INTRODUCTION OF PYTHON

DEEP LEARNING, 2023

OUTLINE

- **Installation**
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- Module
- Function
- Class
- Name and Reference

installation(recommended)

Data science toolkit --- Anaconda

ANACONDA.

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

1. Anaconda 3 安裝教學及說明. Python 開發環境介紹 - Anaconda 3 完整安裝說明及步驟 | by Coding Lab | AI for K-12 | Medium

2. Installing Anaconda on Windows & Add Anaconda to Path Tutorial - DataCamp

installation

Choose your best Python IDE(integrated development environment)

Visual Studio Code

Spyder

Jupyter Notebook

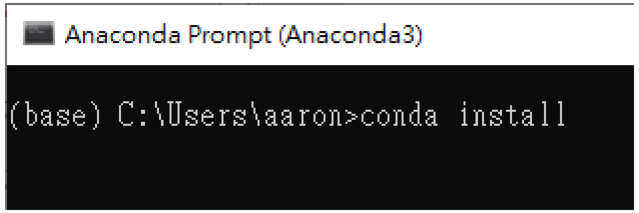
installation

Conda works on your **command line interface such as Anaconda Prompt** on Windows and terminal on macOS and Linux.

Navigator is a desktop graphical user interface that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands.

ANACONDA NAVIGATOR

desktop graphical user interface

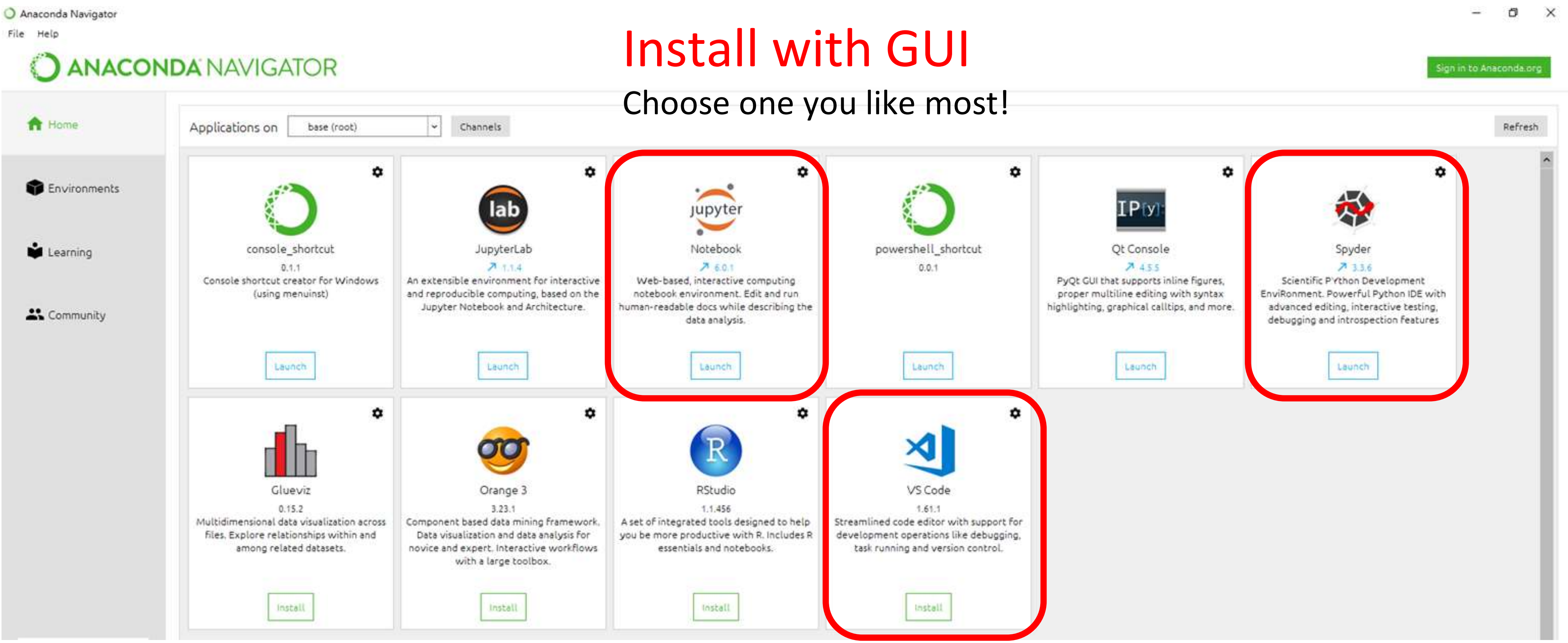


Anaconda Prompt (Anaconda3)

```
(base) C:\Users\aaron>conda install
```

command-line

Installation



Install with GUI

Choose one you like most!

console_shortcut
0.1.1
Console shortcut creator for Windows (using menuinst)

JupyterLab
↗ 1.1.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
↗ 6.0.1
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

powershell_shortcut
0.0.1

Qt Console
↗ 4.5.5
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
↗ 3.3.6
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.23.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

VS Code
1.61.1
Streamlined code editor with support for development operations like debugging, task running and version control.

Tutorial

Website:

Getting started with Anaconda — Anaconda documentation

Quickstart — Spyder 5 documentation (spyder-ide.org)

Jupyter Notebook介紹及安裝說明 - Python4U – Medium

How to Use Jupyter Notebook in 2020: A Beginner’s Tutorial (dataquest.io)

Video:

1-2. Spyder使用教學，Python編輯器最詳細比較－【行銷搬進大程式】 - YouTube

Basics of SPYDER IDE for Python Programmers - YouTube

0-2.Python Jupyter Notebook 使用教學 - YouTube

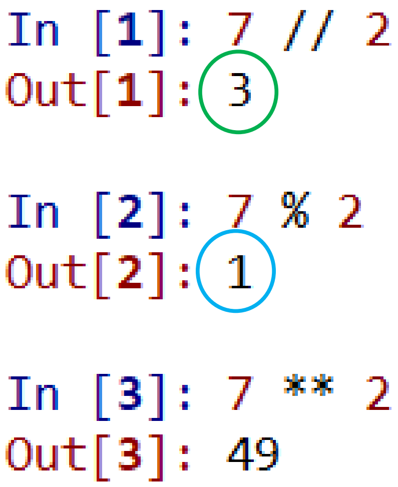
Jupyter Notebook Tutorial: Introduction, Setup, and Walkthrough - YouTube

OUTLINE

- Installation
- **Mathematical Operation**
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- Module
- Function
- Class
- Name and Reference

Mathematical Operation

- + (addition), - (subtraction), * (multiplication), / (division), () (Grouping)
- // (divide and round to integer)
- % (get the remainder)
- **(power)



```
In [1]: 7 // 2
Out[1]: 3

In [2]: 7 % 2
Out[2]: 1

In [3]: 7 ** 2
Out[3]: 49
```

Hint: 7 ÷ 2 = 3 ... 1

OUTLINE

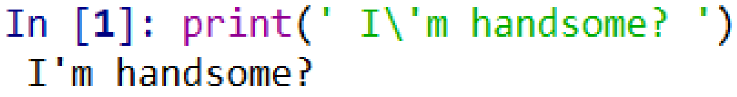
- Installation
- Mathematical Operation
- **Container**
  - **String**
  - **List**
  - **Matrix**
  - **Tuple**
  - **Dictionary**
- Python Syntax
- Module
- Function
- Class
- Name and Reference

String (1)

✓Use ‘ … ’ or “ … ” to express strings

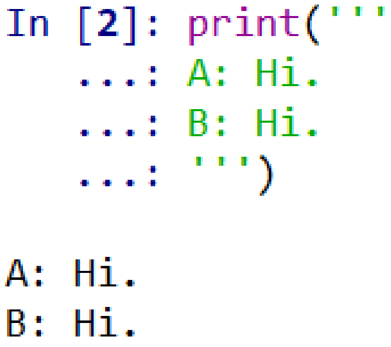
✓Use ‘ \n ’ to express newline

✓Put ‘ \ ’ in front of ‘ ’ ’ or ‘ ” ’ to make them include in the string



```
In [1]: print(' I\'m handsome? ')
 I'm handsome?
```

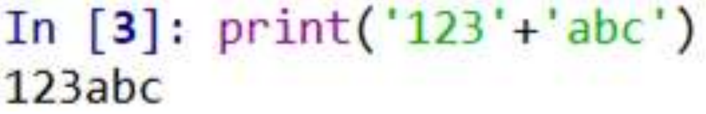
✓Use ‘‘‘ … ’’’ or “““ … ””” to automatically make newlines without ‘ \n ’



```
In [2]: print('''
   ...: A: Hi.
   ...: B: Hi.
   ...: ''')

A: Hi.
B: Hi.
```

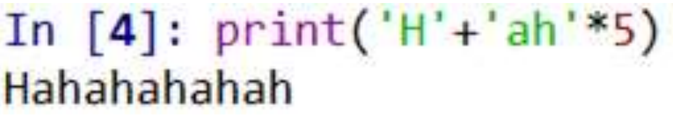
String (2)

- Use the operation + to concatenate strings



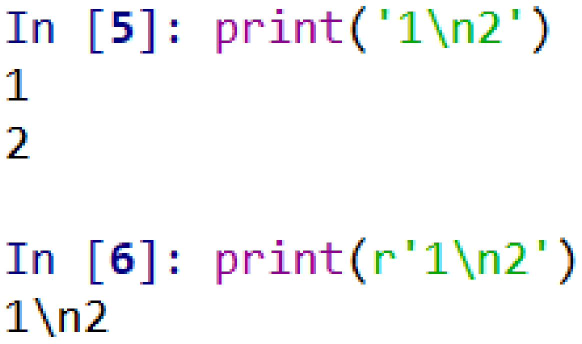
```
In [3]: print('123'+'abc')
123abc
```

- Use the operation * to repeat string



```
In [4]: print('H'+'ah'*5)
Hahahahahah
```

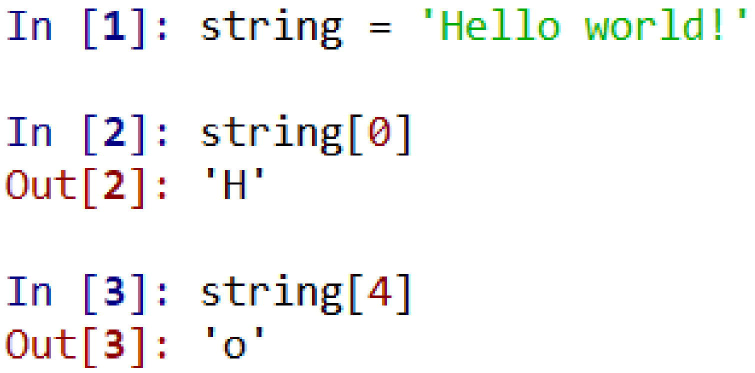
- Put r before ‘ … ’ to disable ‘ \ ’



```
In [5]: print('1\n2')
1
2

In [6]: print(r'1\n2')
1\n2
```

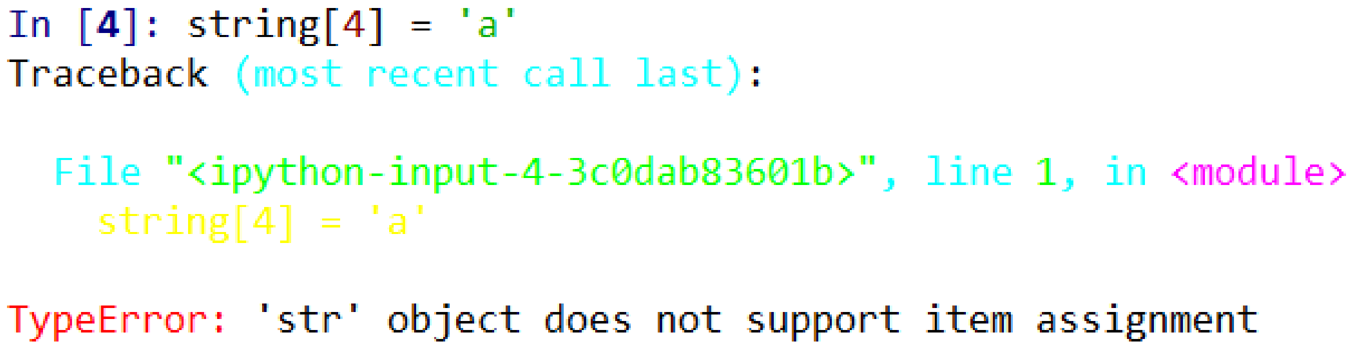
String (3)

- ✓We can use index to find the character in the strings



```
In [1]: string = 'Hello world!'

In [2]: string[0]
Out[2]: 'H'

In [3]: string[4]
Out[3]: 'o'
```

- ✓But string is not allowed to change□   Strings are immutable



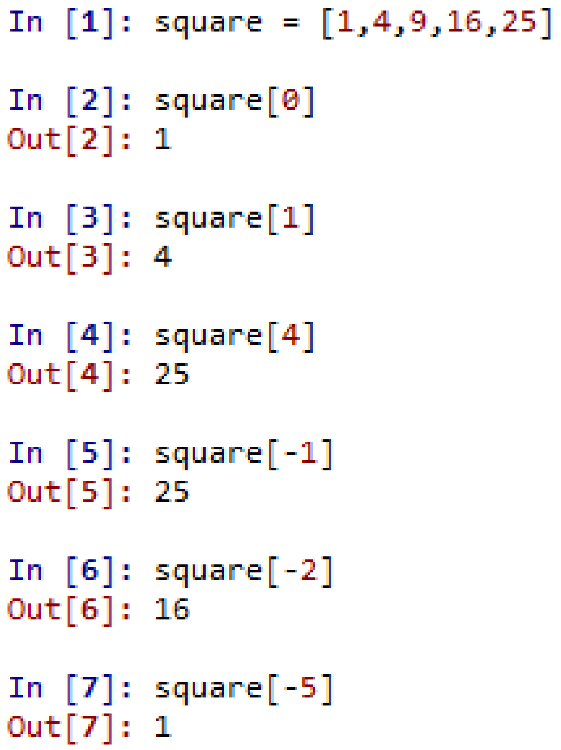
```
In [4]: string[4] = 'a'
Traceback (most recent call last):

  File "<ipython-input-4-3c0dab83601b>", line 1, in <module>
    string[4] = 'a'

TypeError: 'str' object does not support item assignment
```

List (1)

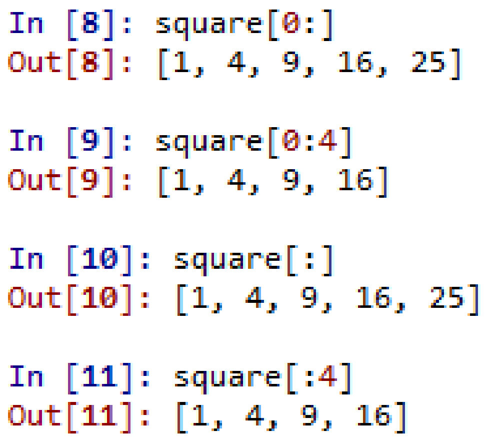
- Content:
  - Can be every kind of data type
    - Integer
    - Float
    - String
    - List
    - etc.

List (2)

- Index:



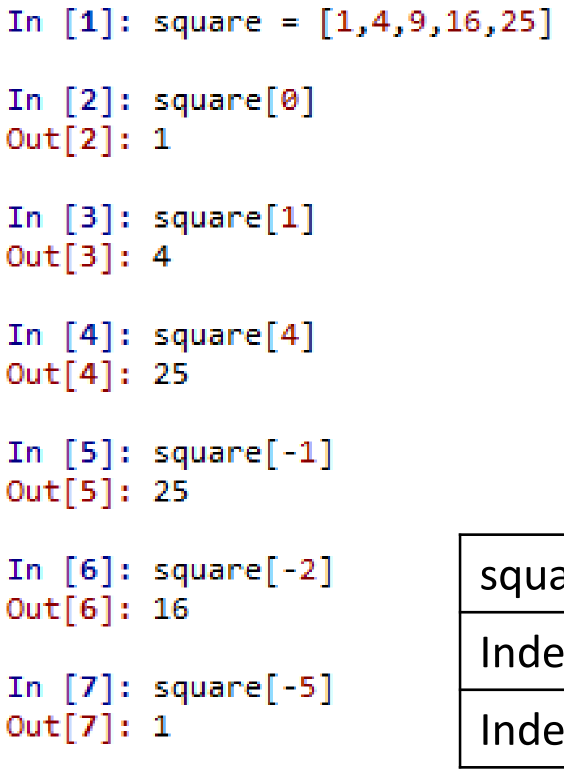
```
In [1]: square = [1,4,9,16,25]

In [2]: square[0]
Out[2]: 1

In [3]: square[1]
Out[3]: 4

In [4]: square[4]
Out[4]: 25

In [5]: square[-1]
Out[5]: 25

In [6]: square[-2]
Out[6]: 16

In [7]: square[-5]
Out[7]: 1
```



```
In [8]: square[0:]
Out[8]: [1, 4, 9, 16, 25]

In [9]: square[0:4]
Out[9]: [1, 4, 9, 16]

In [10]: square[:]
Out[10]: [1, 4, 9, 16, 25]

In [11]: square[:4]
Out[11]: [1, 4, 9, 16]
```

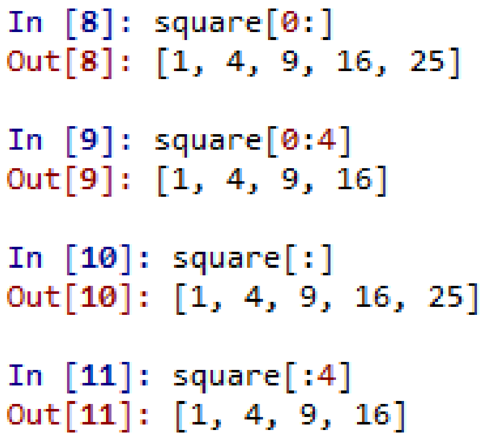
List (2)

✓Index:



```
In [1]: square = [1,4,9,16,25]

In [2]: square[0]
Out[2]: 1

In [3]: square[1]
Out[3]: 4

In [4]: square[4]
Out[4]: 25

In [5]: square[-1]
Out[5]: 25

In [6]: square[-2]
Out[6]: 16

In [7]: square[-5]
Out[7]: 1
```

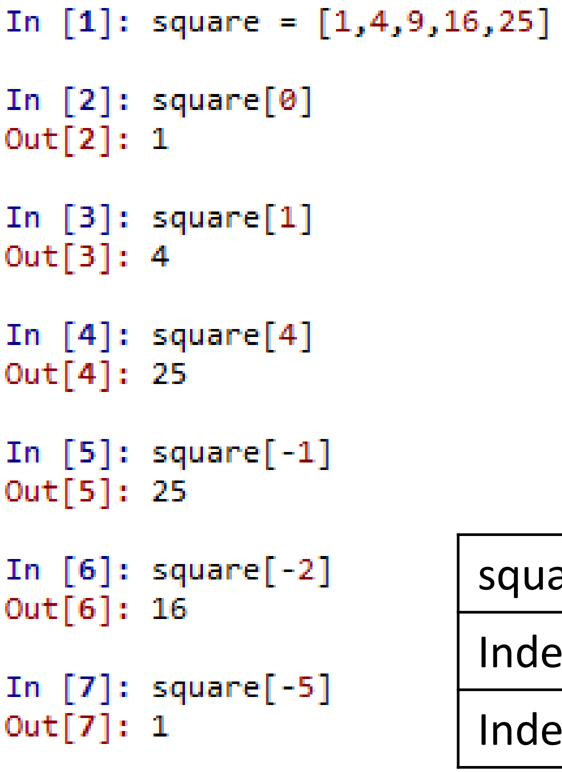
square	1	4	9	16	25
Index-1	0	1	2	3	4
Index-2	-5	-4	-3	-2	-1



```
In [8]: square[0:]
Out[8]: [1, 4, 9, 16, 25]

In [9]: square[0:4]
Out[9]: [1, 4, 9, 16]

In [10]: square[:]
Out[10]: [1, 4, 9, 16, 25]

In [11]: square[:4]
Out[11]: [1, 4, 9, 16]
```

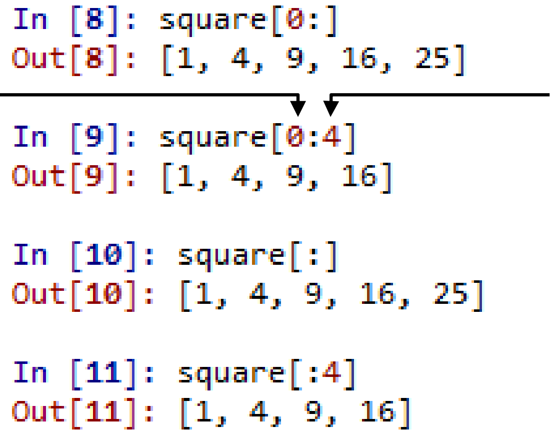
List (2)

✓Index:



```
In [1]: square = [1,4,9,16,25]

In [2]: square[0]
Out[2]: 1

In [3]: square[1]
Out[3]: 4

In [4]: square[4]
Out[4]: 25

In [5]: square[-1]
Out[5]: 25

In [6]: square[-2]
Out[6]: 16

In [7]: square[-5]
Out[7]: 1
```

square	1	4	9	16	25
Index-1	0	1	2	3	4
Index-2	-5	-4	-3	-2	-1



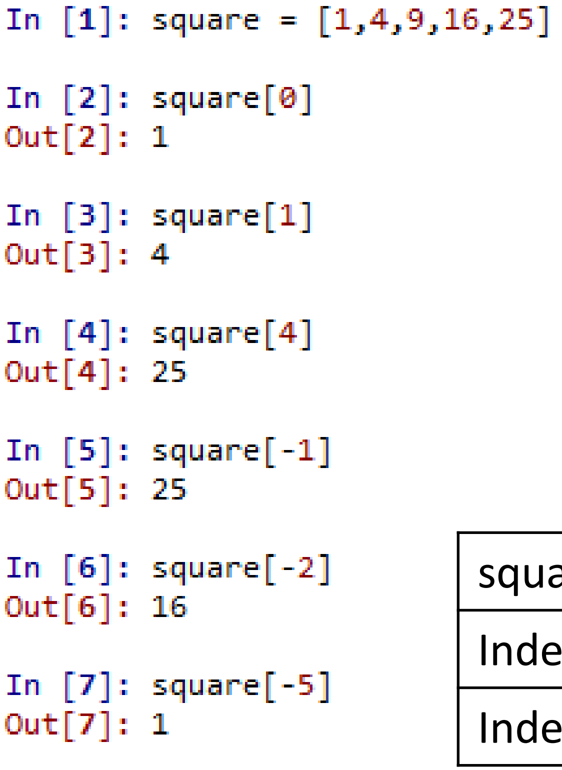
```
In [8]: square[0:]
Out[8]: [1, 4, 9, 16, 25]

In [9]: square[0:4]
Out[9]: [1, 4, 9, 16]

In [10]: square[:]
Out[10]: [1, 4, 9, 16, 25]

In [11]: square[:4]
Out[11]: [1, 4, 9, 16]
```

included

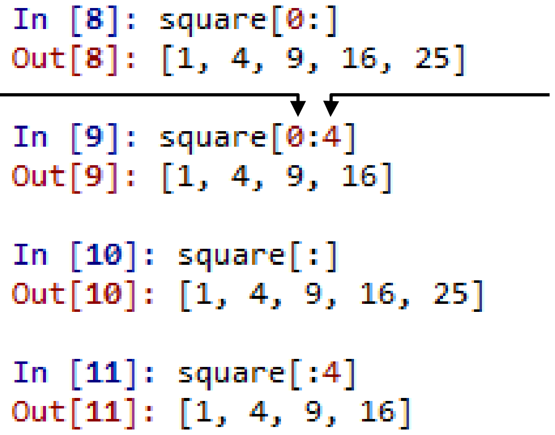
excluded

List (2)

- ✓Index:



```
In [1]: square = [1,4,9,16,25]

In [2]: square[0]
Out[2]: 1

In [3]: square[1]
Out[3]: 4

In [4]: square[4]
Out[4]: 25

In [5]: square[-1]
Out[5]: 25

In [6]: square[-2]
Out[6]: 16

In [7]: square[-5]
Out[7]: 1
```



```
In [8]: square[0:]
Out[8]: [1, 4, 9, 16, 25]

In [9]: square[0:4]
Out[9]: [1, 4, 9, 16]

In [10]: square[:]
Out[10]: [1, 4, 9, 16, 25]

In [11]: square[:4]
Out[11]: [1, 4, 9, 16]
```

included

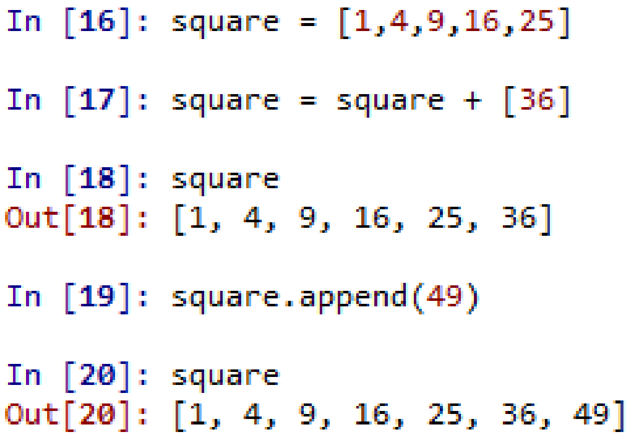
excluded

□ all elements

square	1	4	9	16	25
Index-1	0	1	2	3	4
Index-2	-5	-4	-3	-2	-1

List (3)

✓Concatenation:



```
In [16]: square = [1,4,9,16,25]
```

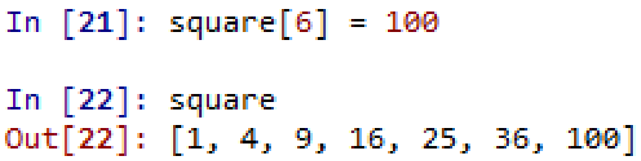
Approach 1:

```
In [17]: square = square + [36]

In [18]: square
Out[18]: [1, 4, 9, 16, 25, 36]
```

Approach 2:

```
In [19]: square.append(49)

In [20]: square
Out[20]: [1, 4, 9, 16, 25, 36, 49]
```

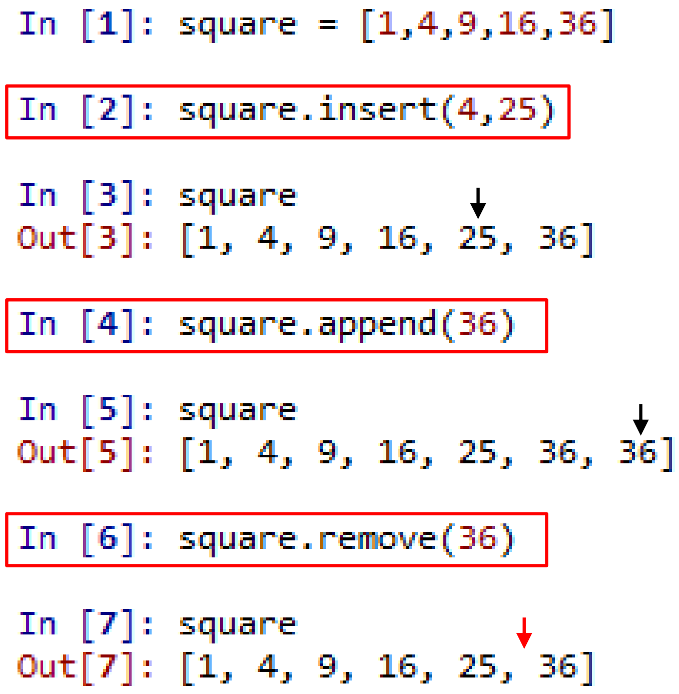
✓Mutable:



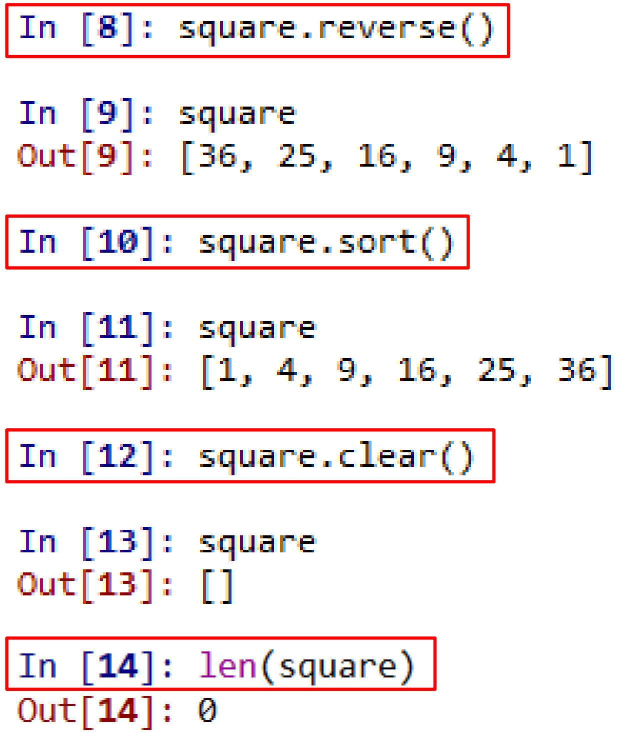
```
In [21]: square[6] = 100

In [22]: square
Out[22]: [1, 4, 9, 16, 25, 36, 100]
```

List (4) – Comprehension

- Extra functions may be used:



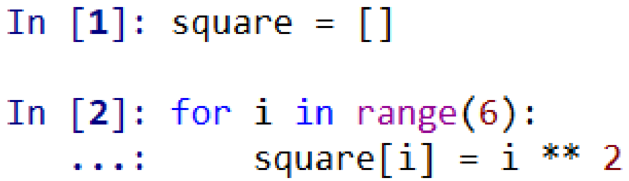
```
In [1]: square = [1,4,9,16,36]

In [2]: square.insert(4,25)

In [3]: square
Out[3]: [1, 4, 9, 16, 25, 36]

In [4]: square.append(36)

In [5]: square
Out[5]: [1, 4, 9, 16, 25, 36, 36]

In [6]: square.remove(36)

In [7]: square
Out[7]: [1, 4, 9, 16, 25, 36]
```



```
In [8]: square.reverse()

In [9]: square
Out[9]: [36, 25, 16, 9, 4, 1]

In [10]: square.sort()

In [11]: square
Out[11]: [1, 4, 9, 16, 25, 36]

In [12]: square.clear()

In [13]: square
Out[13]: []

In [14]: len(square)
Out[14]: 0
```

List (5)

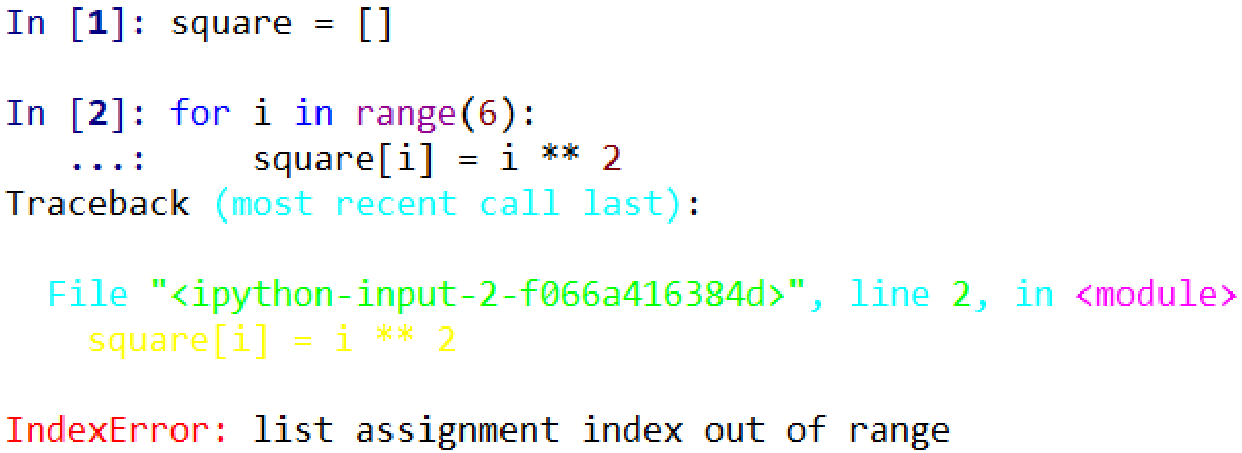
- We must be careful when we using list



```
In [1]: square = []

In [2]: for i in range(6):
   ...:     square[i] = i ** 2
```

List (5)

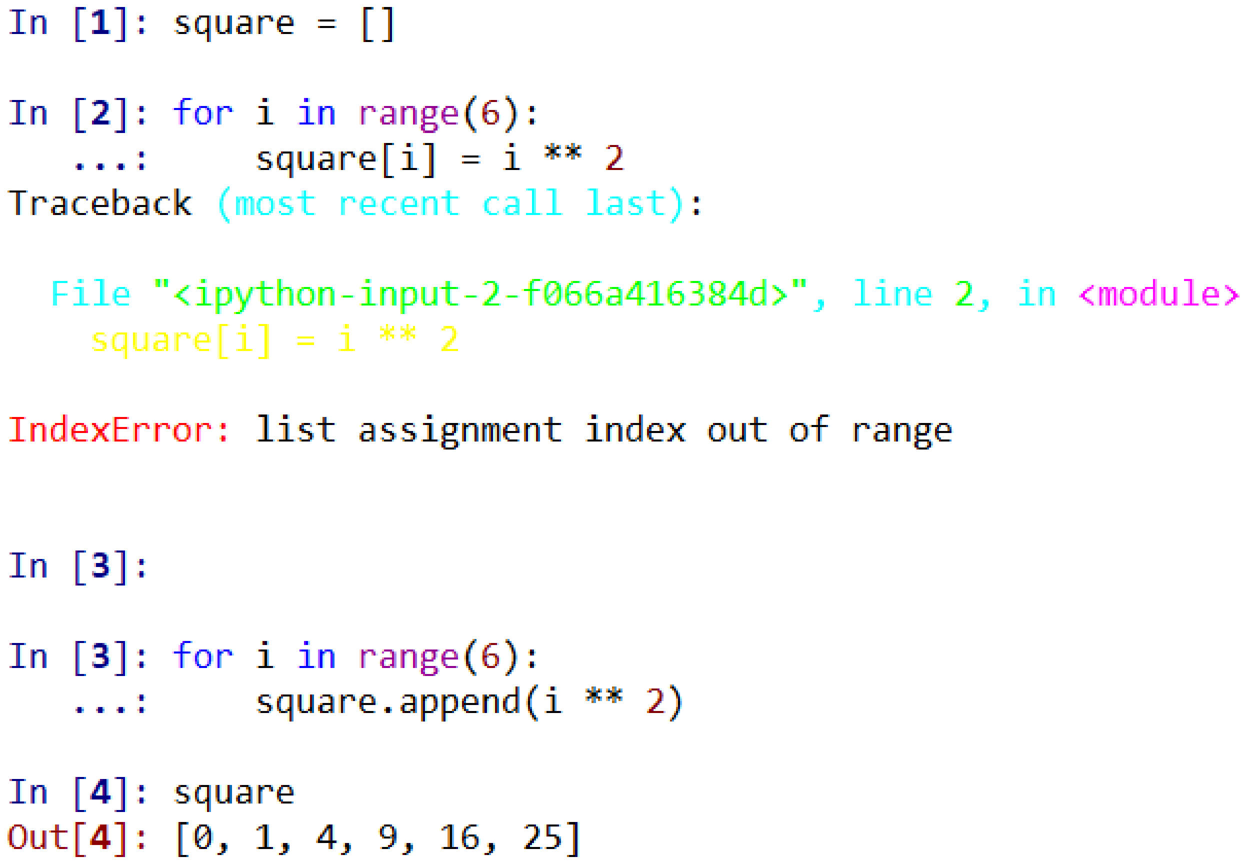
- We must be careful when we using list



```
In [1]: square = []

In [2]: for i in range(6):
   ...:     square[i] = i ** 2
Traceback (most recent call last):

  File "<ipython-input-2-f066a416384d>", line 2, in <module>
    square[i] = i ** 2

IndexError: list assignment index out of range
```

List (5)

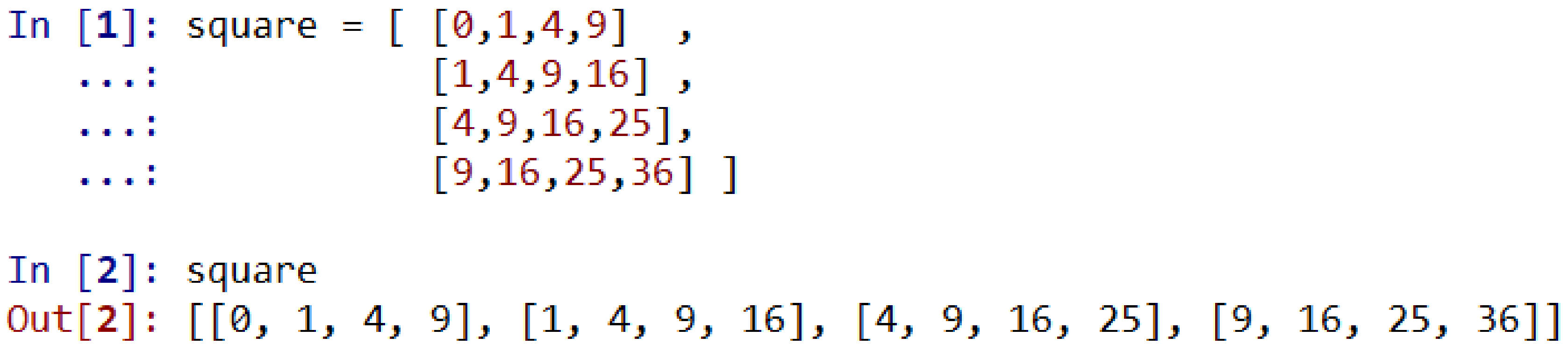
- We must be careful when we using list



```
In [1]: square = []

In [2]: for i in range(6):
   ...:     square[i] = i ** 2
Traceback (most recent call last):

  File "<ipython-input-2-f066a416384d>", line 2, in <module>
    square[i] = i ** 2

IndexError: list assignment index out of range


In [3]:

In [3]: for i in range(6):
   ...:     square.append(i ** 2)

In [4]: square
Out[4]: [0, 1, 4, 9, 16, 25]
```

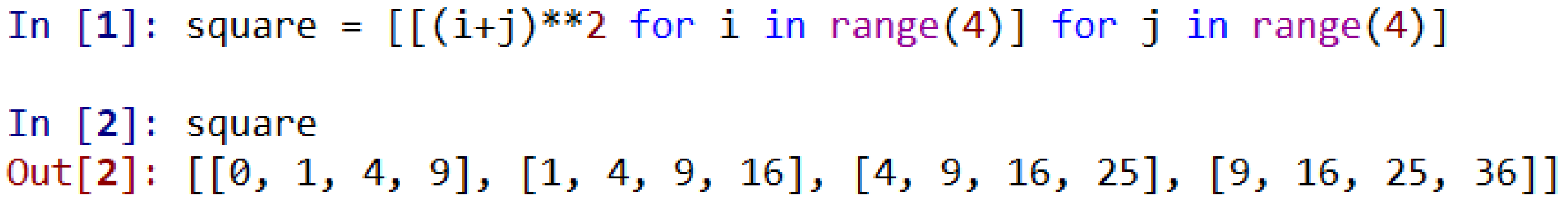
List (6) – Nested List

- ✓ A list which elements are lists



```
In [1]: square = [ [0,1,4,9]   ,
   ...:            [1,4,9,16] ,
   ...:            [4,9,16,25],
   ...:            [9,16,25,36] ]

In [2]: square
Out[2]: [[0, 1, 4, 9], [1, 4, 9, 16], [4, 9, 16, 25], [9, 16, 25, 36]]
```

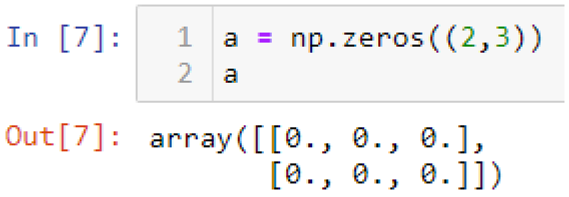
- ✓ In a more efficient way



```
In [1]: square = [[(i+j)**2 for i in range(4)] for j in range(4)]

In [2]: square
Out[2]: [[0, 1, 4, 9], [1, 4, 9, 16], [4, 9, 16, 25], [9, 16, 25, 36]]
```

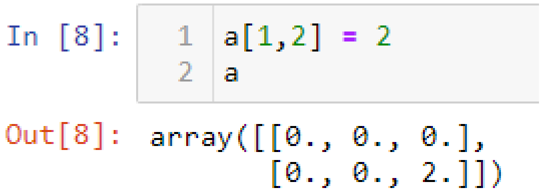
Matrix (1)

✓Define zero matrix



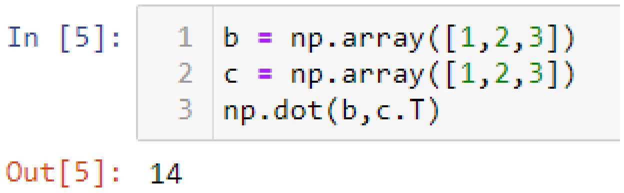
```
In [7]:  a = np.zeros((2,3))
         a

Out[7]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

✓Change element’s value



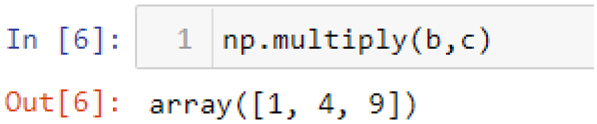
```
In [8]:  a[1,2] = 2
         a

Out[8]: array([[0., 0., 0.],
               [0., 0., 2.]])
```

✓np.dot



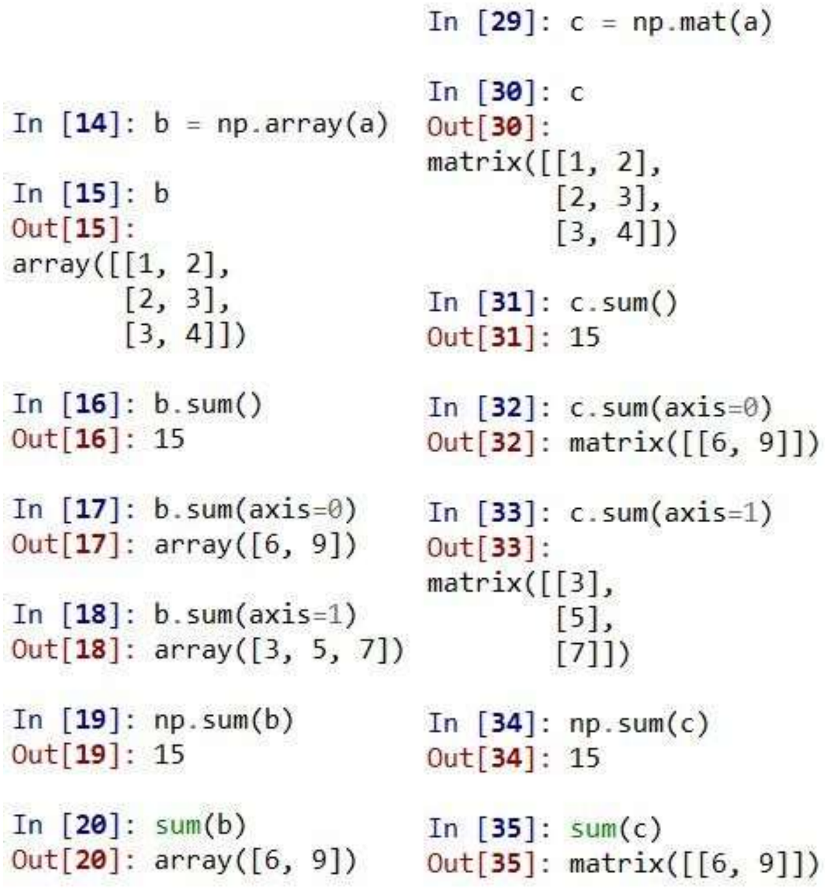
```
In [5]:  b = np.array([1,2,3])
         c = np.array([1,2,3])
         np.dot(b,c.T)

Out[5]: 14
```

✓np.multiply or *



```
In [6]:  np.multiply(b,c)

Out[6]: array([1, 4, 9])
```

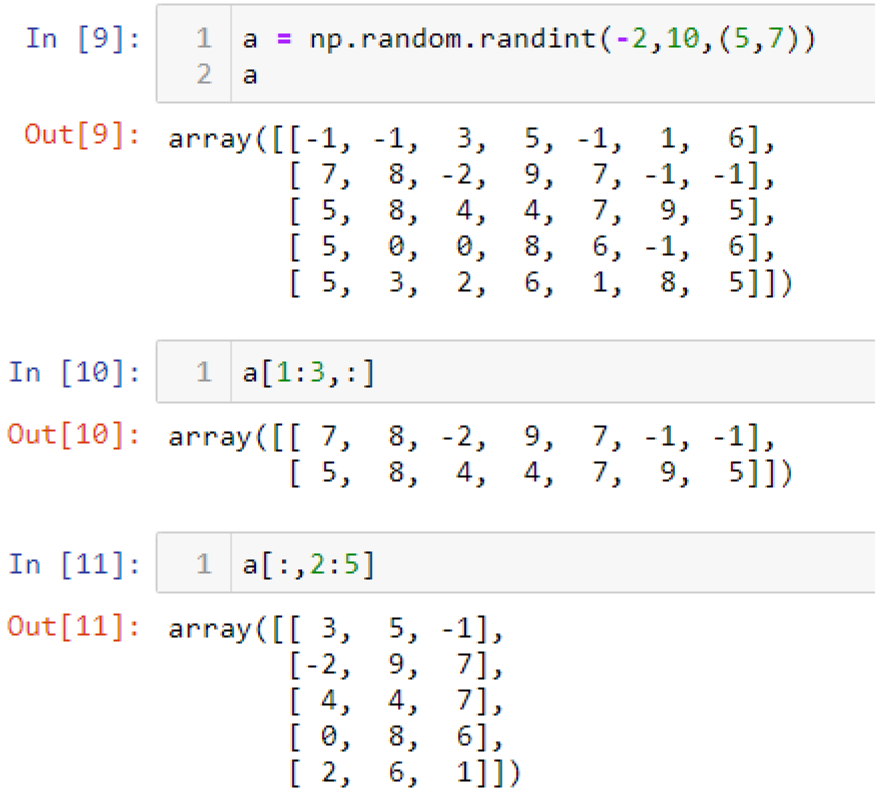
Matrix (2)



```
In [14]: b = np.array(a)

In [15]: b
Out[15]:
array([[1, 2],
       [2, 3],
       [3, 4]])

In [16]: b.sum()
Out[16]: 15

In [17]: b.sum(axis=0)
Out[17]: array([6, 9])

In [18]: b.sum(axis=1)
Out[18]: array([3, 5, 7])

In [19]: np.sum(b)
Out[19]: 15

In [20]: sum(b)
Out[20]: array([6, 9])
```

```
In [29]: c = np.mat(a)

In [30]: c
Out[30]:
matrix([[1, 2],
        [2, 3],
        [3, 4]])

In [31]: c.sum()
Out[31]: 15

In [32]: c.sum(axis=0)
Out[32]: matrix([[6, 9]])

In [33]: c.sum(axis=1)
Out[33]:
matrix([[3],
        [5],
        [7]])

In [34]: np.sum(c)
Out[34]: 15

In [35]: sum(c)
Out[35]: matrix([[6, 9]])
```

Matrix (3)



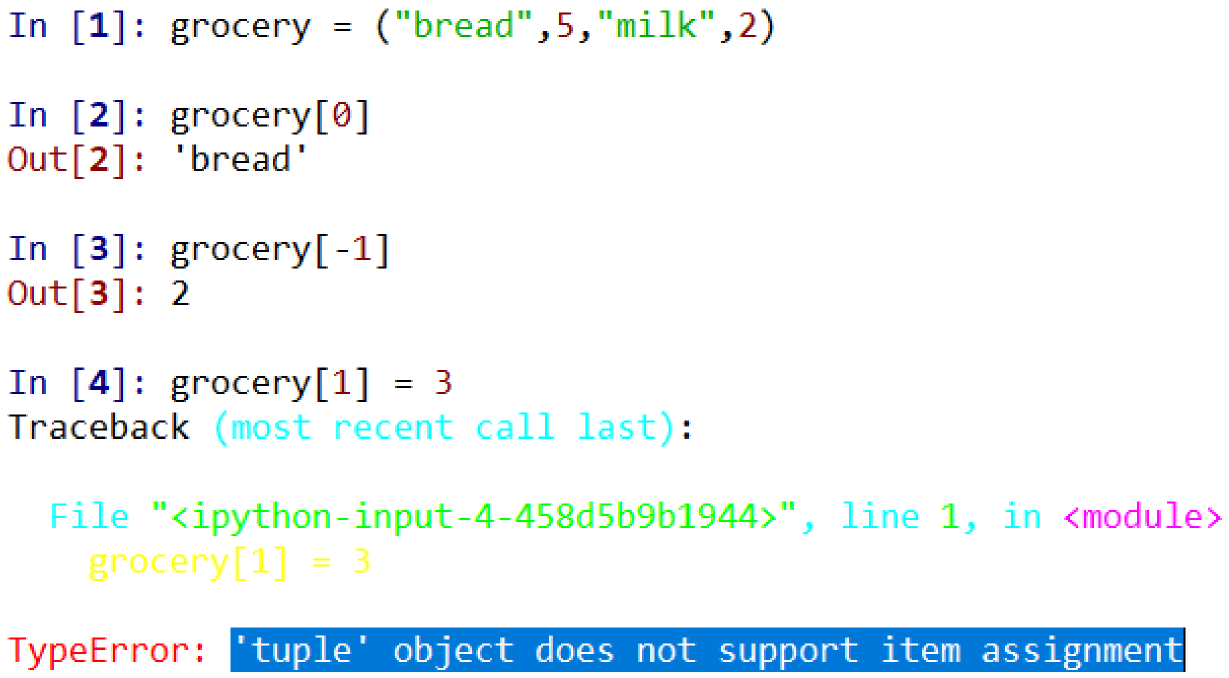
```
In [9]:  a = np.random.randint(-2,10,(5,7))
         a

Out[9]: array([[-1, -1,  3,  5, -1,  1,  6],
               [ 7,  8, -2,  9,  7, -1, -1],
               [ 5,  8,  4,  4,  7,  9,  5],
               [ 5,  0,  0,  8,  6, -1,  6],
               [ 5,  3,  2,  6,  1,  8,  5]])
```

```
In [10]:  a[1:3,:]

Out[10]: array([[ 7,  8, -2,  9,  7, -1, -1],
                [ 5,  8,  4,  4,  7,  9,  5]])
```

```
In [11]:  a[:,2:5]

Out[11]: array([[ 3,  5, -1],
                [-2,  9,  7],
                [ 4,  4,  7],
                [ 0,  8,  6],
                [ 2,  6,  1]])
```

Tuple (1)

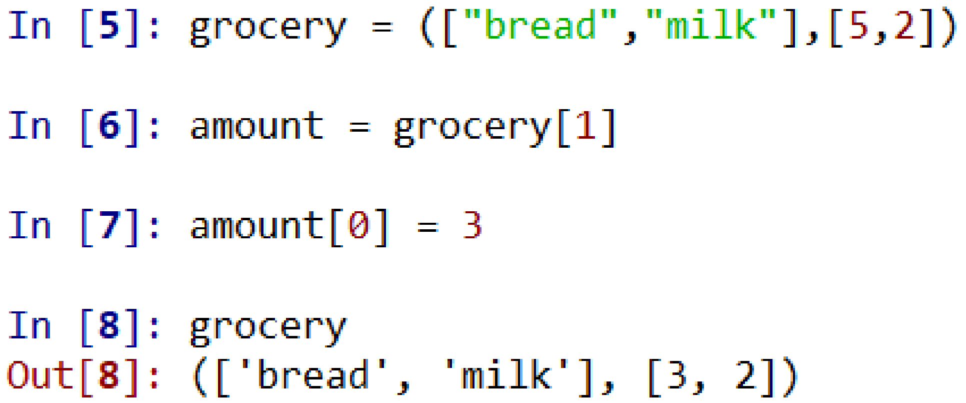
- Sort of an “immutable” list
  - Precisely, the addresses of the tuple points to are immutable.
- For example,



```
In [1]: grocery = ("bread",5,"milk",2)

In [2]: grocery[0]
Out[2]: 'bread'

In [3]: grocery[-1]
Out[3]: 2

In [4]: grocery[1] = 3
Traceback (most recent call last):

  File "<ipython-input-4-458d5b9b1944>", line 1, in <module>
    grocery[1] = 3

TypeError: 'tuple' object does not support item assignment
```

Tuple (2)

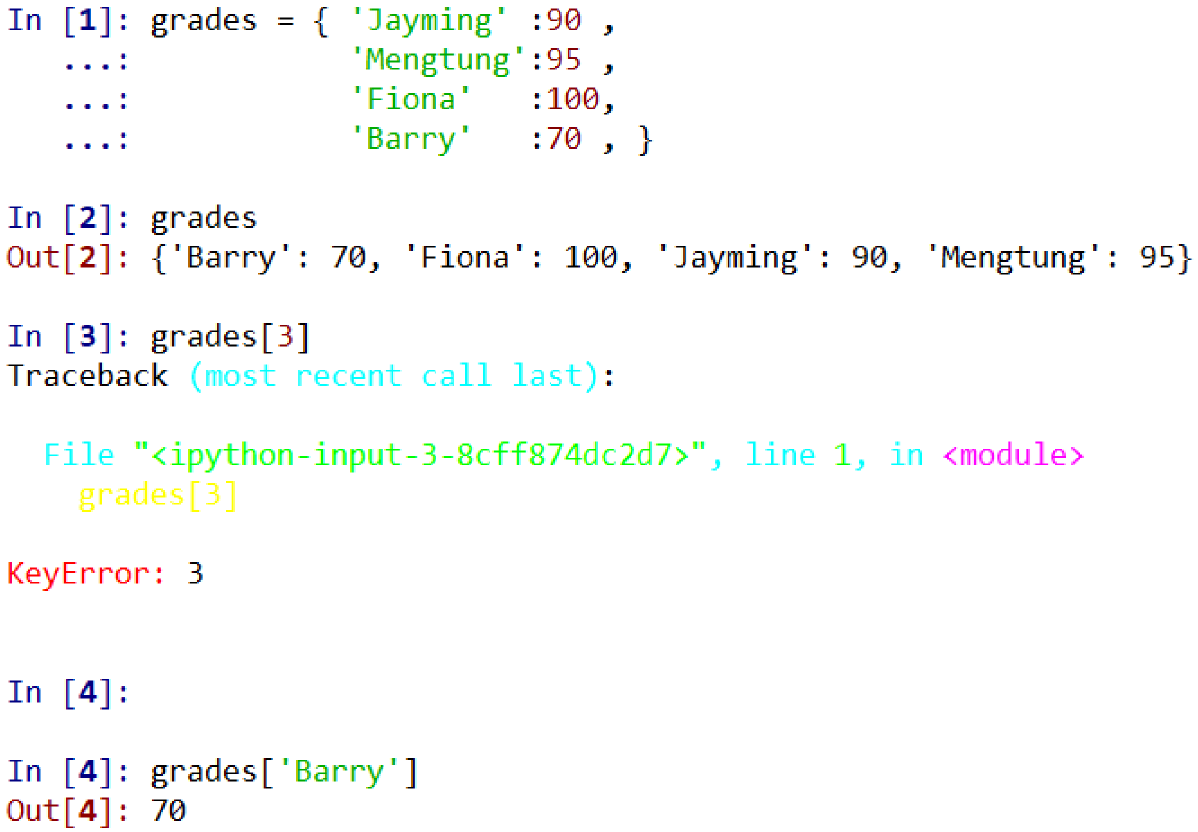
✓ However, we can still find a way to make our tuple mutable by using the attribute of lists.



```
In [5]: grocery = (["bread","milk"],[5,2])

In [6]: amount = grocery[1]

In [7]: amount[0] = 3

In [8]: grocery
Out[8]: (['bread', 'milk'], [3, 2])
```

Dictionary (1)

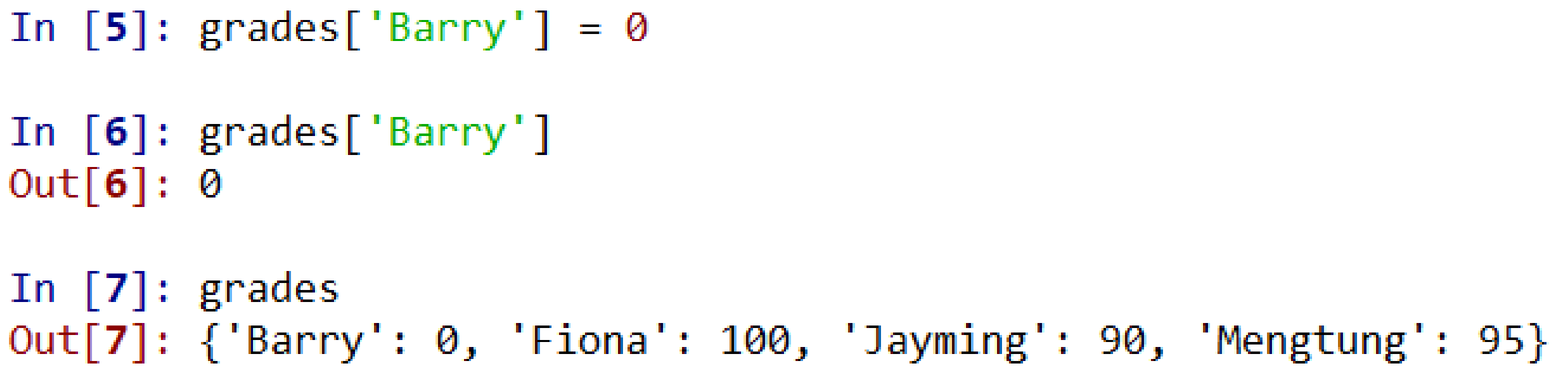
✓Mapping by the keys and the values



```
In [1]: grades = { 'Jayming' :90 ,
   ...:            'Mengtung':95 ,
   ...:            'Fiona'   :100,
   ...:            'Barry'   :70 , }

In [2]: grades
Out[2]: {'Barry': 70, 'Fiona': 100, 'Jayming': 90, 'Mengtung': 95}

In [3]: grades[3]
Traceback (most recent call last):

  File "<ipython-input-3-8cff874dc2d7>", line 1, in <module>
    grades[3]

KeyError: 3


In [4]:

In [4]: grades['Barry']
Out[4]: 70
```

Dictionary (2)

- ✓It’s mutable.

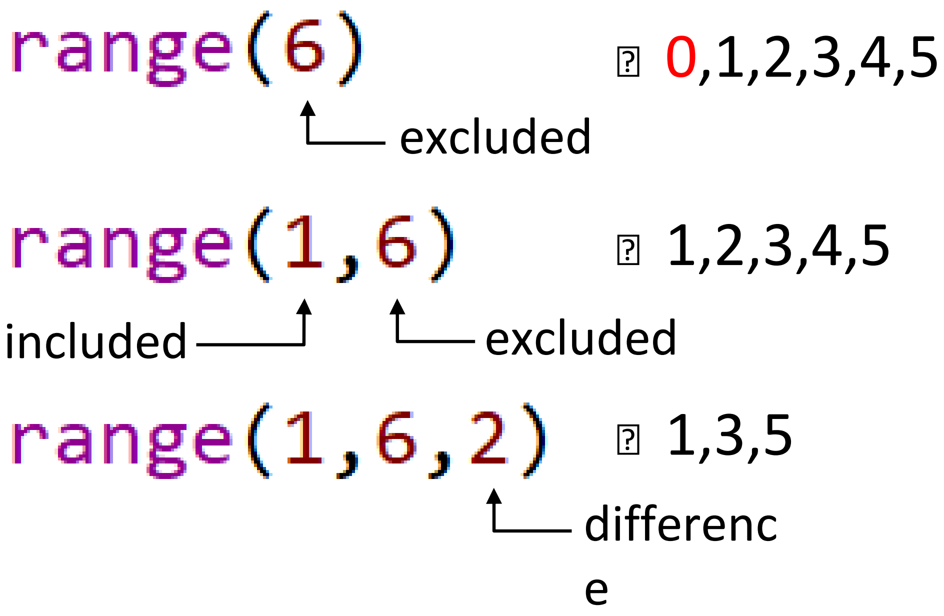


```
In [5]: grades['Barry'] = 0

In [6]: grades['Barry']
Out[6]: 0

In [7]: grades
Out[7]: {'Barry': 0, 'Fiona': 100, 'Jayming': 90, 'Mengtung': 95}
```

OUTLINE

- Installation
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- **Python Syntax**
- Module
- Function
- Class
- Name and Reference

Python Syntax (1) – range



```
range(6)
```

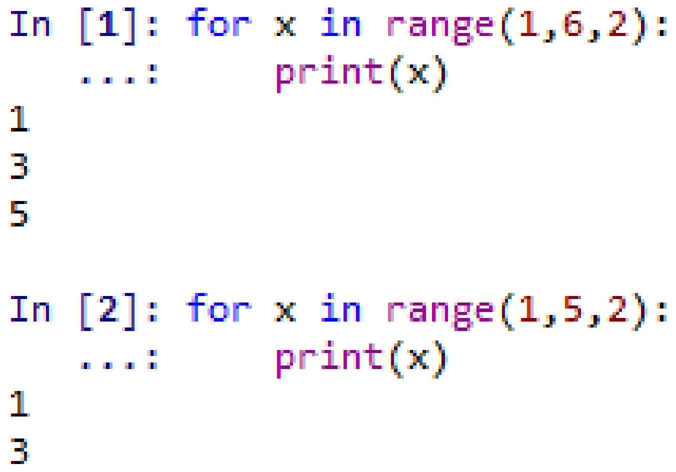
⍰ 0,1,2,3,4,5

```
range(1,6)
```

⍰ 1,2,3,4,5

```
range(1,6,2)
```

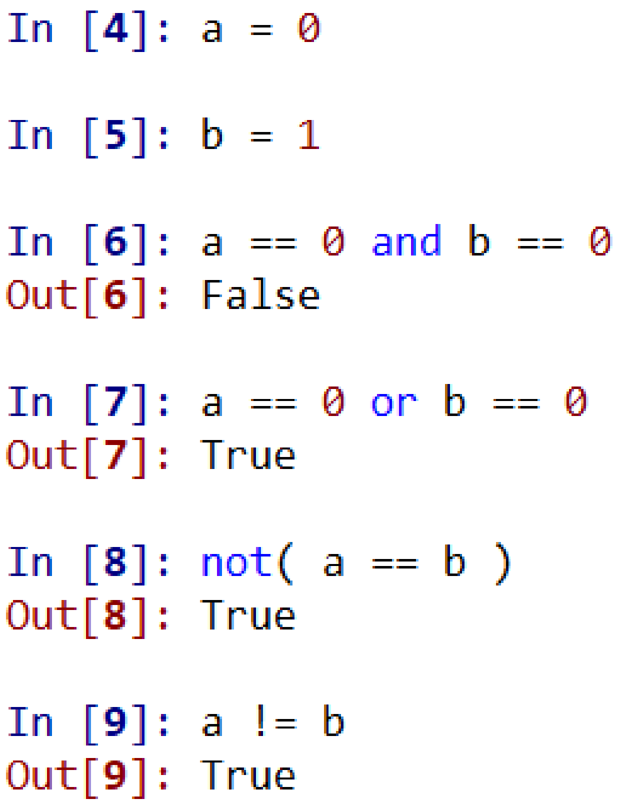
⍰ 1,3,5



```
In [1]: for x in range(1,6,2):
   ...:     print(x)
1
3
5

In [2]: for x in range(1,5,2):
   ...:     print(x)
1
3
```

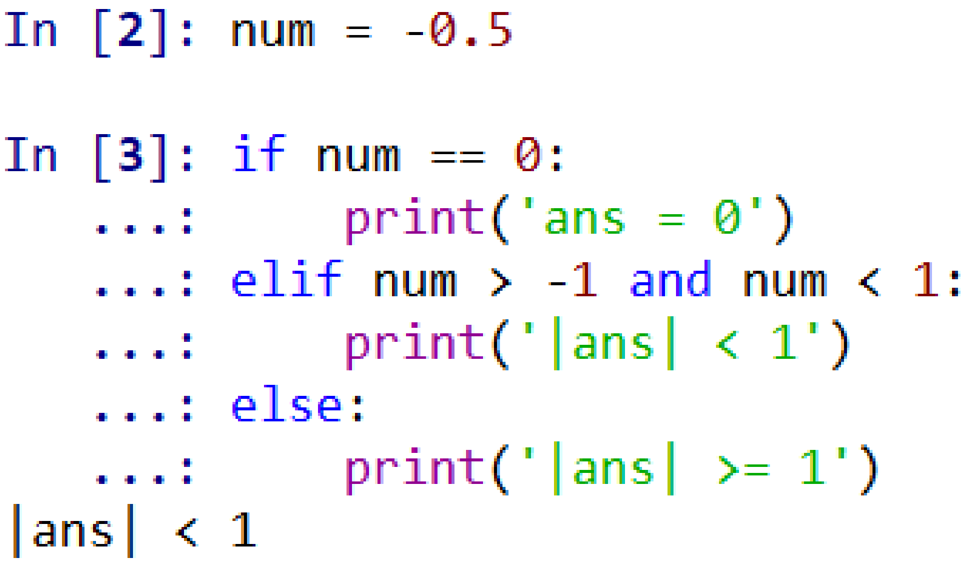
Python Syntax (2) – Boolean logic

- and, or, not



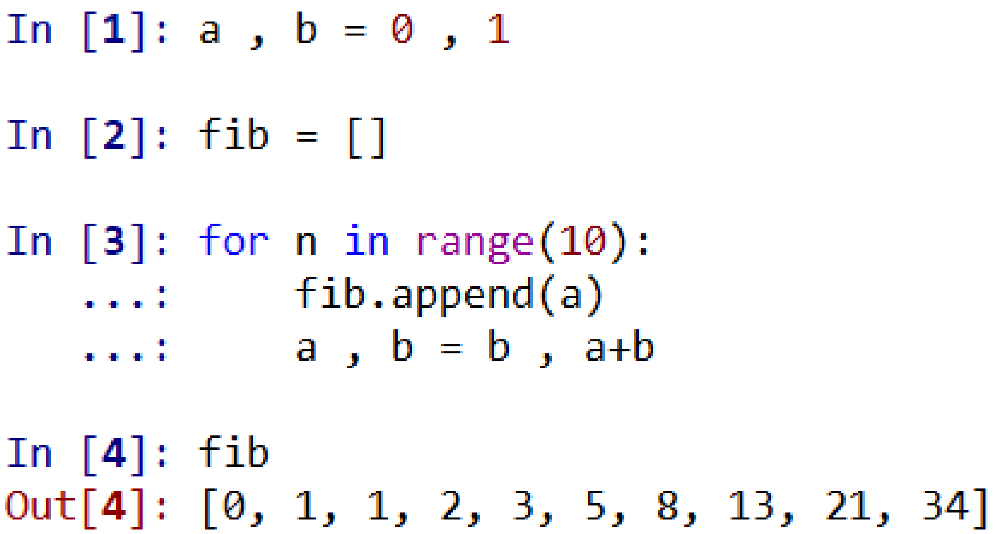
```
In [4]: a = 0

In [5]: b = 1

In [6]: a == 0 and b == 0
Out[6]: False

In [7]: a == 0 or b == 0
Out[7]: True

In [8]: not( a == b )
Out[8]: True

In [9]: a != b
Out[9]: True
```

Python Syntax (3) – if … else



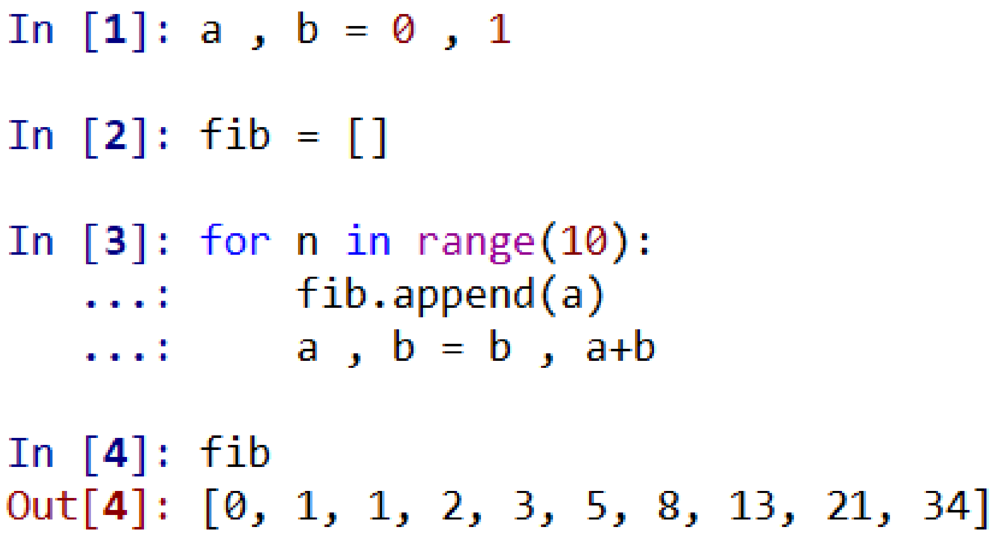
```
In [2]: num = -0.5

In [3]: if num == 0:
   ...:     print('ans = 0')
   ...: elif num > -1 and num < 1:
   ...:     print('|ans| < 1')
   ...: else:
   ...:     print('|ans| >= 1')
|ans| < 1
```

Python Syntax (4) – for loop

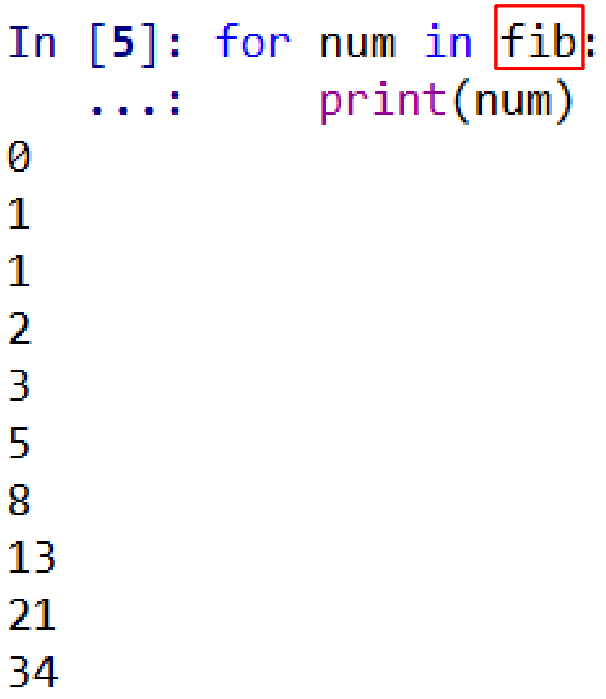


```
In [1]: a , b = 0 , 1

In [2]: fib = []

In [3]: for n in range(10):
   ...:     fib.append(a)
   ...:     a , b = b , a+b

In [4]: fib
Out[4]: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Python Syntax (4) – for loop



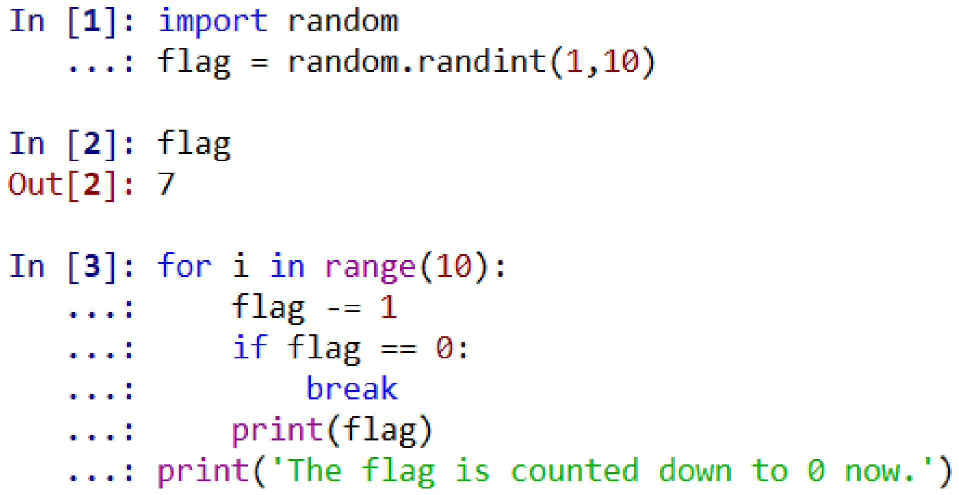
```
In [1]: a , b = 0 , 1

In [2]: fib = []

In [3]: for n in range(10):
   ...:     fib.append(a)
   ...:     a , b = b , a+b

In [4]: fib
Out[4]: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```



```
In [5]: for num in fib:
   ...:     print(num)
0
1
1
2
3
5
8
13
21
34
```

Python Syntax (5) – break

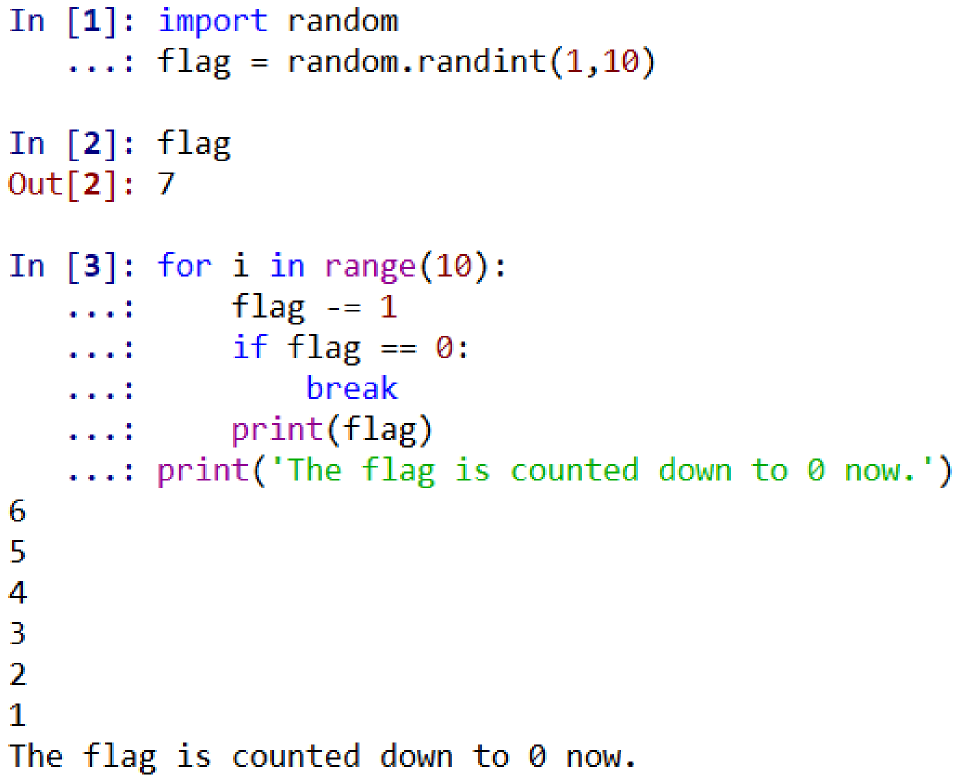
✓Break: skip the following commands and iterations and jump out of the loop where the “break” belongs



```
In [1]: import random
   ...: flag = random.randint(1,10)

In [2]: flag
Out[2]: 7

In [3]: for i in range(10):
   ...:     flag -= 1
   ...:     if flag == 0:
   ...:         break
   ...:     print(flag)
   ...: print('The flag is counted down to 0 now.')
```

Python Syntax (5) – break

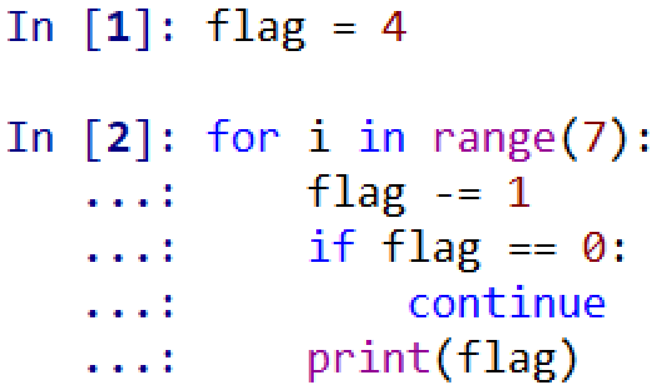
✓ Break: skip the following commands and iterations and jump out of the loop where the “break” belongs



```
In [1]: import random
   ...: flag = random.randint(1,10)

In [2]: flag
Out[2]: 7

In [3]: for i in range(10):
   ...:     flag -= 1
   ...:     if flag == 0:
   ...:         break
   ...:     print(flag)
   ...: print('The flag is counted down to 0 now.')
6
5
4
3
2
1
The flag is counted down to 0 now.
```

Python Syntax (6) – continue

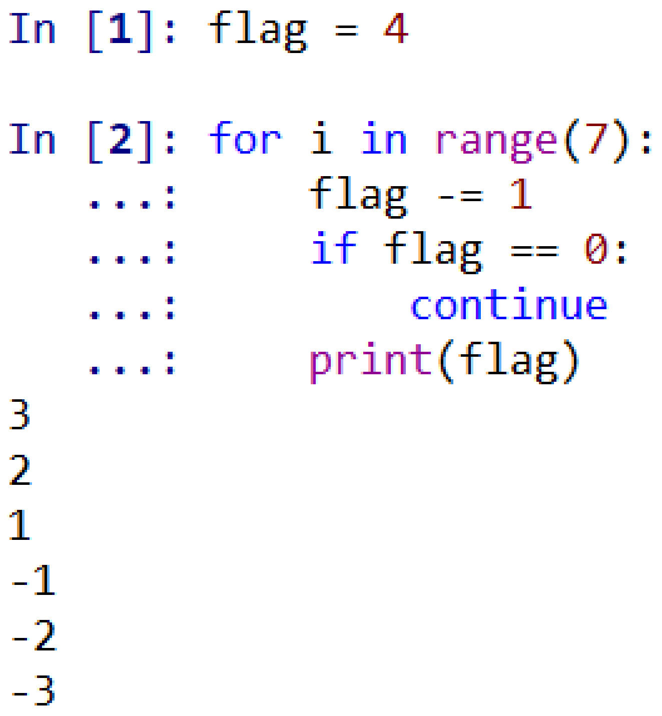
✓Continue: skip the following commands and direct to the next iteration



```
In [1]: flag = 4

In [2]: for i in range(7):
   ...:     flag -= 1
   ...:     if flag == 0:
   ...:         continue
   ...:     print(flag)
```

Python Syntax (6) – continue

- Continue: skip the following commands and direct to the next iteration

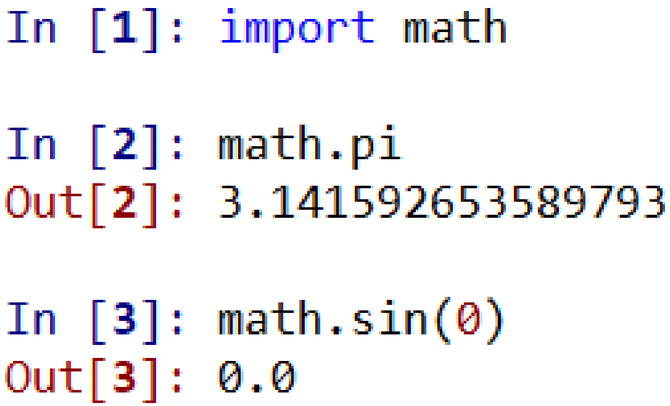


```
In [1]: flag = 4

In [2]: for i in range(7):
   ...:     flag -= 1
   ...:     if flag == 0:
   ...:         continue
   ...:     print(flag)
3
2
1
-1
-2
-3
```

OUTLINE

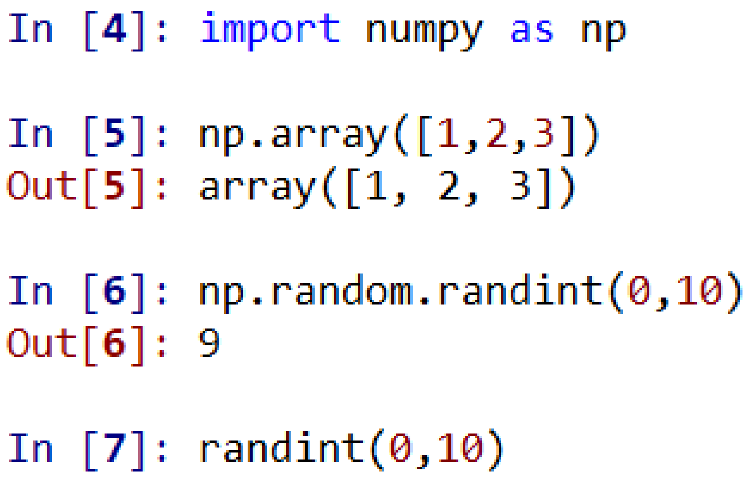
- Installation
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- **Module**
- Function
- Class
- Name and Reference

Module (1) – import



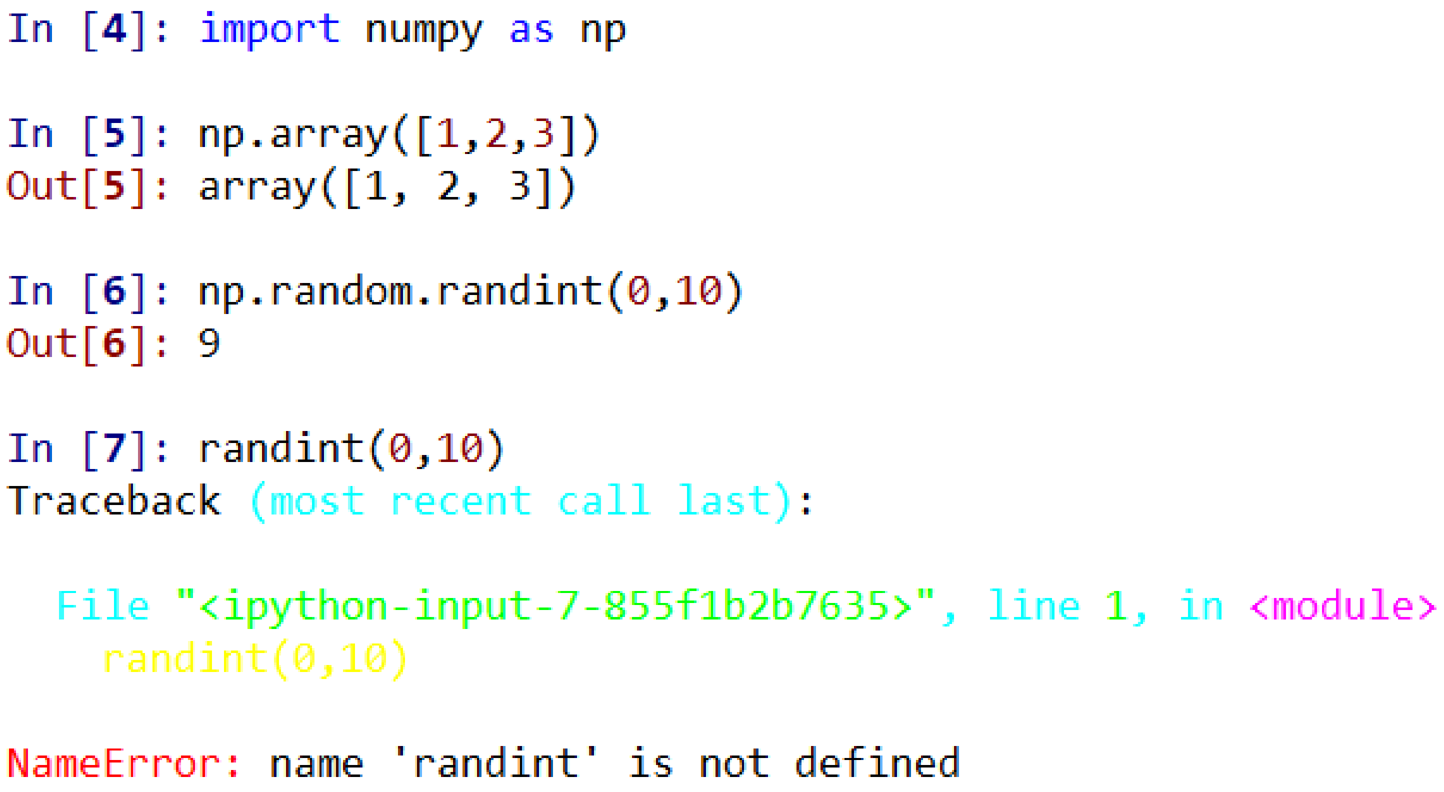
```
In [1]: import math

In [2]: math.pi
Out[2]: 3.141592653589793

In [3]: math.sin(0)
Out[3]: 0.0
```

Module (2) – import … as …



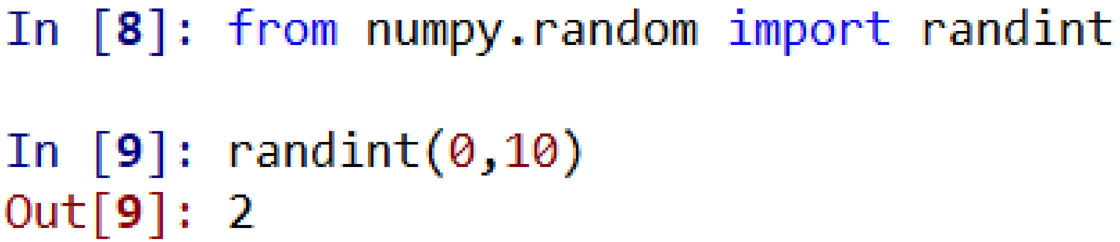
```
In [4]: import numpy as np

In [5]: np.array([1,2,3])
Out[5]: array([1, 2, 3])

In [6]: np.random.randint(0,10)
Out[6]: 9

In [7]: randint(0,10)
```

Module (2) – import … as …



```
In [4]: import numpy as np

In [5]: np.array([1,2,3])
Out[5]: array([1, 2, 3])

In [6]: np.random.randint(0,10)
Out[6]: 9

In [7]: randint(0,10)
Traceback (most recent call last):

  File "<ipython-input-7-855f1b2b7635>", line 1, in <module>
    randint(0,10)

NameError: name 'randint' is not defined
```

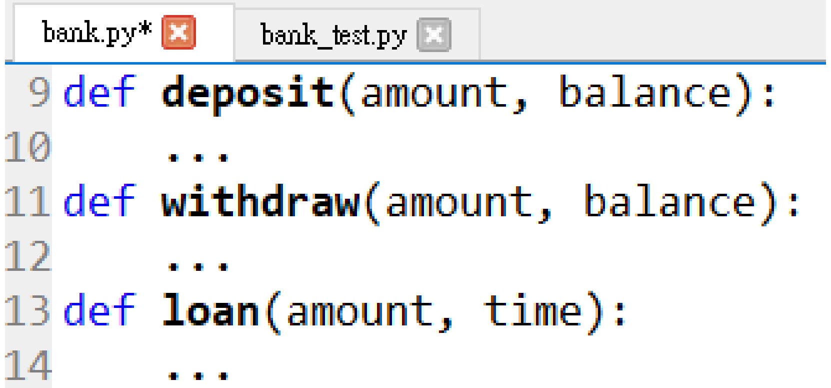
Module (3) – from … import …



```
In [8]: from numpy.random import randint

In [9]: randint(0,10)
Out[9]: 2
```

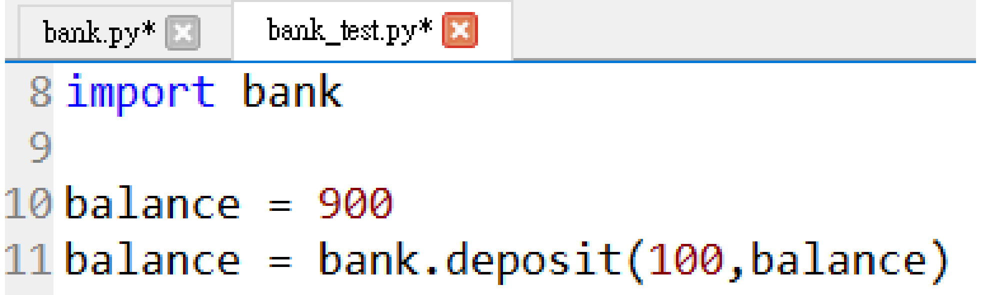
Module (4)

- ✓ Many functions and constants need to be defined
- ✓ Create a source code named “bank.py”



```
def deposit(amount, balance):
    ...
def withdraw(amount, balance):
    ...
def loan(amount, time):
    ...
```

- ✓ import this “bank” module and call the functions in it



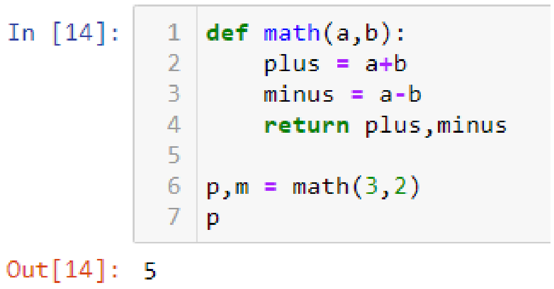
```
import bank

balance = 900
balance = bank.deposit(100,balance)
```

OUTLINE

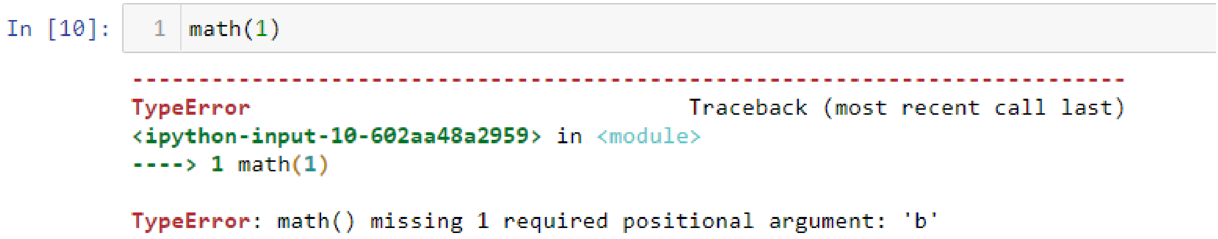
- Installation
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- Module
- **Function**
- Class
- Name and Reference

Function

✓Define function



```
In [14]: def math(a,b):
             plus = a+b
             minus = a-b
             return plus,minus

         p,m = math(3,2)
         p

Out[14]: 5
```



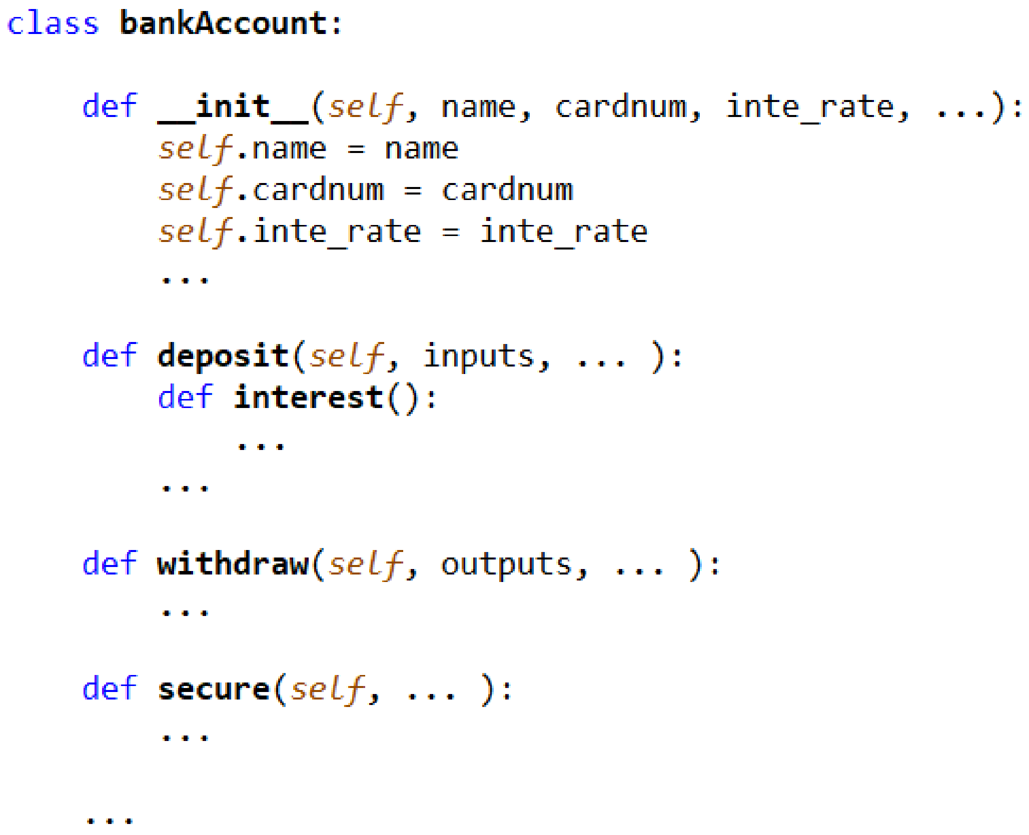
```
In [10]: math(1)

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-10-602aa48a2959> in <module>
         ----> 1 math(1)

         TypeError: math() missing 1 required positional argument: 'b'
```

OUTLINE

- Installation
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- Module
- Function
- **Class**
- Name and Reference

Class (1)

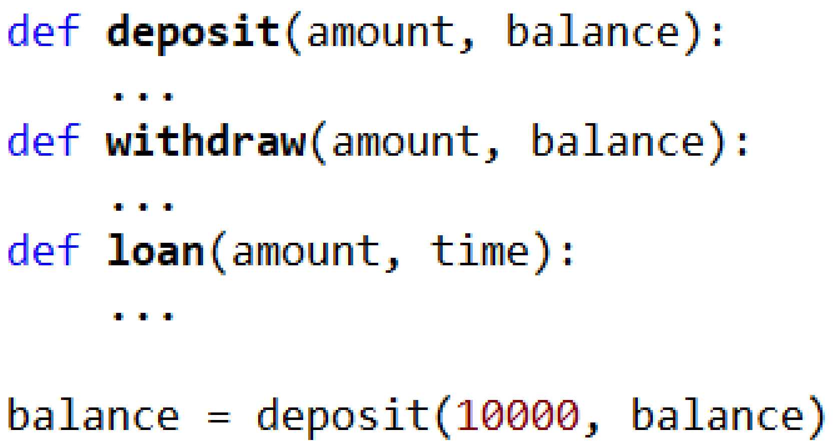
✓Some states and functions need to be combined



```
class bankAccount:

    def __init__(self, name, cardnum, inte_rate, ...):
        self.name = name
        self.cardnum = cardnum
        self.inte_rate = inte_rate
        ...

    def deposit(self, inputs, ... ):
        def interest():
            ...
        ...

    def withdraw(self, outputs, ... ):
        ...

    def secure(self, ... ):
        ...

    ...
```

Class (2)

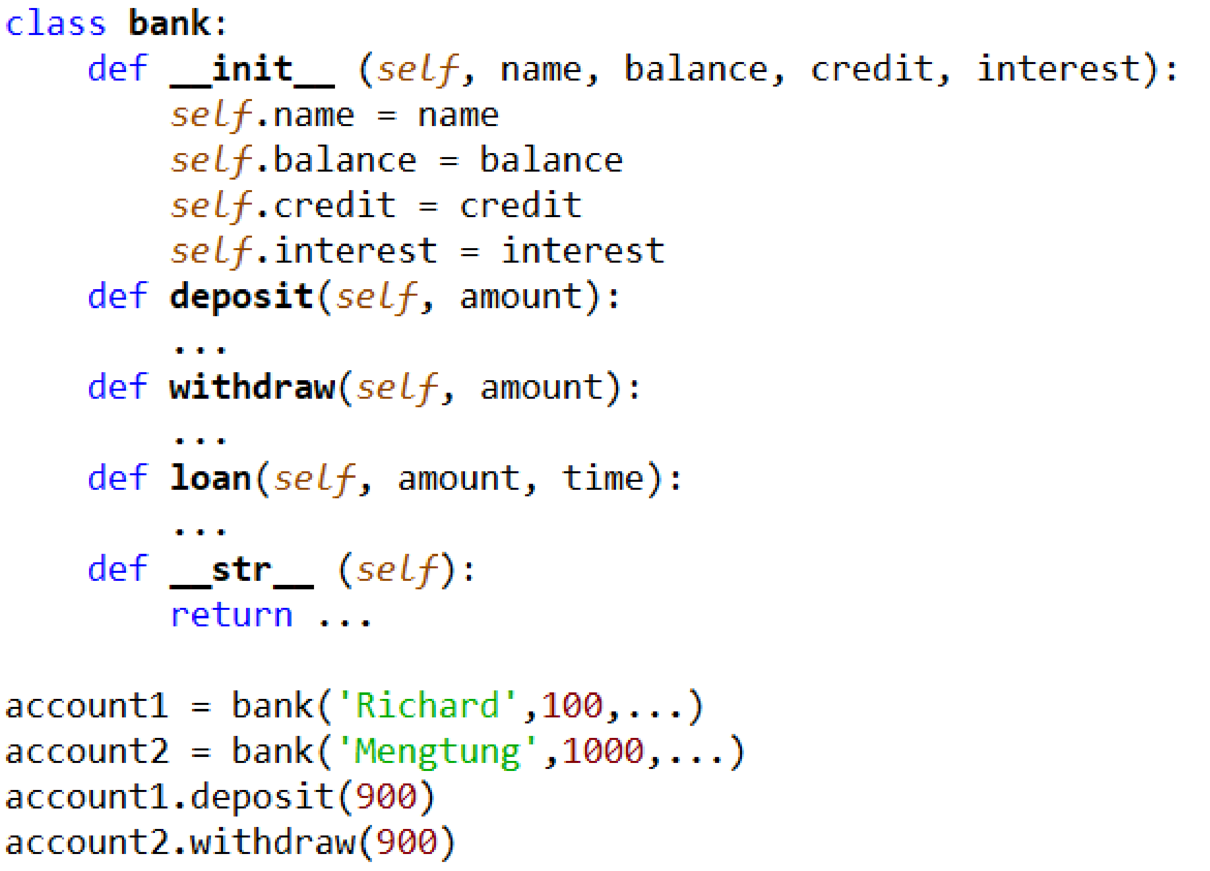
- If you don’t use class,



```
def deposit(amount, balance):
    ...
def withdraw(amount, balance):
    ...
def loan(amount, time):
    ...

balance = deposit(10000, balance)
```

Class (3)

- ✓ If you use class,



```
class bank:
    def __init__ (self, name, balance, credit, interest):
        self.name = name
        self.balance = balance
        self.credit = credit
        self.interest = interest
    def deposit(self, amount):
        ...
    def withdraw(self, amount):
        ...
    def loan(self, amount, time):
        ...
    def __str__ (self):
        return ...

account1 = bank('Richard',100,...)
account2 = bank('Mengtung',1000,...)
account1.deposit(900)
account2.withdraw(900)
```

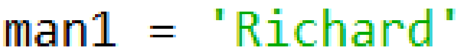
OUTLINE

- Installation
- Mathematical Operation
- Container
  - String
  - List
  - Matrix
  - Tuple
  - Dictionary
- Python Syntax
- Module
- Function
- Class
- **Name and Reference**

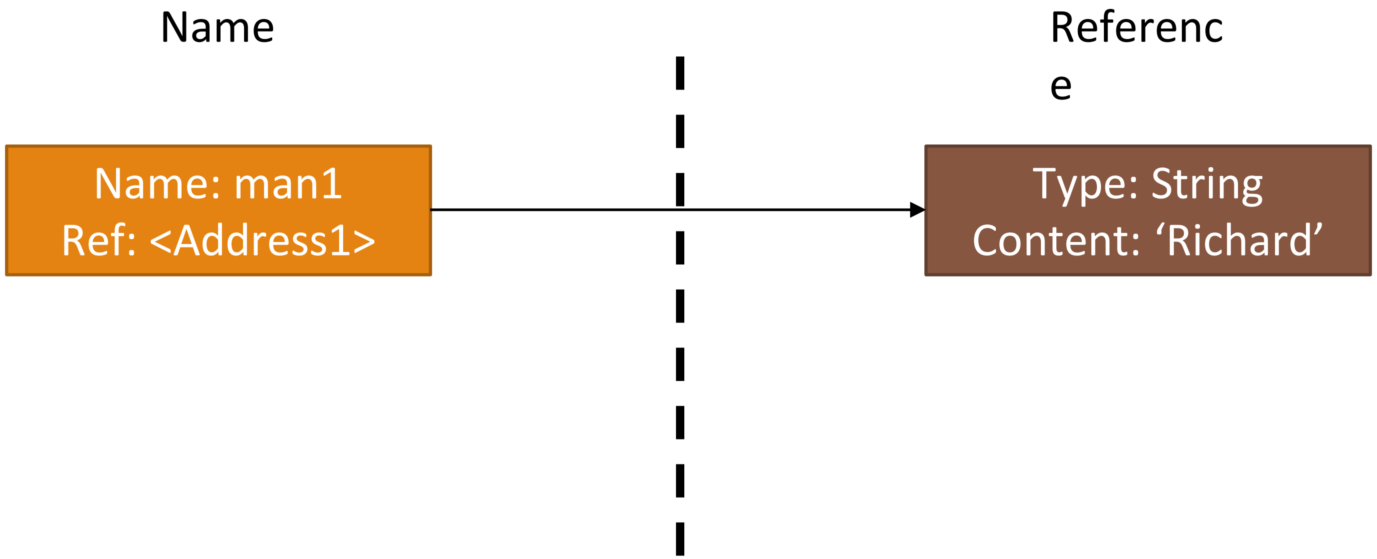
Name and Reference (1)

- ✓ Not “call by value” or “call by reference”
- ✓ But “call by assignment” or “call by object reference”

Name and Reference (2)



```
man1 = 'Richard'
```



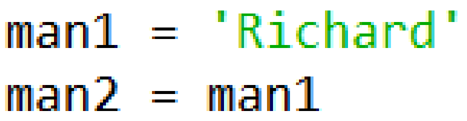
Name

Reference

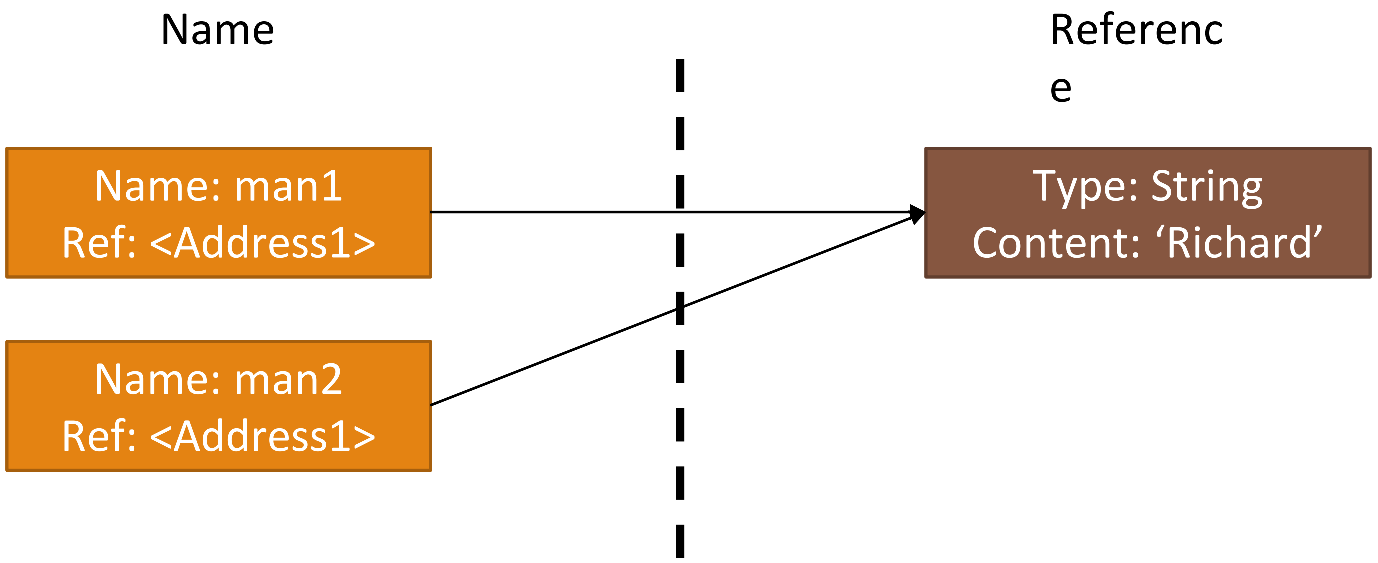
Name: man1
Ref: <Address1>

Type: String
Content: ‘Richard’

Name and Reference (2)



```
man1 = 'Richard'
man2 = man1
```



Name

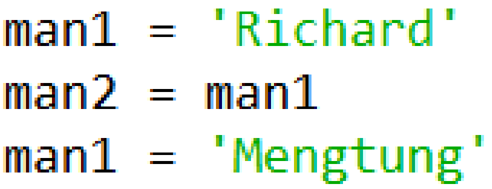
Referenc
e

Name: man1
Ref: <Address1>

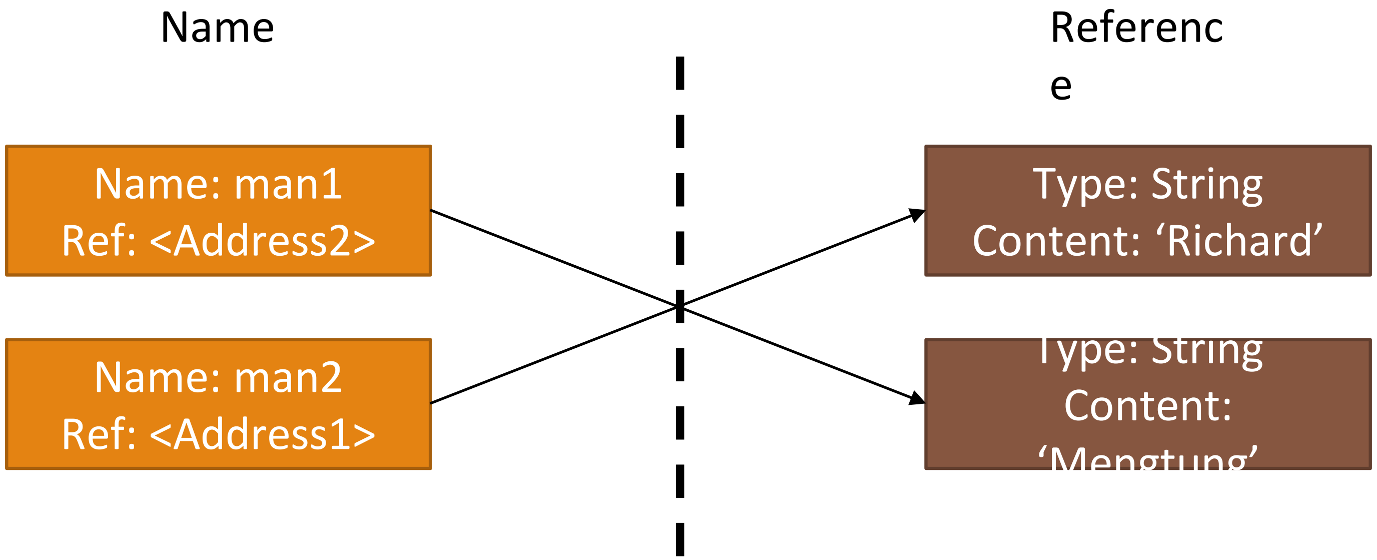
Name: man2
Ref: <Address1>

Type: String
Content: ‘Richard’

Name and Reference (2)



```
man1 = 'Richard'
man2 = man1
man1 = 'Mengtung'
```



Name

Reference

Name: man1
Ref: <Address2>

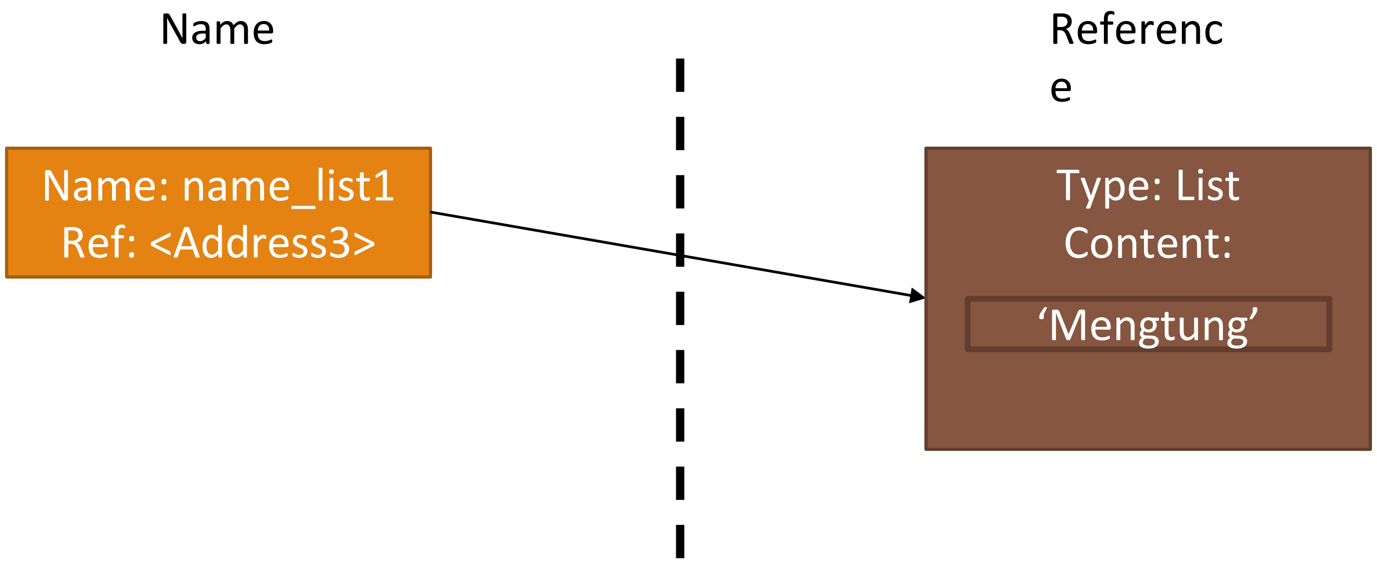
Name: man2
Ref: <Address1>

Type: String
Content: ‘Richard’

Type: String
Content:
‘Mengtung’

Name and Reference (3) – List

```
name_list1 = []
name_list1.append(man1)
```



Name

Referenc
e

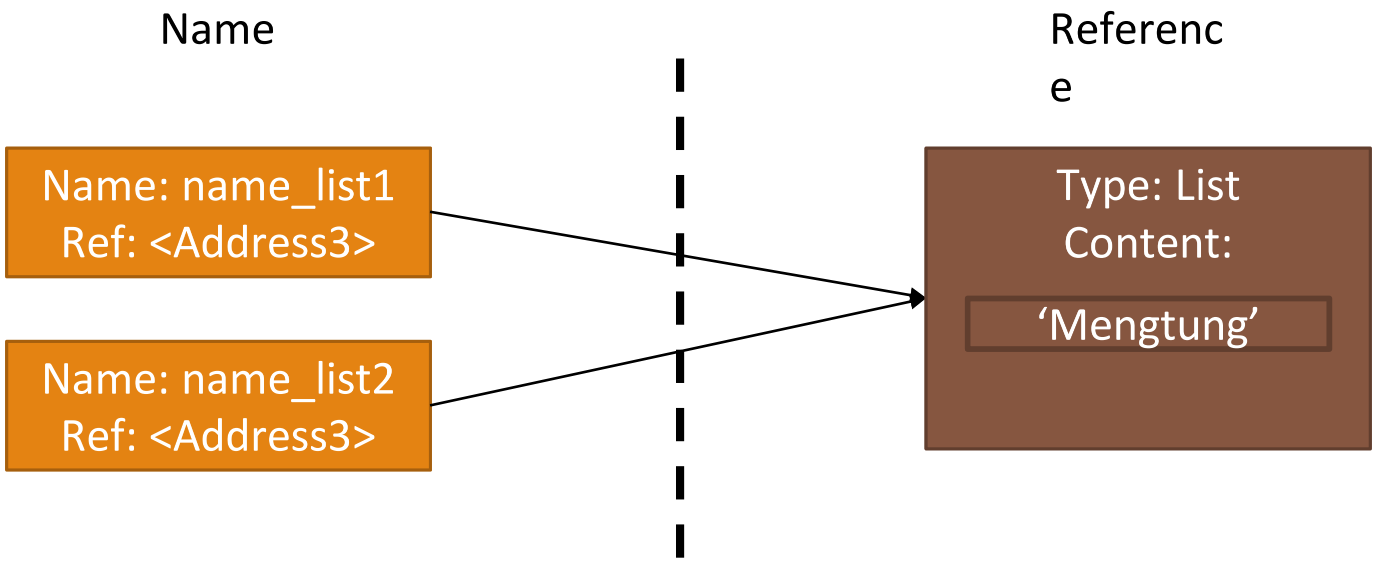
Name: name_list1
Ref: <Address3>

Type: List
Content:

‘Mengtung’

Name and Reference (3) – List

```
name_list1 = []
name_list1.append(man1)

name_list2 = name_list1
```



Name

Referenc
e

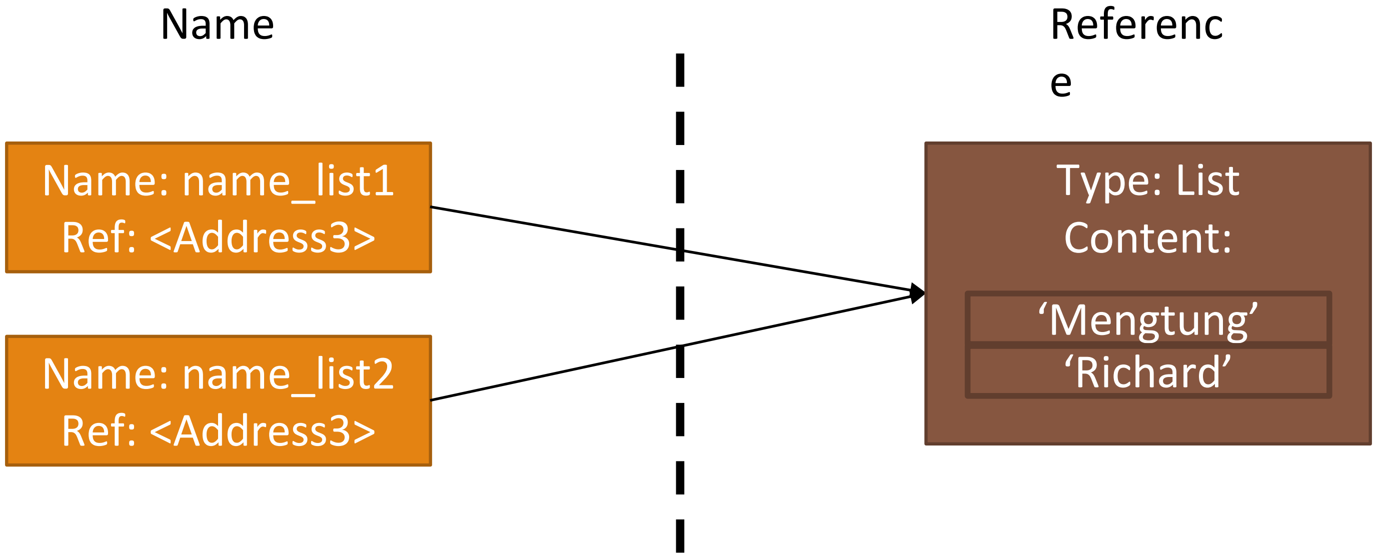
Name: name_list1
Ref: <Address3>

Name: name_list2
Ref: <Address3>

Type: List
Content:
‘Mengtung’

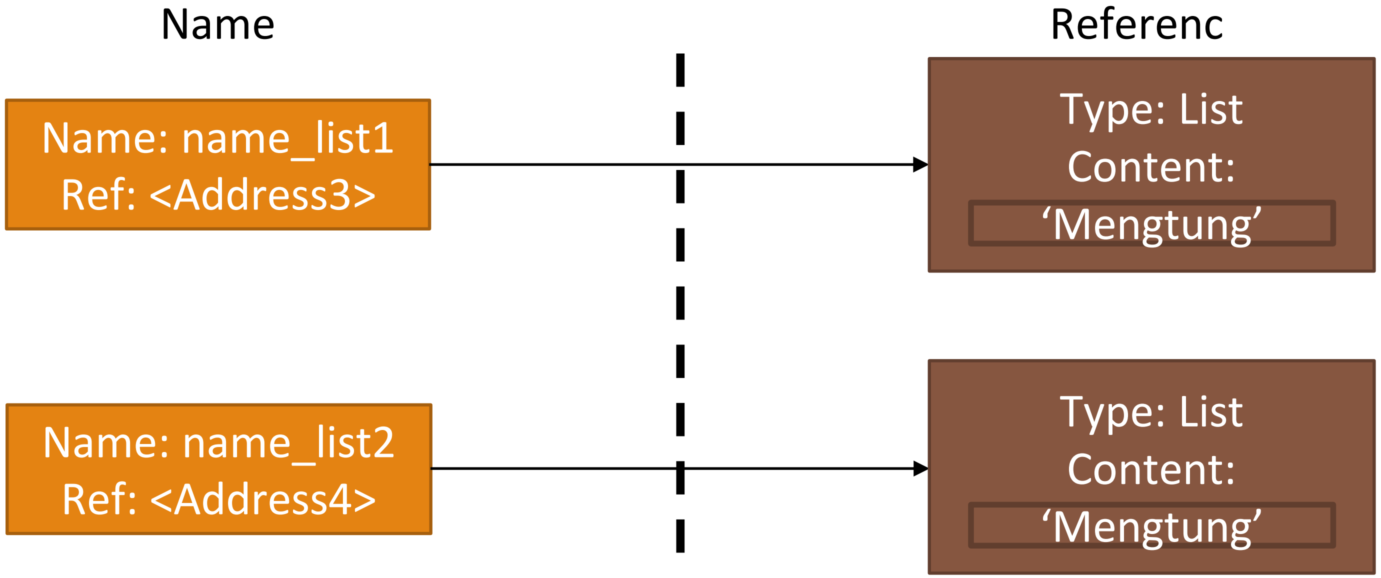
Name and Reference (3) – List

```
name_list1 = []
name_list1.append(man1)

name_list2 = name_list1
name_list2.append(man2)
```



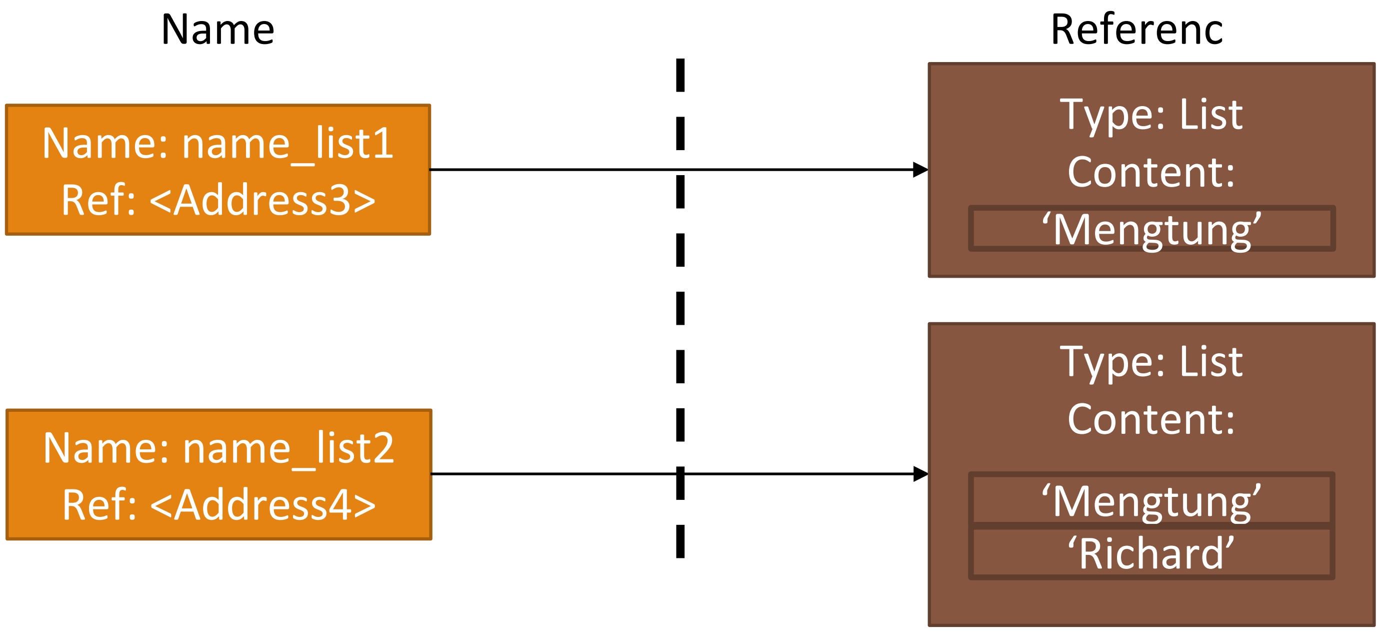
Name and Reference (4) – New List

```
name_list1 = []
name_list1.append(man1)

name_list2 = name_list1[:]
```



Name and Reference (4) – New List

```
name_list1 = []
name_list1.append(man1)

name_list2 = name_list1[:]
name_list2.append(man2)
```



Name

Referenc

Name: name_list1
Ref: <Address3>

Type: List
Content:
‘Mengtung’

Name: name_list2
Ref: <Address4>

Type: List
Content:
‘Mengtung’
‘Richard’

Reference

- Tutorial: https://docs.python.org/3.7/tutorial/index.html#the-python-tutorial
- Library: https://docs.python.org/3/library/index.html
- https://en.wikipedia.org/wiki/Matplotlib
- What is NumPy? — NumPy v1.21 Manual

Thanks for your listening