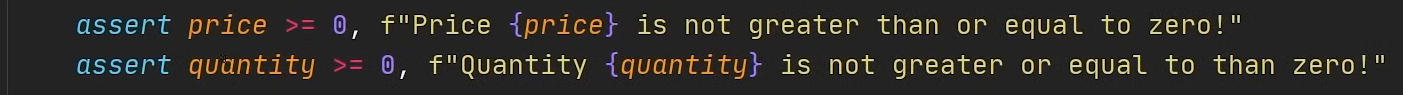
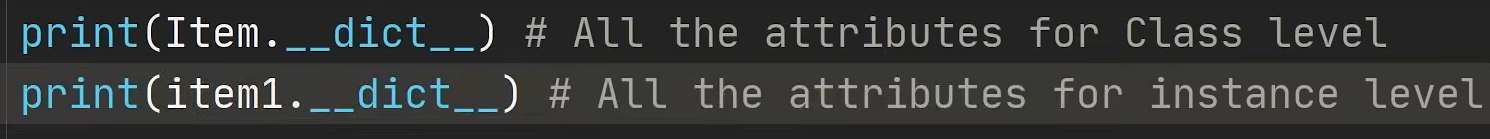
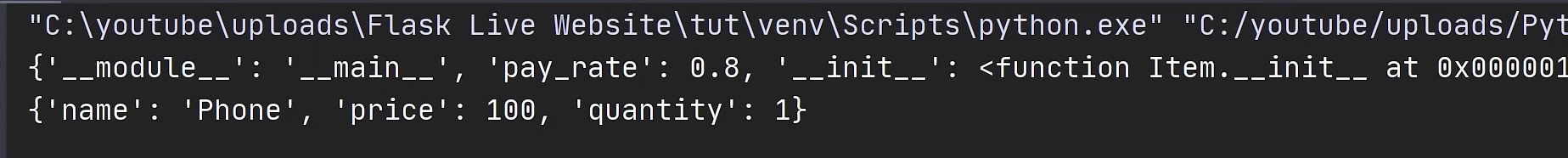
* Object is an instance of Class
* Function which is defined in inside class is called method
* assert: dùng để đặt điều kiện cho một attribute nào đó của 1 object trong class=> nếu khác với điều kiện sau assert thì sẽ trả về exception assertionError, và nếu muốn hiển thị thông báo exception theo yêu cầu:



* self: để trỏ vào đúng object được tạo
* 
* \_\_dict\_\_ : all the attributes for class level and instance level (converted into dictionary format)
* Result: 

Pay\_rate = 0.8 là thuộc tính thuộc về class (được khai báo trực tiếp ở class, không nằm trong method nào cả)

Nếu muốn truy cập vào thuộc tính này thì cần phải sử dụng syntax: <class\_name>. <attribute\_name>

Nếu muốn ghi đè pay\_rate này thì phải chuyển từ Item.payrate thành self.pay\_rate:A screen shot of a computer program

AI-generated content may be incorrect.

* Inside constructor \_\_init\_\_: tạo list các item đã được tạo thuộc class:A black and white text

  AI-generated content may be incorrect.
* In ra: A black and white text

  AI-generated content may be incorrect.
* Cách trực quan và friendly hơn để lấy danh sách tất cả các item đã được tạo thuộc class: sử dụng hàm \_\_repr\_\_: (để chạy sử dụng print(Item.all))

A screenshot of a computer program

AI-generated content may be incorrect.

* Class method, @classmethod là một loại **phương thức (method)** trong class, **gắn liền với class** chứ không phải với từng object (instance). Nó được khai báo bằng decorator @classmethod, và **tham số đầu tiên của nó luôn là cls** (viết tắt của "class").

A screen shot of a computer program

AI-generated content may be incorrect.

| **Loại method** | **Tham số đầu tiên** | **Gọi từ** | **Dùng để** |
| --- | --- | --- | --- |
| Instance method | self | Object | Làm việc với **dữ liệu riêng** của object |
| Class method | cls | Class / Object | Làm việc với **class** và các thuộc tính chung |
| Static method | ❌ Không có self hay cls | Class / Object | Hàm độc lập logic, không liên quan tới object/class |

item.get("name") ⬄item["name"]

isinstance(num) => check xem num có phải là số nguyên hoặc số thực không

| **Đặc điểm** | **@classmethod** | **@staticmethod** |
| --- | --- | --- |
| Tham số đầu tiên | cls (đại diện cho **class**) | ❌ Không có self hay cls |
| Có thể truy cập | Class variables, class methods | Không truy cập gì (giống hàm thường) |
| Gắn với | Class | Class (về mặt cú pháp) |
| Mục đích chính | Factory method, logic liên quan class | Hàm tiện ích (utility method) không phụ thuộc class hay object |

* Class method và static method không cần thiết và không nên gọi từ cấp độ instance
* Quyền truy cập: Mẹo nhớ nhanh:

name → công khai ai cũng dùng được

\_name → "đừng đụng", chỉ nội bộ class/subclass nên dùng

\_\_name → "giấu thật sự", tránh truy cập từ ngoài

* @property : sử dụng để biến 1 method to a attribute
* Cách để có thể thay đổi 1 attribute private:
* Getter and setter:

class Person:

def \_\_init\_\_(self, age):

self.\_age = age # dùng dấu \_ để ẩn đi

@property

def age(self):

print("Getting age...")

return self.\_age

@age.setter

def age(self, value):

print("Setting age...")

if value < 0:

raise ValueError("Age must be >= 0")

self.\_age = value

* Tóm lại 4 tính chất của OOP:

1. Encapsolution

| **Cách viết** | **Ý nghĩa** |
| --- | --- |
| self.name | Public – có thể truy cập thoải mái |
| self.\_name | "Protected" – chỉ nên dùng trong class hoặc subclass |
| self.\_\_name | "Private" – Python sẽ ẩn tên (name mangling) |
| @property | Tạo getter (đọc giá trị) |
| @<prop>.setter | Tạo setter (gán giá trị có kiểm soát) |

**Nếu dùng biến công khai thì sao?**

Vấn đề => Hậu quả nếu dùng công khai

Không kiểm tra đầu vào => Gán sai dữ liệu, gây bug

Không ẩn được xử lý => Dễ bị phụ thuộc vào cách cài bên trong

Không bảo vệ được dữ liệu => Gây lỗi khó truy vết

Không thay đổi được logic => Mọi thay đổi sẽ phá vỡ code cũ

Nếu như dùng @property hoặc @<property>.setter cho biết private (\_\_)thì chỉ có thể gán giá trị khác cho attribute và vẫn giữ nguyên biến gốc, còn cho biến (\_) thì có thể dùng để get và set (thay đổi giá trị luôn)

1. Inheritance (tính kế thừa)
2. Polymophysm (tính đa hình)

Một thực thể duy nhất có thể sử dụng cho các đối tượng khác nhau

Ví dụ về hàm built in: len()có thể đếm số lượng item trong list hoặc đếm số kí tự trong string

Ví dụ trong OOP: sau khi kế thừa từ lớp cha, 1 thực thể của lớp con có thể sử dụng lại method của lớp cha giống như thực thể của lớp cha

1. Abstraction: Tính trừu tượng:

| **Khái niệm** | **Ví dụ** |
| --- | --- |
| Tính trừu tượng là gì? | Ẩn đi chi tiết xử lý, chỉ hiển thị những gì cần thiết |
| Dùng cái gì? | from abc import ABC, abstractmethod |
| Ứng dụng thực tế | Thiết kế hệ thống như: Vehicle, Shape, Model, PaymentMethod,... |

Bắt buộc lớp con phải triển khai các method trừu tượng của lớp cha => mục đích: phân biệt bằng các phương thức đặc trưng của từng lớp con

Không thể tạo instance trực tiếp từ lớp cha mà phải từ lớp con (kế thừa từ lớp cha)