**OBJECT-ORIENTED PYTHON**

Contents

[1. Giới Thiệu 1](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.30j0zll)

[2. Các Class và các đối tượng 3](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.1fob9te)

[3. Khai báo Class 5](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.3znysh7)

[4. Khởi tạo 1 class 6](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.2et92p0)

[5. Tạo các Methods 8](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.tyjcwt)

[6. Hiểu “self” là gì 10](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.3dy6vkm)

[7. Tạo 1 method có chấp nhận đối số 13](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.1t3h5sf)

[8. Các thuộc tính và Init Method 15](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.4d34og8)

[9. Tạo Method append 19](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.2s8eyo1)

[10. Tạo và cập nhật các thuộc tính 21](https://docs.google.com/document/d/1yaiT5T0Vvi58bl3eoLmrGkQifuhJdcHM/edit#heading=h.17dp8vu)

|  |  |  |
| --- | --- | --- |
| Tên | MSSV | Phần việc |
| Tô Gia Huy | 20139003 | Chẵn |
| Trần Nguyễn Quang Lâm | 20139040 | Lẻ |

# 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 tình 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 classs

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

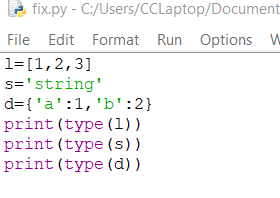
Làm một  bài khởi động nhanh, sử dụng hàm type() để trả về kiểu dữ liệu của biến:

-Dùng hàm print() để hiển thị kiểu dữ liệu của list *l* đã khai báo

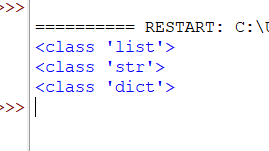
-Dùng hàm print() để hiển thị kiểu dữ liệu của string *s* đã khai báo

-Dùng hàm print() để hiển thị kiểu dữ liệu của dictionary *d* đã khai báo

Code:



Kết quả:



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(varieble) 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

# Classes and Objects

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



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

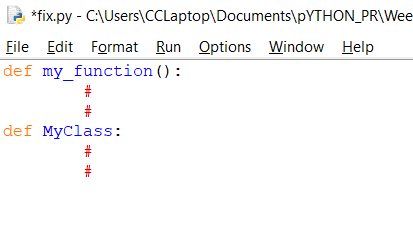


Hàng trăm nhìn xe Tesla- khác với Ford or Toyota- Nhưng trong 1 thời điểm, chúng ta không nhất thiết phải phân biệt chúng với nhau. Ta nói, các xe là Objects thuộc class Tesla.

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.

# Khai báo Class

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



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

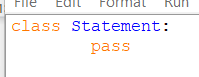
+)Không bắt đầu bằng chữ số

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

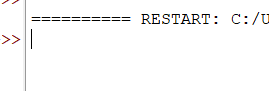
*->Không định nghĩa khai báo được các class trống, trả về lỗi SyntaxError*

*-> Dùng pass để tránh lỗi này*

Code:

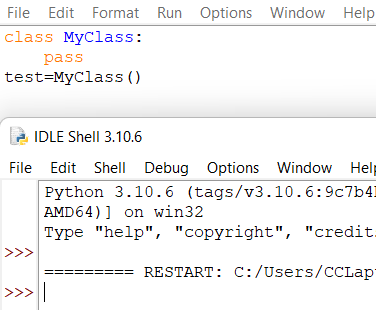


Kết quả:



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



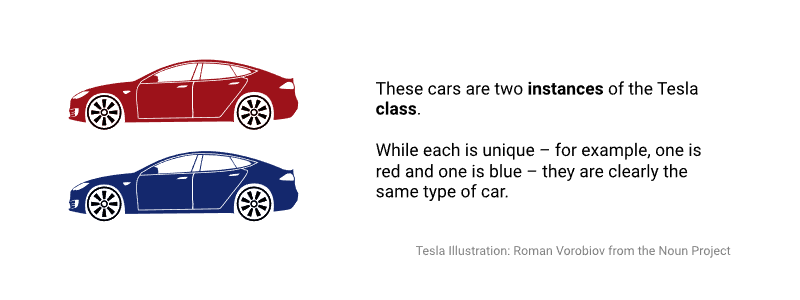
-> Đoạn code khai báo một class trống và dùng lệnh pass để tránh lỗi SyntaxError

-> Code chạy bình thường

# 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 sử khác nhau mỗi đối tượng



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

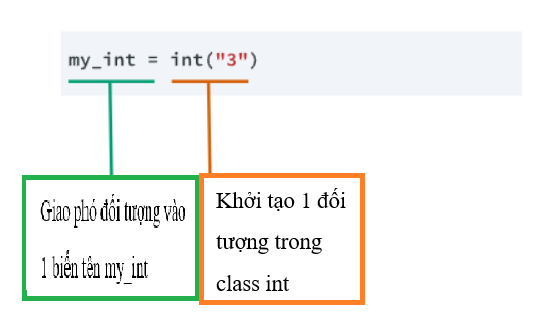
Khi chúng ta định nghĩa class, ta tạo 1 đối tượng (object) thuộc class đó. Nếu ta tạo 1 object của 1 class cụ thể, cụm từ kỹ thuật cho những gì bạn làm là “Khởi tạo 1 đối tượng của class”. Cách để khởi tạo 1 object trong Class:



Dòng trên làm 2 việc:

* Khởi tạo 1 object trong Example class
* Biến chứa object là my\_class\_instance

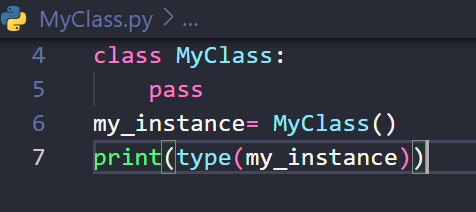
Để minh họa rõ hơn, Hãy nhìn vào ví dụ sự dụng integer class trong python cú pháp int() để chuyển đổi giá trị số được lưu trữ dưới dạng chuỗi đến số nguyên.. Đọc từ trái sang phải



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



Result: in ra kiểu dữ liệu của my\_instance là Class



# Tạo các Methods

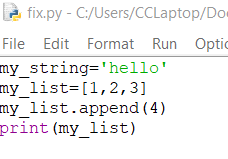
*->Methods có thể hiểu là cách thức thực hiện một hành động trong class, method cũng có thể hiểu là một hàm đặc biệt trong class*

Tương tự với đọan 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

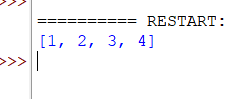
.

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:



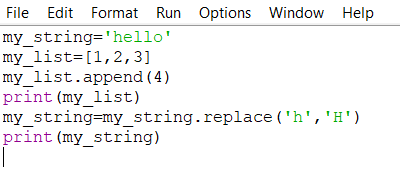
Kết quả xuất ra sẽ là :



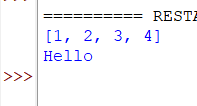
>>Class list có method append: thêm phần tử vào vị trí cuối của playlist

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

Code:



Kết quả:



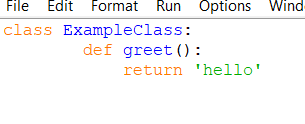
Class string có method replace() thay thế một phần tử tại một vị trí đã cho

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:*



Các bước khai báo method:

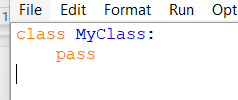
-Bỏ lệnh pass đã khai báo trước đó

-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

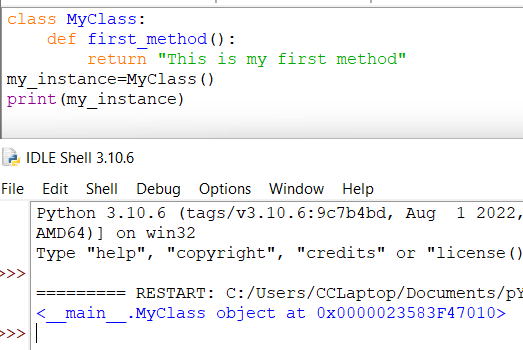


->Thay pass trong class MyClass cũ bằng khai báo method mới, trả về dòng chữ ‘This is my first method’

->Gán biến my\_instance cho kết quả mà class trả về

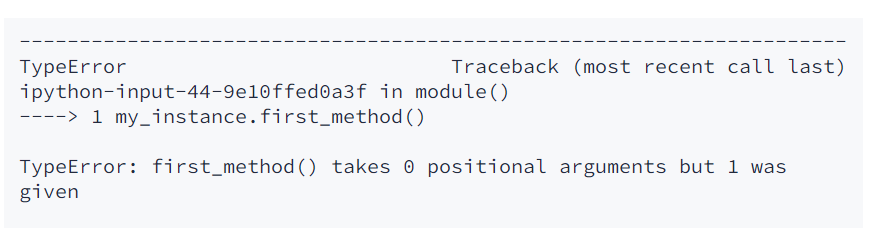
->Code chạy và biến my\_instance có giá trị là ‘This is my first method’

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



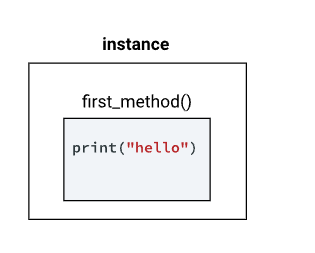
# Hiểu “self” là gì.

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:



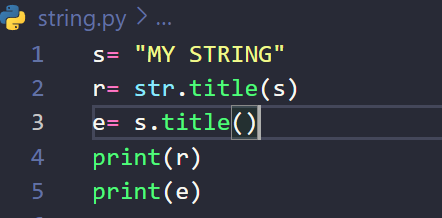
>>my\_instance.first method() được gọi, sẽ trả về 1 data, ta cần 1 biến để lưu trữ, nên chương trình đã bị lỗi do có dữ liệu trả về nhưng không có biến nào để lưu dữ liệu đó

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

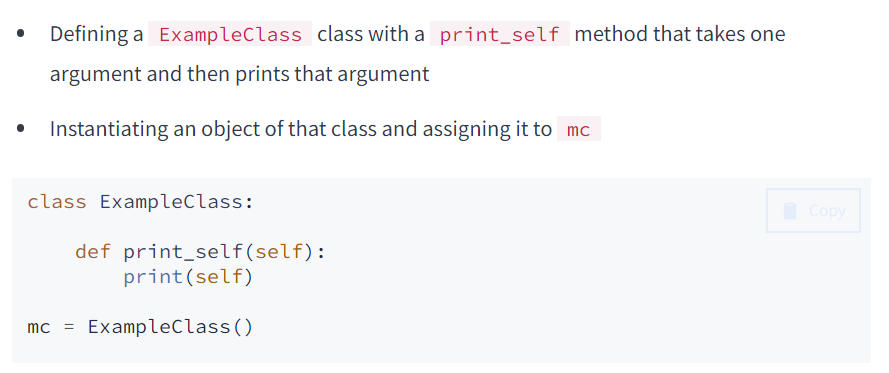


result: chạy được các method trong class str



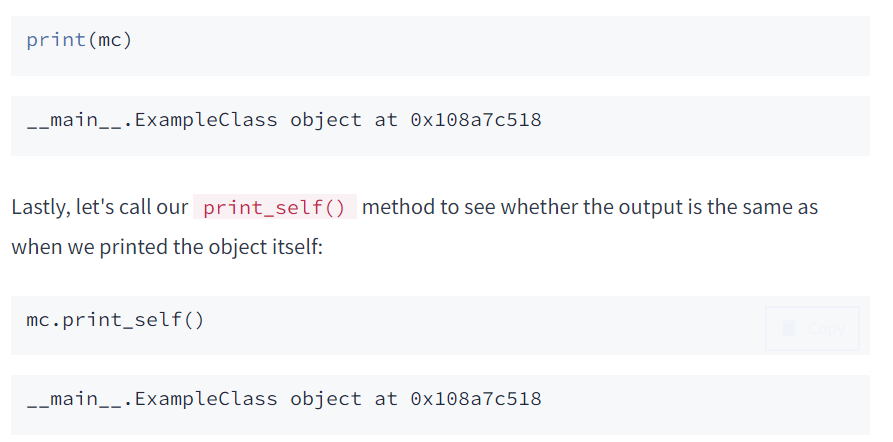
>>Python thêm đối số, đối số đó có thể trả về chính nó, đây là điểm gây ra lỗi ở phía trên. Dưới đây sẽ là phần chứng minh đối số thêm là object cho chính nó.

We'll start by doing the following:



* định nghĩa example class với print\_self method.
* khởi tạo 1 object thuộc class và gán vào biến mc

Next, let's print the mc object so we can understand what the object itself looks like when it prints:



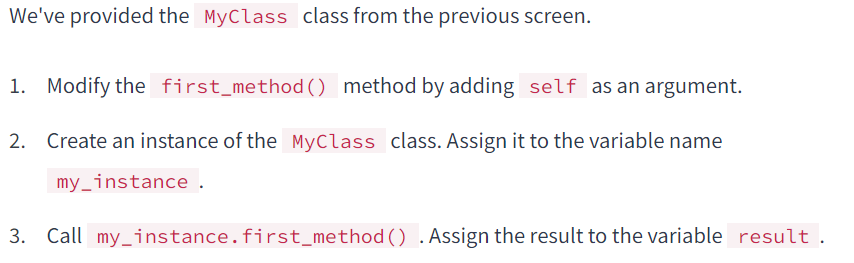
The same output displayed both when we printed the object using the syntax print(mc) and when we printed the object inside the method using print\_self() — which proves that this "phantom" argument is the object itself!

>>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

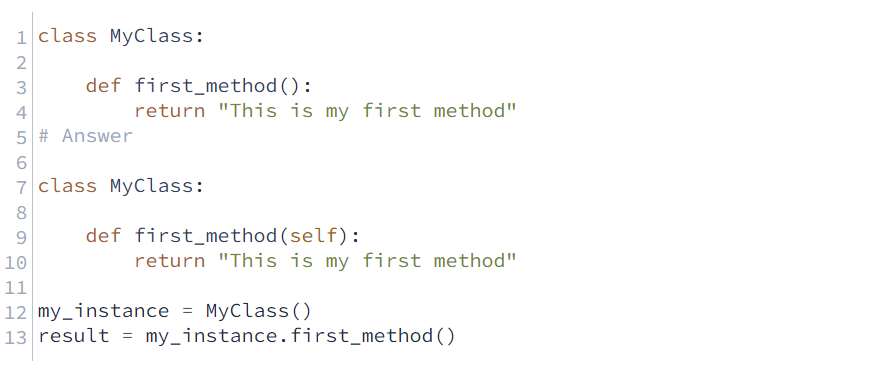
Technically, we can give this first argument — which passes to every method — any parameter name we like. However, the convention is to call the parameter self. This is an important convention because, without it, class definitions can become confusing.

Let's modify the class we created on the previous screen by adding self as an argument to our method. Then, let's call the method to make sure it runs without error.

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



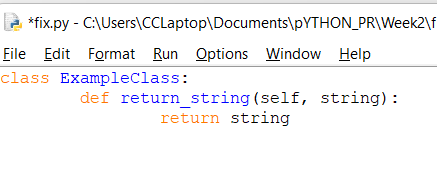
result:



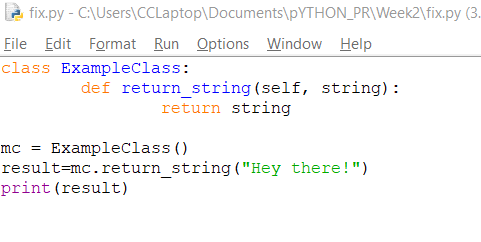
# Tạo một Method có chấp nhận đối số

-Đối số self thực chất là chính đối tượng

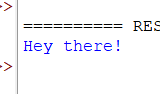
-> Tạo một method với self là đối số truyền vào đầu tiên và chuỗi sẽ là đối số thứ 2, kết quả trả về là một chuỗi



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



Kết quả xuất ra:



*>>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*

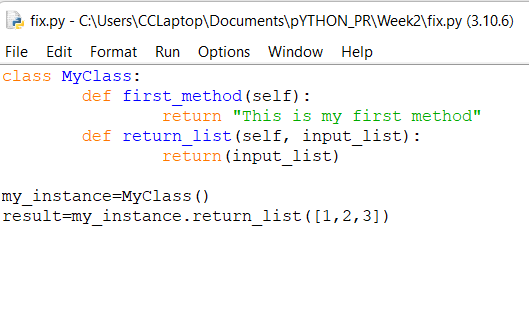
->Trong class MyClass, khai báo method mới tên return\_list() với 2 đối số truyền vào

* 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



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

result:



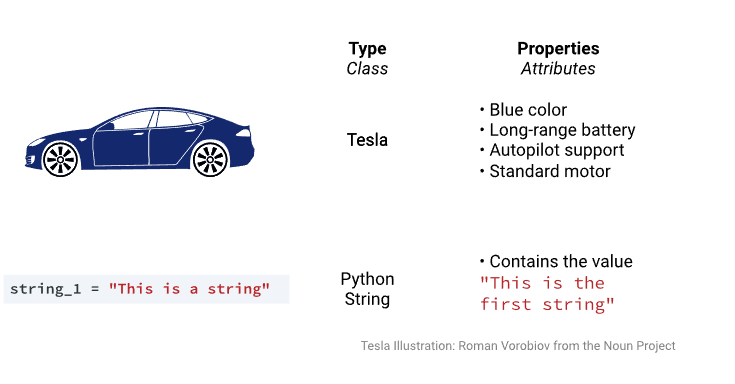
# Attributes and the Init Method

The example we used on the previous screen — a method that takes input and returns output without interacting with the object — isn't a technique we use often.

After all, we could do the same thing with a function without the hassle of defining a class and method. We used this example so you could practice creating a simple class with what you've learned so far.

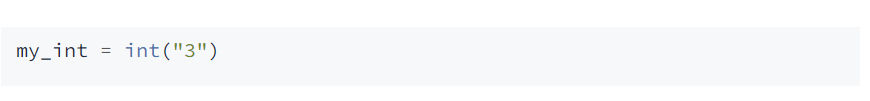
The power of objects is in their ability to store data using **attributes**

Relating back to our Tesla metaphor, an object of the Tesla "class" has attributes like their color, battery, and motor. Similarly, Python strings have attributes — the data stored inside the string:



*You can think of attributes like special variables that belong to a particular class*. Attributes let us store specific values about each instance of our class.

When we instantiate an object, we usually specify the data that we want to store inside that object. Let's look at an example of instantiating an int object:

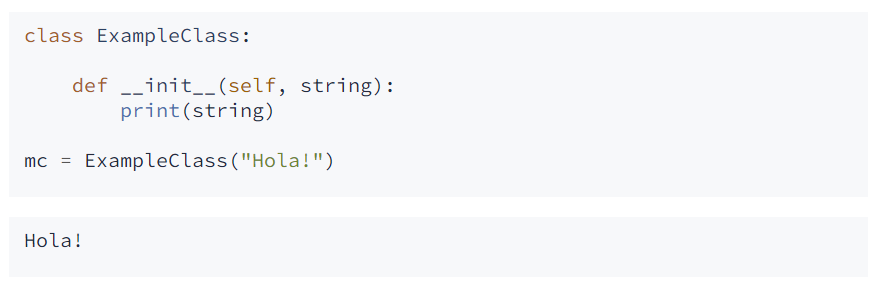


When we used int(), we provided the argument "3", which was converted and stored inside the object. We define any arguments we provide at instantiation using the **init method.**

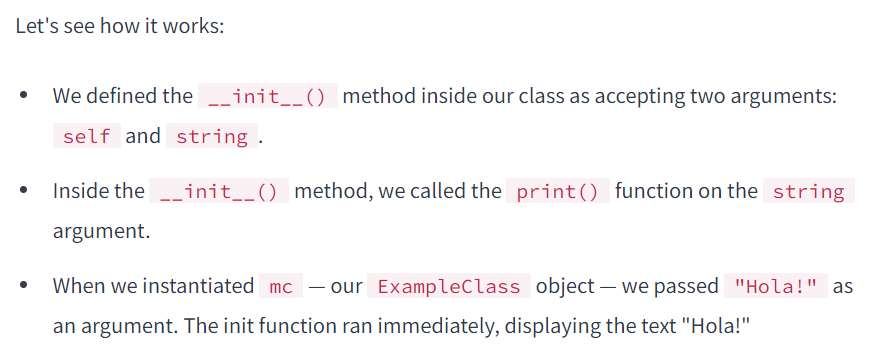
The init method — also called a **constructor** — is a special method that runs when we create an instance so we can perform any tasks to set up the instance.

>>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

The init method has a special name that starts and ends with two underscores: \_\_init\_\_(). Let's look at an example:

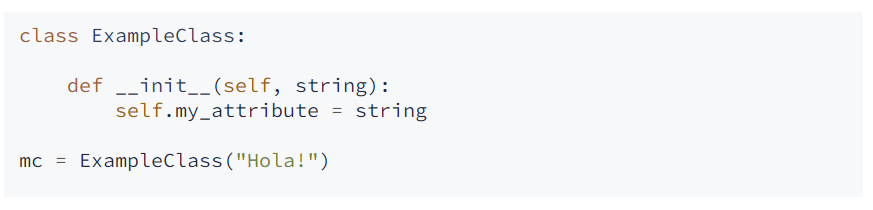


>>chương trình in “Hola!” ngay khi class chỉ vừa khởi tạo.



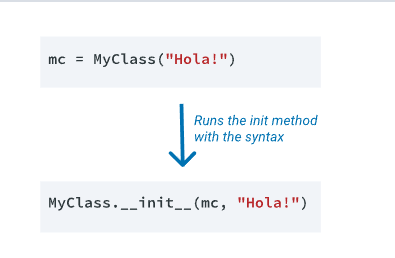
It's unusual to use print() inside an init method, but it helps us understand that the method has access to any arguments passed when we instantiate an object.

The init method's most common usage is to store data as an attribute:

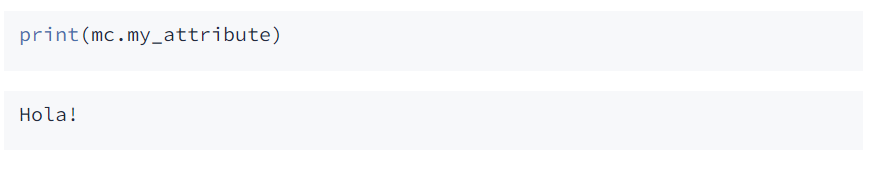


\_\_init\_\_ method thường được dùng để khởi tạo data.

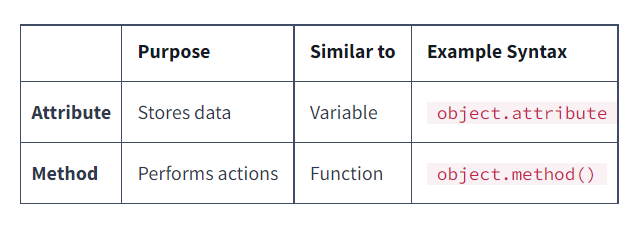
When we instantiate our new object, Python calls the init method, passing in the object:



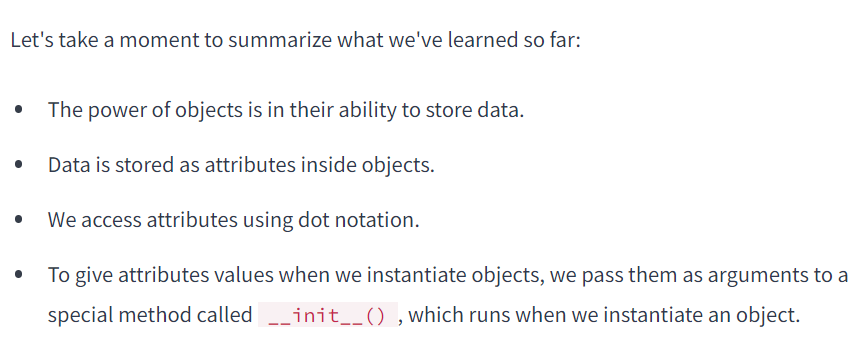
Our code didn't result in any output, but now we have stored "Hola" in the attribute my\_attribute inside our object. Like methods, we access attributes using dot notation, but attributes don't have parentheses like methods do. Let's use dot notation to access the attribute:

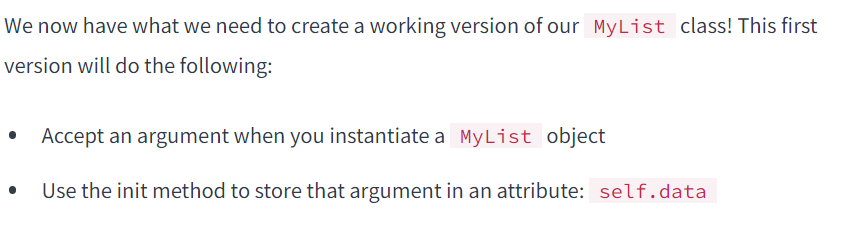


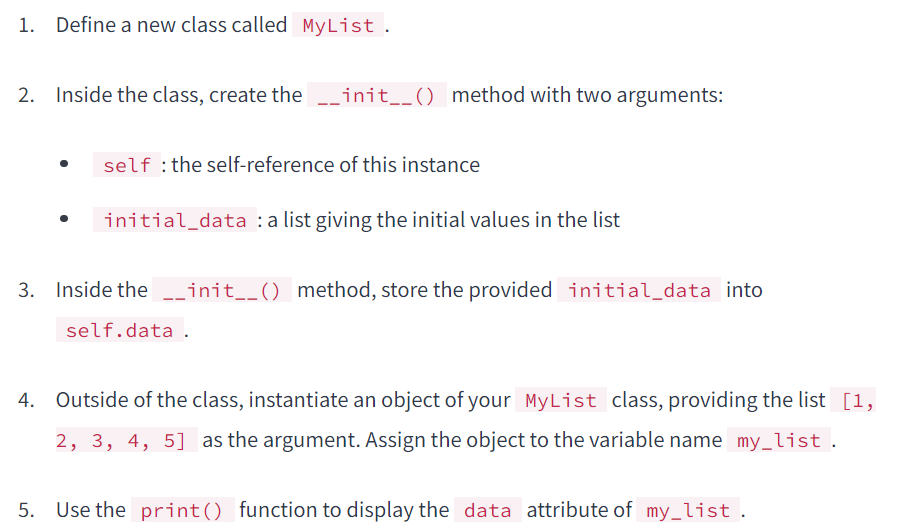
The table below summarizes(tóm tắt) some of the differences between attributes and methods:

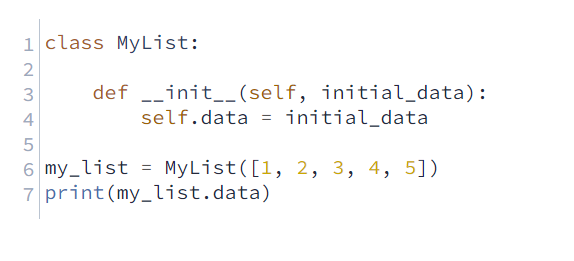


|  |  |  |  |
| --- | --- | --- | --- |
|  | mục đích | giống | cú pháp |
| attribute | lưu trữ data | biến | object.attribute |
| method | thực hiện 1 vài hoạt động | hàm | object.method() |







result:

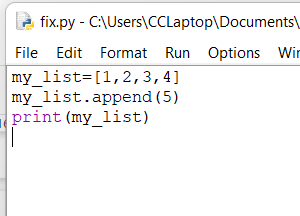


>>Giải thích: my\_list khởi tạo bằng hàm \_\_init\_\_ của class. Hàm \_\_init\_\_ sẽ được gọi ngay khi class được gọi và gán my\_list.data = initial\_data.

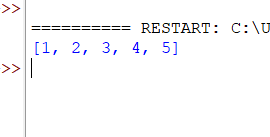
# Tạo Method Append

->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:

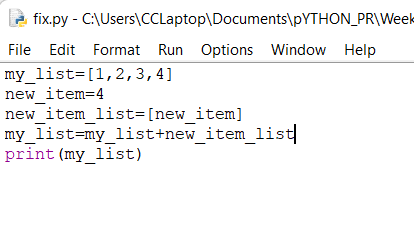


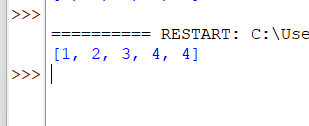
Kết quả:



->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



Kết quả: 

Khi đã hiểu cơ chế method append hoạt động, ta bát 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

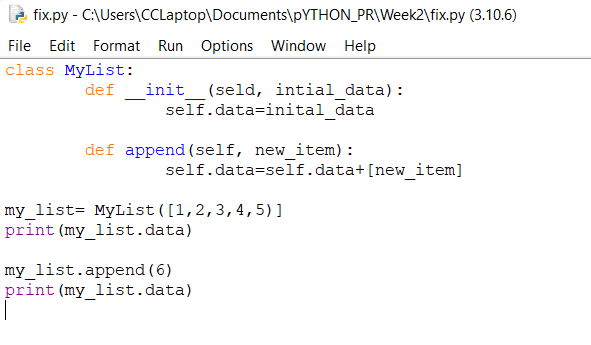
+self: bản thân hàm được gán

+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)

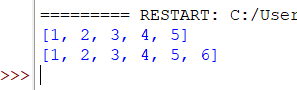


->Class có thêm method mới : append

+)self.data=self.data + [new\_item] :  thêm phần tử mới vào list khởi tạo từ đầu

->Khỏi tạo biến my\_list với classs MyList, giá trị list là [1,2,3,4,5]

in ra màn hình giá trị my\_list.data(phương thức append)

KQ: 

# Creating and Updating an Attribute

Let's summarize the work we've done so far (Tóm tắt các thứ đã học)

A picture containing website

Description automatically generated

* 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

Right now, each behavior we've created for our MyList class is also something a regular Python list does. Now we're going to create some new functionality: a new attribute.

When we want to find the length of a list, we use the len() function. What if we created a new attribute, MyList.length, which stores the length of our list at all times? We can achieve this by adding some to the \_\_init\_\_() method:

Graphical user interface, text, application

Description automatically generated

Let's look at what happens when we use the MyList.length attribute as defined above:

Background pattern

Description automatically generated

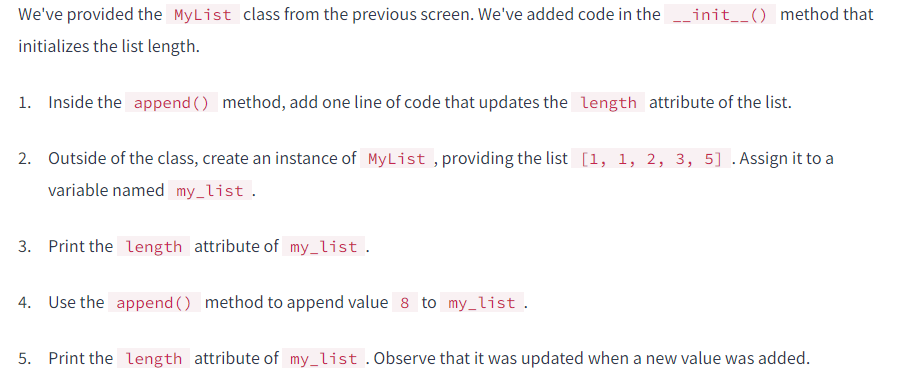
>>Sau khi thêm phần tử “4” vào list, nhưng attribute length vẫn không tăng lên 4. Do chúng ta định nghĩa  attribute length ở init.

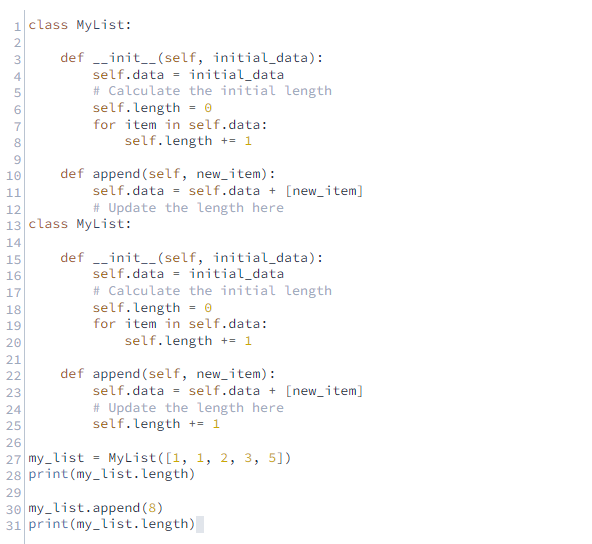
Because the code we added that defined the length attribute was only in the init method, if the list becomes longer using the append() method, our length attribute is no longer accurate.

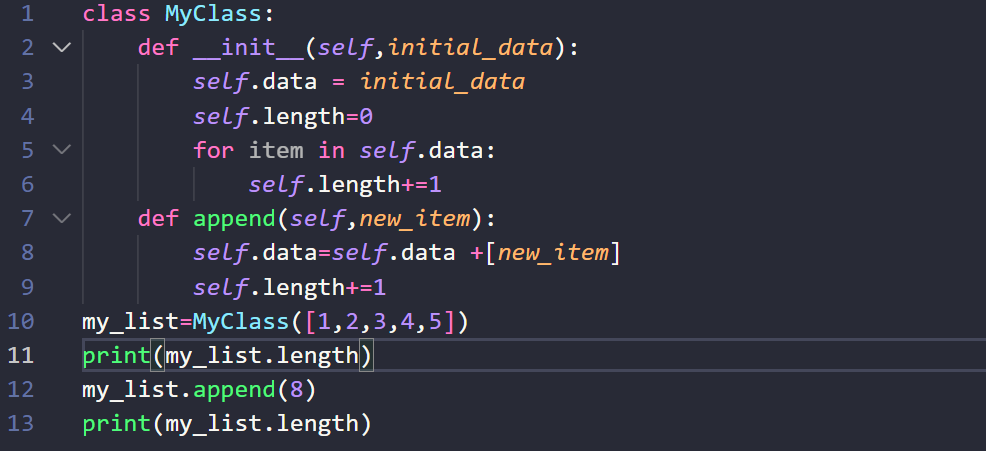
To address this, we need to run the code that calculates the length after any operation that modifies the data, which, in our case, is just the append() method. More precisely, we need to increment the length of the list each time we append a value to it.

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

Let's modify the append() method so that it updates the length of the list when we use it.







result:



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.

