Repeats a set of statements over a group of values.
- Syntax :

  ```
  for variableName in groupOfValues:
      statements
  ```
- We indent the statements to be repeated with tabs or spaces.
- `variableName` gives a name to each value, so you can refer to it in the `statements`.
- `groupOfValues` can be a range of integers, specified with the range function.
- Example:

  ```
  for x in range(1, 6):
      print(x, "squared is", x * x)
  ```

  what is the output?

Range

- `range(start, stop)` - the integers between start (inclusive) and stop (exclusive)
- `range(start, stop, step)` - the integers between start (inclusive) and stop (exclusive) by step
- Example:

```
for x in range(5, 0, -1):
    print(x)
print('Blastoff!!')
```

what is the output?

Loop: While

- while loop: Executes a group of statements as long as a condition is True.
  - good for indefinite loops (repeat an unknown number of times)
- Syntax:

```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print(number,)
    number = number * 2
```

- output: 1 2 4 8 16 32 64 128

Decision: if

- if statement: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
- Syntax:

```
if condition:
    statements
```

- Example:

```
gpa = 3.4
if gpa > 2.0:
    print("Your application is accepted.")
```

  what is the output?

Decision: if/else

- if/else statement: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
- Syntax:

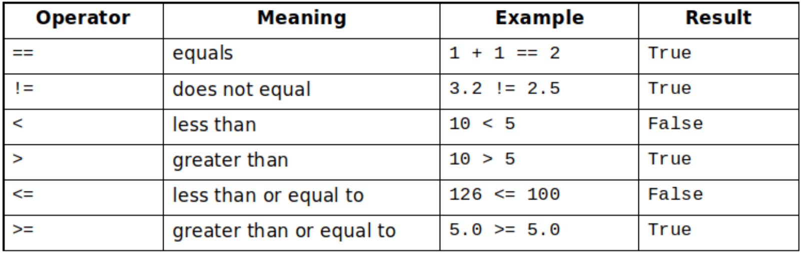
```
if condition:
    statements
else:
    statements
```

- Multiple conditions can be chained with elif ("else if"):
- Syntax:

```
if condition:
    statements
elif:
    statements
else:
    statements
```

Logic

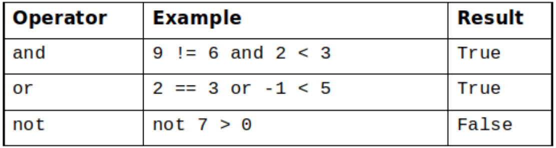
Many logical expressions use relational operators:



Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Logic

Logical expressions can be combined with logical operators:



Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

Function

- return function: A funcion that returns any value

```
def [functionname](argumentIfAny):
    # statements
    # ...
    return [returnedThing]
```

- void function: A function that does not return any value

```
def [functionname](argumentIfAny):
    # statements
    # ...
```

NumPy: What is it?

- a Python extension module that provides efficient operation on arrays of homogeneous data
- allows python to serve as a high-level language for manipulating numerical data, much like IDL, MATLAB, or Yorick
- enabling Linear Algebra (numpy.linalg)
- Vectorisation (e.g. change primitive foor loop into inner product)

NumPy: ndarray

- NumPy's main object is the homogeneous multidimensional array called ndarray.
  - This is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
  - Dimensions usually called axes, number of axes is the rank. [7, 5, -1] #An array of rank 1 i.e. It has 1 axis of length 3. [[1.5, 0.2, -3.7] ,[0.1, 1.7, 2.9]] # An array of rank 2 i.e. It has 2 axes, the first length 3, the second of length 3 (a matrix with 2 rows and 3 columns)

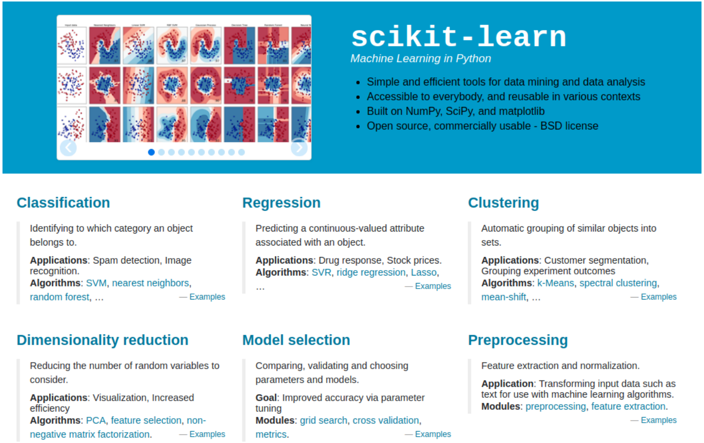
NumPy: Using arrays wisely

- Optimised algorithms - i.e. fast!
- Python loops (i.e. for i in a:. . .) are much slower
- Prefer array operations over loops, especially when speed important
- Also produces shorter code, often more readable

Scikit Learn: Motivation

- Most of our activity in data science for data mining, consists of either finding pattern in data, or function fitting.
- We may develop from scratch, but...
- ...there is some general well-packaged available, for those who want to work on applied area.

Scikit Learn: Modules



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

— Examples

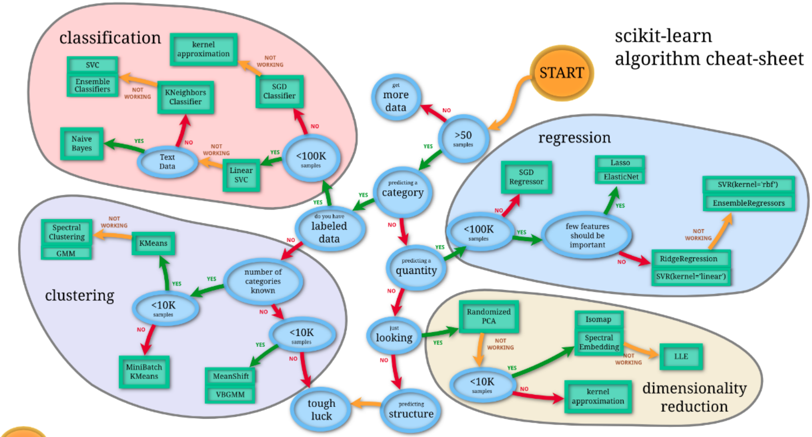
Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

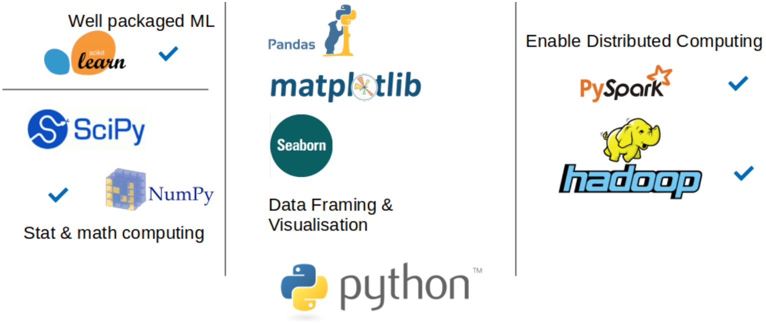
— Examples

Scikit Learn: Choosing the right estimator



Back

How the things related each other



Well packaged ML

Stat & math computing

Data Framing & Visualisation

Enable Distributed Computing

Thank you