

OBJECT-ORIENTED PYTHON

Contents

1. Giới Thiệu	1
2. Các Class và các đối tượng	3
3. Khai báo Class	5
4. Khởi tạo 1 class	6
5. Tạo các Methods	8
6. Hiểu “self” là gì	10
7. Tạo 1 method có chấp nhận đối số	13
8. Các thuộc tính và Init Method	15
9. Tạo Method append	19
10. Tạo và cập nhật các thuộc tính	21

Tên	MSSV
Tô Gia Huy	20139003
Trần Nguyễn Quang Lâm	20139040

1. Giới thiệu

Lập trình hướng chức năng là viết code theo nhiều bước, kết hợp các bước đó thành các lệnh gọi là function = chức năng

-Kiến thức cần biết:

-Lập trình hướng đối tượng tương quan với khoa học dữ liệu như nào

-Các khái niệm chính: class, instance(một cá thể của class), attribute(thuộc tính) và method(phương pháp)

-Cách tạo một class

Khi làm việc với data, người ta thường dùng kiểu lập trình mà gần với lập trình hướng chức năng hơn là lập trình hướng đối tượng, nhưng ta phải hiểu OOP (lập trình hướng đối tượng) làm việc như nào, vì Python là ngôn ngữ lập trình hướng đối tượng.

-> Đối với lập trình hướng đối tượng, các đối tượng có các kiểu khác nhau, mỗi kiểu là một class, các class đó là:

- String class
- List class
- Dictionary class

Làm một bài khởi động nhanh, sử dụng hàm type() để trả về kiểu dữ liệu của biến:

- Dùng hàm `print()` để hiển thị kiểu dữ liệu của list `l` đã khai báo
 - Dùng hàm `print()` để hiển thị kiểu dữ liệu của string `s` đã khai báo
 - Dùng hàm `print()` để hiển thị kiểu dữ liệu của dictionary `d` đã khai báo
- Code:

 fix.py - C:/Users/CCLaptop/Document

File Edit Format Run Options V

```
l=[1,2,3]
s='string'
d={'a':1,'b':2}
print(type(l))
print(type(s))
print(type(d))
```


Kết quả:

```
>>>
===== RESTART: C:\I
<class 'list'>
<class 'str'>
<class 'dict'>
>>> |
```

Giải thích : class là tập hợp các đối tượng có tính chất giống nhau, ở đây các class được ghi trong output là các kiểu đối tượng `list[]`, `string ''`, `dictionary {}`
Hàm `type(variable)` sẽ trả về kiểu dữ liệu của biến đã truyền vào
nên `print type()` các biến đã truyền vào sẽ trả về kiểu dữ liệu của các kiểu đó

2. Classes and Objects

Function `type()` trên màn hình ở trên (phần 1), nó sẽ trả về kiểu dữ liệu được dán nhãn class

 re_2.py

```
1 class 'list'
2 class 'str'
3 class 'dict'
```

Việc này chứng tỏ cách chúng ta sử dụng hàm “type” và “class” thay thế cho nhau. Cho thấy rằng việc chúng ta đã sử dụng classes từ lâu rồi.

- Python lists là các đối tượng của class list
- Python strings là các đối tượng của str class
- Python dictionaries là các đối tượng của dict class

Chúng ta sẽ học cách làm việc của classes bằng cách tạo class của chính mình. Chúng ta sẽ tạo 1 class đơn giản được gọi là `MyList` và tạo lại 1 vài hàm đơn giản của list

class

Như chúng ta đã đề cập ở trên, nó ít phổ biến cho khoa học dữ liệu và phân tích dữ liệu để định nghĩa 1 kiểu dữ liệu mới, nhưng để hiểu cách các đối tượng làm việc sẽ rất hữu ích cho ta tăng hiểu biết và làm việc nhiều hơn với các đối tượng. Học về sự khác biệt tinh tế giữa các đối tượng và các biến. Nhưng bây giờ, hãy nhìn vào mối quan hệ giữa các đối tượng và các lớp.

Object là 1 thực thể lưu trữ data

Object's class định nghĩa đặc tính cụ thể mà objects của class sẽ có. Một cách để hiểu sự khác biệt giữa class và object trong Python là so sánh chúng với đối tượng của thế giới thực. Chúng ta sẽ so sánh object string với xe điện Tesla



Hàng trăm nhìn xe Tesla- khác với Ford or Toyota- Nhưng trong 1 thời điểm, chúng ta không nhất thiết phải phân biệt chúng với nhau. Ta nói, các xe là Objects thuộc class Tesla.

Tesla đã thiết kế xe của họ. Những thiết kế thuộc định nghĩa xe từ trước đến giờ, công dụng của xe và cách nó di chuyển- Tất cả là đặc điểm của 1 chiếc xe. Nhưng thiết kế đó không phải là xe, nó chỉ là những đặc điểm cần thiết để tạo ra 1 chiếc xe. Tương tự, Python, chúng ta tạo ra các class. Những thiết kế này là của định nghĩa chúng ta về class chúng ta cần.

3. Khai báo Class

Khai báo một class cũng tương tự như khai báo một hàm

*fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Wee

File Edit Format Run Options Window Help

```
def my_function():  
    #  
    #  
def MyClass:  
    #  
    #
```

Trên đây là code khai báo 2 class, ta có thể thấy đoạn code bên dưới khai báo không có dấu ngoặc đơn

-> *Khai báo class không cần ngoặc đơn*

Quy tắc đặt tên class:

- +) Chỉ dùng chữ cái, chữ số, gạch chân
- +) Không dùng khoảng cách, kí tự đặc biệt
- +) Không bắt đầu bằng chữ số

2 đoạn code trong hình ảnh đầu bài khai báo 2 class khác nhau, nếu ta sử dụng 2 class đó sẽ có lỗi `SyntaxError` trả về, lí do là vì class chỉ mới được khai báo chưa có method

-> *Không định nghĩa khai báo được các class trống, trả về lỗi `SyntaxError`*

-> *Dùng `pass` để tránh lỗi này*

Code:

```
File Edit Format Run  
class Statement:  
    pass
```

Kết quả:

```
>>> ===== RESTART: C:/U  
>>> |
```

Lệnh `pass` rất hữu dụng khi ta khai báo một class mà chưa có method và muốn khai báo các method sau.

Khai báo một class tên `MyClass` và dùng lệnh `pass` để tránh lỗi `SyntaxError`

```
File Edit Format Run Options Window Help
class MyClass:
    pass
test=MyClass()

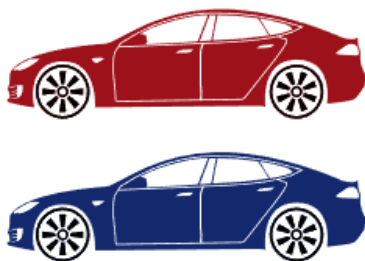
IDLE Shell 3.10.6

File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4l
AMD64)] on win32
Type "help", "copyright", "credit
>>>
===== RESTART: C:/Users/CCLap
>>> |
```

- > Đoạn code khai báo một class trống và dùng lệnh pass để tránh lỗi SyntaxError
- > Code chạy bình thường

4. Khởi tạo 1 class

Ở trên chúng ta đã so sánh các đối tượng với các xe Tesla để giúp ta hiểu sự khác biệt giữa class và object. Hãy tiếp tục làm tương tự để giúp ta hiểu hơn về các class. Trong OOP, chúng ta sử dụng **Ví Dụ** cụ thể để miêu tả sự khác nhau mỗi đối tượng



These cars are two **instances** of the Tesla **class**.

While each is unique – for example, one is red and one is blue – they are clearly the same type of car.

Tesla Illustration: Roman Vorobiov from the Noun Project

Đây là 2 xe ví dụ của class Tesla.

Trong khi 2 xe đều độc nhất- Ví dụ, 1 màu đỏ và 1 màu xanh-Nhưng chúng là 2 dạng xe khác biệt

>>2 xe khác màu- cùng kiểu- Như 2 strings- giữ 2 giá trị khác nhau nhưng hoạt động giống nhau

string_1= “The first string”

string_2= “The second string”

- - - Các Đối tượng này là 2 ví dụ thuộc str class trong Python

Trong khi chúng độc nhất- chúng chứa các giá trị khác nhau- **Nhưng** chúng cùng kiểu đối tượng

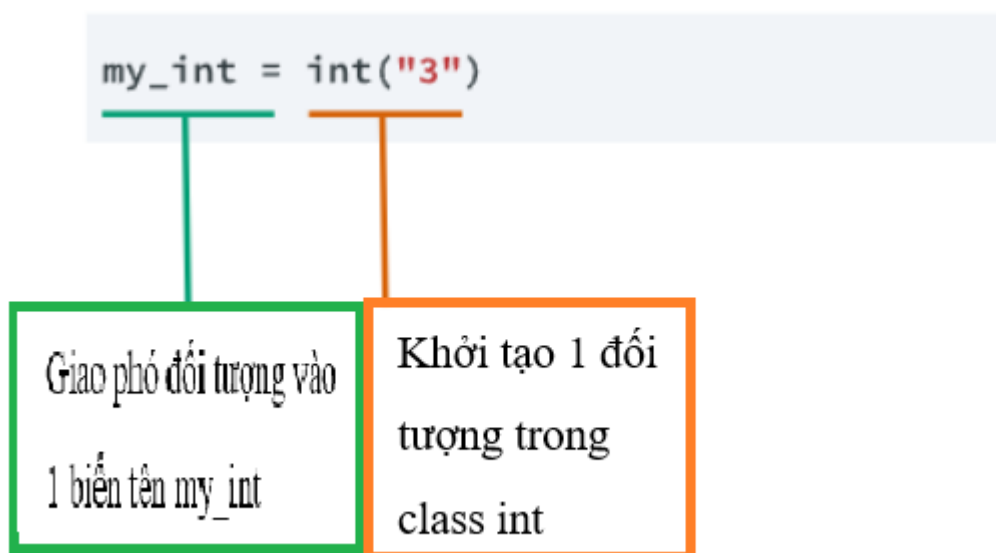
Khi chúng ta định nghĩa class, ta tạo 1 đối tượng (object) thuộc class đó. Nếu ta tạo 1 object của 1 class cụ thể, cụm từ kỹ thuật cho những gì bạn làm là “Khởi tạo 1 đối tượng của class”. **Cách để khởi tạo 1 object trong Class:**

```
my_class_instance = ExampleClass()
```

Dòng trên làm 2 việc:

- Khởi tạo 1 object trong Example class
- Biến chứa object là **my_class_instance**

Để minh họa rõ hơn, Hãy nhìn vào ví dụ sử dụng integer class trong python cú pháp `int()` để chuyển đổi giá trị số được lưu trữ dưới dạng chuỗi đến số nguyên.. Đọc từ trái sang phải



Thông thường ta dùng object và biến thay thế cho nhau. Sự khác biệt thường chỉ quan trọng khi ta nói về OOP Classes

MyClass ở dưới

1. Sử dụng `MyClass()` để khởi tạo biến `MyClass`. Gán nó cho biến `my_instance`
2. Dùng `print()` và `Type()` để print kiểu dữ liệu của `my_instance`

```
MyClass.py > ...
4 class MyClass:
5     pass
6 my_instance= MyClass()
7 print(type(my_instance))
```

Result: in ra kiểu dữ liệu của `my_instance` là Class

```
<class '__main__.MyClass'>
```

5. Tạo các Methods

->Methods có thể hiểu là cách thức thực hiện một hành động trong class, method cũng có thể hiểu là một hàm đặc biệt trong class

Tương tự với đoạn giới thiệu ở trên về Tesla, một đối tượng của class Tesla có thể thực hiện mở khóa và khởi động. Chuỗi của Python cũng có các method có để thực hiện các chức năng với chuỗi

Hiểu methods như là một hàm đặc biệt và nó thuộc về một class cụ thể. VD: ta gọi method `replace` với cú pháp `str.replace()` vì method này thuộc về class `str`

Mỗi class đều có các method của riêng nó, vd:

```
fix.py - C:/Users/CCLaptop/Doi
File Edit Format Run Optic
my_string='hello'
my_list=[1,2,3]
my_list.append(4)
print(my_list)
```

Kết quả xuất ra sẽ là :

```
>>>
===== RESTART:
[1, 2, 3, 4]
>>>
```

>>Class list có method `append`: thêm phần tử vào vị trí cuối của playlist

>>List ban đầu `[1,2,3]` , sau khi gọi method `append(4)` list thành `[1,2,3,4]`

Code:


```
File Edit Format Run Options Window Help
my_string='hello'
my_list=[1,2,3]
my_list.append(4)
print(my_list)
my_string=my_string.replace('h','H')
print(my_string)
|
```

Kết quả:

```
===== REST
[1, 2, 3, 4]
Hello
>>>
```

Class string có method replace() thay thế một phần tử tại một vị trí đã cho
string ban đầu = 'hello'
string.replace('h','H')='Hello'

-Ta không thể gọi method từ class khác với class của đối tượng ta đang sử dụng.

>>Cú pháp khởi tạo một method gần giống như khởi tạo một hàm, chỉ có ngoại lệ là phải khai báo trong định nghĩa class:

```
File Edit Format Run Options Wind
class ExampleClass:
    def greet():
        return 'hello'
```

Các bước khai báo method:

- Bỏ lệnh pass đã khai báo trước đó
- Khai báo **method first_method()**
- Trong method trả về chuỗi **"This is my first method"**
- Bên ngoài class, khai báo một biến, biến này thuộc về class **MyClass** nên nó được coi là một bản thể của MyClass, gán nó với tên **my_instance**

->Đoạn code ban đầu là class trống, dùng lệnh pass để tránh lỗi SyntaxError

```
1 File Edit Format Run Opt
class MyClass:
    pass
|
```

->Thay pass trong class MyClass cũ bằng khai báo method mới, trả về dòng chữ


‘This is my first method’

->Gán biến my_instance cho kết quả mà class trả về

->Code chạy và biến my_instance có giá trị là ‘This is my first method’

-Kết quả trả về là vị trí ô nhớ lưu giá trị của biến

```
class MyClass:
    def first_method():
        return "This is my first method"
my_instance=MyClass()
print(my_instance)
```

 IDLE Shell 3.10.6

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug  1 2022,
AMD64)] on win32
Type "help", "copyright", "credits" or "license()"
>>>
===== RESTART: C:/Users/CCLaptop/Documents/py
<__main__.MyClass object at 0x0000023583F47010>
>>> |
```

6. Hiểu “self” là gì.

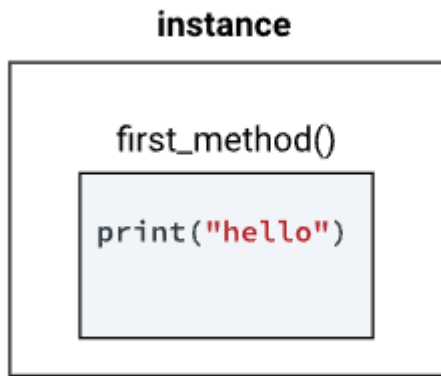
Ta đã định nghĩa 1 class ở trên với 1 method đơn giản. Sau đó ta đã tạo biến tên my_instance thuộc class đó:

```
-----
TypeError                                Traceback (most recent call last)
ipython-input-44-9e10ffed0a3f in module()
----> 1 my_instance.first_method()

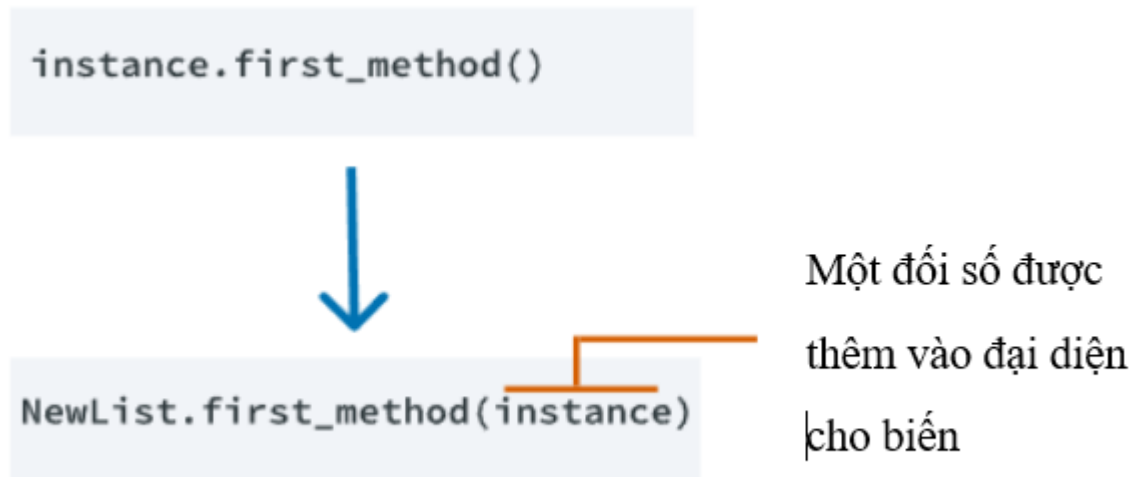
TypeError: first_method() takes 0 positional arguments but 1 was
given
```

>>my_instance.first method() được gọi, sẽ trả về 1 data, ta cần 1 biến để lưu trữ, nên chương trình đã bị lỗi do có dữ liệu trả về nhưng không có biến nào để lưu dữ liệu đó

Lỗi này gây bối rối. Nó nói 1 đối số đã được đưa vào first_method(), nhưng khi ta gọi method, chúng ta không cung cấp bất kỳ 1 đối số nào. Có vẻ như là 1 đối số “ma” đã được đưa vào đâu đó. Để hiểu, Ta sẽ nhìn vào những gì diễn ra khi ta gọi method. Chúng ta sẽ bắt đầu nhìn vào my_instance object chứa đựng 1 method đơn:



Khi gọi `first_method()` thuộc về object `my_instance`, Python chạy cú pháp đó và thêm vào 1 đối số đại diện cho phiên bản mà chúng ta đang gọi:



Ta xác định bằng các sử dụng hàm được built-in str type. Dùng `str.title()` để chuyển đổi 1 str thành chuỗi viết hoa

```
string.py > ...
1  s= "MY STRING"
2  r= str.title(s)
3  e= s.title()
4  print(r)
5  print(e)
```

result: chạy được các method trong class str

```
My String
My String
```

Python thêm đối số, đối số đó có thể trả về chính nó, đây là điểm gây ra lỗi ở phía

trên. Dưới đây sẽ là phần chứng minh đối số thêm là object cho chính nó.
We'll start by doing the following:

```
example.py > ...  
1 class example:  
2     def print_self(self):  
3         print(self)  
4 mc = example()
```

- định nghĩa example class với print_self method.
- khởi tạo 1 object thuộc class và gán vào biến mc

```
print(mc)
```

```
<__main__.example object at 0x000002A17FFAD400>
```

Khi ta gọi print_self() method sẽ ra output tương tự như print chính đối tượng

```
mc.print_self()
```

```
<__main__.example object at 0x000002C0F075D400>
```

>>>Output giống nhau khi sử dụng 2 cú pháp print(mc) và sử dụng print_self()- điều đó chứng tỏ đối số “ma” chính là đối tượng

Self là quy ước để class trở nên dễ hiểu. Bây giờ, chúng ta sẽ sửa đổi code ở trên và thêm self làm đối số cho phương thức.

- Sửa first_method() bằng cách thêm self
- Tạo ví dụ thuộc MyClass. Giao phó cho biến my_instance
- Gọi my_instance.first_method. gán cho biến result

```

p.py > ...
1  class MyClass:
2      def first_method(self):
3          return "this is my first method"
4      def append(self, new_item):
5          self.data=self.data +[new_item]
6          self.length+=1
7  my_instance= MyClass()
8  result=my_instance.first_method()
9  print(result)

```

result:

```

this is my first method

```

7. Tạo một Method có chấp nhận đối số

-Đối số **self** thực chất là chính đối tượng

-> Tạo một method với **self** là đối số truyền vào đầu tiên và chuỗi sẽ là đối số thứ 2, kết quả trả về là một chuỗi

```

*fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Week2\
File Edit Format Run Options Window Help
class ExampleClass:
    def return_string(self, string):
        return string

```

Tạo một đối tượng và gọi đến method đã khai báo, khi gọi method này ta bỏ qua đối số **self**

fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Week2\fix.py (3.

```
File Edit Format Run Options Window Help
class ExampleClass:
    def return_string(self, string):
        return string

mc = ExampleClass()
result=mc.return_string("Hey there!")
print(result)
```

Kết quả xuất ra:

```
.> |
    | ===== RES
    | Hey there!
.> |
```

>> Sau khi định nghĩa class và method của nó, ở trong chương trình chính khi ta gọi đến method đó, không cần phải truyền vào self, chỉ cần truyền vào các đối số cần thiết khác

-> Trong class MyClass, khai báo method mới tên return_list() với 2 đối số truyền vào

+ self: bản thân method đang được khai báo

+ input_list: list đầu vào

-> Triển khai return_list để nó trả về kết quả của input_list

-> Tạo một biến để gán class đã tạo, tên my_instance

-> Gọi method return_list với lệnh my_instance.return_list(), gán kết quả trả về vào biến result

fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Week2\fix.py (3.10.6)

```
File Edit Format Run Options Window Help
class MyClass:
    def first_method(self):
        return "This is my first method"
    def return_list(self, input_list):
        return(input_list)

my_instance=MyClass()
result=my_instance.return_list([1,2,3])
```


->Code chạy bình thường và trả về biến result là kết quả của class MyClass đã khai báo ở trên
result:

```
[1, 2, 3]
```

8. Các thuộc tính và Init method

Sức mạnh của Objects(đối tượng) là khả năng lưu trữ dữ liệu bằng các **thuộc tính**

Lấy lại VD Tesla, 1 đối tượng thuộc Tesla có các thuộc tính như màu, pin, động cơ- Tương tự, Python strings có các thuộc tính- Dữ liệu được lưu trong chuỗi :

	Type Class	Properties Attributes
	Tesla	<ul style="list-style-type: none">• Màu xanh• Thời gian sử dụng pin• Hỗ trợ Tự động lái• Động cơ tiêu chuẩn
<pre>string_1 = "This is a string"</pre>	Python String	<ul style="list-style-type: none">• Chứa giá trị “this is the string”

Tesla Illustration: Roman Vorobiov from the Noun Project

Bạn có thể nghĩ các thuộc tính như là biến đặc biệt thuộc về 1 class cụ thể
Các thuộc tính cho ta lưu các giá trị đặc biệt về 1 biến thuộc class

```
my_int = int("3")
```

>>init method là một method đặc biệt mà nó sẽ chạy ngay khi chúng ta tạo 1 biến, vậy nên chúng ta có thể tạo nhiều nhiệm vụ khi ta set up 1 biến

Khởi tạo Init method bằng cách :**def __init__(self):**

```

init.py > example > __init__
1 class example:
2     def __init__(self, string):
3         print(string)
4 mc = example("Hola!")
5

```

Hola!

>> chương trình in “Hola!” ngay khi class chỉ vừa khởi tạo.

__init__ method thường được dùng để khởi tạo data.

```

init.py > ...
1 class example:
2     def __init__(self, string):
3         self.my_attribute = string
4 mc = example("Hola!")
5

```

Khi ta tạo đối tượng, Python sẽ gọi Init method thông qua đối tượng đó

```
mc = MyClass("Hola!")
```



Chạy init method
với cú pháp

```
MyClass.__init__(mc, "Hola!")
```

Code chưa đưa ra output nhưng, bây giờ “Hola!” được lưu trong thuộc tính attribute trong đối tượng.


```

class example:
    def __init__(self, string):
        self.my_attribute = string
mc= example("Hola!")
print(mc.my_attribute)

```

Hola!

Bảng tóm tắt sự khác nhau giữa thuộc tính và method

	mục đích	giống	cú pháp
attribute	lưu trữ data	biến	object.attribute
method	thực hiện 1 vài hoạt động	hàm	object.method()

```

init_2.py > ...
1  class MyList:
2      def __init__(self, i):
3          self.data = i
4  my_list=MyList([1,2,3,4])
5  print(my_list.data)

```

Result:

[1, 2, 3, 4]

>>Giải thích: my_list khởi tạo bằng hàm __init__ của class. Hàm __init__ sẽ được gọi ngay khi class được gọi và khởi tạo thuộc tính data bằng i.

9. Tạo Method Append

->Tái tạo hàm append trong class list thay cho hàm có sẵn trong built-in, bắt đầu với việc quan sát hàm append trong built-in hoạt động

Code:

```
fix.py - C:\Users\CCLaptop\Documents\
File Edit Format Run Options Wir
my_list=[1,2,3,4]
my_list.append(5)
print(my_list)
```

Kết quả:

```
>>>
===== RESTART: C:\U
[1, 2, 3, 4, 5]
>>>
```

->Method append trong built-in chấp nhận một đối số, không trả về bất kì giá trị nào

->Cách thức hoạt động của method này có thể hiểu là thêm một phần tử vào list, ta có thể thực hiện bằng cách cho phần tử mới là một list, cộng list ban đầu vào list phần tử mới, kết quả sẽ giống với method append

```
fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Week
File Edit Format Run Options Window Help
my_list=[1,2,3,4]
new_item=4
new_item_list=[new_item]
my_list=my_list+new_item_list
print(my_list)
```

```
>>>
===== RESTART: C:\Use
[1, 2, 3, 4, 4]
>>>
```

Kết quả:

Khi đã hiểu cơ chế method append hoạt động, ta bắt đầu xây dựng method mới tương tự:

->Với class MyList được cho từ bài trước, khai báo method append mới, với 2 đối số truyền vào


+self: bản thân hàm được gán

+new item: input được thêm vào list

->Triển khai append để list lưu trong self.data có thêm phần tử mới, new item

->Sau khi khai báo class xong, tạo một biến tham chiếu đến MyList với giá trị [1,2,3,4,5], tên biến là my_list

->In giá trị của my_list.data (gọi method append)

 fix.py - C:\Users\CCLaptop\Documents\pYTHON_PR\Week2\fix.py (3.10.6)

```
File Edit Format Run Options Window Help
class MyList:
    def __init__(self, initial_data):
        self.data=initial_data

    def append(self, new_item):
        self.data=self.data+[new_item]

my_list= MyList([1,2,3,4,5])
print(my_list.data)

my_list.append(6)
print(my_list.data)
|
```

->Class có thêm method mới : append

+self.data=self.data + [new_item] : thêm phần tử mới vào list khởi tạo từ đầu

->Khởi tạo biến my_list với class MyList, giá trị list là [1,2,3,4,5]
in ra màn hình giá trị my_list.data(phương thức append)

```
===== RESTART: C:/User
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
KQ: >>> |
```

10. Creating and Updating an Attribute

Tóm tắt các thứ đã học

- Tạo MyList class lưu list ngay khi khởi tạo sử dụng init constructor
- Lưu list đó trong attribute của MyList gọi là data
- Tạo method append để thêm phần tử vào cuối mảng

Tạo 1 class. Khi ta muốn xem độ dài của List, ta sử dụng len(). Tạo 1 thuộc tính là mylist.length, thứ sẽ lưu độ dài của list. Ta có thể làm khi sử dụng Init method

```

class MyClass:
    def __init__(self, initial_data):
        self.data = initial_data
        self.length=0
        for item in self.data:
            self.length+=1
    def append(self, new_item):
        self.data=self.data +[new_item]

```

Sử dụng Append():

```

my_list=MyClass([1,2,3,4,5])
print(my_list.length)
my_list.append(8)
print(my_list.length)

```

5

5

>>Sau khi thêm phần tử “8” vào list, nhưng attribute length vẫn không tăng lên 6.
Do chúng ta định nghĩa attribute length ở init.

>>Để sửa lỗi này, ta sẽ tính toán length sau khi chỉnh sửa data.

```
class MyClass:
    def __init__(self, initial_data):
        self.data = initial_data
        self.length=0
        for item in self.data:
            self.length+=1
    def append(self, new_item):
        self.data=self.data +[new_item]
        self.length+=1

my_list=MyClass([1,2,3,4,5])
print(my_list.length)
my_list.append(8)
print(my_list.length)
```

Result: đã đúng với những gì ta muốn.

```
5
6
```

Giải thích: ta thêm 1 lệnh là `self.length+= 1` ngay khi ta `append` 1 phần tử mới, từ đó, attribute `length` sẽ được cập nhật mỗi khi có phần tử mới.