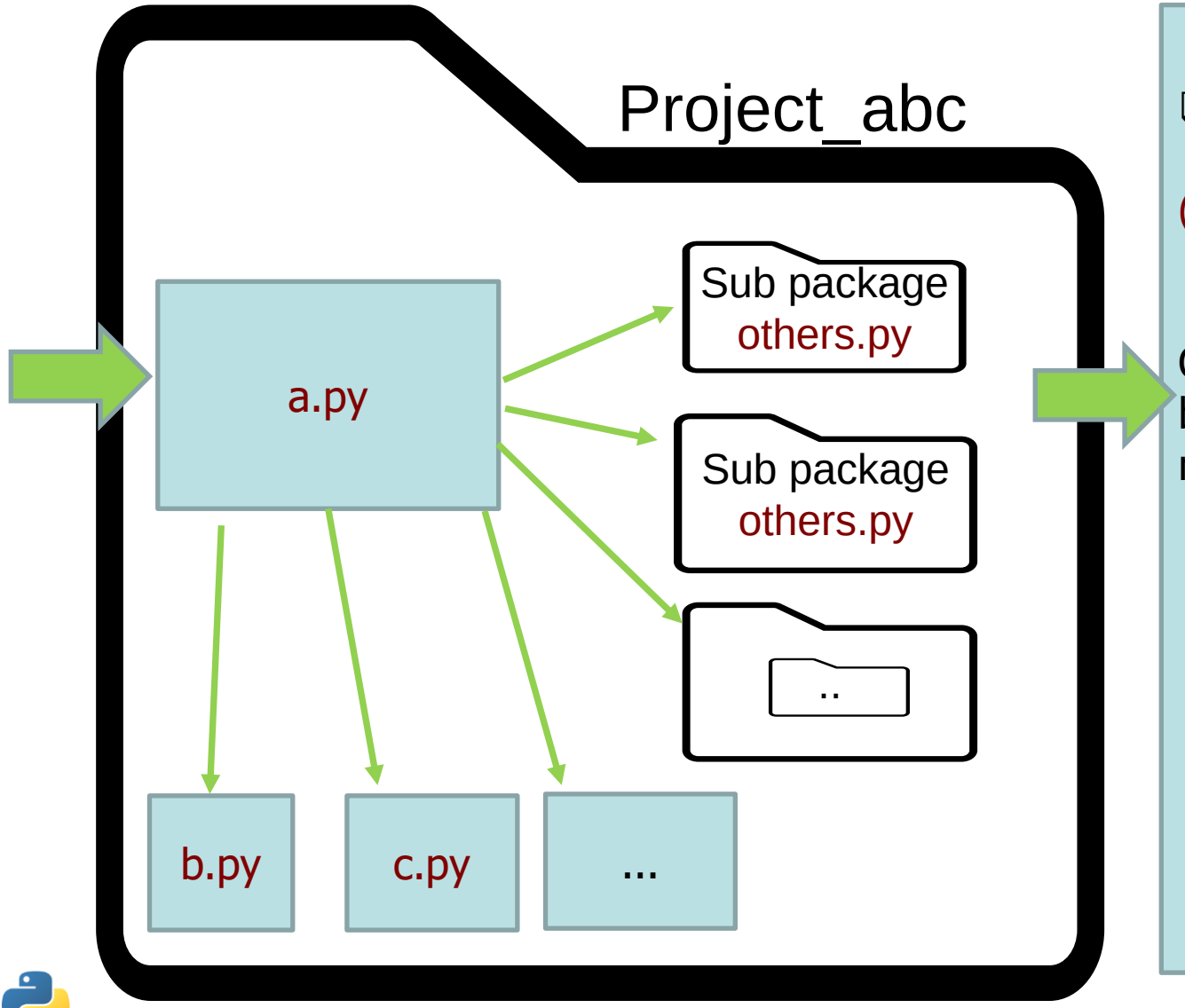
Cấu trúc căn bản chương trình

Cấu trúc chương trình

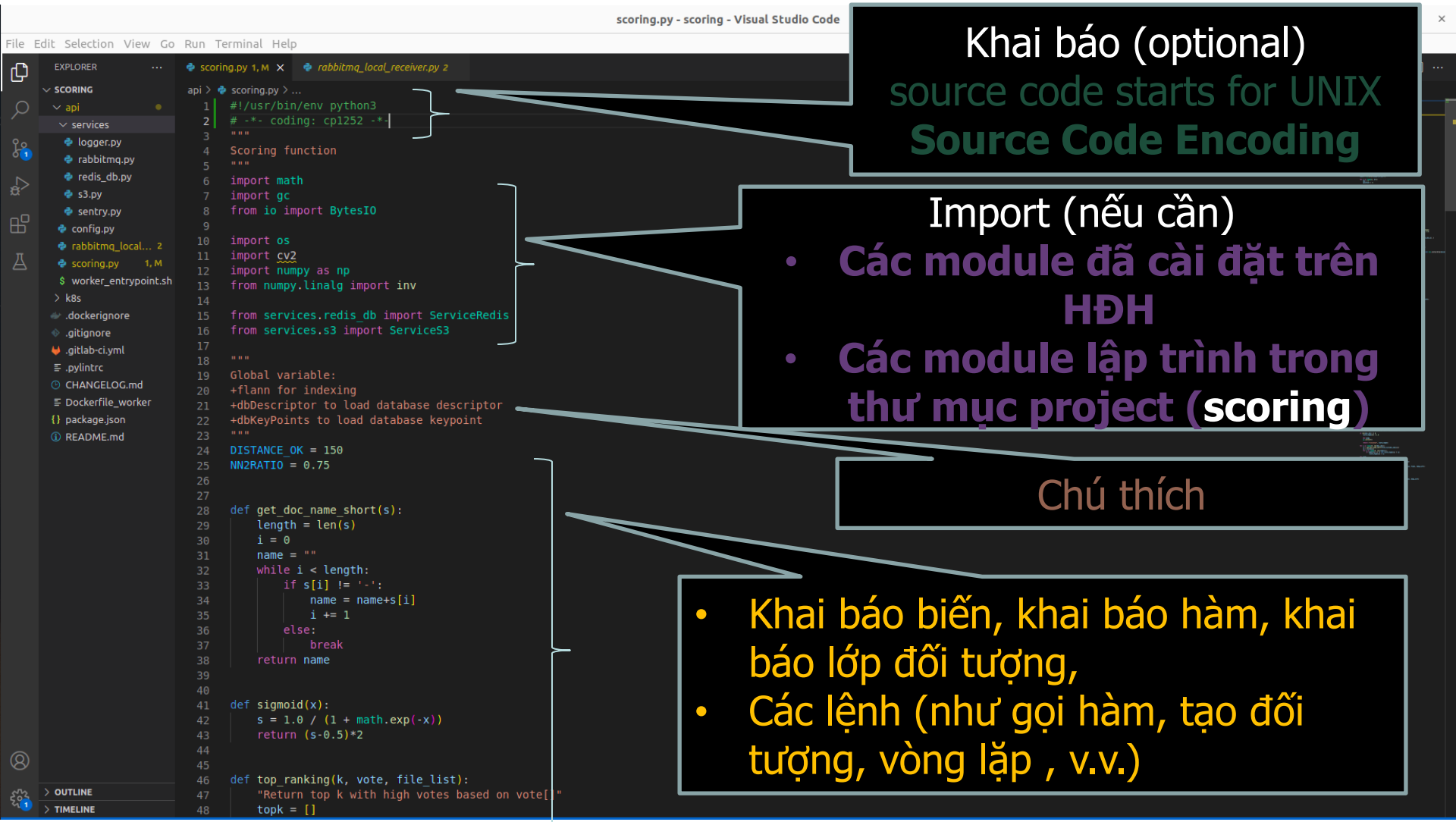


- Modules python cài đặt trong HĐH
(pip install package)

Có thể phân nhóm riêng biệt theo từng nhu cầu mỗi project

- Quản lý môi trường ảo(virtual environment)
- Tránh đụng độ phiên bản (version)
- **Anaconda, *virtualenv ...***

Cấu trúc chương trình

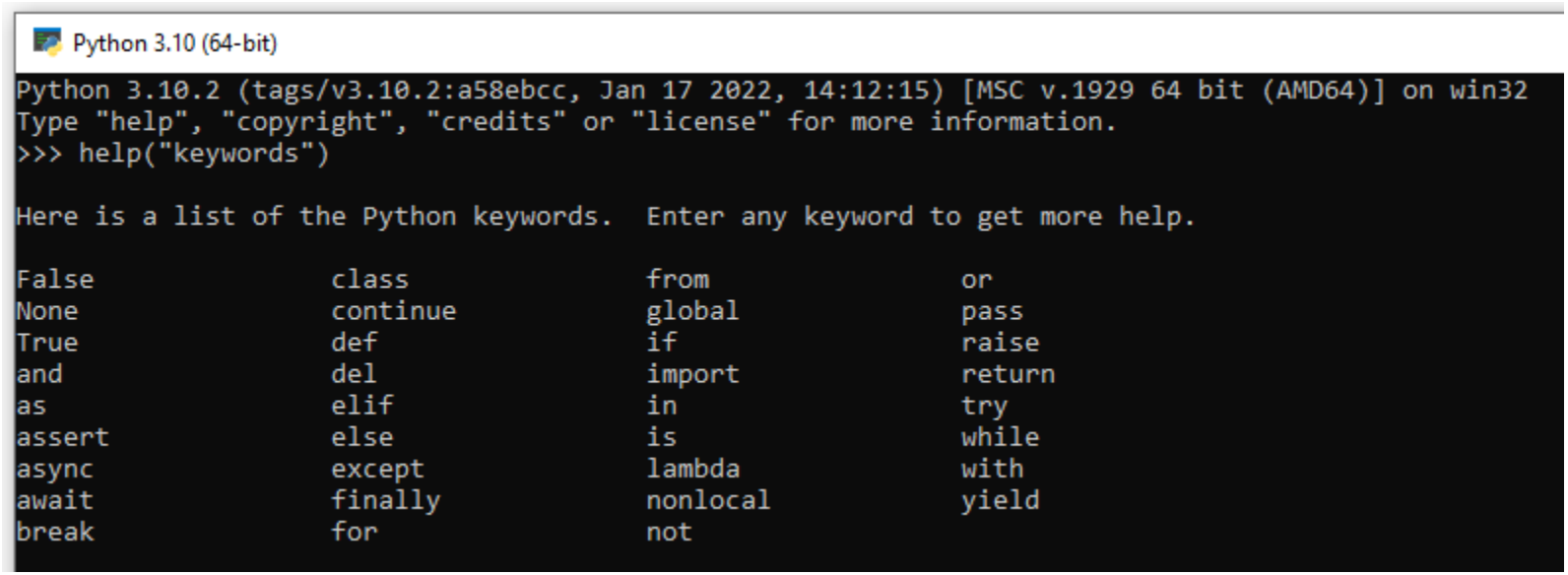


Từ khóa (keyword)

- **Từ khóa (keyword)** là những từ (word) được dành riêng trong Python
- Không thể sử dụng từ khóa để đặt tên biến, tên hàm hoặc bất kỳ định danh (identifier) nào khác
- Ngôn ngữ python phân biệt chữ hoa và chữ thường khác nhau trong định danh và từ khóa

Từ khóa (keyword)

- lệnh help("keywords") trong trình thông dịch Python để xem danh sách tất cả các từ khóa trong Python



Định danh (identifier)

- **Định danh** (identifier) là tên được đặt cho các đối tượng như lớp, hàm, biến, .v.v.

- Ký tự bắt đầu định danh phải là alphabet hoặc _

- **Không được sử dụng các ký hiệu đặc biệt** như !, @, #, **$,%,... phép toán hay khoảng trắng** trong định danh

- Phân biệt chữ in hoa và chữ thường

Định danh (identifier)

- Nên đặt tên định danh có ý nghĩa và dễ nhớ.

  Ví dụ: c = 10 thì nên đặt là count = 10.

- Các từ trong một tên định danh có thể được nối với nhau bởi dấu gạch dưới

  Ví dụ: this_is_a_long_variable

- Luôn kiểm tra tên định danh đã đặt cẩn thận tránh nhầm lẫn do gõ nhầm phím hay vô ý

  Ví dụ: count, couNt và Count là khác nhau

Câu lệnh (statement), khối lệnh

- Một statement trong Python thường được viết trong 1 dòng

```
a=5
print (a)
```

- Lệnh được viết trên nhiều dòng sử dụng ký tự \

```
total = item_one + \
                   item_two + \
                   item_three
```

- Lệnh được bao bằng các cặp dấu ngoặc: [], {}, () không cần phải sử dụng ký tự \ để tiếp tục dòng

```
days = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
```

Câu lệnh (statement), khối lệnh

- Dấu ; để cách nhiều lệnh trên dòng

  a=5; s="Xin chao"; c=8

```
1   import sys
2                                   khối lệnh chính
3   file = "data.txt"
4   try:
5       myfile = open(file, "r")
6       myline = myfile.readline()          khối lệnh thân try
7       while myline:
8           print(myline)
9           myline = myfile.readline()      khối lệnh thân while
10      myfile.close()
11  except IOError as e:
12      print("I/O error({0}): {1}".format(e.errno, e.strerror))   khối lệnh thân except
13  except: #handle other exceptions such as attribute errors
14      print("Unexpected error:", sys.exc_info()[0])              khối lệnh thân except
15  print("done")
```

Canh lề (indentation)

```
############################################################
def getAllTextLines(self):
    texts = []
    for i in range(0, len(self.textlines)):
        results_line = self.textlines[i]
        text_row = ""
        endX_Pre = -1
        for j in range(0, len(results_line)):
            startX = results_line[j][0][0]
            s = results_line[j][1]
            if self.isAmountOfNumber(s):
                first_space = s.find(" ")
                if first_space > 2:
                    s = s[0:first_space]  # remove  too far group
                s = s.replace(" ", "")
                # print(s)
            if j == 0:
                text_row = s
            elif startX <= endX_Pre:
                text_row = text_row + s
            else:
                text_row = text_row + " " + s
            endX_Pre = results_line[j][0][2]
            # print(results_line[j][1],end =" ")
            # text=results_line[j][1]
            # print(text,self.isAddress(text))
        texts.append(text_row)
        # print('')
    return texts
```

Canh lề (indentation)

- Indentation xác định một khối lệnh (code block) như thân hàm, thân vòng lặp,...
- Indentation thay cho cặp {code block } như các ngôn ngữ C/C++, Java,...
- 4 khoảng trắng được sử dụng cho thụt đầu dòng và được ưu tiên hơn các tab.
- Phải thống nhất xuyên suốt module
- Nếu sử dụng indentation không nhất quán trình thông dịch sẽ báo lỗi: IndentationError
- Để chuẩn hóa có thể dùng VS code +autopep8,black...

Ghi chú (comment)

- Sử dụng # để chú thích 1 dòng trong chương trình

```
# comment
```

- Sử dụng ''' để chú thích trên nhiều dòng

```
''' We are in a comment
We are still in a comment
'''
```

Docstring

- Sử dụng để mô tả hàm, phương thức, lớp, module và có thể truy xuất nếu cần object.__doc__
- Docstring là chuỗi ký tự xuất hiện ngay sau khi định nghĩa của một phương thức, lớp hoặc module
- Sử dụng bộ 3 **dấu nháy** “”” phần văn bản mô tả “”” để viết docstring

Docstring

example.py

```
def double(num):
    """Function to double the value"""
    return 2*num
```

Chúng ta có thể truy cập docstring với thuộc tính **___doc___**. **Ví dụ**:

example.py

```
def double(num):
    """Function to double the value"""
    return 2*num
print(double.__doc__)
```

Kết quả

```
Function to double the value
```

Biến và kiểu dữ liệu

Data types

Kiểu	Từ khóa
Text Type	str
Numeric Types	int, float, complex
Sequence Types	list, tuple, range
Mapping Type	Dict
Set Types	set, frozenset
Boolean Type	bool
Binary Types	bytes, bytearray, memoryview
None Type	NoneType

Example	Data Type	Ghi chú
x = "Hello World"	str	Độ dài tùy vào
x = 20000000000000000000000000000000000000	int	Dung lượng RAM
x = 20.5	float	
x = 1j	complex	
x = ["apple", "banana", "cherry"]	list	
x = ("apple", "banana", "cherry")	tuple	
x = range(6)	range	
x = {"name" : "John", "age" : 36}	dict	
x = {"apple", "banana", "cherry"}	set	
x = frozenset({"apple", "banana", "cherry"})	frozenset	
x = True	bool	
x = b"Hello"	bytes	
x = bytearray(5)	bytearray	
x = memoryview(bytes(5))	memoryview	
x = None	NoneType	

Biến (Variables)

- **Một biến (variable)** là một một **vùng nhớ** (trên RAM) lưu trữ dữ liệu trong chương trình
- Vùng nhớ có địa chỉ bắt đầu, có thể thu hồi và giải phóng bởi bộ thu gom rác (Garbage Collector): module gc
- Biến phải đặt tên theo quy tắc định danh
- Biến sinh ra (khai báo, khởi tạo) bằng cú pháp (assignment syntax):
  - <Tên_biến> = <giá trị>
- Phải có ý thức tiết kiệm bộ nhớ khi sử dụng biến lưu dữ liệu lớn, ví dụ giải phóng khi không còn dùng đến

Biến (Variables)

- Biến không cần khai báo
- Kiểu dữ liệu biến **tự động xác đinh qua giá trị gán**
- Biến có thể thay đổi kiểu dữ liệu khi gán giá trị kiểu mới
- **type(tên_biến)** ➔kiểu dữ liệu của biến
- **id(tên_biến** ➔ địa chỉ của biến
- **del tên_biến** ➔ xóa biến và giải phóng vùng nhớ
- Module **gc** ➔ cung cấp công cụ để lập trình viên quản lý biến, ví dụ để giải phóng tất các các biến rác.

```
gc.collect()
```

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

Biến (Variables)

```
x = 5
y = "John"
print(x)
print(y)
z=12222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222212
print(z % 2)
```

==========➔

```
5
John
0
```

##########

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

==========➔

```
Orange
Banana
Cherry
```

Ví dụ

```
ten = 10
twenty = 20
thirty = 30
forty = 40
fifty = 50
sixty = 60
seventy = 70
eighty = 80
ninety = 90
Total = ten + twenty + thirty + forty + \
    fifty + sixty + seventy + eighty + ninety
print("Print Total numeric value : ", Total)

==============================➔
Print Total numeric value :  450
```

Ví dụ

```
ten = " 10 "
twenty = " 20 "
thirty = " 30 "
forty = " 40 "
fifty = " 50 "
sixty = " 60 "
seventy = " 70 "
eighty = " 80 "
ninety = " 90 "

Total = ten + " " + twenty + " " + thirty + " " + forty + " " + \
    fifty + " " + sixty + " " + seventy + " " + eighty + " " + ninety
print("Print Total text value : ", Total)
=========================================➔
Print Total text value :   10   20   30   40   50   60   70   80   90
```

Ép kiểu (casting) theo mong muốn

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

example.py

```
a = 1.1 + 2.2
print(a)
print("a == 3.3?", (a==3.3))
```

➔ `abs(a-3.3)<=0.0000001`

Kết quả

```
3.3000000000000003 a == 3.3? False
```

Hàm xác định kiểu

```
x = 5
y = "John"
z=float(x)
t=str(x)
u=["Year",2022]
v={"hocphan":"Python","tc":2}
print(type(x))
print(type(y))
print(type(z))
print(type(t))
print(type(u))
print(type(v))
=========➔
<class 'int'>
<class 'str'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'dict'>
```

String value

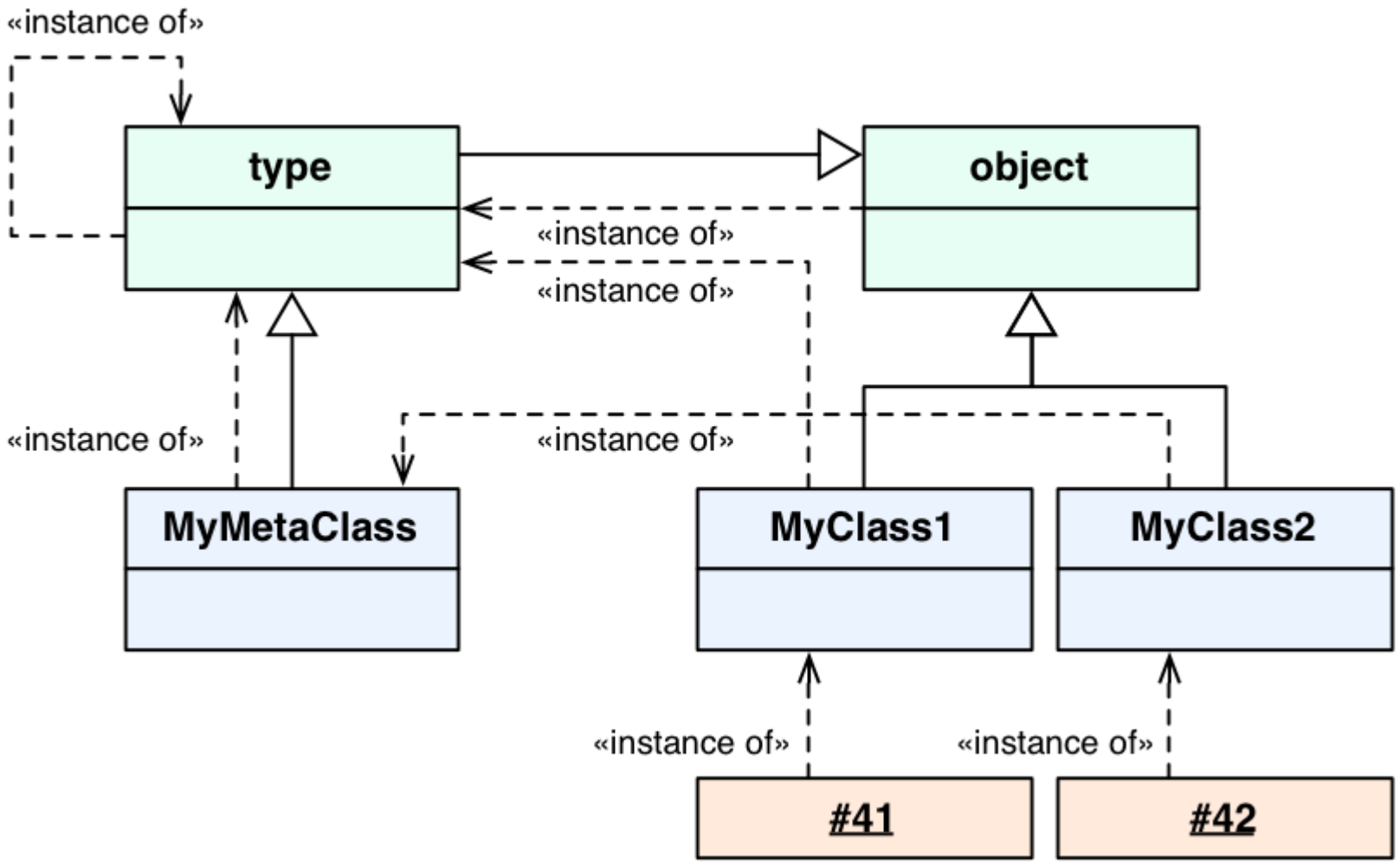
```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive

- ```
  a = 4
  A = "Sally"
  #A will not overwrite a
  ```

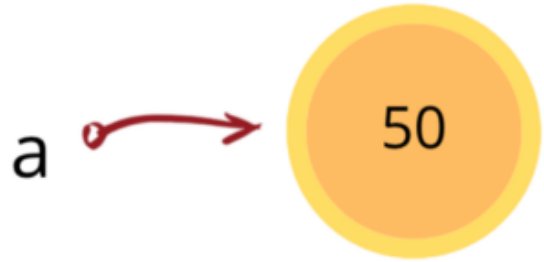

  ```

Everything is an object



Địa chỉ biến

```
a = 50
```



PYTHON OBJECT

type	integer
value	50
reference count	1

Mutable and Immutable Objects

mutable Objects	Immutable Objects
❑ lists, dictionaries, sets, bytearray, and etc.	❑ integers, float, complex, strings, tuple, frozensets, and bytes.
❑ Địa chỉ tham chiếu (id) không đổi khi đối tượng thay đổi giá trị hay mở rộng giá trị	❑ Địa chỉ thay đổi khi đối tượng được gán giá trị mới (cấp phát vùng nhớ mới)

```
list1=["Year",2022]
list2=list1
list1.append(1000.0)
print(list1)
print(list2)
==============➔
['Year', 2022, 1000.0]
['Year', 2022, 1000.0]
```

```
s1="Python"
s2=s1
s1=s1+" 3.10"
print(s1)
print(s2)
====================➔
Python 3.10
Python
```

Mutable and Immutable Objects

Mutable Objects	Immutable Objects
• Khi cần bản sao mà không làm ảnh hưởng đối tượng, tạo ô nhớ mới, có phương thức copy()	• Không có phương thức copy

```
list1=["Year",2022]
list2=list1.copy()
list1.append(1000.0)
print(list1)
print(list2)
==================>
['Year', 2022, 1000.0]
['Year', 2022]
```

```
s1="Python"
s2=s1.copy()



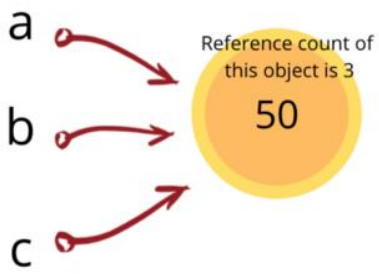

==================>
AttributeError: 'str' object has no attribute 'copy'
```

Hàm xác định địa chỉ

```
a = 50
b = a
c = 50
print(id(a))
print(id(b))
print(id(c))
```

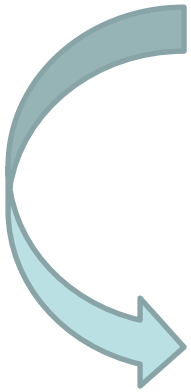
```
4364962480
4364962480
4364962480
```

There is 1 object, 3 names and 3 references!



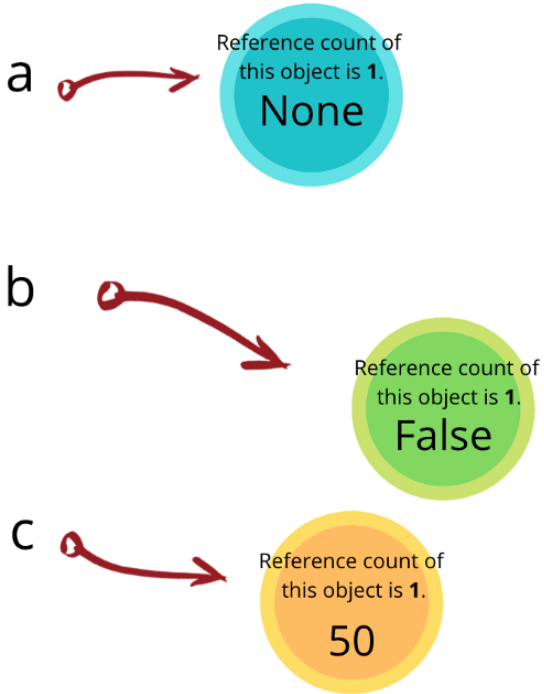

```
print(a is b)
print(c is b)
print(a is c)
```

```
True
True
True
```



```
a = None
b = False
print(id(a))
print(id(b))
print(id(c))
```

```
4364658792
4364558736
4364962480
```



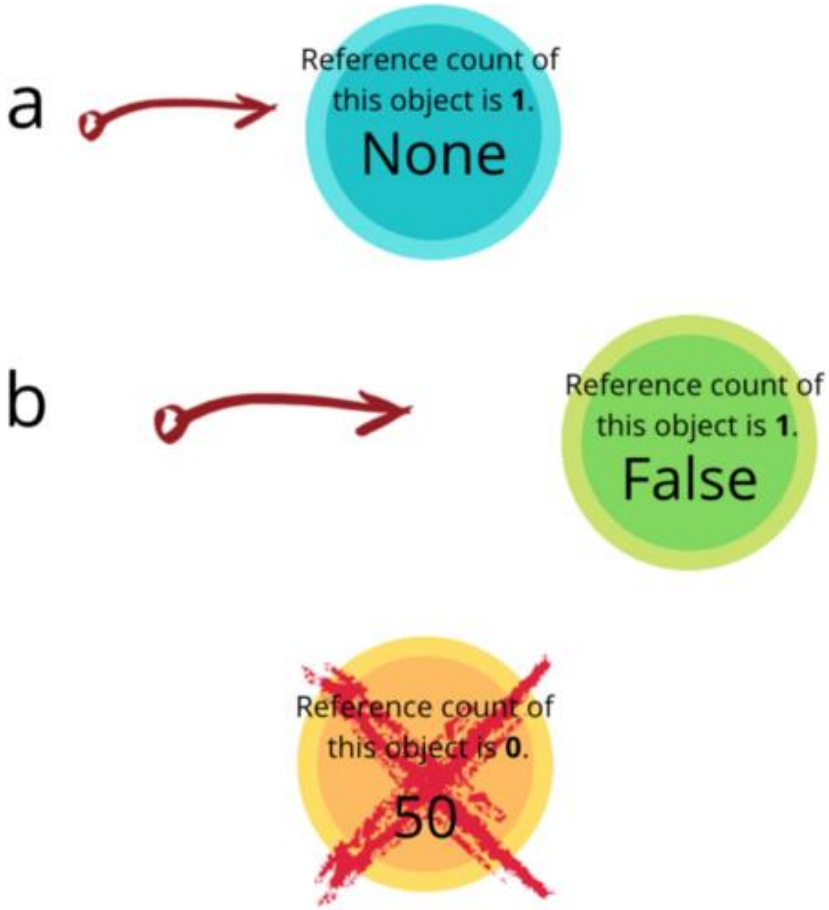
Hàm xác định địa chỉ

```
del(c)
print(id(a))
print(id(b))
print(id(c))
```

```
4364658792
4364558736


---------------------------------
NameError
<ipython-input-98-e81976fdfa9f> in
      2 print(id(a))
      3 print(id(b))
----> 4 print(id(c))

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



Phép toán == khác is

- Phép so sánh == so sánh giá trị
- Phép toán `is` để xét hai đối tượng cùng tham chiếu đến một địa chỉ.

  `x is y` tương đương `id(x) == id(y)`

Ví dụ:

```
>>> dish = ["rice", "eggs"]
>>> id(dish)
3009428812360
>>> container = dish
>>> id(container)
3009428812360
>>> container == dish
True
>>> container is dish
True
>>> id(container) == id(dish)
True
>>> bowl = ["rice", "eggs"]
>>> id(bowl)
3009428812808
>>> bowl == dish
True
>>> bowl is dish
False
>>> id(bowl) == id(dish)
False
```

Q & A

Thank you!