Lectures Overview

Python Basics

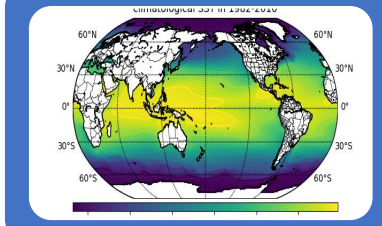
- Background
- Installation & setup
- Python Language

Python Advanced

- I/O & Exceptions Handling
- Modules & Packages
- Object-Oriented Programming in Python

Python for Scientific Computation

- Array computation with Numpy
- Scipy
- Draw common 2D figures with Matplotlib



Python for Oceanography

- Read/Write netCDF files
- Draw data on maps with basemap

Introduction to Python3

Wenming Wu

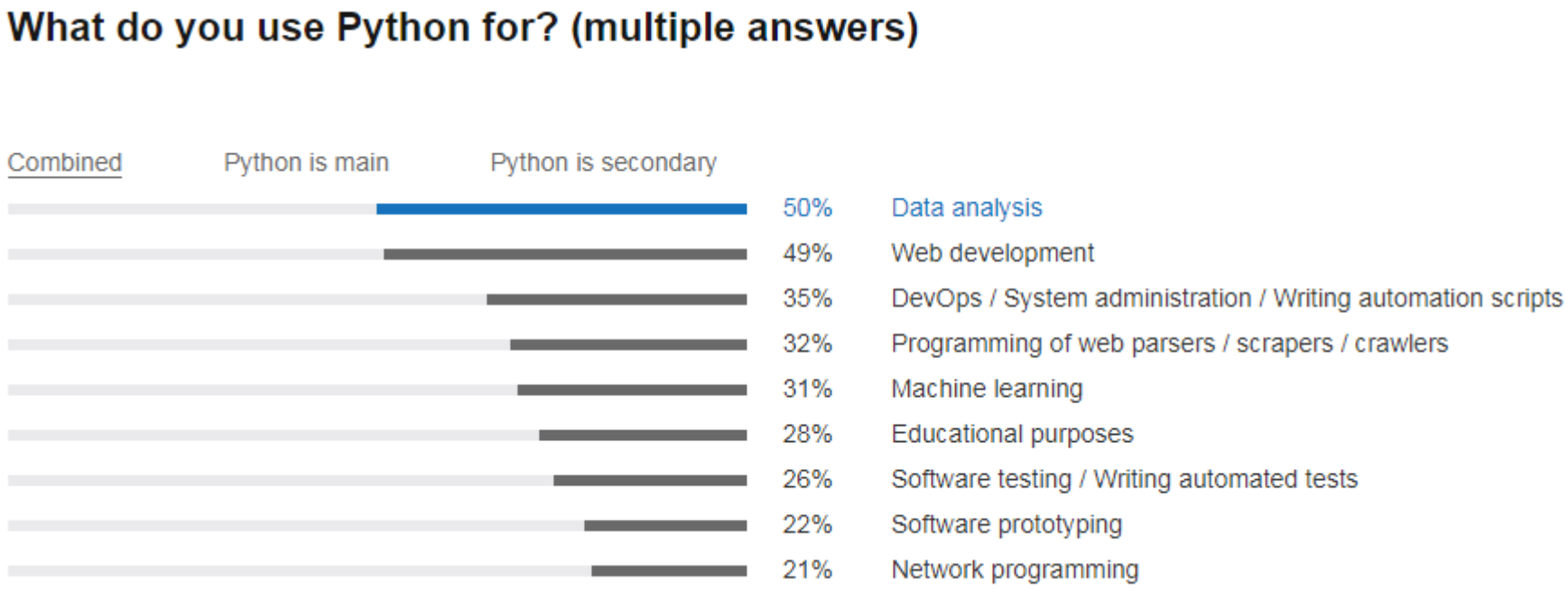
06/20/2018

Basics

- Why Python ?
- Installation & setup
- Identifies & keywords
- Data type
  - Basic types
    - Integral
    - Floating-point
    - String
  - Collection data types
    - Sequence
    - Mapping
- Control flow
- Function

Why Python ?

- Open source general-purpose language



Source: [Python Developers Survey 2017 Results]

Why Python ?

- “People want to use Python because of its **intuitiveness**, **beauty**, **philosophy**, and **readability**.”*

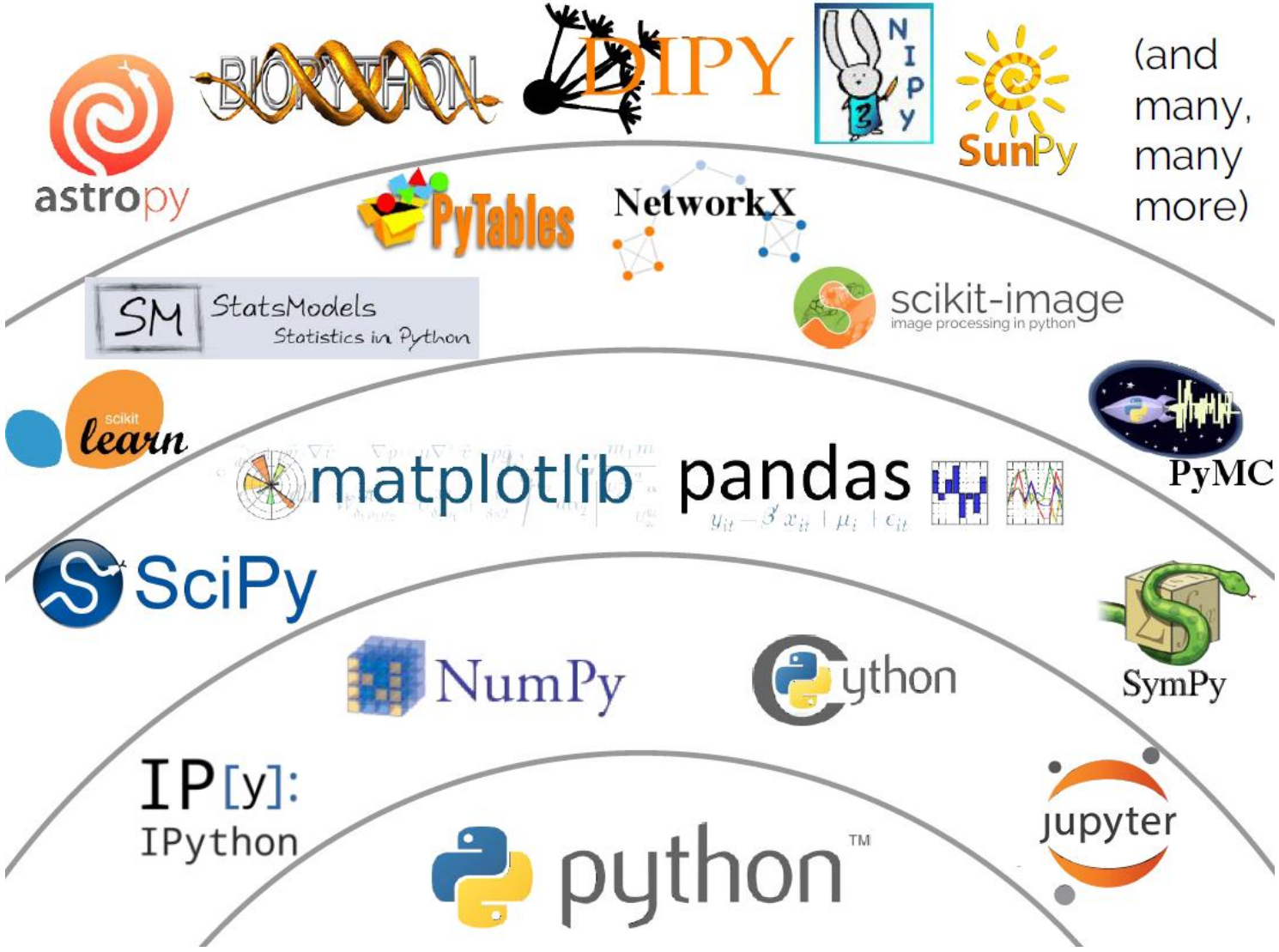
```
x = 34 - 23         # A comment.
y = "Hello"         # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"     # String concat.
print (x)
print (y)



12
HelloWorld
```

- “So people build Python packages that incorporate lessons learned in other tools & communities.”*

* quotes from: Jake VandPlas 2017 PyData Talk

Why Python for Scientific Computation ?

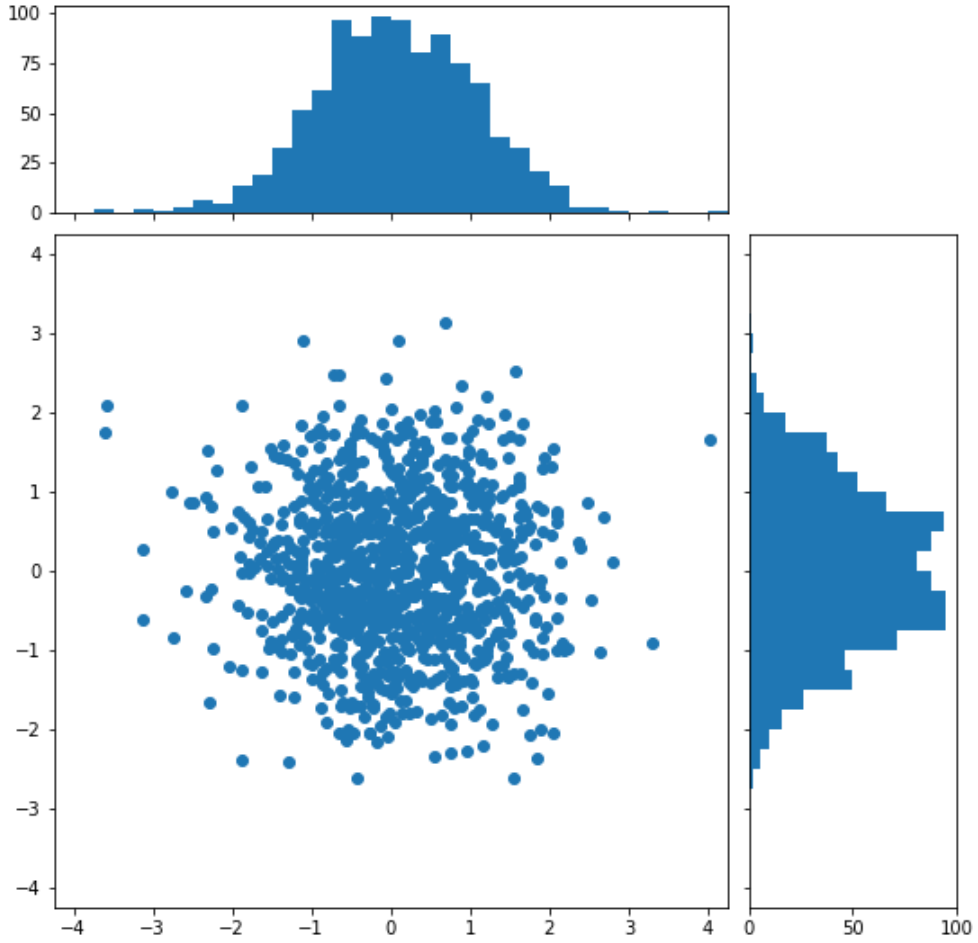
- Easily work with Fortran/C/C++ etc.
- Rich built-in and 3rd party packages for both common and specific domains

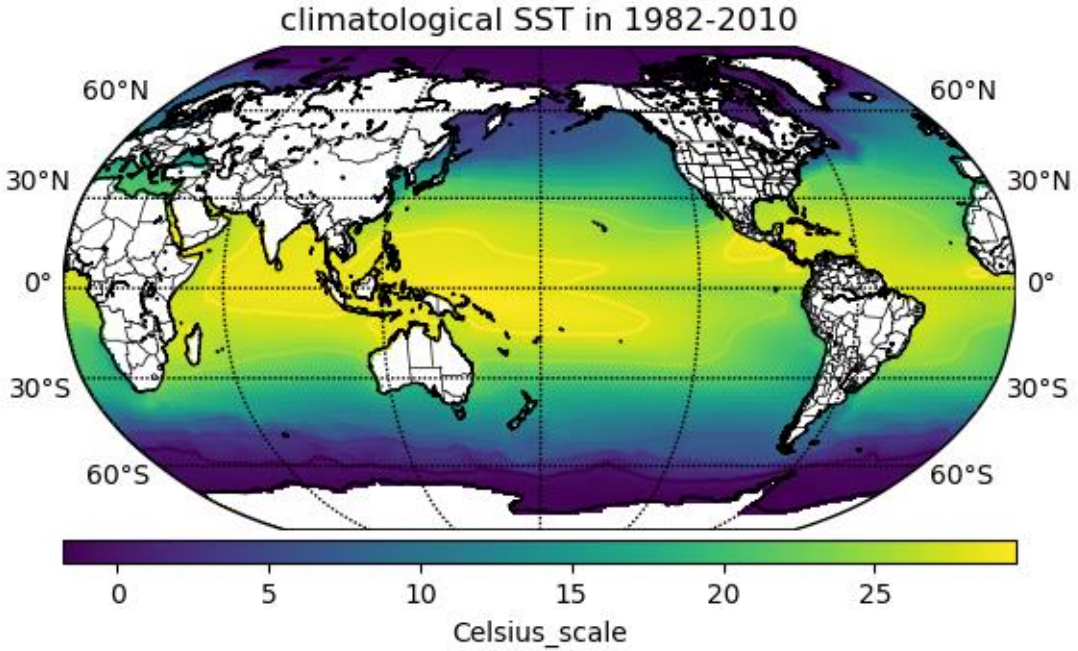


source: Jake VandPlas 2015 SciPy Talk

Why Python for Scientific Computation ?

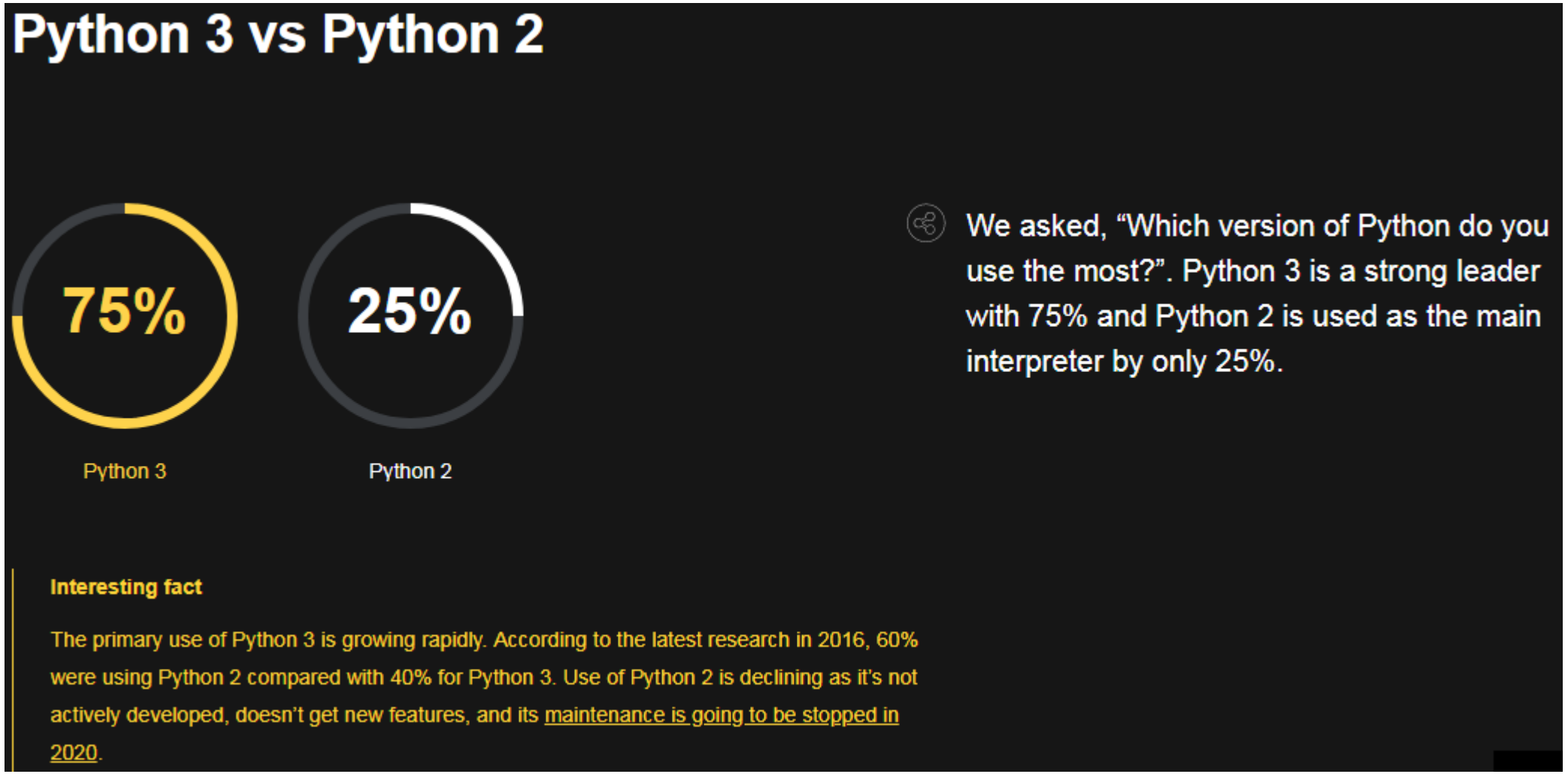
- High-quality figure plotting





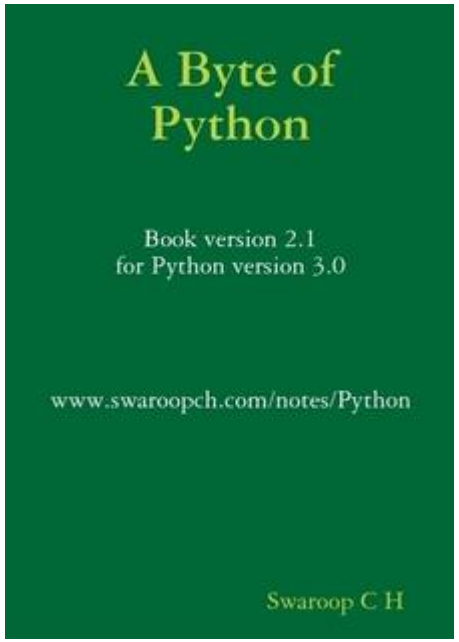
Why Python 3 instead of Python 2 ?

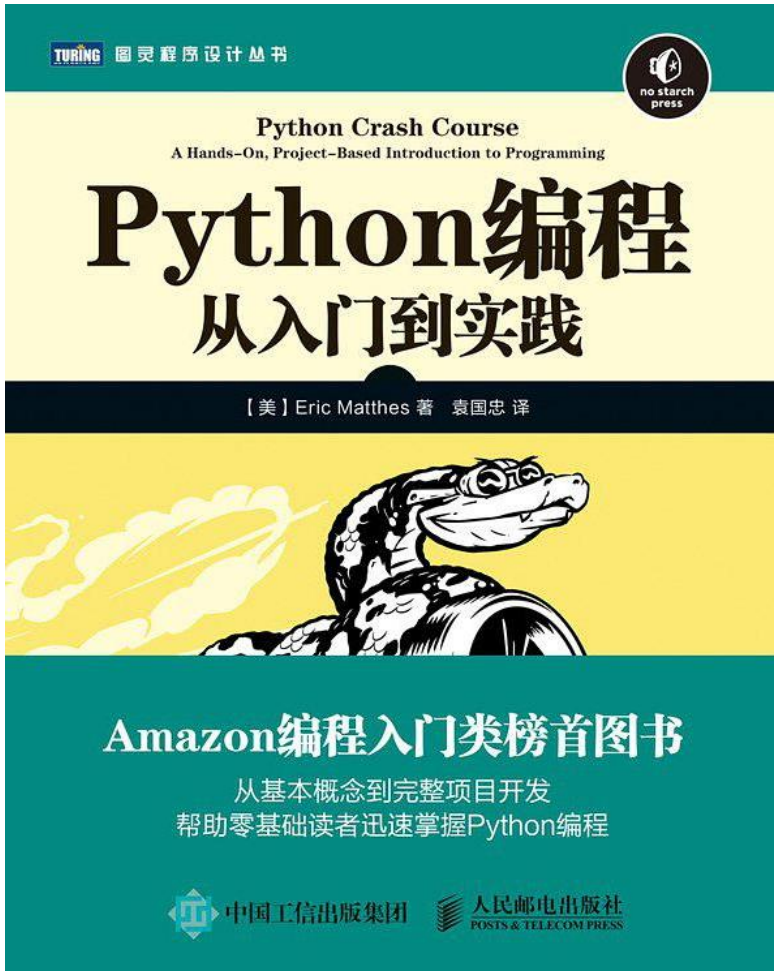
- “Python 2.x is legacy, Python 3.x is the present and future of the language”

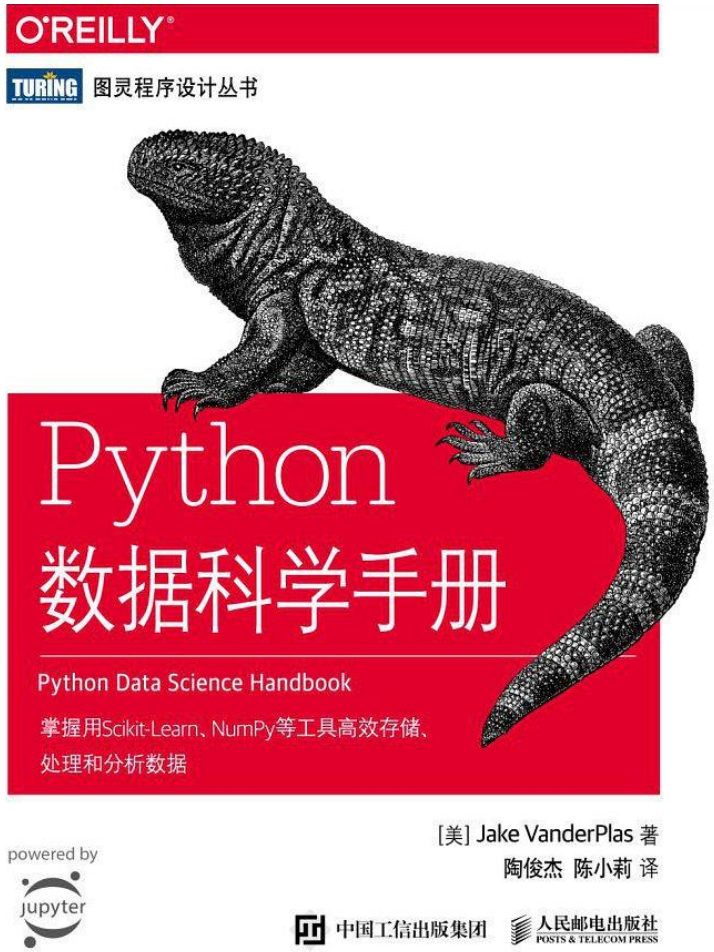


Source: Python Developers Survey 2017 Results

Textbooks


A Byte of Python
Book version 2.1
for Python version 3.0
www.swaroopch.com/notes/Python
Swaroop C H


TURING 图灵程序设计丛书
no starch press
Python Crash Course
A Hands-On, Project-Based Introduction to Programming
Python编程
从入门到实践
【美】Eric Matthes 著 袁国忠 译
Amazon编程入门类榜首图书
从基本概念到完整项目开发
帮助零基础读者迅速掌握Python编程
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS


O'REILLY
TURING 图灵程序设计丛书
Python
数据科学手册
Python Data Science Handbook
掌握用Scikit-Learn、NumPy等工具高效存储、
处理和分析数据
powered by
jupyter
[美] Jake VanderPlas 著
陶俊杰 陈小莉 译
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Installation

- Conda is an open source package management system focused on Python for scientific computing, which runs on Windows, macOS and Linux.
- Two approaches
  - Miniconda: a minimal installation of the conda command-line tool
  - Anaconda: miniconda plus lots of common used Python packages
- Suggest to download from local repository mirrors instead
  - https://mirrors.ustc.edu.cn/anaconda/archive

Anaconda3-5.2.0-Linux-ppc64le.sh	288.3 MiB	2018-05-31 02:37
Anaconda3-5.2.0-Linux-x86.sh	507.3 MiB	2018-05-31 02:37
Anaconda3-5.2.0-Linux-x86_64.sh	621.6 MiB	2018-05-31 02:38
Anaconda3-5.2.0-MacOSX-x86_64.pkg	613.1 MiB	2018-05-31 02:38
Anaconda3-5.2.0-MacOSX-x86_64.sh	523.3 MiB	2018-05-31 02:39
Anaconda3-5.2.0-Windows-x86.exe	506.3 MiB	2018-05-31 02:41
Anaconda3-5.2.0-Windows-x86_64.exe	631.3 MiB	2018-05-31 02:41

Setup

- Verify the installation

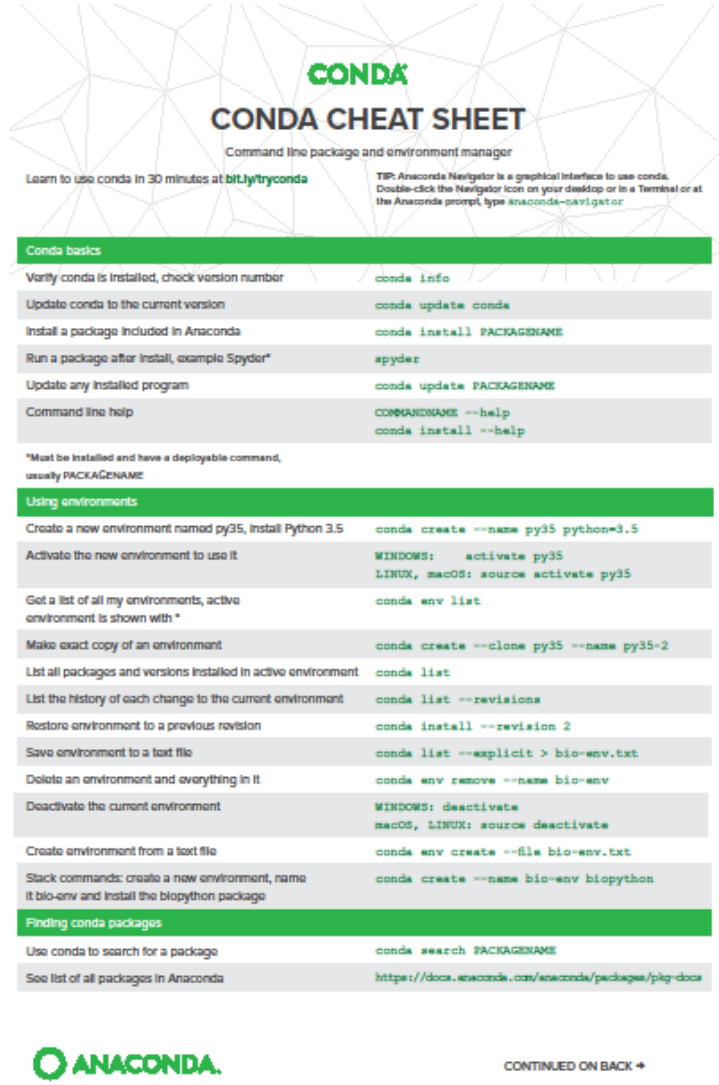
```
Anaconda Prompt - conda  install numpy pandas matplotlib

(base) C:\Users\wewu>python --version
Python 3.6.5 :: Anaconda custom (64-bit)
```

- Setup updating from local repository channels

```
(base) C:\Users\wewu>conda config --add channels https://mirrors.tuna.tsinghua.e
du.cn/anaconda/pkgs/main

(base) C:\Users\wewu>conda config --add channels https://mirrors.tuna.tsinghua.e
du.cn/anaconda/pkgs/free

(base) C:\Users\wewu>conda config --show channels
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - Defaults
```

- Install 3rd party packages according to your need

```
(base) C:\Users\wewu>conda install numpy pandas matplotlib netcdf4
Solving environment: done

## Package Plan ##

  environment location: C:\Users\wewu\Anaconda3

  added / updated specs:
    - matplotlib
    - netcdf4
    - numpy
    - pandas


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    libssh2-1.8.0              |           vc14_0         189 KB  https://mirror
s.tuna.tsinghua.edu.cn/anaconda/pkgs/free
    matplotlib-2.0.2           |      np113py36_0         8.5 MB  https://mirror
```



CONDA

CONDA CHEAT SHEET

Command line package and environment manager

Learn to use conda in 30 minutes at bit.ly/tryconda

TIP: Anaconda Navigator is a graphical interface to use conda. Double-click the Navigator icon on your desktop or in a Terminal or at the Anaconda prompt, type anaconda-navigator

Conda basics	
Verify conda is installed, check version number	conda info
Update conda to the current version	conda update conda
Install a package included in Anaconda	conda install PACKAGENAME
Run a package after install, example Spyder*	spyder
Update any installed program	conda update PACKAGENAME
Command line help	COMMANDNAME --help conda install --help

*Must be installed and have a deployable command, usually PACKAGENAME

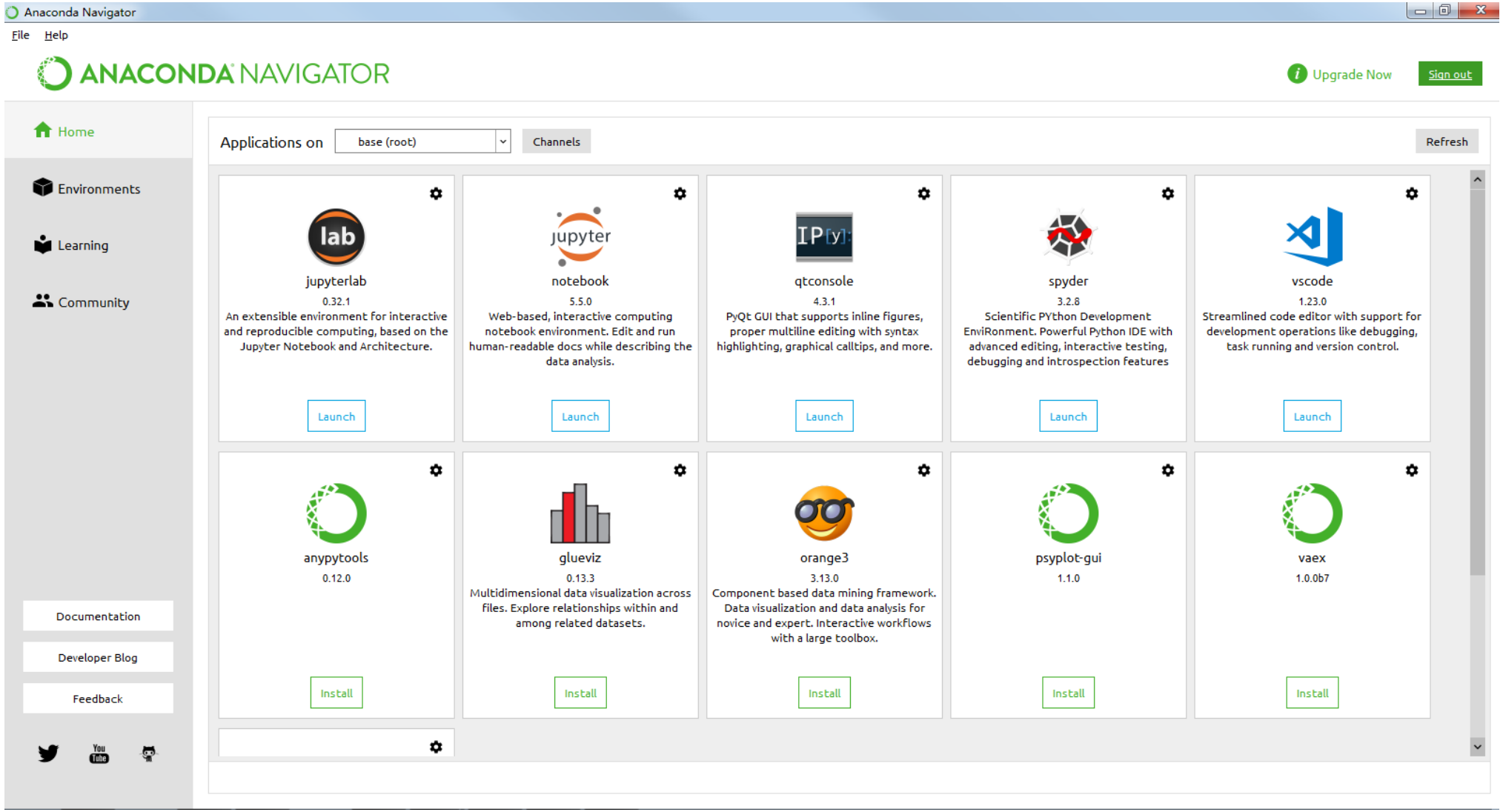
Using environments	
Create a new environment named py35, install Python 3.5	conda create --name py35 python=3.5
Activate the new environment to use it	WINDOWS: activate py35 LINUX, macOS: source activate py35
Get a list of all my environments, active environment is shown with *	conda env list
Make exact copy of an environment	conda create --clone py35 --name py35-2
List all packages and versions installed in active environment	conda list
List the history of each change to the current environment	conda list --revisions
Restore environment to a previous revision	conda install --revision 2
Save environment to a text file	conda list --explicit > bio-env.txt
Delete an environment and everything in it	conda env remove --name bio-env
Deactivate the current environment	WINDOWS: deactivate macOS, LINUX: source deactivate
Create environment from a text file	conda env create --file bio-env.txt
Stack commands: create a new environment, name it bio-env and install the biopython package	conda create --name bio-env biopython

Finding conda packages	
Use conda to search for a package	conda search PACKAGENAME
See list of all packages in Anaconda	https://docs.anaconda.com/anaconda/packages/pkg-docs

ANACONDA.

CONTINUED ON BACK →

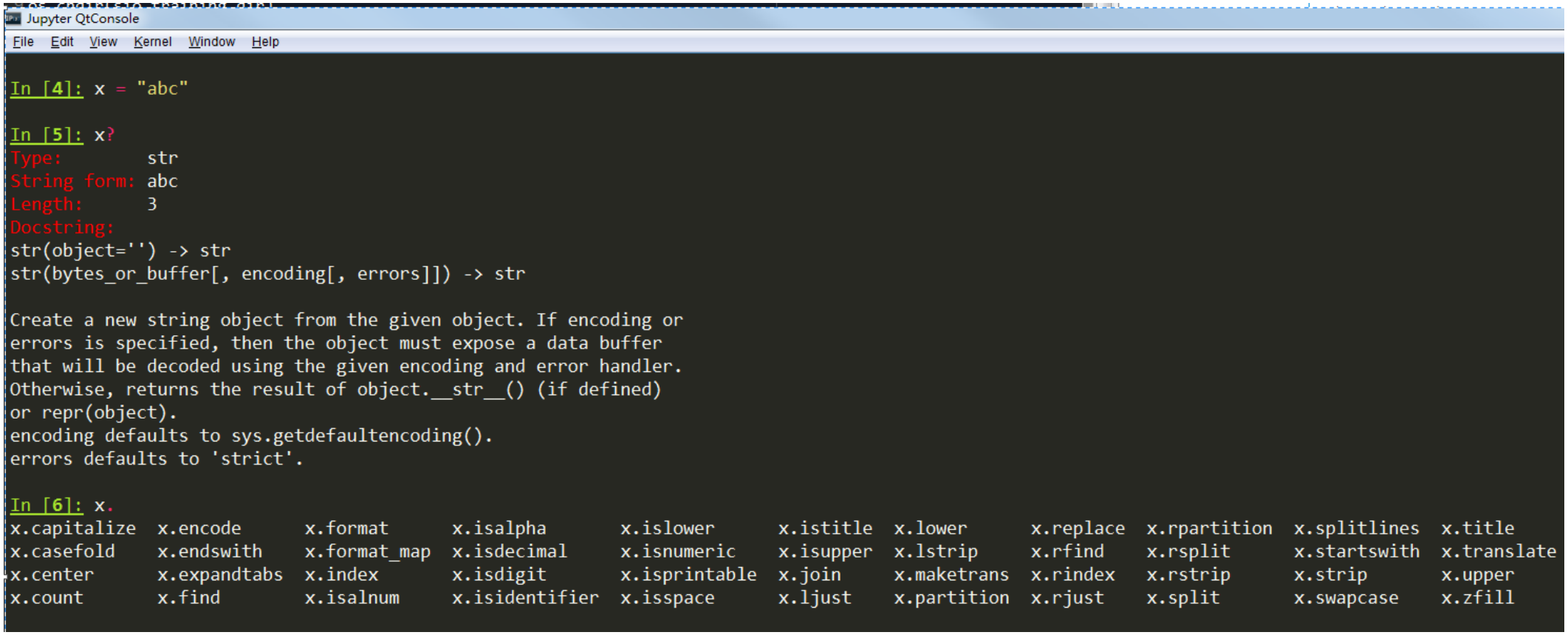
Anaconda Navigator



Get Help & Documents in IPython

"When a technologically-minded person is asked to help a friend, family member, or colleague with a computer problem, most of the time it's less a matter of knowing the answer as much as knowing how to quickly find an unknown answer. "

- Accessing Documentation/Source Code with ***?/??***
- Auto-Completion with ***Tab key***



A Code Sample

```
x = 34 - 23        # A comment.
y = "Hello"        # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
   x = x + 1
   y = y + "World"     # String concat.
print (x)
print (y)



12
HelloWorld
```

To understand the codes:

- ***Assignment*** uses **=** and comparison uses **==**
- Start comments with **#**: the rest of line is ignored
- For numbers, **+ - * / %** are as expected
  - Special use of **+** for string concatenation
  - Special use of **%** for string formatting (as with printf in C)
- Logical operators are ***keywords*** (**and**, **or** , **not**), not ***identifiers***
- The basic printing command is **print**
- The first ***assignment*** to a variable create it
  - **Variable types don't need to be declared**
  - Python figures out the variable types on its own
- Whitespace is meaningful in Python
  - **Use consistent indentation instead of braces { } to mark blocks of code**
  - Use \ when must go to next line prematurely

Identifiers & keywords

- Identifiers
  - Rule1: contain non-empty string with ***letters***, ***numbers***, and ***underscores***. Can not start with a number
  - Rule2: case sensitive and can not be reserved keywords
  - Recommend: to follow the PEP8 Python coding style
- Keywords

and	as	assert	break	class	continue	def	del
elif	else	except	False	finally	for	from	global
if	import	in	is	lambda	None	nonlocal	not
or	pass	raise	return	True	try	while	with
yield							

Assignment

- You create a name the first time it appears on the left side of an assignment expression:

  x = 3

- Names in Python do not have an intrinsic type. Objects have types.
  - Python determines the type of the reference automatically based on the data object assigned to it

```
>>>x = 3
>>>x
3
>>>type(x)
<class 'int’>
>>>type(3)
<class 'int'>
```

- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Binding a variable in Python means setting a name to hold a reference to some object.
  - Assignment manipulates references, not copies

Multiple Assignment

- You can also assign to multiple names at the same time :

```
>>> x, y =2, 3
>>> x
2
>>> y
3
```

Python Built-in Data Types

Boolean
True/False

```
if (number % 2) == 0:
    even = True
else:
    even = False
```

Numbers
Integers, Floats, and Complex

```
a = 5
b = 7.3
c = 2 +3j
```

Strings
Immutable sequences of Unicode characters

```
a = “This is a string”
```

Tuples
Immutable sequences of arbitrary Python objects

```
a= (“Edward”, 42)
```

Bytes
Immutable sequences of 8-bit bytes in range [0, 255]

Bytes Arrays
Mutable sequences of 8-bit bytes in range [0, 255]

```
b = bytes(b’Abc’)
ba = bytearray(b’Abc’)
```

Lists
Mutable sequences of arbitrary Python objects

```
a = [“Edward”, 42]
```

Sets
Mutable finite **sets** of unique and immutable Python object

Frozen Sets
Immutable finite **sets** of unique and immutable Python object

```
s = {“Edward”, 42}
fs = frozenset(s)
```

Dictionaries
Mutable finite **sets** of Python objects indexed by arbitrary index sets

```
d = {"name": "Edward", "age": 42}
```

Data Type: integer

- int: unlimited value (min/max value depends on machine memory capacity)
  - 123 #decimal 0b110011 #binary
  - 0o675351 #octal 0xDEF0A #hexadecimal
  - Operator:
    - Numerical operator: +, -, *, /, //, %, **
    - Bit operator: &, |, ^, <<, >>, ~
  - x operator= y ←→ x = x operator y
  - Functions: int(i), hex(i), bin(i)
- bool
  - True, False
  - Logical operator: and, or, not (***short-circuit calculation***)
  - Function: bool(x) return False when x is false or omitted, return True otherwise.
- **Methods** for int & bool types

```
In [114]: int.
int.bit_length  int.conjugate  int.denominator  int.from_bytes  int.imag  int.mro  int.numerator  int.real  int.to_bytes
```

Data Type: float-point

- float: ***float(x)***
  - 0.0 -2.5 8.98e-4
  - Double precision value
  - abs(a-b) <= sys.float_info.epsilon
  - ***math*** module: math.acos(x), math.ceil(x), math.e, math.log(x, b), …
- complex: ***complex(real, imag)***
  - a pair (***real***, ***imag***) of float: z = -89.5+2.125j z.real, z.imag: (-89.5, 2.125)
  - ***cmath*** module: cmath.pi
- decimal.Decimal from standard library: decimal.Decimal(x)
  - decima.Decimal("54321.01234567898765432 1")
  - Precision controllable decimal value
- **Methods** for float & complex types

```
In [119]: float.
float.as_integer_ratio  float.conjugate  float.fromhex  float.hex  float.imag  float.is_integer  float.mro  float.real
```

```
In [119]: complex.
complex.conjugate  complex.imag  complex.mro  complex.real
```

Data Type: strings

- Definition: **immutable** sequences of Unicode characters
- Valid string
  - Can use “” or ‘’ to specify: x=“abc” x=‘abc’
  - x = “matt’sin” ←→ x = ‘matt”sin’ (same thing)
  - Use triple double-quotes(“””) for multi-line strings or strings that contain both ‘ and “ inside of them: “””ab’cd’’ef”””
- Operators and functions
  - ***<, <=, ==, !=, >, >=***
  - **Slicing** :
    - ***str[index]***
    - ***str[start:end]***
    - ***str[start:end:step]***
  - ***str.format():***
    - ***“The {who} was {0} last week.”.format(12, who=“boy”)***
  - ***Methods:***

```
In [8]: str = "abc'def"

In [9]: str.
str.capitalize  str.endswith    str.index      str.isidentifier  str.istitle  str.lstrip     str.rindex     str.split       str.title
str.casefold    str.expandtabs  str.isalnum    str.islower       str.isupper  str.maketrans  str.rjust      str.splitlines  str.translate
str.center      str.find        str.isalpha    str.isnumeric     str.join     str.partition  str.rpartition str.startswith  str.upper
str.count       str.format      str.isdecimal  str.isprintable   str.ljust    str.replace    str.rsplit     str.strip       str.zfill
str.encode      str.format_map  str.isdigit    str.isspace       str.lower    str.rfind      str.rstrip     str.swapcase
```

Data Type: tuple

- **Definition: immutable** sequences of arbitrary Python objects
- **Creation**

```
>>> tuple_sample = (1, 2, 3)
>>> edward = ['Edward Gumby', 42]
>>> tuple_sample = tuple(edward)
>>> tuple_sample
('Edward Gumby', 42, 'Male')
```

- **Methods**

```
In [160]: tuple.
tuple.count  tuple.index  tuple.mro
```

- **Indexing & Slicing**

```
>>> x = 1, 2, 3
>>> x[1]
2
>>> x[0:2]
(1, 2)
```

Data Type: list

- **Definition: mutable** sequences of arbitrary Python objects
- **Creation**

```
>>> edward = ['Edward Gumby', 42]
>>> strings = list(“abc”)
>>> strings
['a', 'b', 'c’]
>>> number = list(range(10))
>>> number
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **Methods**

```
In [119]: list.
list.append  list.clear  list.copy  list.count  list.extend  list.index  list.insert  list.mro  list.pop  list.remove  list.reverse  list.sort
```

```
>>> edward.append(‘Male’)
>>> Edward
['Edward Gumby', 42, 'Male’]
>>> edward.index(42)
1
```

Data Type: list – indexing & Slicing

- **Indexing**: starts from 0 and supports negative number.

```
>>> greeting = 'Hello'
>>> greeting[0]
'H'
>>> greeting[-1]
'o'
```

- **Slicing**

```
>>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers[3 : 6: 1]   # [start : end : interval] , [start, start + interval , … end)
[3, 4, 5]
```

Data Type: list – slicing

- **Slicing**

```
>>> numbers[7:]
[7, 8, 9]
>>> numbers[:3]
[0, 1, 2]
>>> numbers[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers[-3:-1]
[7, 8]
>>> numbers[-1:-3]
[]
>>> numbers[-1:-3:-1]
[9, 8]
>>> numbers[: :2]
[0, 2, 4, 6, 8]
```

Data Type: list – assignment operator

- **Assignment**

```
>>> numbers = [1, 5, 6]
>>> numbers[0] = 0
>>> numbers
[0, 5, 6]
>>> numbers[1:] = [1, 2, 3]
>>> numbers
[0, 1, 2, 3]
>>> numbers[1:1] = [4, 5, 6]
>>> numbers
[0, 4, 5, 6, 1, 2, 3]
>>> numbers[1:4] = []
>>> numbers
[0, 1, 2, 3]
```

Data Type: list – special operators

- **+**

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> ‘Hello ‘ + ‘world’
‘Hello world’     # two string
>>> [1, 2, 3] + ‘Hello’
Error       # list and string
```

- *****

```
>>> ‘Hi’ * 3
‘HiHiHi’
>>> [32] * 5
[32, 32, 32, 32, 32]
>>> [None] * 5
[None, None, None, None, None]
```

- Example

```
>>>endings = ['st', 'nd', 'rd'] + 17 * ['th'] + ['st', 'nd', 'rd'] + 7 * ['th'] + ['st']
```

Data Type: list – keywords

- **del**

```
>>> names = [‘Tom’, ‘Jack’, ‘Rose’]
>>> del names[1]
>>> names
[‘Tom’, ‘Rose’]
```

- **in**

```
>>> strings = [‘abc’, ‘def’]
>>> ‘abc’ in strings
True
>>> ‘a’ in strings
False
>>> permissions = ‘rwx’
>>> ‘w’ in permissions
True
>>> ‘a’ in permissions
False
>>> ‘rw’ in permissions
True
>>> ‘rx’ in permissions
False
```

Data Type: list – copy and sort a list

- **Wrong way**

```
>>> y = x
>>> y.sort()
>>> x
[1, 2, 3]
>>> y
[1, 2, 3]
```

- **Right ways:**

```
>>> y = x[:]
>>> y.sort()
>>> x
[2, 3, 1]
>>> y
[1, 2, 3]
```

or

```
>>> y = sorted(x)
>>> x
[2, 3, 1]
>>> y
[1, 2, 3]
```

Data Type: dictionary

- **Definition**: **mutable** finite sets of Python objects indexed by arbitrary index sets

- **Creation**

```
>>>phonebook = {‘Alice’:’2341’, ‘Beth’:’9102’}

>>> items = [(‘Alice’, ’2341’), (‘Beth’, ’9102’)]
>>> phonebook = dict(items)
>>> phonebook
{‘Alice’:’2341’, ‘Beth’:’9102’}
```

- **Methods**:

```
In [201]: dict.
dict.clear   dict.fromkeys  dict.items  dict.mro  dict.popitem     dict.update
dict.copy    dict.get       dict.keys   dict.pop  dict.setdefault  dict.values
```

Data Type: dictionary

- **Basic operations:**
  - len(d): returns the number of key-value pairs.
  - d[key]: returns the value corresponds to key
  - d[key] = value: assign value to key in d.
  - **del** d[key]: delete item key in d.
  - key **in** d: check whether key is in d.

- **Advantages**
  - More types of key other than index, compared to sequences data types (string, tuple, list)
  - Much more flexible even using index as key

```
>>> x = []
>>> x[20] = ‘Hello’
Error
>>> y = {}
>>> y[20] = ‘Hello’
```

Shallow and deep copy operations

- **Copy one dictionary and make some changes**

```
>>> x = {'user_name': 'admin', 'lst': ['a', 'b', 'c']}
>>> y = x.copy()
>>> y['user_name'] = 'guest'
>>> y['lst'][0] = 'd'
>>> y
{'user_name': 'guest', 'lst': ['d', 'b', 'c']}
>>> x
{'user_name': 'admin', 'lst': ['d', 'b', 'c']}          # SURPRISE! x is also changed…
```

- **Right way**

```
>>> import copy
>>> y = copy.deepcopy(x)
>>> y['user_name'] = 'guest'
>>> y['lst'][0] = 'd'
>>> y
{'user_name': 'guest', 'lst': ['d', 'b', 'c']}
>>> x
{'user_name': 'admin', 'lst': ['a', 'b', 'c']}
```

Control flow: conditional statement

- **The followings will be considered as false in python:**
  False None 0 “” () [] {}

```
>>> True==1
True
>>> False==0
True
>>> True + False + 42
43
>>> bool(42)
True
>>> bool(‘’)
False
```

- **keywords: if, else, elif**

```
num = input(‘Enter a number: ‘)
if num > 0:
    print ‘The number is positive’
elif num < 0:
    print ‘The number is negative’
else:
    print ‘The number is zero’
```

Control flow: conditional statement

- Logical operations
  ==, <, >, <=, >=, !=, is, is not, in, not in.

  if 3 < num < 7:

  ```
  >>> x = y = [1, 2, 3]
  >>> z = [1, 2, 3]
  >>> x == y
  Ture
  >>> x == z
  True
  >>> x is y
  True
  >>> x is z
  False
  ```

- **assert(condition)**
  if not condition:
  crash program

  ```
  >>>assert(age >= 0)
  ```

Control flow: loop statement

- **while**

```
x = 1
while x <= 100:
    print(x)
    x+=1
```

- **for … in …**
  - **List iteration**

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in numbers:
    print(number)
```

  - **Dictionary iteration**

```
d = {'x':1, 'y':2, 'z':3}
for key   in d:
    print key, 'corresponds to', d[key]

for key, value in d.items():
    print key, 'corresponds to', d[key]
```

Control flow: loop statement

- **for … in …**
  - **Parallel iteration**

```
names = ['anne', 'beth', 'damon']
ages = [12, 30, 45]
>>> zip(names, ages)
[('anne', 12), ('beth', 30), ('damon', 45)]
for name, age in zip(names, ages):
    print(name, age)
```

  - **Index iteration**

```
names = ['anne', 'beth', 'damon']
index = 0
for name in names:
    print(index, name)
    index += 1

for index, number in enumerate(numbers):
    print (index, number)
```

Control flow: loop statement

- **break, continue**

```
x = 3
while x < 10:
    if x > 7:
        x += 2
        continue
    x = x +1
    print("still in the loop.")
    if x == 8:
        break
print("outside of the loop.")
```

Control flow: special loop statement

- **List traversal**

```
>>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x*x for x in numbers]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]


>>> [x*x for x in numbers if x%3==0]
[0, 9, 36, 81]


>>> [(x, y) for x in numbers[0:2] for y in numbers[0:2]]
[(0, 0), (0, 1), (1, 0), (1, 1)]

>>> girls = ['alice', 'bernice']
>>> boys = ['bob', 'arnold']
>>> [b + '+' + g for b in boys for g in girls if b[0] == g[0]]
['bob+bernice', 'arnold+alice']
```

Function: definition

- **def creates a function and assign it a name**
- **return sends a result back to the caller**
- **Arguments and return types are not declared**

```
def square(x):
    return x*x

def printHi():
    print('Hi')
    return
    print("Not printed")
```

```
>>> x = printHi()
Hi
>>> x
>>>
>>> print(x)
>>> None
```

Function: arguments passing

- **Arguments are passed by assignment**
- Passed arguments are assigned to local names
- Assignment to argument names don’t affect the caller
- Changing a mutable argument may affect the caller

```
def changer(name, age, organizations):
    name = name.title()
    age += 1
    organizations.append('SIO’)
    print(name, age, organizations)
    return
```

```
>>> name = 'wenming wu’
>>> age = 30
>>> orgs = ['ASML', ‘DJEL’]
>>> changer(name, age, orgs)
 Wenming Wu 31 ['ASML', ‘DJEL', 'SIO’]

>>> name
 'wenming wu’
>>> age
30
>>> orgs
 ['ASML', ‘DJEL', 'SIO']
```

Function: passing arguments by keyword

Two ways to assign argument values to function parameters: by position or by keyword

```
def hello_1(greeting, name):
    print(greeting, name)

def hello_2(name, greeting):
    print(greeting, name)
```

- **By position**

```
>>> hello_1('Hello', 'Tom')
Hello Tom
>>>hello_2('Tom', 'Hi')
Hi Tom
```

- **by keyword**

```
>>>hello_1(greeting='Hello', name='Tom')
Hello Tom
>>>hello_2(greeting='Hi', name='Tom')
Hi Tom
```

Function: optional arguments with default values

- **Define default values for arguments that need not be passed**

```
def hello_3(greeting='Hello', name = 'Tom'):
    print(greeting, name)
```

```
>>> hello_3()
Hello Tom
>>> hello_3('Hi')
Hi Tom
>>> hello_3(name='Jack')
Hello Jack
```

Function: passing multiple arguments with * and ** symbols

- ***params represents a tuple receiving any excess positional arguments**

```
def print_params(first, *therest):
        print(first, therest)

>>> print_params(1)
1 ()
>>> print_params(1, 2, ‘name’)
 1 (2, 'name’)
```

- ****params represents a dictionary receiving any excess keyword arguments**

```
def print_params_2(first, **therest):
        print(first, therest)

>>> print_params_2(1, name='Jack', age=21)
 1 {'name': 'Jack', 'age': 21}
```

Function: recursion

- **Recursion allows functions to call themselves**
- **Recursive functions solve problems by reducing them to smaller sub-problems of the same form**

```
def factorial(x):
    ret = 1
    if x == 0:
        return ret

    a = 1
    while a <=x:
        ret *= a
        a += 1

    return ret

>>>factorial(0)
1
>>>factorial(10)
3628800
```

```
def factorial(x):
    if x == 0:
        return 1

    return x * factorial(x - 1)


>>>factorial(0)
1
>>>factorial(10)
3628800
```

Function: others

- **All functions in Python have a return value**
  - Functions without a return statement will return the special value None
- **There is no function overloading in Python**
  - Two different functions can’t have the same name, even if they have different arguments
- **Functions can be used as any other data type. They can be:**
  - arguments to function
  - return values of functions
  - assigned to variables
  - parts of tuples, list, etc.

Python basics: reference & exercise

- References:
  - https://www.python.org/
  - Swaroop C H, “A Byte of Python”
  - Eric Matthes, “Python Crash Course: A Hands-On, Project-Based Introduction to Programming”
  - https://www.python.org/dev/peps/pep-0008/
  - https://www.cnblogs.com/vamei/archive/2012/09/13/2682778.html
  - https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000
  - https://www.fullstackpython.com/table-of-contents.html
  - http://www.pythontutor.com/
- Exercises 1
  - Write a python function to sort any list of integers. (You could not use any built-in sort functions.)

    >>>sort([9, 5, 18, -2, 0])

    [-2, 0, 5, 9, 18]