Machine Learning

Tutorial 1: Python

Dr Patrick Chan **Mr. Xavier Xie**

School of Computer Science and Engineering

South China University of Technology

About this lab

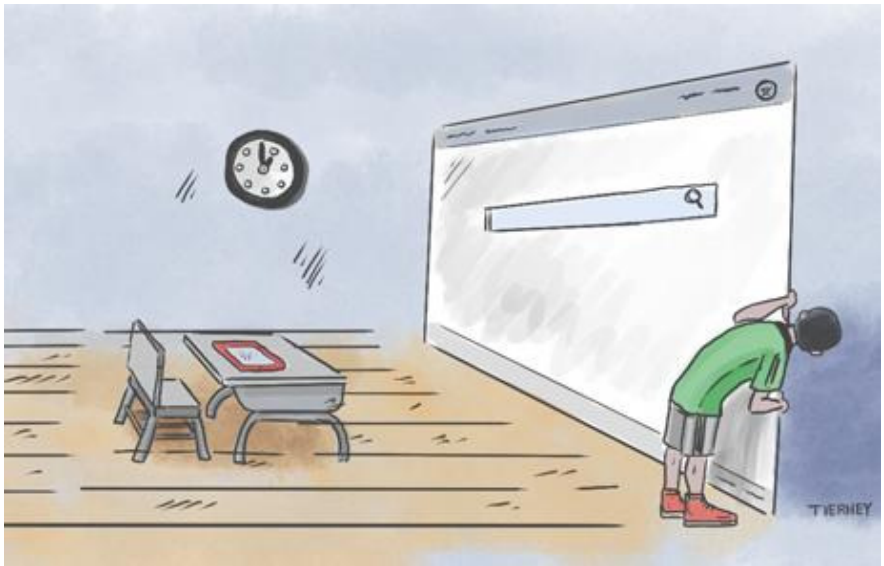
- Python
  - Anaconda
  - Jupyter
- Python Programming
  - Basic Syntax
  - Libraries

Useful Links

- Python: https://docs.python.org/3.7/
- Anaconda: https://www.anaconda.com/
- Jupyter Notebook: https://jupyter.org/
- NumPy: https://www.numpy.org/
- Pandas: https://pandas.pydata.org/
- Matplotlib: https://matplotlib.org/

Find the solution by yourself!

- Google:
  http://www.google.com/



What is Python?

- Developed by Guido Van Rossum in the late 1980s
- A simple programming language
  - More readable than others, e.g. C/C++/Java
  - Very easy to use
  - Many libraries for specific requirements
    - NumPy, Pandas, are Matplotlib will be discussed

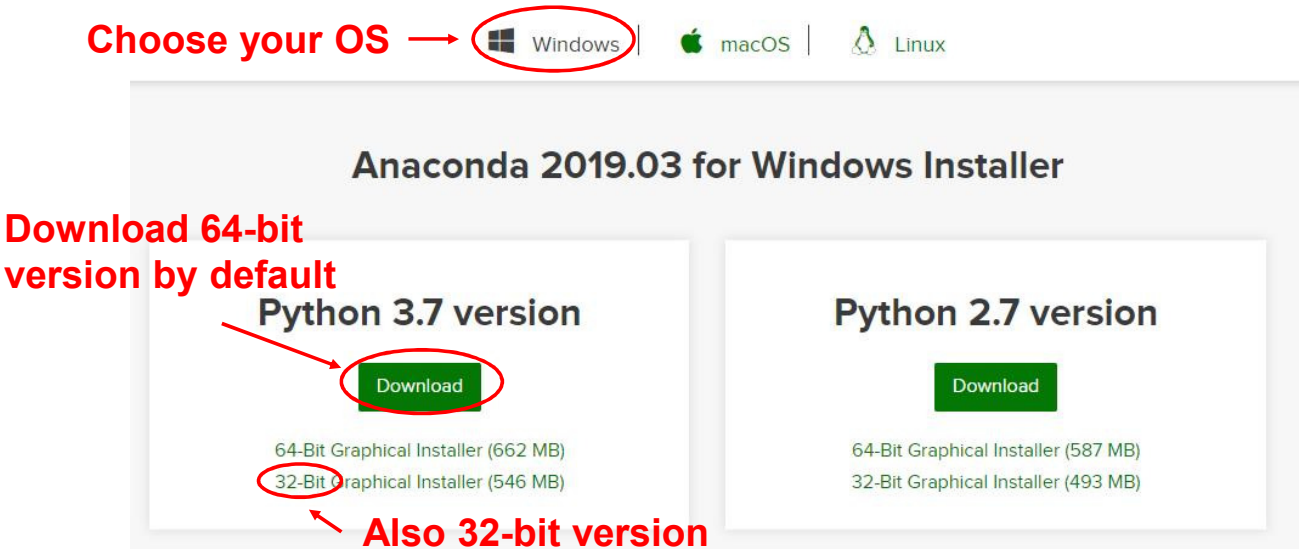
How to start with Python?

- **Anaconda**

  A powerful **library manager** for Python
  - Easy to install and manage libraries
  - Contains more than 1500 packages for data science
  - Provide user-friendly IDE: **Jupyter Notebook**
    - Easy to run
    - Easy to debug

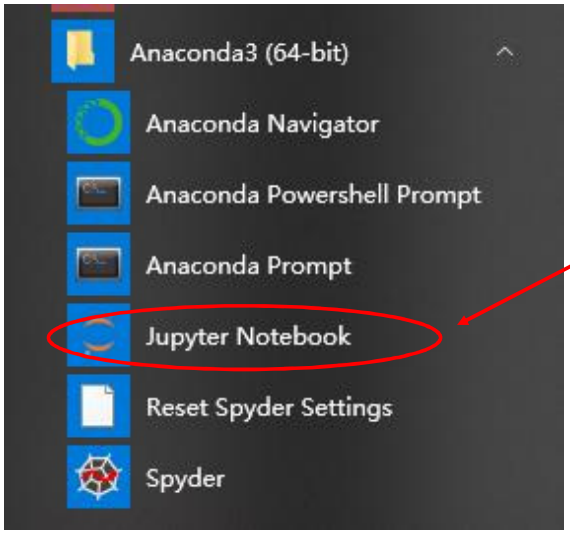
Anaconda Installation

- Download: https://www.anaconda.com/distribution/



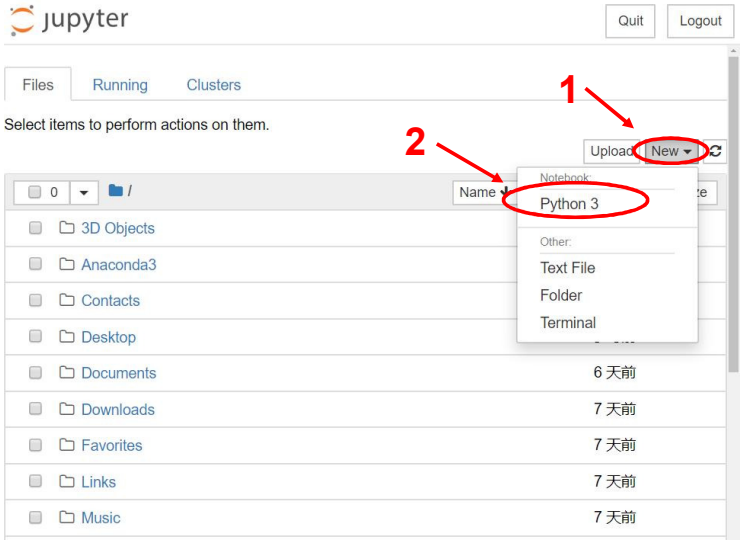
Python: How to start?

- Start “Jupyter Notebook”
  - Click “Jupyter Notebook” in “Anaconda3” folder



Python: How to start?

- Create a new program
  - Click “New” on right top menu
  - Select “Python 3”

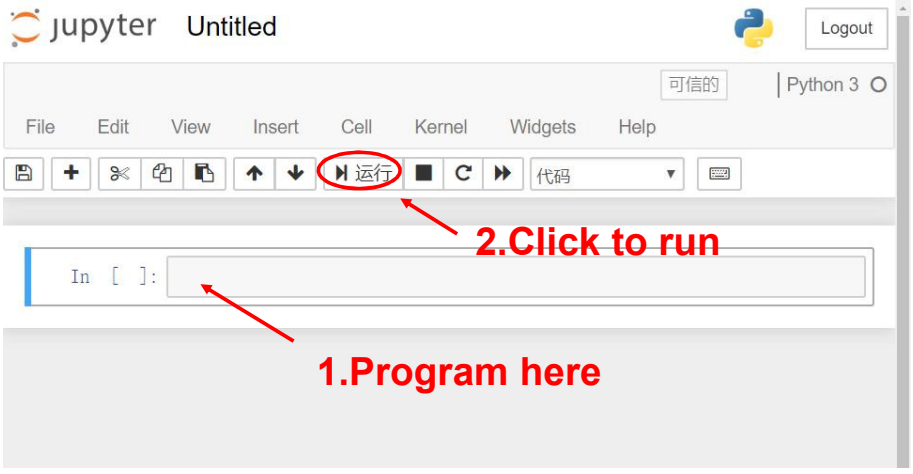


Python: How to start?

- Type the following code in code cell

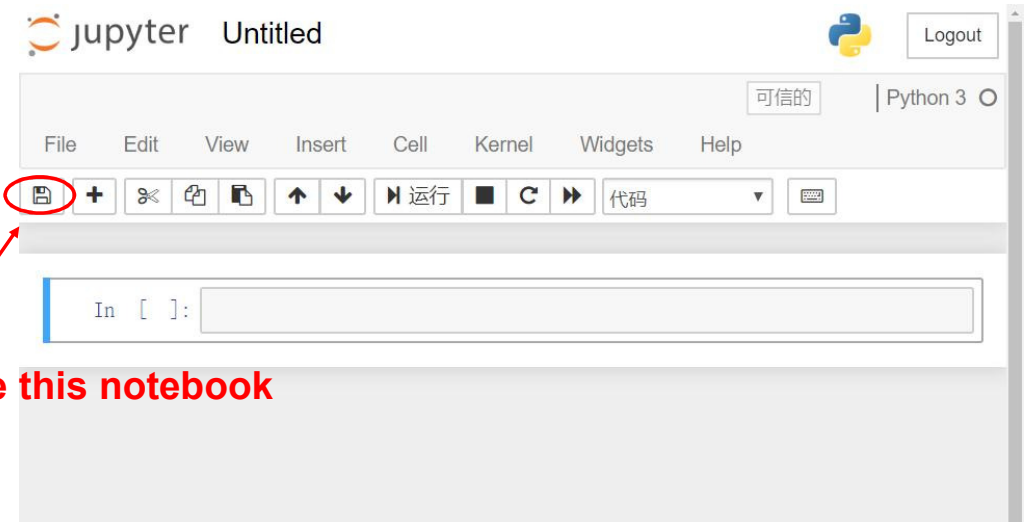
```
a = 'Hello world'
a
```

- Click “Run”
- See what happens



Python: How to start?

- Save a file



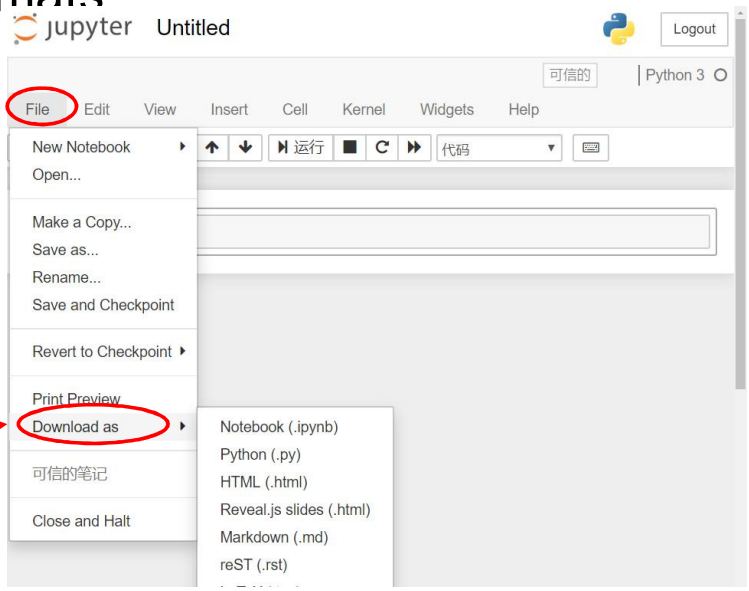
Click to save this notebook

Python: How to start?

- Save a file in other formats

Click to save your code to different file types

Select the file type



Python: Basic

Print to screen

- `print()`
- *`variable`*

```
a = 1
print(a)
a
```

```
1

1
```

Comment

- `# TEST`

```
a = 1  # hello
```

Python: Variable

- Case sensitive (e.g. a is not equal to A)
- Cannot start with a number
- No special characters (e.g. @ # $ %)
- No reserved word or function name (e.g. max, True)
- Valid character: a-z, A-Z, 0-9, _ , Chinese

Python: Variable

- Example

```
variable = "variable 1"
print(variable)
_variable = "variable 2"
print(_variable)
variable3 = "variable 3"
print(variable3)
变量4 = "variable 4"
print(变量4)
```

```
variable 1
variable 2
variable 3
variable 4
```

Supported,
but not recommended

Python: Variable Type

- Boolean: `bool`
- Integer: `int`
- Real: `float`
- String: `str`
  - Single quote ' '
  - Double quote " "

```
a = 100
b = 100.123
c = "hello"
d = False
```

- No declaration is needed!

Python: Variable Type

Check the type of a variable

- `Type()`

```
a = 100
b = 100.123
c = "hello"
d = False
print("a is a:", type(a))
print("b is a:", type(b))
print("c is a:", type(c))
print("d is a:", type(d))
```

```
a is a: <class 'int'>
b is a: <class 'float'>
c is a: <class 'str'>
d is a: <class 'bool'>
```


Python: Variable Type

- A variable type can be changed

```
a = 10
print("a:",type(a))
a = 10.5
print("a:", type(a))
a = 'hello'
print("a:", type(a))
```

```
a: <class 'int'>
a: <class 'float'>
a: <class 'str'>
```

Python: Type Conversion

- `bool()` : change to Boolean
- `int()` : change to integer
- `float()` : change to real
- `str()` : change to string

- How about change a char to a real number?

```
print("str(1): ", str(1))
print("float(1): ", float(1))
print("bool(1)", bool(1))
```

```
str(1):  1
float(1):  1.0
bool(1) True
```


Python: Print

- Print a value

```
print("str(1): ", str(1))
print("float(1): ", float(1))
print("bool(1)", bool(1))
```

```
str(1):  1
float(1):  1.0
bool(1) True
```

Python: Print format

Print	Operator	format()
int	%d	{}
str	%s	{}
float/double	%f	{}
precision-specified float/double	%.2f	{:.2f}

Python: Print format

Using operator

```
print("%d" % 10)
print("%s" % "string")
print("%f" % 1.23456)
print("%.2f" % 1.23456)
```

```
10
string
1.234560
1.23
```

Using format()

```
print("{}".format(10))
print("{}".format("string"))
print("{}".format(1.23456))
print("{:.2f}".format(1.23456))
```

```
10
string
1.23456
1.23
```

Python: Print format

Multi values

- %

```
"%s, %d, %f" % ("hello", 10, 1.23456)
```

- format()

```
"{}, {}, {}".format("hello", 10, 1.23456)
```


Try It!

- Given string: "Int: 10, Float: 123.4567, String: hello, world"
  - Print the string using %
  - Print the string using format()

Python: ";"

- ";" is not necessary generally
- Only be useful when more than 1 statements in 1 line

```
a = 10
b = 20
```

```
c = 30; d = 40; e = 50
```


Python: Arithmetic operators

Operator	Description	Example
+	Add: a + b	a + b = 9 + 2 = 11
-	Minus: a - b	a – b = 9 – 2 = 7
*	Multiply: a x b	a * b = 9 x 2 = 18
**	Power: a^b	a ** b = 9^2 = 81
/	Devided: a / b	a / b = 9 / 2 = 4.5
//	Returns the integer portion of the quotient: a / b	a // b = ⌊9 / 2⌋ = 4
%	Complementation: a % b	a % b = 9 % 2 = 1

- The parenthesis （ ） can be used
  - e.g. (a + b) * c

Try It!

```
4 * 5;
4 * 5
4 ** 5

4 / 2
4 // 2

(2 + 2) * 3
2 + (2 * 3)
2 + 2 * 3
```

What is the difference?

Given a=10, b=20, c="15"

```
a == b
a > b
a < b
a <> b
a == c
a >= c
```

Are a and c comparable?

Python: Data Type

`List`	mutable	ordered	[ ]
`Tuple`	immutable	ordered	( )
`Set`	mutable	unordered	{ }
`Frozenset`	immutable	unordered	

- Mutable : can be modified after creation

Python: List

Create List

- `[ ]` `myList = [1,2,3,4]`
- `list([ ])` `myList = list([1,2,3,4])`

Can store heterogeneous data

- myList = [1, “hello”, 2.34, [5, 6, 7]]

Python: List

- Index starts from 0
- a = [0, 1, 2, 3, 4, 5]

Indexing

- `mylist[2]` 2

Python: List

- Index starts from 0
- a = [0, 1, 2, 3, 4, 5]

Slicing

- `myList[:]` [0, 1, 2, 3, 4, 5]
- `myList[2:]` [2, 3, 4, 5]
- `myList[:2]` [0, 1]
- `myList[2:4]` [2, 3, 4]
- `myList[2:-1]` [2, 3, 4]

Python: List

- `index(object)`
- `insert(position, object)`
- `append(object)`
- `extend([object, object, …])`
- `remove(object)`
- `count(object)`
- `sort()`
- `sort(reverse=True)`

Python: List

- ls.sort(key=None, reverse=False)

```
ls = [1, 6, -3, 4, 2, -5]
ls.sort()
ls
```

```
[-5, -3, 1, 2, 4, 6]
```

by default, the list will be sorted according to the original value

```
def cal_abs(a):
    return a * -1 if a < 0 else a

ls = [1, 6, -3, 4, 2, -5]
ls.sort(key=cal_abs)
ls
```

```
[1, 2, -3, 4, -5, 6]
```

given key, the list will be sorted according to the value calculated by key

Try It!

- Given list [1, “3”, 6, -5, “-4”, 2]
  - Sort by the original value
  - Sort after converting the original value to int
  - Sort after converting the original value to int and square it

Python: Tuple

Create Tuple

- `( )` `myTuple = (1,2,3,4)`
- `tuple(( ))` `myTuple = tuple((1,2,3,4))`

Can store heterogeneous data

- `myTuple = (1, "hello", 2.34, [5, 6, 7])`

Python: Tuple

- `count(object)`
- `index(object)`

- `len(tuple)`

Python: Tuple

Mutable item in tuple can be modified

- `a = ('1', 1)`
- `a[1] = 2` ERROR

- `a = ('1', [1])`
- `a[1][0] = 2` OK

Python: Set

Create Set

- `{ }` `mySet = {1, 2, 3, 4, 5}`
- `set({ })` `mySet = set({1,2,3,4,5})`
- `set([ ])` `mySet = set([1,2,3,4,5])`

Can store heterogeneous data

- `s = {1, "hello", 2.34, [5, 6, 7]}`

Python: Set

- Don’t support indexing or slicing as no order

A set contains unique items

- `mySet = {1, 2, 2, 3, 3, 3}`
- mySet is equal to {1, 2, 3}

Python: Set

Union (∪)	`A|B`	items in A or B
Intersection (∩)	`A&B`	items in both A and B
Difference (–)	`A-B`	items in A but not in B
Symmetric Differences (⊕)	`A^B`	items in either A or B, but not both

Python: Set

- Subset ($\subseteq$) `A<=B` all items in A are in B
- Proper Subset ($\subset$) `A<B` all items in A are in B, some items in B are not in A
- Superset ($\supseteq$) `A>=B` all items in B are in A
- Proper Superset ($\supset$) `A>B` all items in B are in A, some items in B are not in A

Python: Set

- `add(object)`
- `remove(object)`

- How to check if a element is in a set???

Using operator: in

```
s = {"a", "b"}
"c" in s
```

```
False
```

Python: Frozenset

Create Frozenset

- `forzenset([ ])`

```
mySet = frozenset([1,2,3,4,5])
```

- Frozenset is as the same as Set but immutable

Try It!

- Given the list [1, 2, 2, 3, 3, 4, 5, 5]
  - Remove duplicate items from the list

Python: Dictionary

- Contains **key-value** pairs

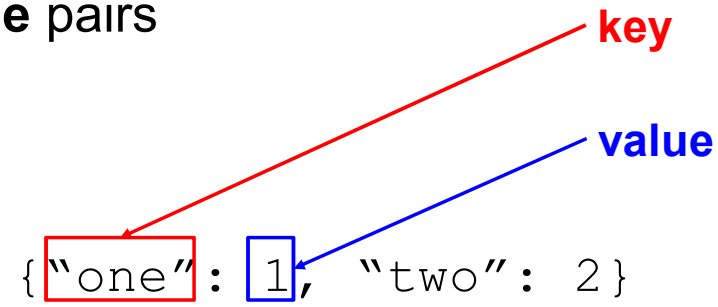
Create a dictionary:

- { : , : , ..}

```
myDict = {"one": 1, "two": 2}
```

- dict({ : , : , …})

```
myDict = dict({"one": 1, "two": 2})
```



Python: Dictionary

Dictionary's value and key can both be heterogeneous

- s = {"one": 1, 2: "two"}

- Dictionary's key must be immutable

Python: Dictionary

- check if an identifier is in the dictionary
- `key in Dict` `"one" in "myDict"`

Python: Dictionary

- `keys()`
- `values()`
- `items()`
- `has_key(key)`
- `get(key)`
- `del()`
- `remove()`

Try It!

- Given dict1 {"one": 1, "two": 2},dict2 {1: “one”, 2: “two”}
  - Remove the value 1 from dict1
  - Add one key-value pair: “three”: 3 to dict1
  - Add all key-value pairs in dict2 to dict1
  - Remove the value with key of 3 from dict2

What will happen since dict2 doesn’t have key 3, how to avoid this

Python: Common Functions

- `max(val1, val2, …)`
- `min(val1, val2, …)`
- `sum(val1, val2, …)`

Try It!

- Given the list [2, 4, 11, 6, 5, -4, 14, 9]
  - Get the maximum of the list
  - Get the minimum of the list
  - Get the sum of the list

Python: Conditional Statement (If Then Else)

- `if condition :`
  `elif condition :`
  `else:`

- Indentation is used
- No endif or { }

```
a = 15
if a > 20:
    print("a is bigger than ")
    print("20. ")
elif a > 10:
    print("a is bigger than ")
    print("10. ")
else:
    print("a is small. ")
print("FINISH!")
```

```
a is bigger than
10.
FINISH!
```

Python: Pass

An empty statement

- Pass
- To keep the program structure intact
- If no operation is provided after indentation, a pass statement is used

```
if 100 < 10:
    pass
else:
    print("100 > 10")
```

```
100 > 10
```

```
if 100 < 10:
else:
    print("100 > 10")
```

```
  File "<ipython-input-38-832472c816d5>", line 2
    else:
    ^
IndentationError: expected an indented block
```

Python: Comparison Operators

Operator	Description	Example a=9,b=2
==	if a is equal to b	a == b: False
!=	if a is not equal to b	a != b: Ture
<>	if a is not equal to b	a <> b: True
>	if a is bigger than b	a > b: True
<	if a is smaller than b	a < b: False
>=	if a is not smaller than b	a >= b: True
<=	if a is not bigger than b	a <= b: False

Python: Looping (For Loop)

For Loop

- `for iterating_var in sequence:`

  `    statements`

- Iterating variable types:
  - str list tuple set dictionary **range**

Python: Looping (For Loop)

Create range

- `range(int1, int2[, step])`

Can store heterogeneous data

- `r = range(10)` 0, 1, 2, …, 10
- `r = range(1, 10)` 1, 2, …, 10
- `r = range(1, 10, 2)` 1, 3, 5, 7, 9
- `r = range(3, 1, -1)` 3, 2, 1

Python: Looping (For Loop)

- range() can be used to create a list

```
ls = [i for i in range(10)]
(type(ls))
```

```
list
```

Generator expression: a recommended style to create a list

```
ls
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


Python: Looping (For Loop)

- Str

```
string = "hello"
for s in string:
    print(s)
```

```
h
e
l
l
o
```

- Tuple

```
tp = (1, 2, 3, 4)
for i in tp:
    print(i)
```

```
1
2
3
4
```

- List

```
ls = [1, 2, 3, 4]
for i in ls:
    print(i)
```

```
1
2
3
4
```

- Range

```
r = range(1, 5)
for i in r:
    print(i)
```

```
1
2
3
4
```

Python: Looping (For Loop)

- Dictionary

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}
for key, value in d.items():
    print(key, ":", value)
```

```
one : 1
two : 2
three : 3
four : 4
```


Python: Looping (While Loop)

While Loop

- `while condition:`

  `statements`

- Iterating variable types:
  - str list tuple set range

Python: Looping (While Loop)

- Str

```
string = "hello"
index = 0
while index < len(string):
    print(string[index])
    index += 1
```

```
h
e
l
l
o
```

- Tuple

```
tp = (1, 2, 3, 4)
index = 0
while index < len(tp):
    print(tp[index])
    index += 1
```

```
1
2
3
4
```

- List

```
ls = [1, 2, 3, 4]
index = 0
while index < len(ls):
    print(ls[index])
    index += 1
```

```
1
2
3
4
```

- Range

```
r = range(1, 5)
index = 0
while index < len(r):
    print(r[index])
    index += 1
```

```
1
2
3
4
```


Try It!

- Given the dictionary {"one": 1, "two": 2, “three”: 3}
  - Visit all key-value pair in the dictionary

Try It!

- Visiting all key-value pairs in dictionary

```
d = {"one": 1, "two": 2}
for k, v in zip(d.keys(), d.values()):
    print(k, v)
```

1

```
one 1
two 2
```

```
for k, v in d.items():
    print(k, v)
```

2

```
one 1
two 2
```

```
for k in d.keys():
    print(k, d[k])
```

3

```
one 1
two 2
```

```
for k, v in enumerate(d):
    print(k, v)
```

4

```
0 one
1 two
```


Python: Looping

- `enumerate()`

```
iteration = [1, 2, 3]
for index, value in enumerate(iteration):
    print(index, value)
```

```
0 1
1 2
2 3
```

- `zip()`

```
iteration1 = [1, 2, 3]
iteration2 = ["one", "two", "three"]
for iter1, iter2 in zip(iteration1, iteration2):
    print(iter1, iter2)
```

```
1 one
2 two
3 three
```

Python: Looping

- continue

```
iteration = [1, 2, 3, 4]
for i in iteration:
    if i is 2:
        continue
    print(i)
```

Only skip this time

```
1
3
4
```

- break

```
iteration = [1, 2, 3, 4]
for i in iteration:
    if i is 2:
        break
    print(i)
```

End loop

```
1
```


Python: String

- Single quote (' ')
- Double quote (" ")
- Triple quote (''' ''' or """ """) for multi-line string
- Indexing and Slicing

```
str1 = 'Hello, Python!'
str2 = "Hello, Python!"
str3 = """Hello, World!
and hello, Python!"""

print(str1)
print(str2)
print(str3)
```

```
Hello, Python!
Hello, Python!
Hello, World!
and hello, Python!
```

Python: String

Escape characters, starts with

- `\n` newline
- `\r` carriage return
- `\t` horizontal tabs
- `\'` single quote

```
print("aaa\rbbb")
```

```
bbb
```

```
print("aaa\nbbb")
```

```
aaa
bbb
```


Python: String

- `+` concatenate
- `in` check substring
- `r` or `R` ignore escape character (\)
- `%` format

```
string = "hello," + "world!"
print("string:", string)
print("\"hello\" in \"hello,world!\"", "hello" in string)
```

```
string: hello,world!
"hello" in "hello,world!" True
```

```
string1 = "hello,\nworld!"
string2 = r"hello,\nworld!"
string3 = R"hello,\nworld!"
print("string1:", string1)
print("string2:", string3)
print("string3:", string3)
```

```
string1: hello,
world!
string2: hello,\nworld!
string3: hello,\nworld!
```

```
string4 = "%s,world!" % "hello"
print("string4:", string4)
```

```
string4: hello,world!
```

Python: String

- `find(str [, start] [, end])`
- `replace(findStr, replaceStr [, m])`
- `split([str])`
- `capitalize()`
- `lower()`
- `upper()`
- `startswith(str, beg=0, end=len(string))`
- `endswith(str, beg=0, end=len(string))`

Try It!

- Given string “hello, python and world!”
  - Visit the item at position 3 of the string
  - Get the last 4 item in the string
  - Replace “o” with “k”
  - Find the index of “n”
  - Find the index of “and”

The string has more than 1 “n”, what will the index be?

Python: Function

- Create Function
- def *functionName* [(*Inputpara*, ..)]:

  ["*Comments*"]

  Statement

  [return expression]

Python: Function

- Example

```
def func_a(a):
    "This function prints parameter a, no return value"
    print("in func_a, a:", a)

a = 100
func_a(a)
```

```
in func_a, a: 100
```

Python: Function

- Example

```
def func_b(a, b):
    "This function calculates the sum of a and b"
    s = a + b
    return s

a = 100
b = 10
c = func_b(a, b)
c
```

```
110
```


Python: Function

- Example

```
def func_c(a, b):
    "This function calculates the sum and the multi of a and b"
    s = a + b
    m = a * b
    return s, m

a = 100
b = 10
c, d = func_c(a, b)
print(c, d)
```

Return multi variables

Received as 2 variables

```
110 1000
```

```
e = func_c(a, b)
e
```

Received as 1 tuple variable

```
(110, 1000)
```

Python: Class

Create Class

- `class classname ([baseClass]):`

```
    ["Comments to describe the class"]
    [def __init__(self[, varName1, ..])]
    def functionName1(self[, varName1, ..])
```

The pointer to itself

Python: Class

```
class Class1():

    def __init__(self, a):
        self.b = a

    def func1(self, b):
        print("in class Class1, function func1, self.b", self.b)
        print("in class Class1, function func1, b", b)

class1 = Class1(10)
class1.func1(20)
```

Global (Public) variable

Local (private) variable

```
in class Class1, function func1, self.b 10
in class Class1, function func1, b 20
```

Python: Class

```
class BaseClass():
    def func_a(self):
        print("func_a is a function defined in BaseClass")

class DerivedClass1(BaseClass):
    pass

class DerivedClass2(BaseClass):
    pass

derived_class1 = DerivedClass1()
derived_class1.func_a()
derived_class2 = DerivedClass2()
derived_class2.func_a()
```

2 derived classes

```
func_a is a function defined in BaseClass
func_a is a function defined in BaseClass
```

2 derived classes both inherit from BaseClass, both have func_a()

Try It!

- Define a function to calculate the sum of a list, the prototype of the function is: def calc_sum(ls)
- Define 3 classes, each class should have at least 1 function(not initial function), then define a class that inherits from these 3 classes

Python: Library

Import library

- `import libName [as aliasName]`
- `from libName import libName [as aliasName]`

```
import numpy
import numpy as np  # use np as alias, easier to use
from numpy import array  # only import array in numpy
```

- Python is famous and fancied for its rich and powerful libraries

Python: Library

- Libraries are discussed
  - `pickle` Save environment (variables, object..) as a file
  - `os` Read the files from a hard disk
  - `NumPy` Matrix operation, mean, variance
  - `Pandas` Data analysis, data preprocessing
  - `Matplotlib` Plot a graph

Python: Data Persistence

```
import pickle
```

Save objects as a file

- `f = open(path, "wb")`
- `pickle.dumps(object, f)`

w	to write
r	to read
a	to append
b	binary data

object will be stored to file

Load data from disk

- `f = open(path, "rb")`
- `pickle.loads(f)`

Python: Data Persistence

- Save an object

```
import pickle as pkl

a = {"one": 1}
f = open("data.pkl", "wb")
pkl.dump(a, f)
f.close()
```

- Load an object

```
f = open("data.pkl", "rb")
c = pkl.load(f)
f.close()
c
```

```
{'one': 1}
```

Python: Data Persistence

- Save multi objects

```
import pickle as pkl

a = {"one": 1}
b = {"two": 2}
f = open("data.pkl", "wb")
pkl.dump(a, f, pkl.HIGHEST_PROTOCOL)
pkl.dump(b, f, pkl.HIGHEST_PROTOCOL)
f.close()
```

- Load multi objects

```
f = open("data.pkl", "rb")
c = pkl.load(f)
d = pkl.load(f)
f.close()
print(c)
print(d)
```

```
{'one': 1}
{'two': 2}
```

Specify protocol to support multi objects

Python: File

```
import os
```

Read folders and files

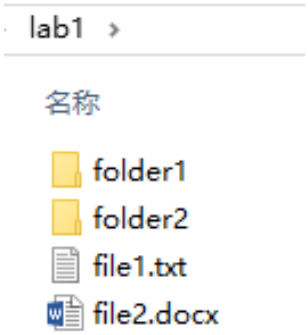
- `os.listdir(path)`
  - return: a list containing all folders and files in the given path

Check if a path is existed(folder or file)

- `os.path.exists(path)`

Python: File

Assume, the folder "lab1" contains:



- Get the paths of all folders and files

```
import os

fs = os.listdir("lab1")
fs
```

```
['file1.txt', 'file2.docx', 'folder1', 'folder2']
```

- Check if folder or file exists

```
os.path.exists("lab1/folder3")
```

```
False
```

```
os.path.exists("lab1/file3.txt")
```

```
False
```

NumPy: Import

- An array contains the same type of items
- Support matrix operation

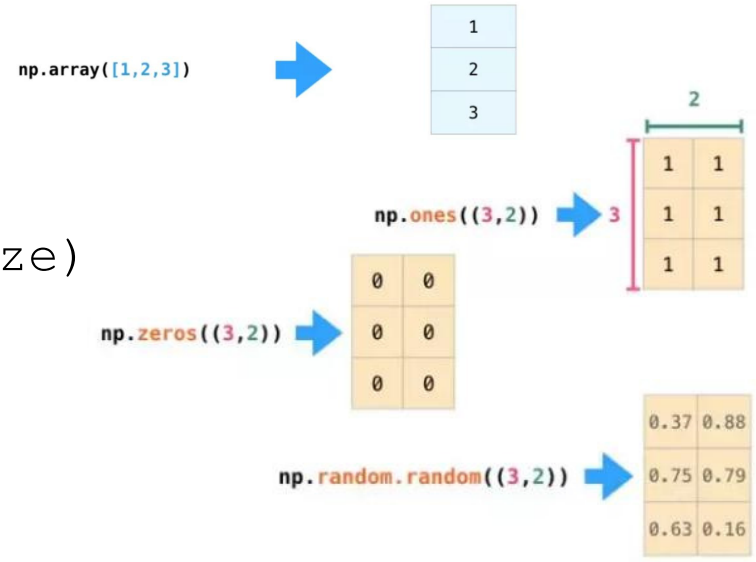
Import NumPy

- `import NumPy as np`
- np is commonly set as alias

NumPy: n-dimensional array

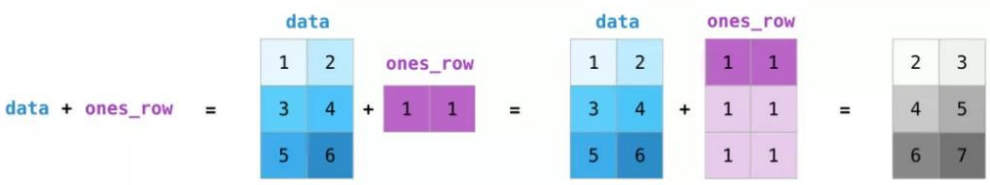
Create an array

- `np.array(list)`
- `np.ones(size)`
- `np.zeros(size)`
- `np.random.random(size)`



NumPy: Arithmetic Operations

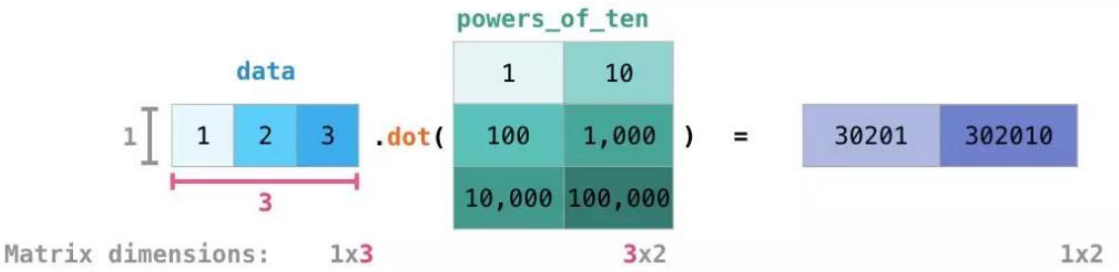
- array and scalar + - * /
- array and array + - * /
  - at least one dimension is same



NumPy: Matrix operation

Dot Product

- np.dot(*array*)

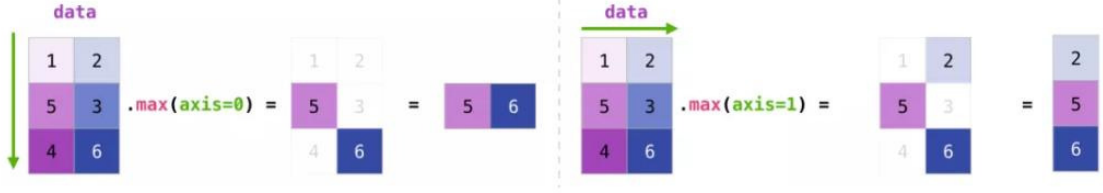


NumPy: Summary Function

- np.max()
- np.min()
- np.mean()
- np.var()
- np.std()
- *myNumPy*.median(*array*)

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print("max:", arr.max())
print("min:", arr.min())
print("mean:", arr.mean())
print("median:", np.median(arr))
print("variance:", arr.var())
print("standard deviation:", arr.std())
```

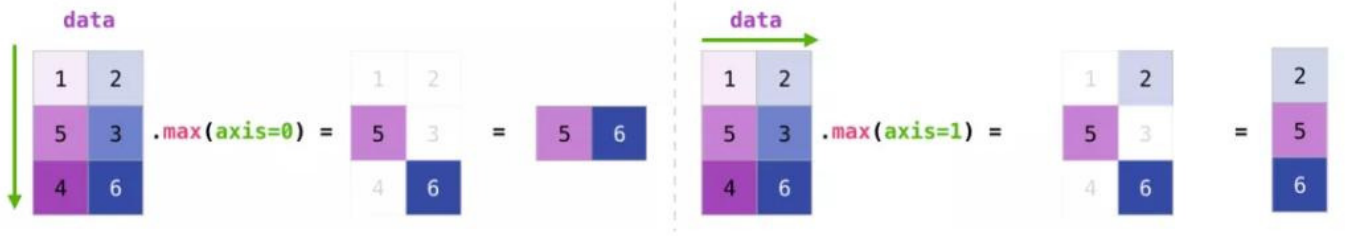
```
max: 8
min: 1
mean: 4.5
median: 4.5
variance: 5.25
standard deviation: 2.29128784747792
```



NumPy: Summary Function

Dimension Specification

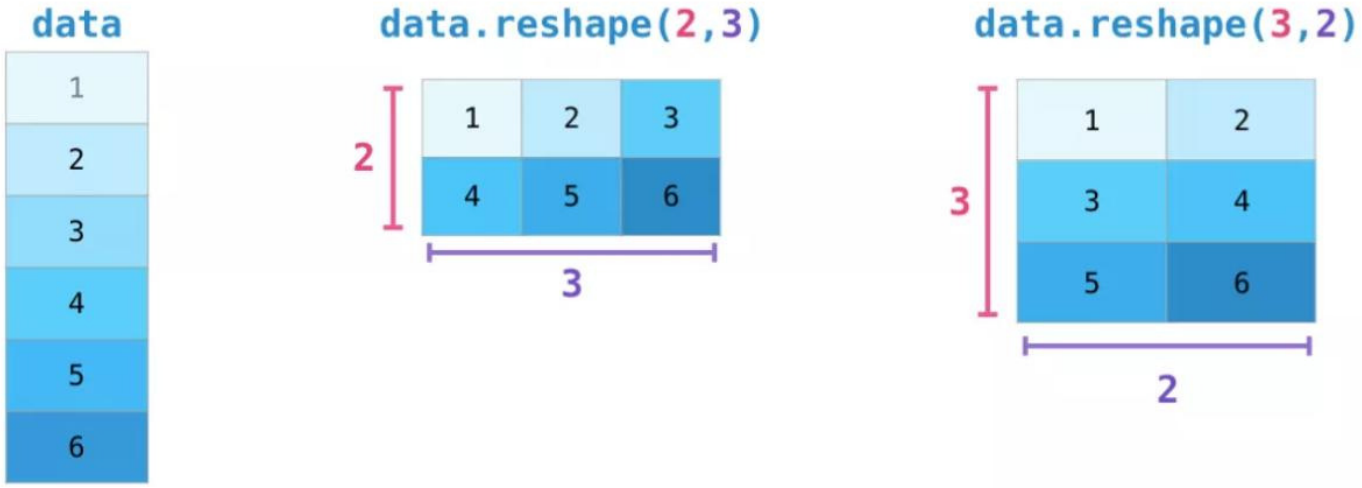
- `axis`
- E.g. `max(axis=1)`



NumPy: Summary Function

Change the architecture of the array

- `np.reshape()`



Try It!

- Given arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
- Calculate arr's max, min, mean, median, variance, standard deviation
- What if we specify the axis? Try different axis and observe

NumPy: Indexing

Get a column

- `np.array[:, columnNumber]`

Get a row

- `np.array[rowNumber]`

Get an item

- `np.array[dim1,dim2,…]`
- `np.array[(dim1,dim2,…)]`

NumPy: Indexing

```
a = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
a
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

- Column 1

```
a[:, 1]
```

```
array([2, 5, 8])
```

- Row 1

```
a[1]
```

```
array([4, 5, 6])
```

- Item at [1, 1]

```
a[1, 1]
```

```
5
```

```
a[(1, 1)]
```

```
5
```


NumPy: Slicing

Get columns columnNumber1 through columnNumber2

- `np.array[:, [colNum1]:[colNum2]]`

Get rows rowNumber1 through rowNumber2

- `np.array[[rowNum1]:[rowNum2]]`

NumPy: Slicing

```
a = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
a
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

- Columns 0 through 1

```
a[:, :2]
```

```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

- Rows 0 through 1

```
a[:2]
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```


NumPy: Array Persistence

Save Array

- `save(file, array)`
  - save array to “npy” file

Load Array

- `load(file)`
  - load array from “npy” file

Pandas outlines

- A brief introduction to Pandas
- Data structure
  - Series (1D data)
  - DataFrame (2D data)
  - Pannel (3D data)
- Data analysis

Pandas: Import

- Support non-numerical data
- Index for each dimension data
- Series (1D array)
- DataFrame (2D array)

Import Pandas

- `import pandas as pd`

Pandas: Series

- Series are usually used to represent a record

Create Series

- `pd.Series()`
- `pd.Series(datatype)`
  - `Datatype`: list, tuple, dictionary and np.array

Pandas: Series

```
ls = ["one", "two", "three", "four"]
s = pd.Series(ls)  # list converts to Series
print(type(s))
print(s)
```

```
<class 'pandas.core.series.Series'>
0      one
1      two
2    three
3     four
dtype: object
```

Values

default indexes

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}
s = pd.Series(d)  # dictionary converts to Series
print(type(s))
print(s)
```

```
<class 'pandas.core.series.Series'>
one      1
two      2
three    3
four     4
dtype: int64
```

Keys

Pandas: DataFrame

- DataFrame are usually used to represent data of a table

Create DataFrame

- `pd.DataFrame()`
- `pd.DataFrame(datatype)`
  - *datatype*: list, tuple, dictionary and np.array
  - Dictionary is the most suitable datatype

Pandas: DataFrame

- Create DataFrame from dictionary

Column index

```
data = {'name': ['AA','IBM','GOOG'],
        'date': ['2001-12-01','2012-02-10','2010-04-09'],
        'shares': [100, 30, 90],
        'price': [12.3, 10.3, 32.2]}
df = pd.DataFrame(data)
print(type(df))
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>
   name        date  shares  price
0    AA  2001-12-01     100   12.3
1   IBM  2012-02-10      30   10.3
2  GOOG  2010-04-09      90   32.2
```

default row index

Pandas: DataFrame

- Create DataFrame from dictionary

Column index

specified row index

```
data = {'name': ['AA','IBM','GOOG'],
        'date': ['2001-12-01','2012-02-10','2010-04-09'],
        'shares': [100, 30, 90],
        'price': [12.3, 10.3, 32.2]}
df = pd.DataFrame(data, index=["one", "two", "three"])
print(type(df))
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>
       name        date  shares  price
one      AA  2001-12-01     100   12.3
two     IBM  2012-02-10      30   10.3
three  GOOG  2010-04-09      90   32.2
```


Pandas: DataFrame

- Create DataFrame from List

```
data = [
    [1, 2, 3, 4, 5],
    ["one", "two", "three", "four", "five"]
]
df = pd.DataFrame(data)
print(type(df))
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>
      0    1      2      3      4
0     1    2      3      4      5
1   one  two  three   four   five
```

Pandas: DataFrame

Convert to numpy.array

- `dataframe.values`

```
ls = [
    [1, 2],
    [3,4]
]
df = pd.DataFrame(ls)
df
```

	0	1
0	1	2
1	3	4

```
df.values
```

```
array([[1, 2],
       [3, 4]], dtype=int64)
```


Pandas: DataFrame: Access Data

Access by **column index**

- `dataframe[columnName]`

Access by **row index**

- `dataframe.iloc[rowNumber]`

```
df["name"]
```

```
0      AA
1     IBM
2    GOOG
Name: name, dtype: object
```

```
df.iloc[0]
```

```
name              AA
date      2001-12-01
shares           100
price           12.3
Name: 0, dtype: object
```

Pandas: View

Show **first rows**

- `head(rowNumber)`

Show **last rows**

- `tail(rowNumber)`

```
data.head(3)
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0

```
data.tail(3)
```

	id	name	gender	age
7	8	Nat	female	29.0
8	9	Anna	female	NaN
9	10	Jack	male	31.0

Pandas: Data Description

Display statistic on data

- `describe()`
- Only numerical data

In this case, column “name” and “gender” are not numerical data, so are not included.

```
data.describe()
```

	id	age
count	10.00000	8.000000
mean	5.50000	31.250000
std	3.02765	8.154753
min	1.00000	19.000000
25%	3.25000	27.000000
50%	5.50000	32.500000
75%	7.75000	35.750000
max	10.00000	43.000000

Pandas: Filter

Select the rows that meets the conditions

- `dataframe[conditions]`

- How to filter two or more columns?

```
elder_than_20 = data[data["age"] > 20]
elder_than_20
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
9	10	Jack	male	31.0

Pandas: Filter

Select the rows that meets the conditions

- `dataframe[conditions]`

```
elder_than_20_male = data[(data["age"] > 20) & (data["gender"] == "female")]
elder_than_20_male
```

	id	name	gender	age
1	2	Nancy	female	34
5	8	Nat	female	29

Pandas: Data Analysis

Sort data by the value of given column

- `dataframe.sort_values(columnName)`

```
sort_female = female.sort_values("age")
sort_female
```

	id	name	gender	age
7	8	Nat	female	29.0
1	2	Nancy	female	34.0
4	5	Jully	female	NaN
8	9	Anna	female	NaN

Pandas: Data Analysis

Show if every item is null

- `dataframe.isnull()`

```
data.isnull()
```

	id	name	gender	age
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	True
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	True
9	False	False	False	False

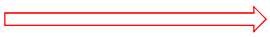
Pandas: Data Analysis

Replace null data

- `dataframe.fillna(value)`

```
data
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
3	4	Jason	male	19.0
4	5	Jully	female	NaN
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
8	9	Anna	female	NaN
9	10	Jack	male	31.0



```
data = data.fillna(0)
data
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
3	4	Jason	male	19.0
4	5	Jully	female	0.0
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
8	9	Anna	female	0.0
9	10	Jack	male	31.0

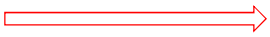
Pandas: Data Analysis

Replace null data at specified column(s)

- `dataframe.fillna({columnName:value})`

```
data
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
3	4	Jason	male	19.0
4	5	Jully	female	NaN
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
8	9	Anna	female	NaN
9	10	Jack	male	31.0



```
data = data.fillna({"age":0})
data
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
3	4	Jason	male	19.0
4	5	Jully	female	0.0
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
8	9	Anna	female	0.0
9	10	Jack	male	31.0

Pandas: Data Persistence

Load data from **csv** file

- `pd.read_csv(file)`

Load data from **excel** file

- `pd.read_excel(file)`

```
data = pd.read_csv('lab_pandas.csv')
data
```

	id	name	gender	age
0	1	Kobe	male	21.0
1	2	Nancy	female	34.0
2	3	John	male	43.0
3	4	Jason	male	19.0
4	5	Jully	female	NaN
5	6	Andy	male	35.0
6	7	Leon	male	38.0
7	8	Nat	female	29.0
8	9	Anna	female	NaN
9	10	Jack	male	31.0

Pandas: Data Persistence

Save data to **csv** file

- `dataframe.to_csv(file)`

Save data to **excel** file

- `dataframe.to_excel(file)`

Pandas: Exchange with NumPy

NumPy to Pandas

- `pd.Series(array)`
  - `array`: must be 1-dimensional

```
a = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
df = pd.DataFrame(a)
df
```

	0	1	2
0	1	2	3
1	4	5	6

- `pd.DataFrame(array)`
  - `array`: must be 2-dimensional

```
a = np.array([1, 2, 3])
s = pd.Series(a)
s
```

```
0    1
1    2
2    3
dtype: int32
```

Pandas: Exchange with NumPy

Pandas to NumPy

- `pd.values`

```
d = {
    "one": [1, 2, 3],
    "two": [4, 5, 6]
}

dddd = pd.DataFrame(d)
dddd.values
```

```
array([[1, 4],
       [2, 5],
       [3, 6]], dtype=int64)
```

- `pd.to_numpy()`

```
aaaa = dddd.to_numpy()
aaaa
```

```
array([[1, 4],
       [2, 5],
       [3, 6]], dtype=int64)
```

Matplotlib

- Plot module of MatLab

Matplotlib: Functions

Create Figure

- `figure(num[, figureSize])`
  - num: optional, index of the created figure, 0 by default
  - figsize: optional, size of the figure

Draw a curve

- `plot(x, y[, label])`
  - x, y provide the data and must have same number of row
  - label: legend

Matplotlib: Functions

Specify the **title** of the figure

- `title(title)`

Specify the **title** of y axis

- `ylabel(label)`

Matplotlib: Functions

Specify the title of x axis

- `xlabel(label)`

Show the legends on the figure

- `legend()`
  - by default, the legends are not showed

Matplotlib: Functions

Show the figure

- `show(label)`

Save the figure

- `savefig(fname)`

Matplotlib: Draw Figure

- The standard procedure to draw a figure using Matplotlib

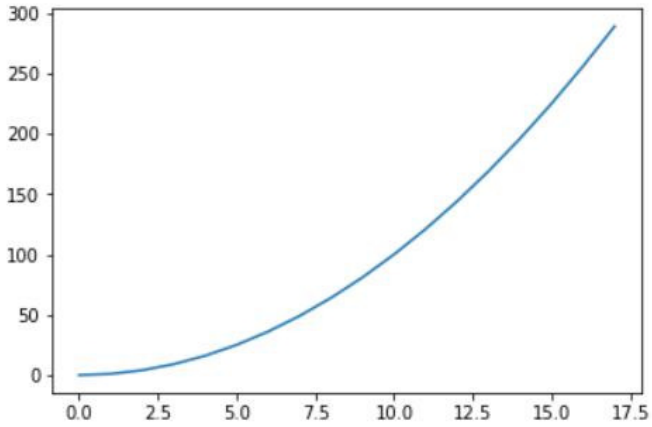
```
import matplotlib.pyplot as plt

plt.figure()

x = [i for i in range(18)]
y = [i * i for i in x]

plt.plot(x, y, label="square")
plt.savefig("lab1_matplotlib.png")
plt.show()
```

1. create a new figure
2. prepare data
3. plot the data
4. [optional] save the figure
5. show the figure

Matplotlib: Draw Figure

- An example

```
# 1. create a new figure
plt.figure()
# 2. prepare some data to plot
x = [i for i in range(18)]
y = [i * i for i in x]
# 3. plot the data
plt.plot(x, y, label="square")
# 4. [optional] save the figure
plt.savefig("lab1_matplotlib.png")
# 5. show the figure
plt.show()
```

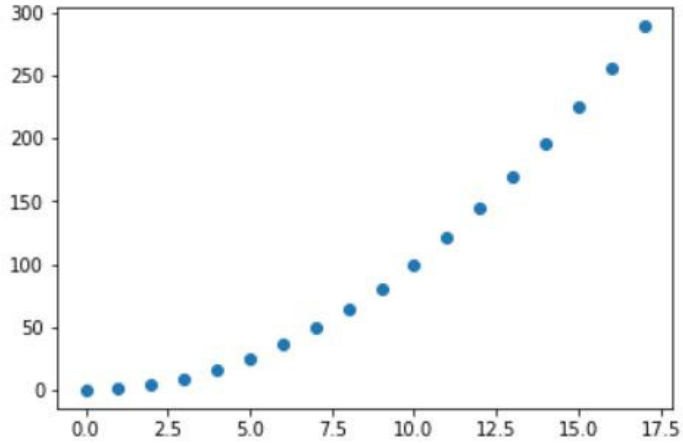


Matplotlib: More Functions

Draw scatter figure

- `scatter(x, y[, c])`
  - c: to specify the color, blue by default
  - Some commonly used colors
    - b blue
    - y yellow
    - g green
    - k black
    - r red

```
plt.figure()
x = [i for i in range(18)]
y = [i * i for i in x]
plt.scatter(x, y)
plt.show()
```



Matplotlib: More Functions

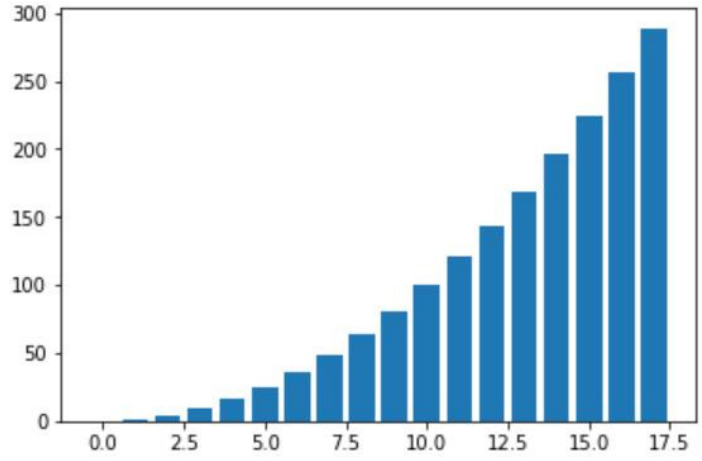
Draw a bar chart

- `bar(left, height, width=0.8, bottom=None)`
  - left, height, width and bottom specify the bar **position** and **size**
  - left stands for **the center of the x axis of the bar**
  - (left-width / 2, bottom) is the bottom left corner
  - (left + width / 2, bottom + height) is the upper right corner

Matplotlib: More Functions

- Bar chart example

```
plt.figure()
x = [i for i in range(18)]
y = [i * i for i in x]
plt.bar(x, y)
plt.show()
```



Matplotlib: More Functions

Import image lib from matplotlib

- `import matplotlib.image as mpimg`

Read an image

- `mpimg.imread(file)`
- return a numpy.array typed object of the image

Matplotlib: More Functions

Draw a picture

- `imshow(array)`
  - array: numpy.array typed data, contains each value of the pixels of the picture

Matplotlib: More Functions

Draw multi sub-figures in one figure

- `subplot(numRows, numCols[, sharex, sharey])`
  - numRows and numColumns: divide the entire figure into numRows rows and numCols columns
  - sharex: if True, share the x axis, False by default
  - sharey: if True, share the y axis, False by default
  - E.g. plt.subplot(2, 2)

2 rows, 2 columns

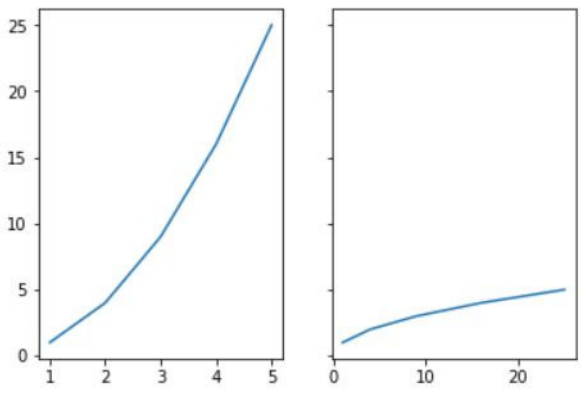
(1, 1)	**(1, 2)**
(2, 1)	**(2, 2)**

Matplotlib: More Functions

- Subplot example

```
import matplotlib.pyplot as plt

plt.figure()

x = [i for i in range(1, 6)]
y = [i * i for i in x]

f, (a1, a2) = plt.subplots(1, 2, sharey=True)
a1.plot(x, y)
a2.plot(y, x)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Try It!

- Draw and show the sine function (x from -2π to 2π)