Comp2211

※python fundamentals:

- Python: dynamically type multiparadigm programming language.
- import sys
  print(sys.version) ⇒ check the version.
- Data Type:
  - basic type: int, float, bool (use and, or, not.), string (str)
  - container: lists, dictionaries, set, tuples. → these are built-in container types.

  → actually they are classes.
  → len() works for all container, string.

  (list) [10, "Hello", 20.5]
  (dict) {"key1": "value1", "key2": "value2"}
  (set) {"a", "b"}
  (tup) ("a", "b")

  type(<obj>) ⇒ check type.

- Arithmetic operators: +, -, *, /, //, %, **, (), +=, -=, *=, /=.

  (python don't have increment, decrement, i.e. x++, ++x)

- String:
  - 用法: hello = 'hello' world = "world"
    len(hello) → 长度
    hw = hello + ' ' + world

    hw12 = '%s %s %d' % (hello, world, 12)
    type(hw12)
    print(hello * 3)

  - add a prefix in front of strings:

prefix	Meaning	
u	unicode (default)	u"abc"
b	Byte (ASCII) string.	b"abc"
r	Raw string (原始string)	r"\nabc\n"
f	Formatted string	a=1, b=2, print(f'a+b={a+b}')

→ directly stored on disk, machine readable. (need decoding, encoding)

eg: unicode_str = u'Comp'
byte_str = b'Comp'
decode_str = byte_str.decode('utf-8')
encode_str = unicode_str.encode('utf-8')

- strings are object in Python.

- String objects:
  - s.capitalize() → 首字母变大写.
  - s.upper() → 所有字母变大写.
  - s.rjust(7) → 将string里的字符向右对齐.
  - s.center(7) → 向中间对齐.
  - s.replace("l","ell") → 将所有"l"换为"ell".
  - s.strip() → 将前后空格去除.
- type conversion:
  - int('3.14')+3 ⇒ 6
- print: out data to the standard output device(screen)
  - print(*<obj>, sep=' ', end='\n', <file>=sys.stdout, flush=False)
    - sep: ";"中间插入
    - end: 以…结尾
    - 在哪个地方print
    - 确保能够多output在print()调用时.
  - str.format():
    x='Bt'; y='A'
    print('abcd{}efg{}'.format(x,y))
    → {1}.{0} : index.
- input:
  eg: age = input("Enter your age:")

※ Containers:

Data Structure	Ordered?	Duplicate?	Indexing/ Slicing?	Changeable/ Mutable?	Constructor	Example
List	Yes	Yes	Yes	Yes	[ ] or list()	[5.7, 4, 'yes', 5.7]
Dictionary	No	No	Yes	Yes	{ } or dict()	{'Jun':75, 'Jul':89}
Set	No	No	No	Yes	{ } or set()	{5.7, 4, 'yes'}
Tuple	Yes	Yes	Yes	No	( ) or tuple()	(5.7, 4, 'yes', 5.7)

① List: resizeable, contain different types.

- index: 0, 1, ..., n
  -(n+1), ..., -1
- list.append(<item>)
- list_1 += list_2
- list_1.pop(<index>) —— remove and return the given index value from the list. (default is the last one)
- list_1.remove(<element>) —— remove the specified value from list. (只除去一个)

· Slicing :

index after the last element.

list [<start> : <end (exclusive)> : <jump>]

default: 0 # of elements 1

eg: num[2:4] = [8,9]

· list_1 = list(range(5)) >>> [0, 1, 2, 3, 4]

· loop : for element in list:
print(element)

· list comparison :

new_list = [x+2 for element in list if <condition>]

· join two list:

list1 = [1,2,3] , list2 = [4,5,6]

I. + operator: list_new = list1+list2 ⇒ [1,2,3,4,5,6]

II. list.extend(): list_new = list1.extend(list2)

III. unpacking: list_new = [*list_1, *list_2]

IV. itertools.chain():

list_new = list(itertools.chain(list1, list2))

② dictionary:

- dict_1 = {key_1 : value_1, ...} —— accessed by its key name, using [key]
- dict_1[new_key] = new_value
- dict_1.get(key, value) —— return value of a specific key,
  value (optional) default value: None. —— None ≡ "N/A"
- dict_1.pop(key, <default_value>) —— remove the specified key-value, return the value

  error: 1. parameter is not specified.
  2. key is not found.
- if access an element not exist ⇒ error
- del d['fish'] —— delete the element in dictionary.

- for loop:

  ① for key in dict:
  ...

  ② for key, value in d.items():

- dictionary comprehension:

```
nums = [0, 1, 2, 3, 4]
dict_1 = {x: x**2 for x in nums if x % 2 == 0}
dict_1
>>> {0: 0, 2: 4, 4: 16}
```

③ Set: unordered collection of distinct elements. (i.e. no duplicates)

set_1 = {value_1, value_1, ...} ← ignore duplicates.

- unchangeable, unindexed, but we can add, remove items.
- set_1[0] ⇒ error

  we can convert set to list: list_1 = list(set) → order is not guaranteed.
- value_1 in set_1 —— to check whether the item is in the set.
- len(set_1) —— get length.
- add(value_1), remove(value_2)
- no pop —— since it's unordered.
- enumerate:

```
for index, value in enumerate(set_1):
    ...
for value in set_1:
    ...
```

- Set comprehension:

```
set_1 = {int(sqrt(x)) for x in range(30)}
>>> {0, 1, 2, 3, 4, 5}
```

④ Tuple: unchangeable list (ordered)

tuple = (value_1, value_2, ...)

- access elements with [ ] → only readable.
- we can use tuple as index in dictionary (but list can't):

```
dict_1 = {(x, x+1): x for x in range(5)}
>>> {(0,1):0, (1,2):1, (2,3):2, (3,4):3, (4,5):4}
```

※ Parallel Iteration:

```
list_1 = [name_1, ... ]
list_2 = [property_1, ...]
for name, property in zip(list_1, list_2):
    ...
```

(can be used for more than 2 lists)

※ Function:

```
def func(x):
    ...
```

- variable inside a function is local, only accessible inside function scope.
- global var ——— define a global variable, can be accessed everywhere.
- when we don't know how many arguments.

① add an * before a parameter name:

in this way, the function will receive a tuple of arguments.

```
eg: def func(*kids):
        print(kids[2])
    func("a", "b", "c")        >>> c
```

② add an ** before a parameter name:

- default values → use "parameter_1 = 1" for readability.

※ Class:

```
class Person(object):
    def __init__(self, name='Tom', ...):
        self.__name = name
        :
    def get_name(self):
        ...
```

must; default value.

- Public. Private, Protected:

Public → default in Python, accessible anywhere

Private → only in class

Protected → accessible for class and sub-classes.

by convention: private: adding a prefix __
protected: adding a prefix _
(no protection by python but just remind programmer not to modify them)

※ Module, Package, Library

- module: a python file.
- package: folder of python files
- library: collection of packages.

※ Import.

① from [module] import [function / value]
(module → .py file.)
(* → all)

② import sklearn.cluster —— import all the modules inside package sklearn.cluster
import sklearn.cluster.KMeans —— import the module KMeans only.
→ call things in KMeans: sklearn.cluster.KMeans
(to tideous so use: from sklearn.cluster import KMeans

③ from ... import ... as <nickname>.

※ Data: in .csv file

① import Data:

```
from google.colab import drive.          —— access to google drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My drive/ training_data.csv', index=False)
```

② export data:

```
output = pd.DataFrame({...})   → dict
output.to_csv('....', index=False)
                  → path.
```

* Numpy (Numerical Python) —— core library for numeric and scientific computing in Python.

↳ it provide high-performance multidimensional array object.

① numpy array: → advantage: 1. less memory
2. fase
3. convienent

- rank: # of dimensions.
- shape: a tuple of size of array along each dimensions.

eg: (2,3) — 2 rows, 3 columns.

- example:

```
a = np.array([1,2,3])
type(a)          >>> <class 'numpy.ndarray'>
a.shape          >>> (3,)
```

shape is an attribute of np.array.

(3,) — Tuple
(3) — int

a [, , , ···] —— access by [ ]

- create arrays:
  - np.zeros((2,2)) → shape —— [[0. 0.]
    [0. 0.]]
  - np.ones((1,2)) —— [[1. 1.]]
  - np.full((2,2),7) → constant array —— [[7. 7.]
    [7. 7.]]
  - np.eye(2) → 2×2 indentity matrix —— [[1. 0.]
    [0. 1.]]
  - np.random.random(2,2) —— random nums in 0~1

library, package, module.

- indexing:

a[1,1] —— one element
a[1, :] —— 1d
a[:, :] —— 2d

a[1,:] ⇔ a[1]

- b = a[1:3, 2:4] —— shalow copy. (when modifing b, a is modified)

- Integer Array Indexing:

```
a = np.array([[1,2],[3,4],[5,6]])
print(a[[0,1,2],[0,1,0]])
>>> [1 4 5]
b = a[[0,1,2],[0,1,0]]
```

—— 1 2
3 4
5 6

—— (0,0) (1,1) (2,0)

—— get back a new ndarray (deep copy)

```
a[[0,1,2],[0,1,0]] += 10
```
—— not only retrive but also modify.

- Boolean Array Indexing:

```
a = np.array([[1,2],[3,4],[5,6]])
bool_idx = (a>2)
>>> [[False False]
     [True  True]
     [True  True]]
```

```
a[bool_idx]   or a[a>2]
>>> [3 4 5 6]
```
—— only retrive those index are true.
and always get back 1D array.

② Data Type:

```
x = np.array([1,2], dtype=np.int64)
x.dtype
>>> int64
```

③ Array Math:

- x+y 、 np.add(x,y)
- x-y 、 np.subtract(x,y)
- x·y 、 np.multiply(x,y)
- x/y 、 np.divide(x,y)
- np.sqrt(x)
- x.dot(y) or np.dot(x,y)

eg: $x = \begin{bmatrix}1&2\\3&4\end{bmatrix}$ $y = [9,10]$

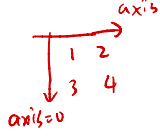
do x.dot(y), and y will be converted to $y^T = \begin{pmatrix}9\\10\end{pmatrix}$

↳ only 1-D can be converted in this way.

④ Numpy Functions:

1. sum:

```
x = np.array([[1,2],[3,4]])
np.sum(x)
np.sum(x, axis=0)
np.sum(x, axis=1)
```



—— Sum all the elements → 10
—— Sum of each column. → [4 6]
—— Sum of each row. → [3 7]

2. np.empty_like(x)
—— create an empty matrix with the same shape as x
(fill in some random values)

3. np.tile(v,(4,1))
—— stack 4 copies of v on top of each other

eg. v = [1 0 1]
⇒ [1 0 1
1 0 1
1 0 1
1 0 1]

④ Transpose: x.T

(note that: 1D array does nothing)

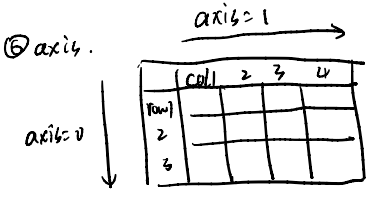
⑤ Broadcasting:

np.arange(3) + 5 ⇒ [5 6 7]

np.one((3,3)) + np.arange(3) ⇒ [[1 2 3]
[1 2 3]]

np.arange(3).reshape(3,1) + np.arrange(3) ⇒ [[0 1 2
1 2 3
2 3 4]]

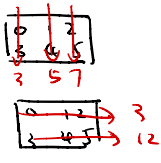
⑥ axis.



I. sum: [[0 1 2]
[3 4 5]]

np.sum(arr, axis=0) ⇒ [3, 5, 7]

np.sum(arr, axis=1) ⇒ [3, 12]



II. Concatenate:

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

b: [[9, 9, 9]
[9, 9, 9]]

np.concatenate([a, b], axis=0) ⇒ [[1 1 1]
[1 1 1]
[9 9 9]
[9 9 9]]

np.concatenate([a, b], axis=1) ⇒ [[1 1 1 9 9 9]
[1 1 1 9 9 9]]

⑥ delete:

default.

numpy.delete(arr, obj, axis=None)

I. (10, 30, 15) —delete, obj=1, axis=0→ (9, 30, 15)

axis=0 ↓ 1 ↓ 2

II. arr = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

np.delete(arr, [1,3,5], None) ⇒ [1 3 5 7 8 9 10 11 12]

output is a flattened array.

⑦ mean.

(300, 5) —mean, axis=1→ (300,)

⑧ np.split(data, k, axis=) —— 将 data 分成 k 份

✲ enumerate:

① for i, value in enumerate(list, 1):

i: index

1: optional, starting index of the counter. (here: index 1,2,3 ...)

② counter_list = list(enumerate(my_list, 1))

[(1, "a"), (2, "b") ...]

tuple in list.

✲ transpose: np_arr.T

slice 易错:

① slicing: [0:-1] 不取.

✲ plot:

x axis. y axis.

import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [10,20,30,40])

plt.ylabel('...') —— write sth on y axis

plt.xlabel('...') —— x axis

plt.show()

a = np.array([[1,2,3],[10,20,30]])

左端点 右端点

plt.plot(a)