Big Data and Data Mining

02. Introduction to Python

Why Python for Data Science?

NumPy

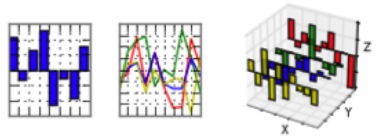
SciPy

matplotlib

IP[y]: IPython
Interactive Computing

pandas

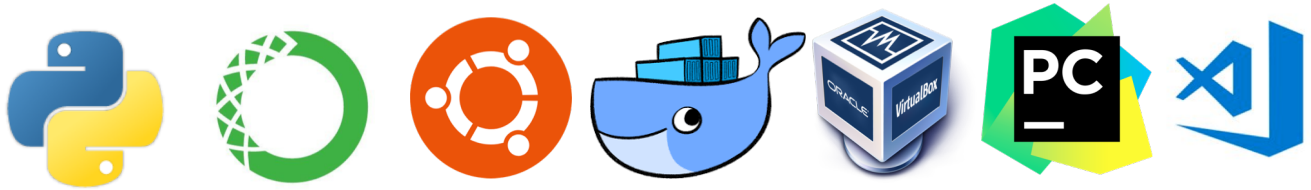
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



jupyter

scikit learn

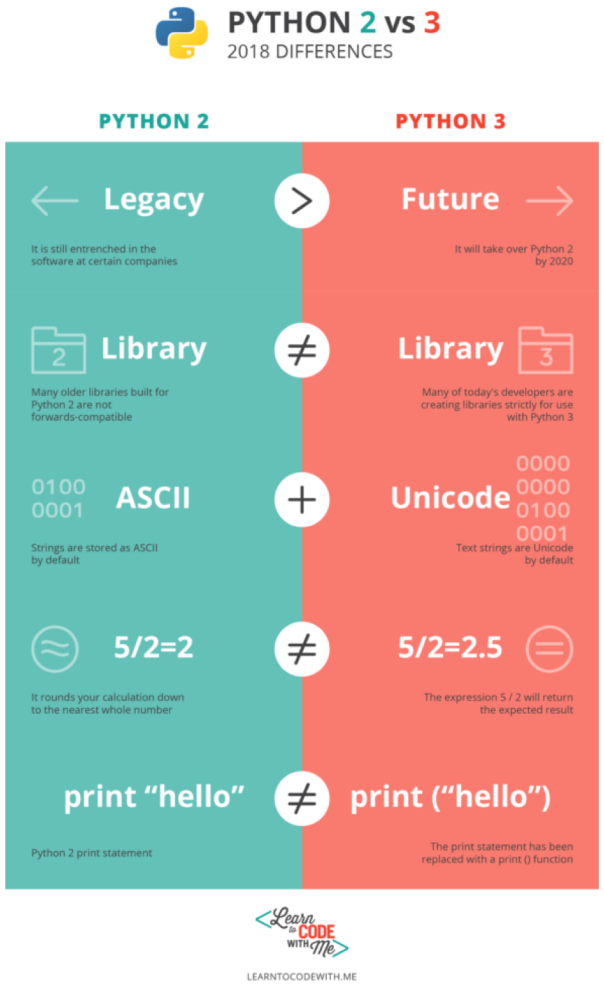
Requirement



- Operation System
  - Linux OS: Ubuntu distribution is highly recommended.
  - macOSX is also a decent platform
  - Windows 10 with native WSL Ubuntu
- Python Programming Environment
  - Interpreter, Library *etc.*
- Editor or **I**ntegrated **D**evelopment **E**nvironment
  - Syntax Highlight, Easy Configuration *etc.*
- Option For Enterprise Production
  - Any Laptop with Text Editor (*VSCode*) and SSH toolchain
  - IDEs, such as *PyCharm Professional*, are always handy
  - Access credential to the docker, virtual machine on remote DEV server

a) https://docs.microsoft.com/en-us/windows/wsl/install-win10

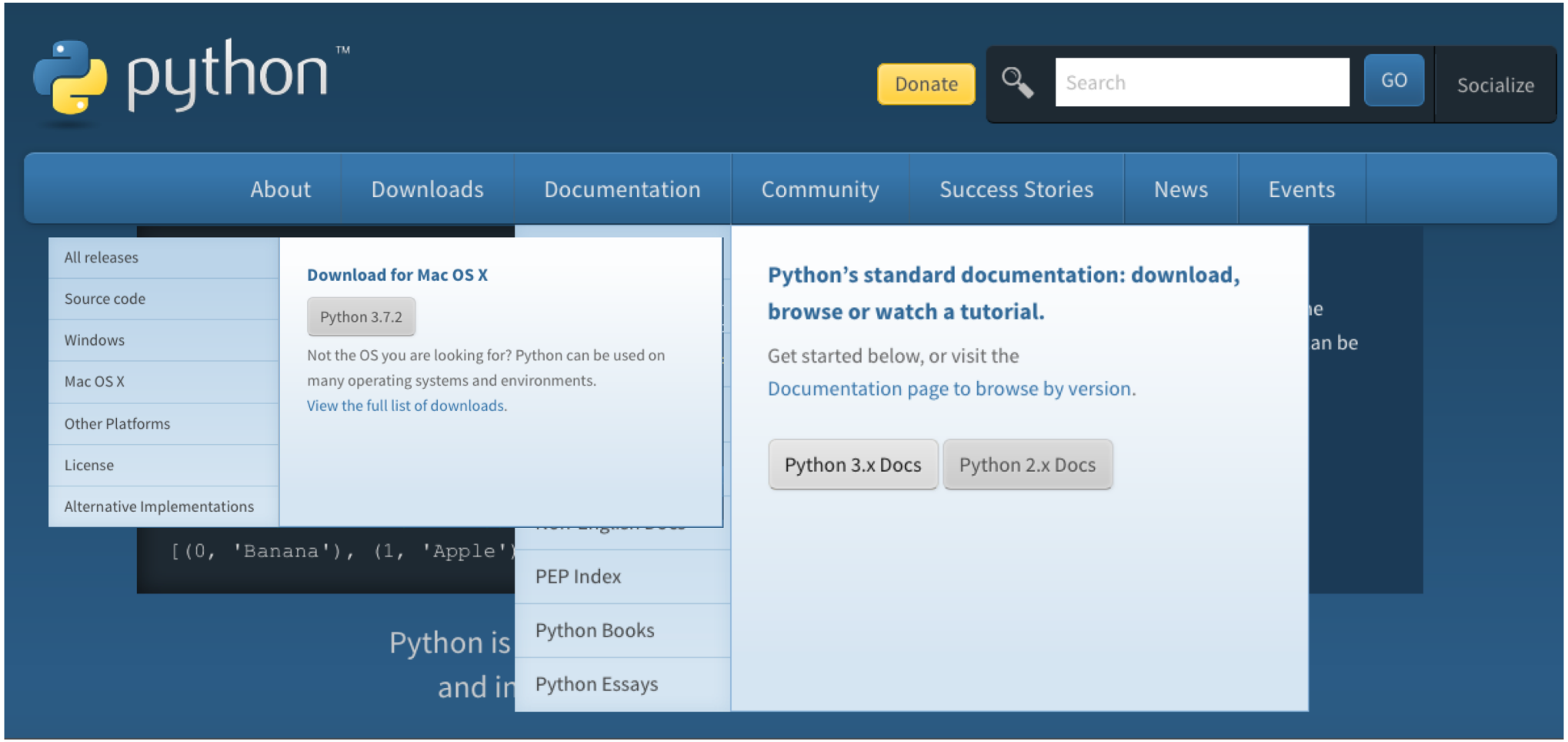
Python 2 or Python 3



a) https://dev.to/kostassar/python-2-vs-python-3-2f03
b) https://learntocodewith.me/programming/python/python-2-vs-python-3/

Official Resource

Practice is the key to learning any programming language!

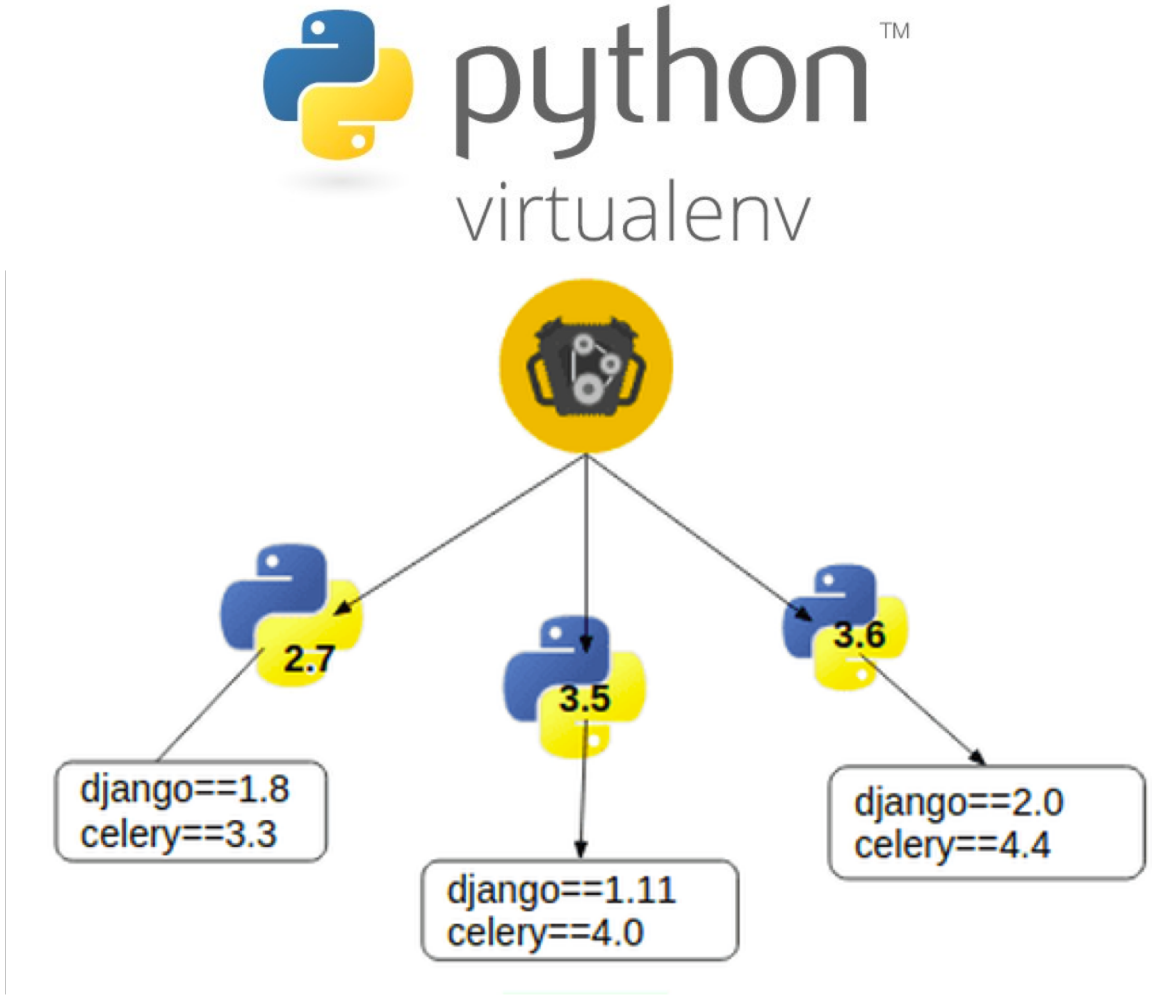


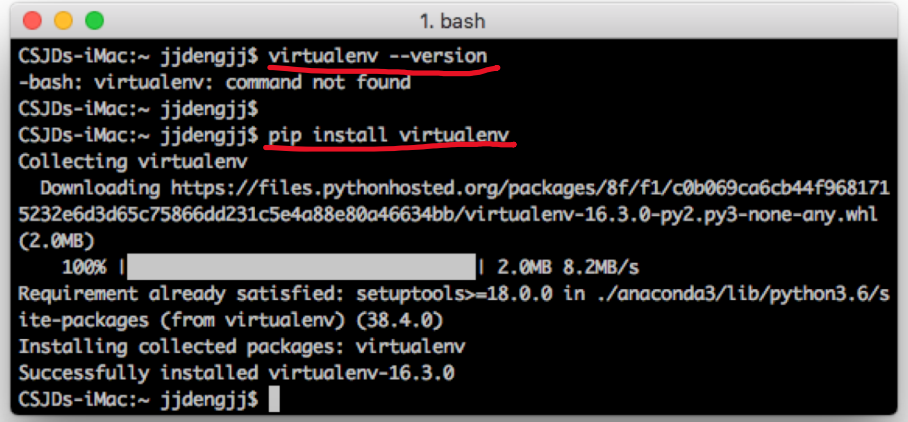
a) https://www.python.org/

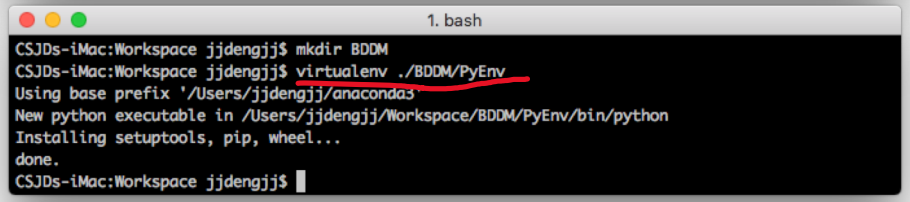
Installation – Basic

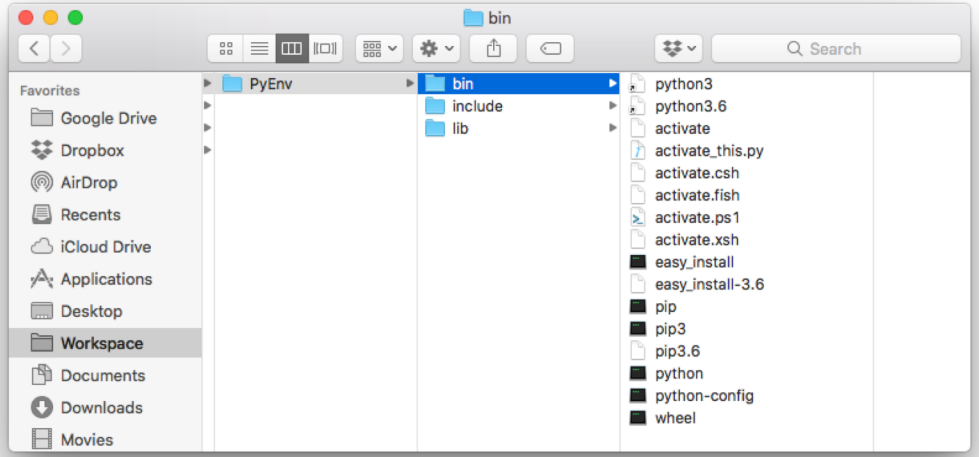
- Interpreter and Standard Library
  - Official Python Distribution: Highly recommended for professional
    - Lightweight distribution with essential components
    - For Windows & macOSX: Binary Installer
    - For Linux: *apt-get install …* or other package manager
    - *pip* as library management
  - Anaconda Distribution: If you are new to python, check this out
    - Fully packed but with redundant packages
    - *conda* as library management
    - *pip* as library management
    - Third-party library repositories
- Official Doc is the best material for learning python as always

Installation – Advance



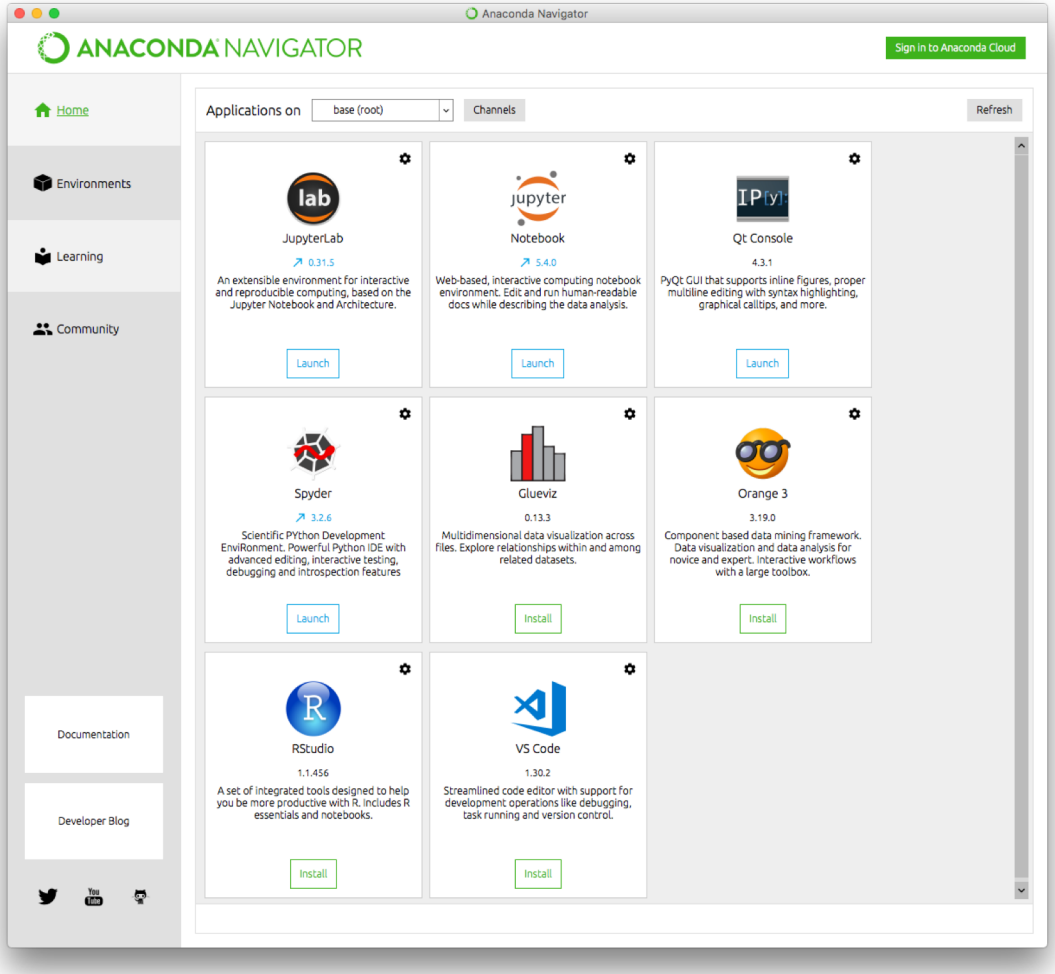


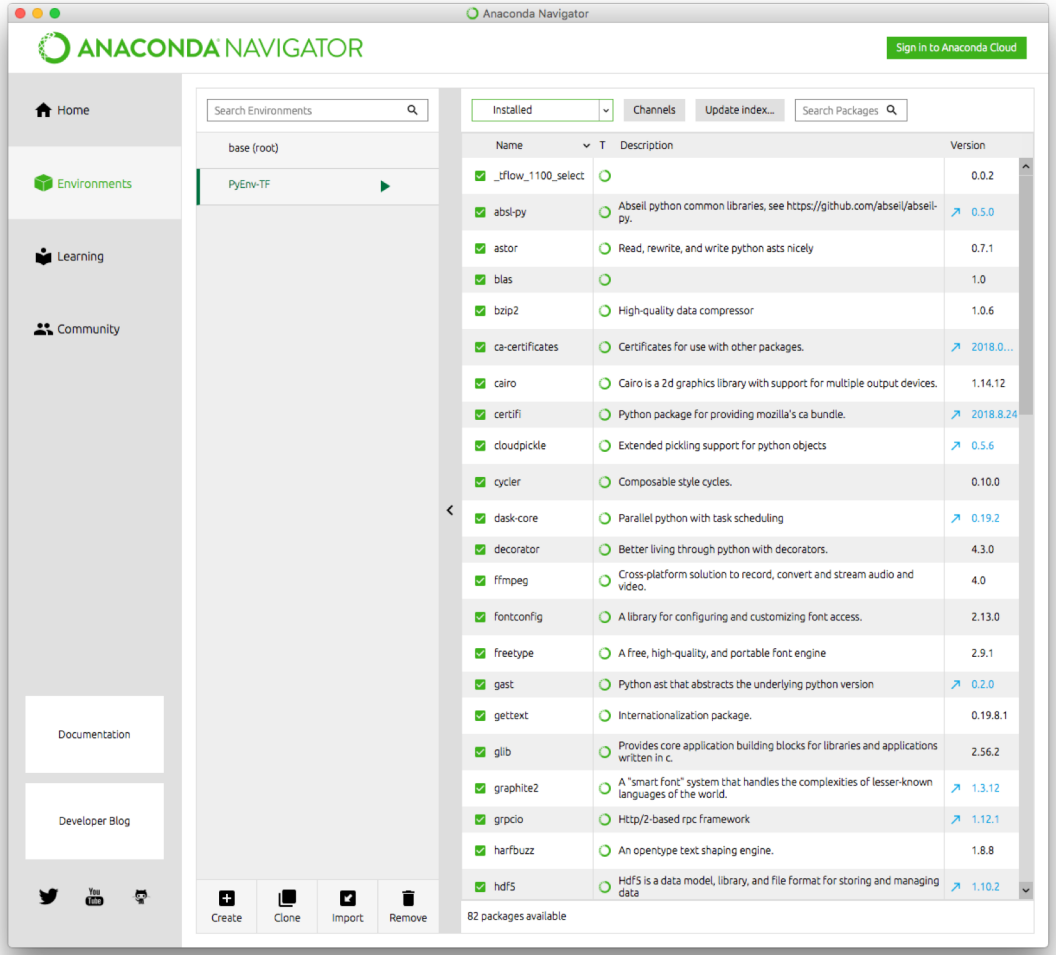




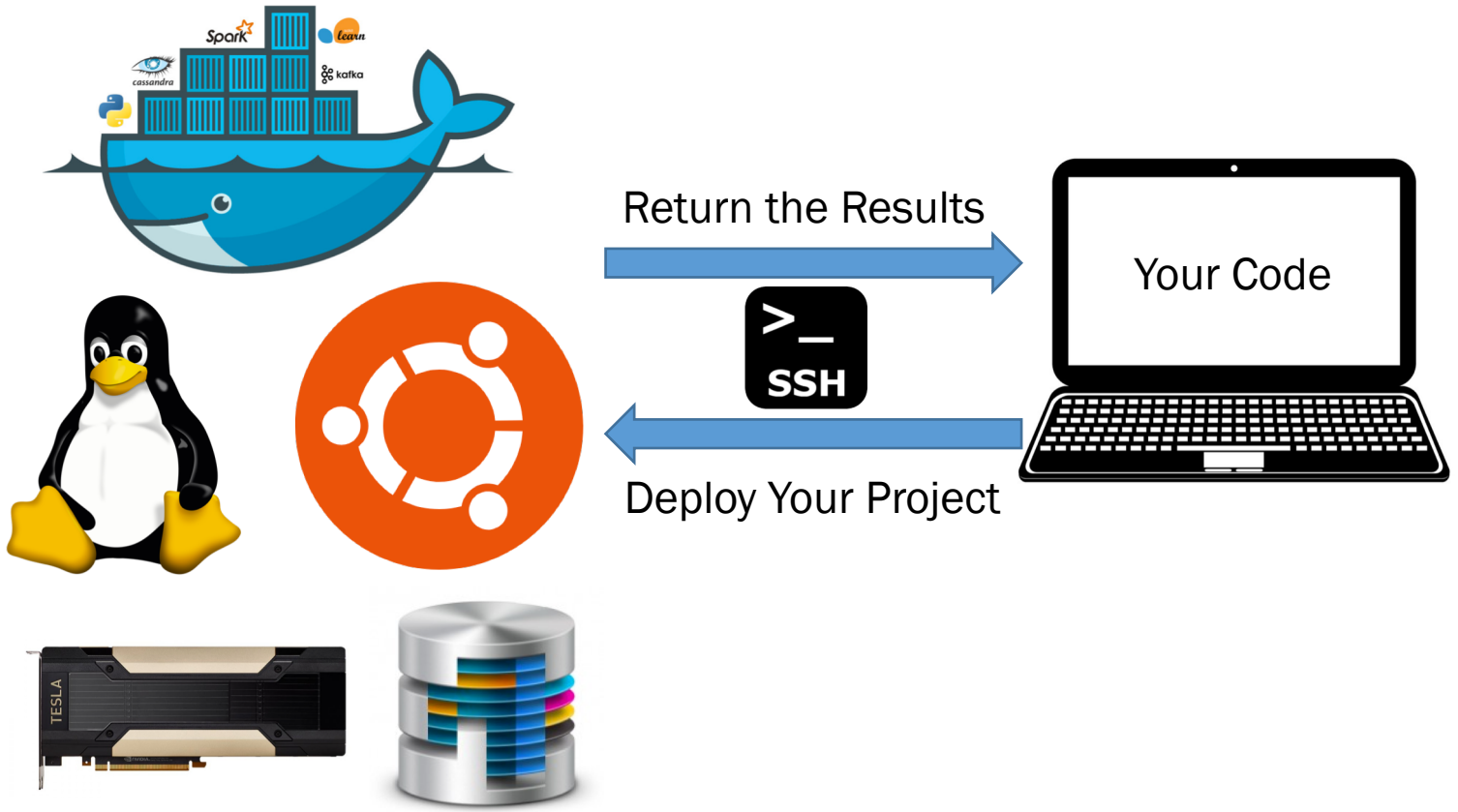
a) https://virtualenv.pypa.io/en/latest/

Anaconda for Newbie



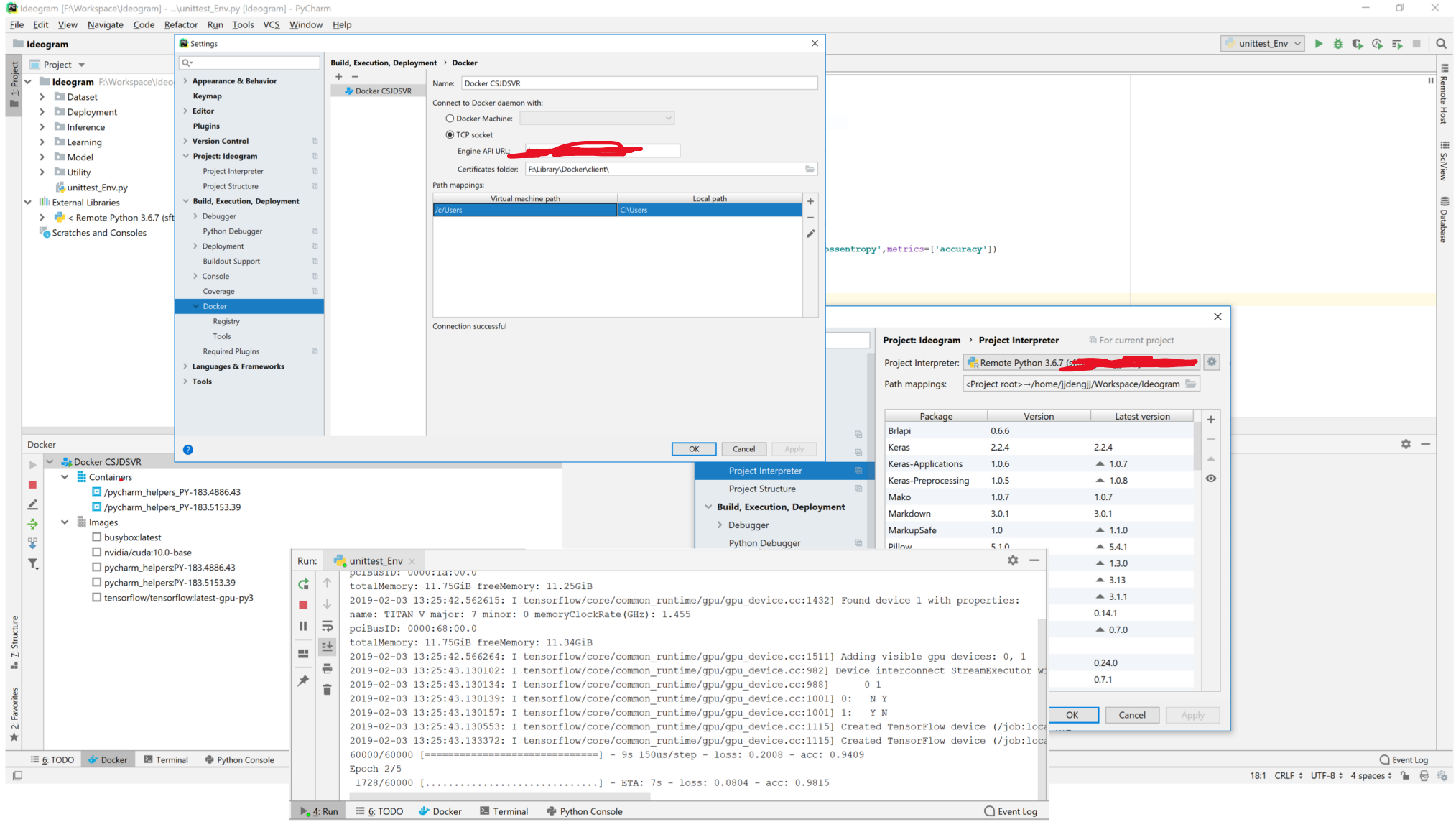


Remote Server or Docker for Professional



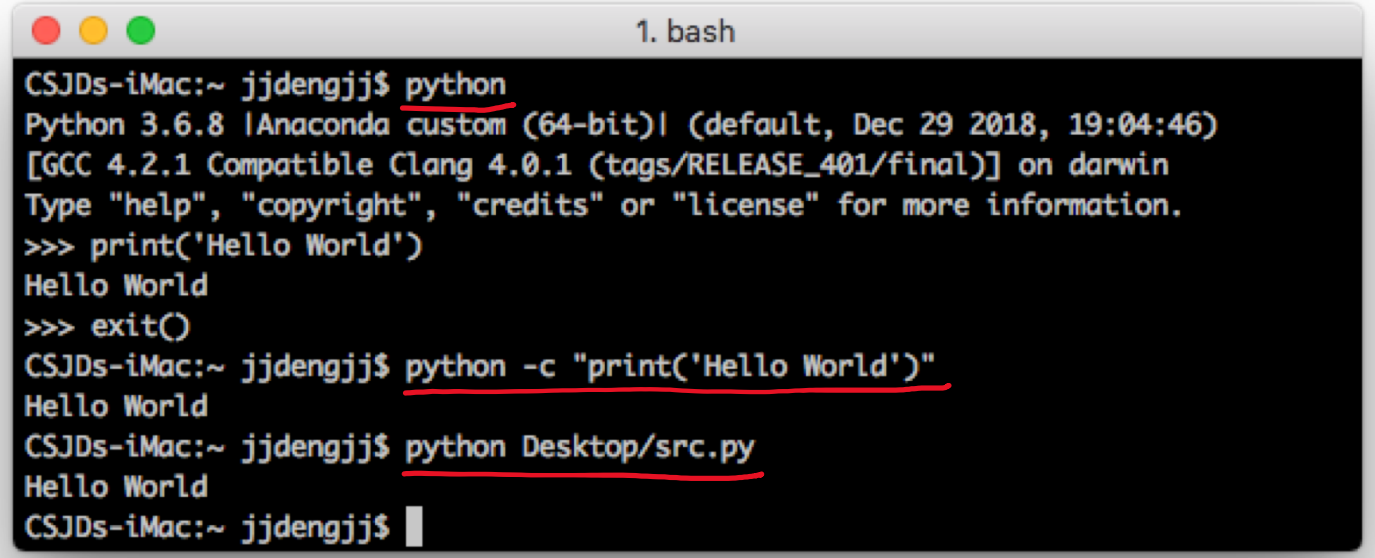
a) Welcome to my office for viewing those lovely pieces.

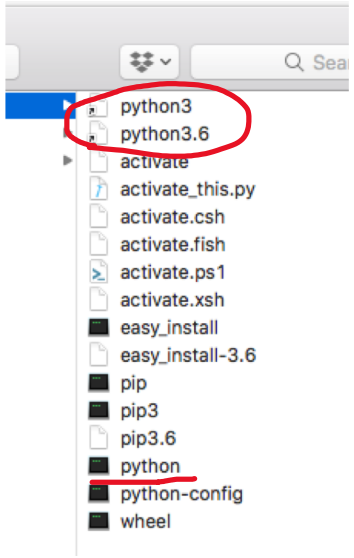
Developing with PyCharm



Using Python Interpreter

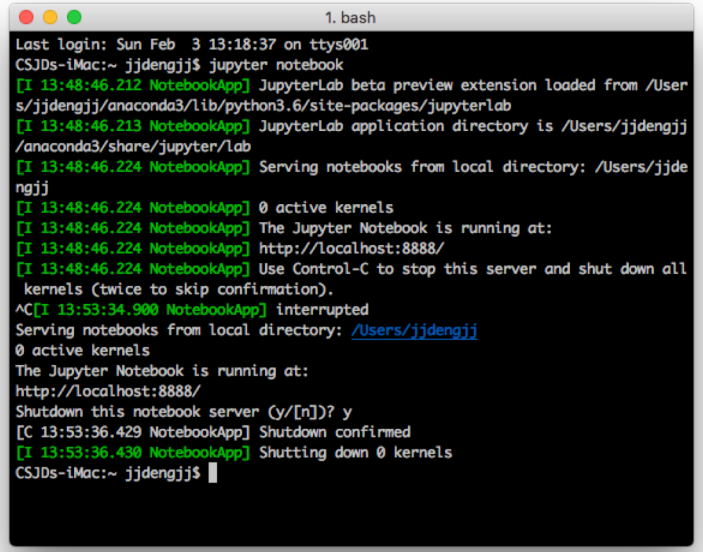
- The binary distribution of *python or python.exe* is the python language interpreter, similar to *java* or *java.exe*.

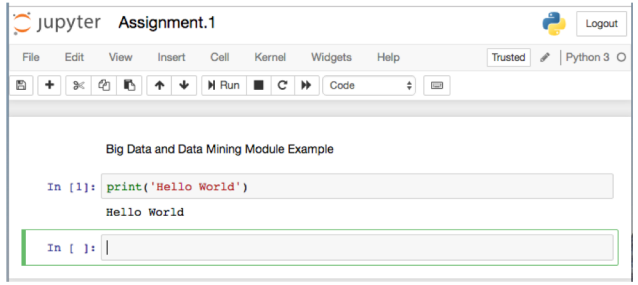


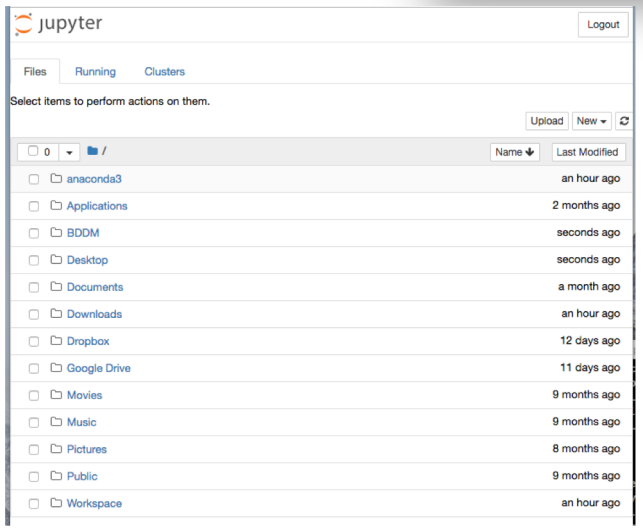


Using Jupyter Environment

- Web-based Python programming environment
  - Ideally for python beginner
- Installation: *pip install jupyter*
- Get Started: *jupyter notebook*
  - Running a small web server on your local machine
  - http://localhost:8888/tree
- Stop Jupyter:
  - Press "*CTRL+C*" in the terminal





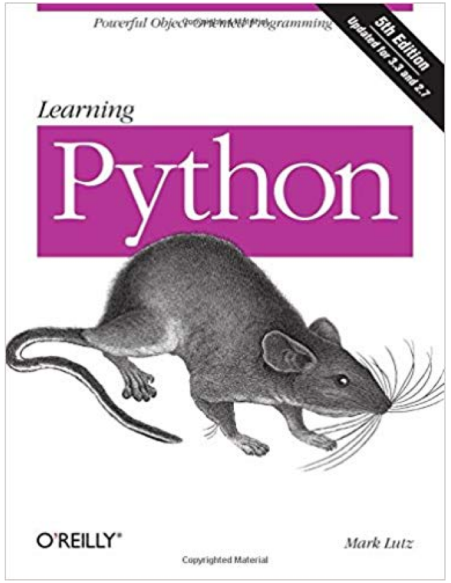


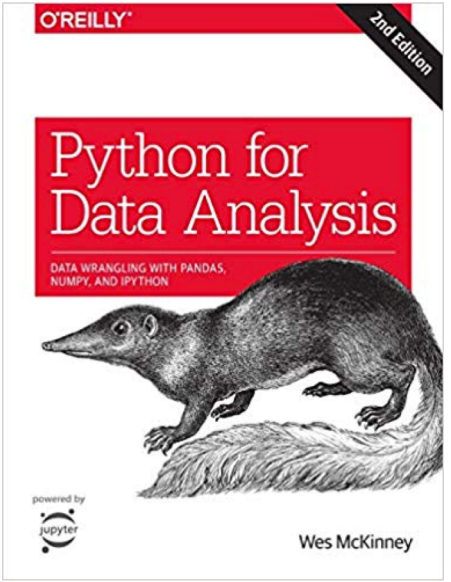
a) https://jupyter.org/

Essential Programming

- Logic and Numerical Operation
  - *i.e. ==, =, +, -, *, / etc.*
- Data Structure
  - *i.e. List, Dictionary, String etc.*
- Programme Structure
  - *i.e. Condition, Loop, Function etc.*
- Project Structure and OOP
  - *i.e. Module Import, Class etc.*
- Standard Library
  - i.e. built-in function, basic IO *etc.*
- External Library and ***More…***

This is NOT another programming module, so we are NOT going to cover all language features, but we will review the essential knowledge to get you started in this lecture.





Logic and Numerical Operations

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * 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
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`float(x)`	*x* converted to floating point
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

Logic and Numerical Operations

```
In [ ]: a=10
        a.
```

```
bit_length
conjugate
denominator
from_bytes
imag
numerator
real
to_bytes
```

Operation	Result
`x | y`	bitwise *or* of *x* and *y*
`x ^ y`	bitwise *exclusive or* of *x* and *y*
`x & y`	bitwise *and* of *x* and *y*
`x << n`	*x* shifted left by *n* bits
`x >> n`	*x* shifted right by *n* bits
`~x`	the bits of *x* inverted

```
In [6]: a=10
        print(bin(a))
        b=14
        print(bin(b))
        c=a^b
        print(c)
        print(bin(c))
```

```
0b1010
0b1110
4
0b100
```

```
In [7]: a=10
        print(a.bit_length())
        print(bin(a))
```

```
4
0b1010
```

a) More standard numerical operations/functions are defined in math *etc.* modules.

Data Structure – Sequence Types

- List – Mutable
  - Using a pair of square brackets to denote the empty list: []
  - Using square brackets, separating items with commas: [a], [a, b, c]
- Tuple – Immutable
  - Using a pair of parentheses to denote the empty tuple: ()
  - Using a trailing comma for a singleton tuple: a, or (a,)
- Range – Immutable
  - The *range* type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in *for* loops.
- An object with a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered. A new object has to be created if a different value has to be stored.

Data Structure – Sequence Types

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

Note: brackets and square brackets

Operation	Result	Notes
`s[i] = x`	item *i* of *s* is replaced by *x*	
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*	
`del s[i:j]`	same as `s[i:j] = []`	
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*	(1)
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list	
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)	
`s.clear()`	removes all items from *s* (same as `del s[:]`)	(5)
`s.copy()`	creates a shallow copy of *s* (same as `s[:]`)	(5)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)	
`s *= n`	updates *s* with its contents repeated *n* times	(6)
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)	
`s.pop([i])`	retrieves the item at *i* and also removes it from *s*	(2)
`s.remove(x)`	remove the first item from *s* where `s[i]` is equal to *x*	(3)
`s.reverse()`	reverses the items of *s* in place	(4)

Mutable type only

Data Structure – Sequence Types

- *class* range(*stop*)
- *class* range(*start*, *stop*[, *step*])
  - For a positive *step*, the contents of a range r are determined by the formula *r[i] = start +step*i* where *i >= 0 and r[i] < stop*.
  - For a negative *step*, the contents of the range are still determined by the formula *r[i] = start +step*i*, but the constraints are *i >= 0 and r[i] > stop*.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
>>> list(range(1, 0))
[]
```

```
In [11]: i = range(10)
         print(i)
         print(list(i))

         range(0, 10)
         [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Note: *range* gives an object not a list of value

Data Structure – Dictionary

- A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the dictionary.
  - *i.e.* vehicle registration number

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```

In [14]:

```
a = dict(one=1, two=2, three=3)
b = {'one': 1, 'two': 2, 'three': 3}
c = b
print(a==b)
print(a is b)
print(c is b)
c['one'] = 0
print(b)
```

```
True
False
True
{'one': 0, 'two': 2, 'three': 3}
```

Data Structure – Set

- A set object is an unordered collection of distinct hashable objects. Common uses include membership testing, **removing duplicates from a sequence**, and computing mathematical operations such as intersection, union, difference, and symmetric difference.

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Data Structure – String

A='100' *v.s.* A=100 ?

- Single quotes as *str* variable: 'allows embedded "double" quotes'
- Double quotes as *str* variable: "allows embedded 'single' quotes".
- Triple quoted: '''Three single quotes''', """Three double quotes"""
  - Triple quoted is for multiple line comments.
  - Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.
  - # is for single line comment.

```
def architecture(self, sizeCBlocks=_defaultCBlock, sizeDBlocks=_defaultDBlock, nonLinear='relu', onDropout=0, onBatchNorm=False):
    """
    Build LeNet Architecture
    :param sizeCBlocks: A List of Tuple of Convolutional Kernel Configuration, (nFilter, sizeKernel, sizePooling, sizeStride)
    :param sizeDBlocks: A List of Integer of Dense Layer Configuration, [nNeuros]
    :param nonLinear: Name of Activation Function, See https://keras.io/activations/
    :param onDropout: 0 = No Random Dropout, Positive Non-Zero Value (0, 1) = Percenta
    :param onBatchNorm: Perform Batch Normalization After Activation
    :return: self._model
    """
```

```
# ==============================================================================
# Copyright 2017 The DLPractice Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
ler the License.

gjing Deng (jjdengjj@gmail.com)
==============================================================
```

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
```

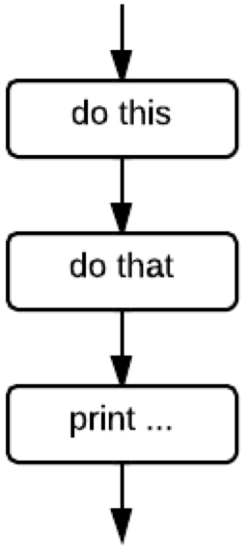
a) More *str* operations/functions can be found in python reference document.

Programme Structure – Condition Control

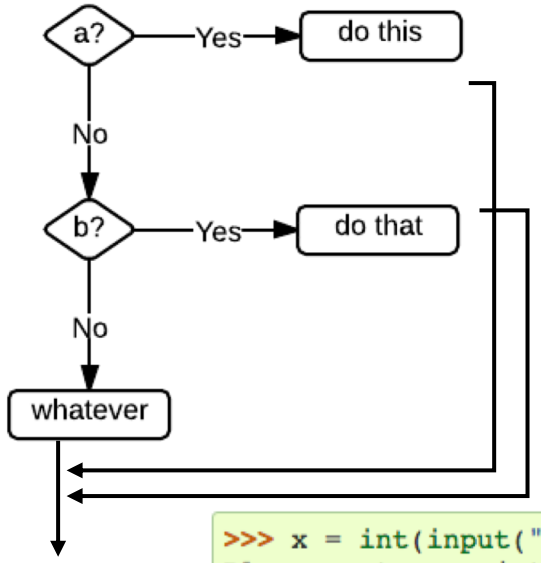
```
do this

do that

print ...
```




```
if a:
    do this
elif b:
    do that
else:
    whatever
```



Don’t forget the Colon.

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Note: Common logic mistake is that the conditions are not exclusive between each others.

a) https://www.codecademy.com/en/forum_questions/51684a3d4ce76309b4001b9c

Programme Structure – For Loop Flow

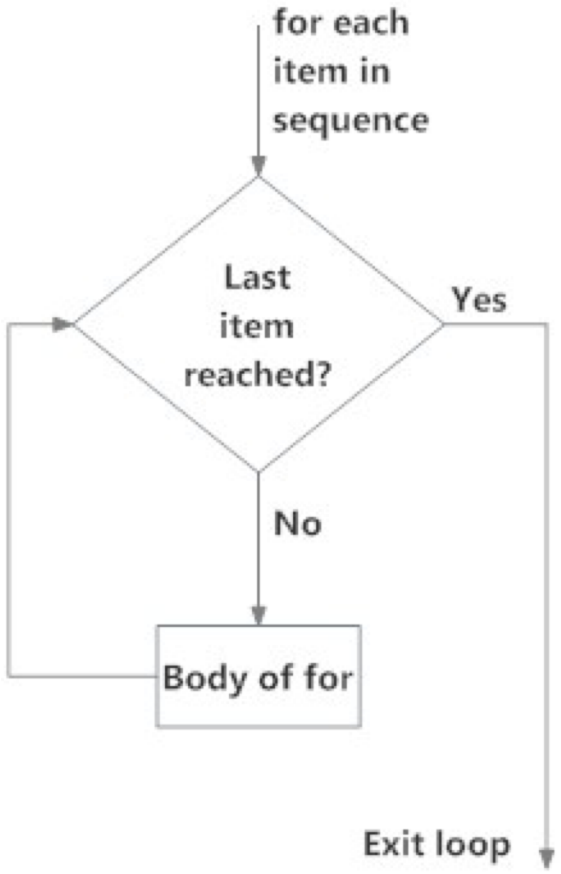


Fig: operation of for loop

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Don’t forget the Colon.

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

a) https://www.datamentor.io

Programme Structure – Loop with else

- Loop statements may have an else clause; it is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while), but not when the loop is terminated by a break statement.
- *break* and *continue*

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Don’t forget the Colon.

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found a number", num)
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

Programme Structure – While Loop Flow

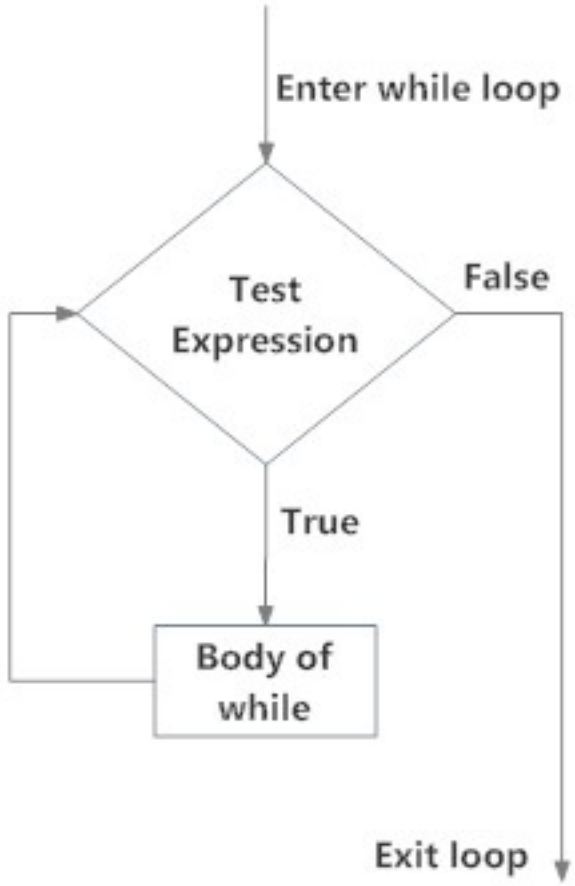


Fig: operation of while loop

Don’t forget the Colon.

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

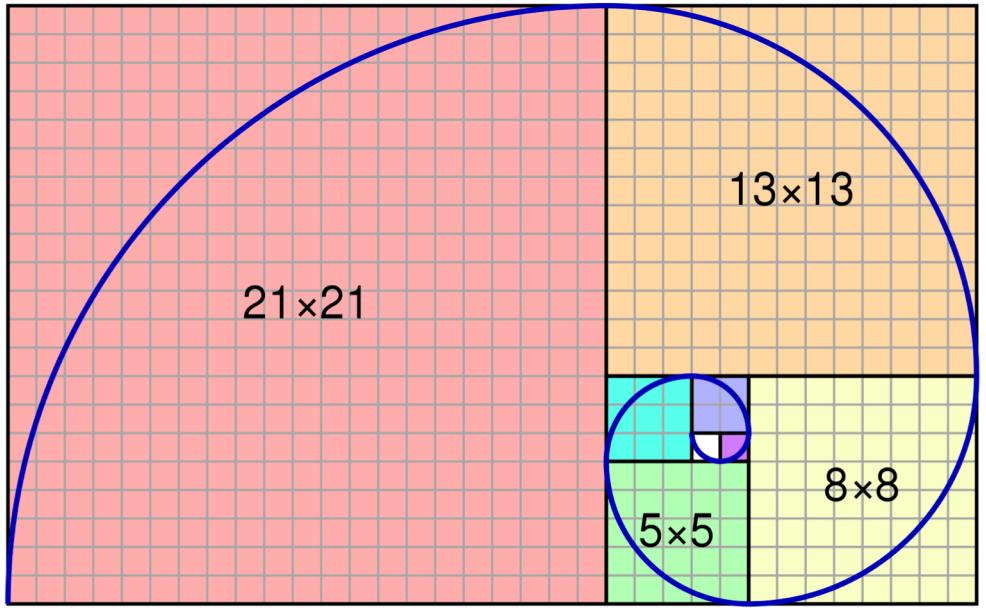
a) https://www.datamentor.io

Programme Structure – Function

```
>>> def fib2(n):  # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

The keyword *def* introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line, and must be indented.

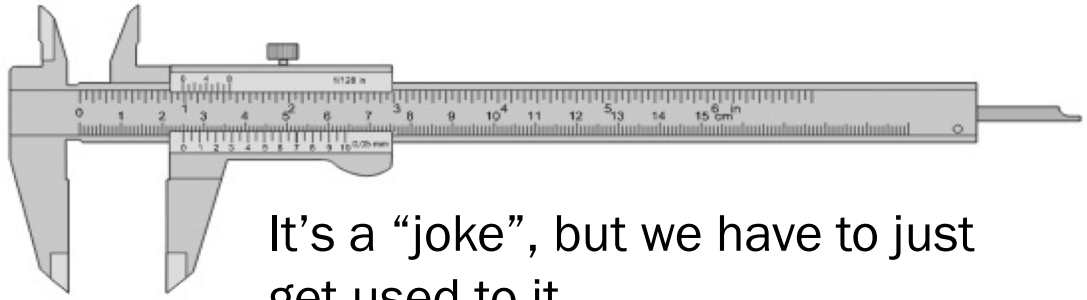
Don't forget the Colon.



The Fibonacci spiral: an approximation of the golden spiral created by drawing circular arcs connecting the opposite corners of squares in the Fibonacci tiling

Python Code Block – Indentation

- *if-elif-else* block
- *while* loop block
- *for* loop block
- function *def* block
- PEP 8 -- Style Guide for Python Code
  - PyCharm enforce this style and check your code again the rules by default.
- It includes Code layout, comment *etc.*
- It helps to improve the readability.



It's a "joke", but we have to just get used to it.

- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module Level Dunder Names
- String Quotes

a) https://www.python.org/dev/peps/pep-0008/

Project Structure – Module and Import

- If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost.
  - Use a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a *script*.
  - Split it into several files for easier maintenance.
  - Reuse a handy function that you've written in several programs without copying its definition into each program.

fibo.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

```
>>> import fibo
```

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import *
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> import fibo as fib
>>> fib.fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

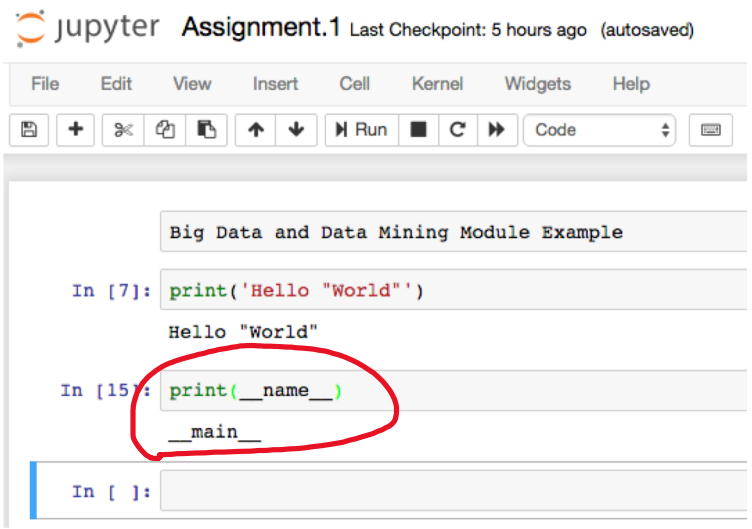
```
>>> from fibo import fib as fibonacci
>>> fibonacci(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Running your module as a script

```
>>> import fibo
```

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```




```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Run as a script

```
$ python fibo.py 50
0 1 1 2 3 5 8 13 21 34
```

Import as a module

```
>>> import fibo
>>>
```

Project Structure – Where is my module

- The Module Search Path
  - When a module named spam is imported, the interpreter first searches for a built-in module with that name.
  - If not found, it then searches for a file named *xxxx.py* in a list of directories given by the variable sys.path.

```
In [24]: import sys
         for x in sys.path:
             print(x)

''
/Users/jjdengjj/anaconda3/lib/python36.zip
/Users/jjdengjj/anaconda3/lib/python3.6
/Users/jjdengjj/anaconda3/lib/python3.6/lib-dynload
/Users/jjdengjj/anaconda3/lib/python3.6/site-packages
/Users/jjdengjj/anaconda3/lib/python3.6/site-packages/aeosa
/Users/jjdengjj/anaconda3/lib/python3.6/site-packages/IPython/extensions
/Users/jjdengjj/.ipython
```

Project Structure – Package or Library

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

- Import individual modules from the package
  - *import sound.effects.echo*
  - *from sound.effects import echo*
  - *echo.echofilter(input, output, delay=0.7, atten=4)*
- Import individual function from the module
  - *from sound.effects.echo import echofilter*
  - *echofilter(input, output, delay=0.7, atten=4)*
- Intra-package references
  - *from . import echo*
  - *from .. import formats*
  - *from ..filters import equalizer*

Object Oriented Programming

In [26]:
```
class Person:

    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last

    def Name(self):
        return self.firstname + " " + self.lastname

class Employee(Person):

    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum

    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber

x = Person("Marge", "Simpson")
y = Employee("Homer", "Simpson", "1007")

print(x.Name())
print(y.GetEmployee())
```

```
Marge Simpson
Homer Simpson, 1007
```

- Class Definition
- Initialisation Function
- Member Variable
- Member Function
- Inheritance
  - Initialise base class
- Object Instantiation
- Object Usage

Standard Library – Basic File IO

```
>>> f = open('workfile', 'w')
```

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

Always forget to close your file?
Python has a good solution!

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

file.read([size])
Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes).

file.readline([size])
Reads one entire line from the file. A trailing newline character is kept in the string.

file.write(str)
Writes a string to the file. There is no return value.

file.writelines(sequence)
Writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.

file.seek(offset[, whence])
Sets the file's current position

file.flush()
Flush the internal buffer, like stdio's fflush. This may be a no-op on some file-like objects.

Standard Library – Built-in Function

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Standard Library – Overview

10. Brief Tour of the Standard Library
    - 10.1. Operating System Interface
    - 10.2. File Wildcards
    - 10.3. Command Line Arguments
    - 10.4. Error Output Redirection and Program Termination
    - 10.5. String Pattern Matching
    - 10.6. Mathematics
    - 10.7. Internet Access
    - 10.8. Dates and Times
    - 10.9. Data Compression
    - 10.10. Performance Measurement
    - 10.11. Quality Control
    - 10.12. Batteries Included

11. Brief Tour of the Standard Library — Part II
    - 11.1. Output Formatting
    - 11.2. Templating
    - 11.3. Working with Binary Data Record Layouts
    - 11.4. Multi-threading
    - 11.5. Logging
    - 11.6. Weak References
    - 11.7. Tools for Working with Lists
    - 11.8. Decimal Floating Point Arithmetic

Note: Always refer to the official documents, we cannot remember them all.

a) https://docs.python.org/3.7/tutorial/index.html

External Library – For Data Mining

- *NumPy* and *SciPy*
  - Numerical computation
- *Matplotlib* and *OpenCV*
  - Data visualisation and computer vision
- *pandas* and *scikit-learn*
  - Data pre-process and traditional machine learning
- *BeautifulSoup*
  - Data scrapping from internet
- *Tensorflow* and *Keras*
  - Deep Neural Network