**- Big-O Notation** (Time/Space Complexity)

**- Data Structures** (4 operations: Creating, Deleting, Inserting, Locating)

- Arrays (Fixed/Dynamic Size)

- Linked List (Single/Double Linked)

- Queue and Stack

- Simple Trees (Binary Tree and Binary Search Tree)

- Heap (Min, Max, Priority Queue)

- Graphs ((Un)Directed, (Un)Weighted)

- Hash Map

- **Algorithms**

- Recursion - Searching (Linear/Binary)

- Sorting (Insertion, Selection, Bubble, Merge, Heap, Quick)

- Graph (Depth/Breadth First Search, Kruskal, Prims)

- Path Finding (Dijstra, A\*)

- Greedy

- Divide and Conquer

- Dynamic Programming

- Backtracking

**- Advanced**

- Trees (Tries, B/AVL/Red-Black/Segment/Fenwick Trees)

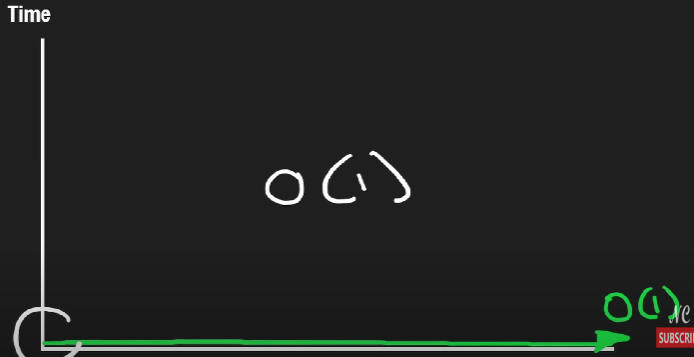
- Skip Lists

- Disjoint Set

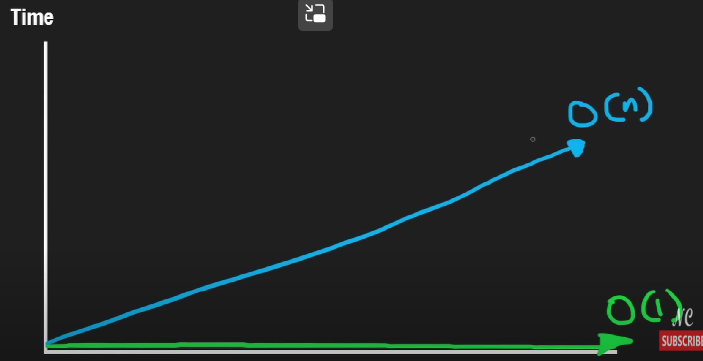
- Math (Combinatorics, Probabilty, Discrete Math, Discrete Structures)

Big o

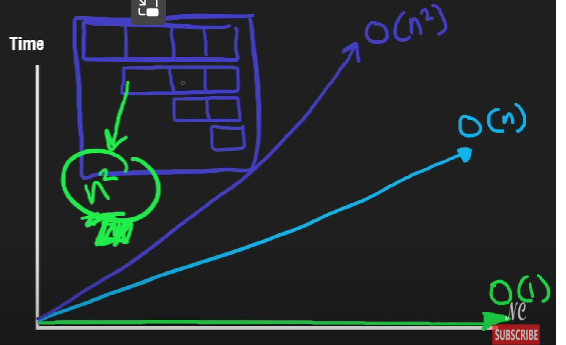
O(1) – constant time , accessing elements, appending etc…



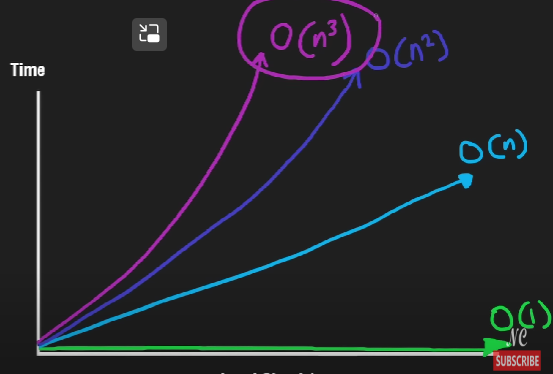
O(n) – increase proportionally, for loop, inserting elements in middle, remove in the middle, searching in the array



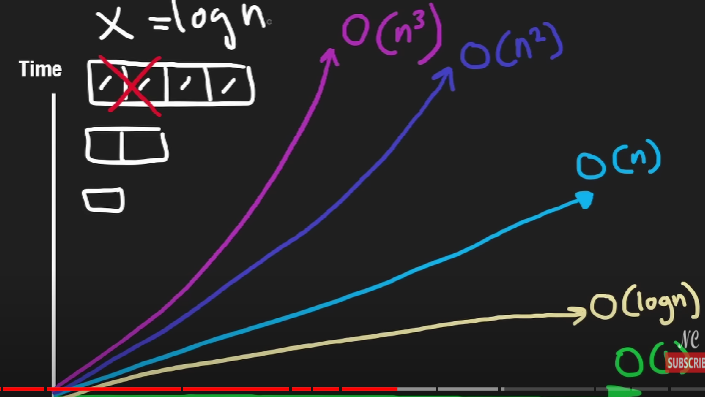
O(n^2) – nested loops, looping single array, insertion sort- insert middle of the array



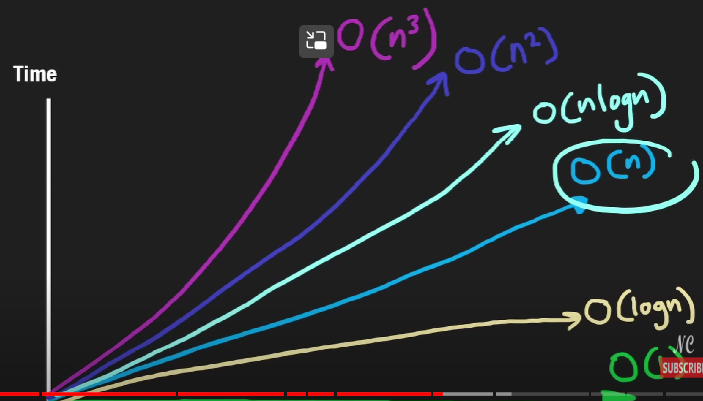
O(n^3) - 3 nested loops,



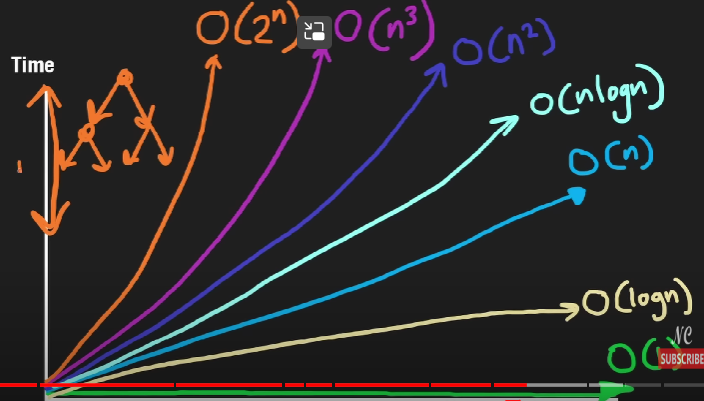
O(log n) – binary search, binary search tree BST, heap push and pop



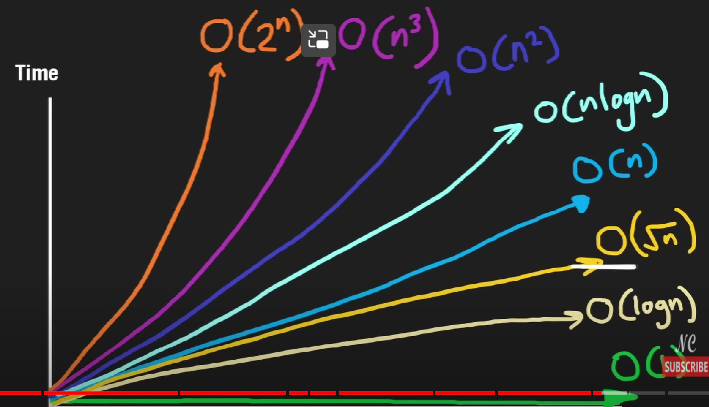
O(n log n) – built in sorting functions, merge sort, heap sort



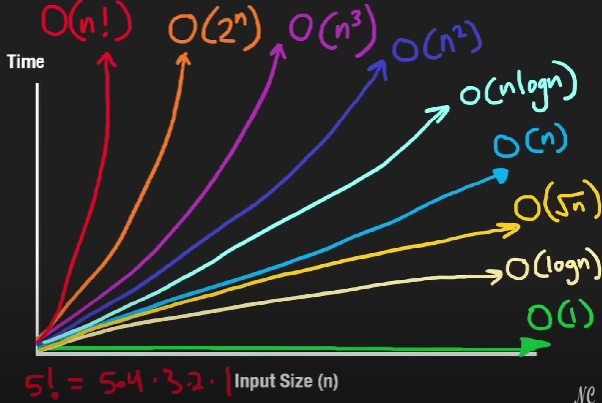
O(2^n) – recursion, tree height n, two branches



O(root(n)) – get all factors of n



O(n!) – permutation, travelling salesman problem



Backend developer :







Becoming a Python backend developer involves several steps, including learning Python programming, mastering backend development concepts, and gaining hands-on experience with relevant frameworks and tools. Here's a roadmap to help you get started:

1. \*\*Learn Python\*\*: Start by learning Python programming language fundamentals such as syntax, data structures, control flow, functions, and object-oriented programming (OOP). There are many online resources available, including tutorials, courses, and books. Some popular resources include Codecademy, Coursera, and the official Python documentation.

2. \*\*Understand Web Development Basics\*\*: Familiarize yourself with fundamental web development concepts such as HTTP protocol, client-server architecture, RESTful APIs, and CRUD operations (Create, Read, Update, Delete). This knowledge forms the foundation for backend development.

3. \*\*Learn Backend Development\*\*: Dive into backend development principles and technologies. Understand how servers work, handle requests and responses, manage databases, and maintain application state. Learn about different backend frameworks and their usage.

4. \*\*Choose a Backend Framework\*\*: Python offers several backend frameworks such as Django, Flask, and FastAPI. Each framework has its own strengths and use cases. Start with one framework and become proficient in it. Django is a popular choice for its feature completeness, while Flask offers simplicity and flexibility.

5. \*\*Master a Database\*\*: Learn about databases and how to interact with them from Python. Common databases used in backend development include SQL databases like PostgreSQL, MySQL, and SQLite, as well as NoSQL databases like MongoDB.

6. \*\*Build Projects\*\*: Practice your skills by building real-world projects. Start with simple projects and gradually increase complexity. Building projects helps solidify your understanding of concepts and gives you practical experience.

7. \*\*Version Control with Git\*\*: Learn version control using Git, a crucial skill for collaboration and managing code changes. Understand concepts like repositories, branches, commits, and merging.

8. \*\*Understand Deployment\*\*: Learn how to deploy your backend applications to production environments. Explore deployment platforms like Heroku, AWS, Google Cloud Platform, or DigitalOcean.

9. \*\*Continuous Learning and Improvement\*\*: Backend development is a continuously evolving field. Stay updated with the latest trends, tools, and best practices. Follow blogs, participate in online communities, and attend conferences or meetups.

10. \*\*Build a Portfolio\*\*: Showcase your projects, skills, and contributions on platforms like GitHub or personal portfolio websites. A strong portfolio demonstrates your capabilities to potential employers.

11. \*\*Networking and Job Search\*\*: Connect with other developers and professionals in the industry through networking events, online forums, and social media platforms like LinkedIn. Start applying for backend developer positions, and be prepared for interviews by practicing coding challenges and discussing your projects and experiences.

Remember, becoming a proficient backend developer takes time and dedication. Stay patient, keep practicing, and don't hesitate to seek help from the community or mentors along the way. Good luck on your journey!

Classes and objects:

Class Item:

Def \_\_init\_\_(self): # always initiate automatically

Print(“I am created”)

Def calculate\_total\_price(self): #methods (functions of inside of a class)

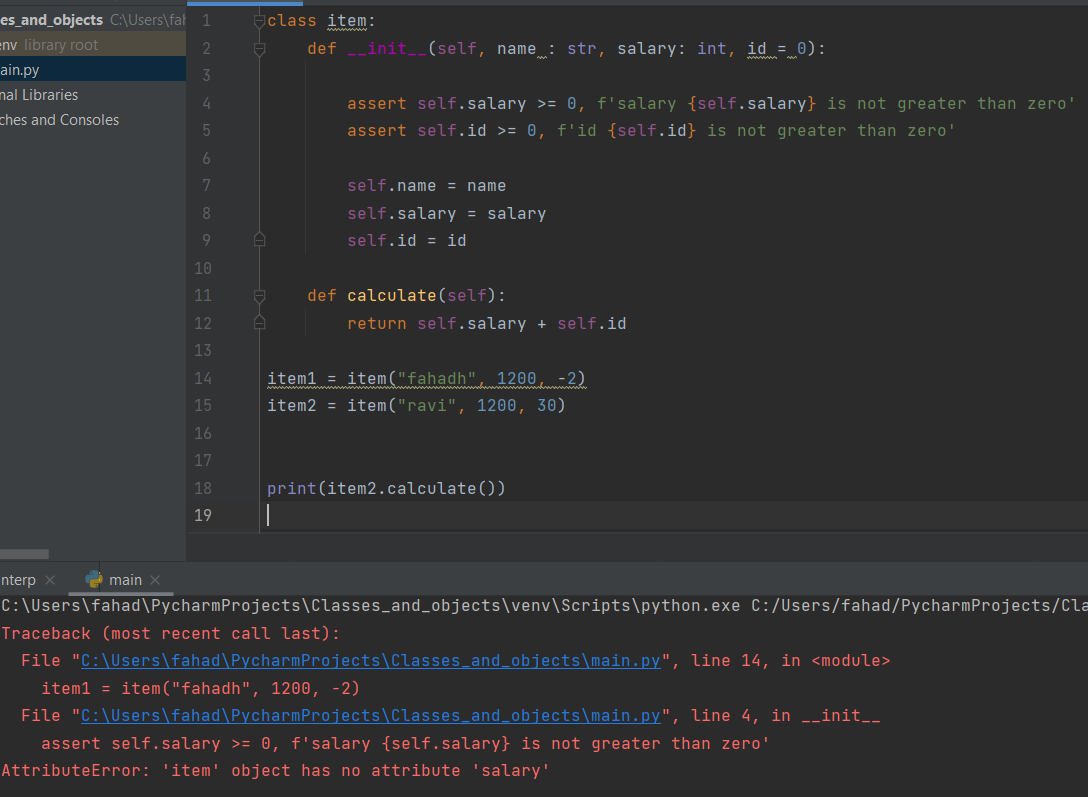
pass

Item1 = Item() # Item1 = object, item() = instance of a class

Item1.name = “Phone” #attributes of a class

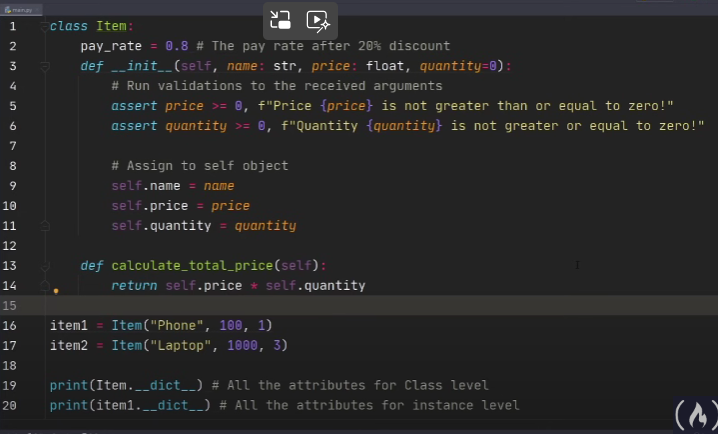
Item2 = Item() # Item2 = object, item() = instance of a class, init execute automatically when u call this instance

We can use “assert” to validate the class.

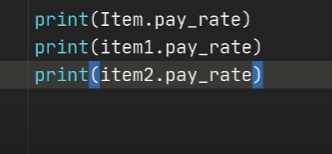


Class level attribute:

Pay\_rate = 0.8 # class level attribute, can be accessed by both in class level and instance level.



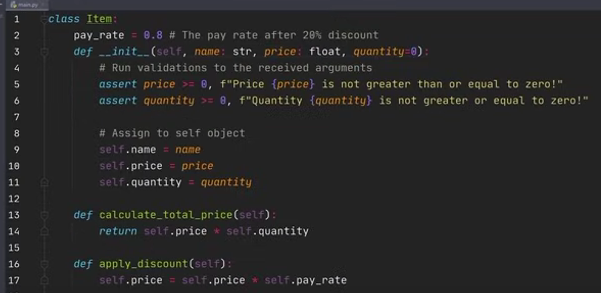
\_\_dict\_\_ > magic keyword, used to show all the attributes in both class level and instance level

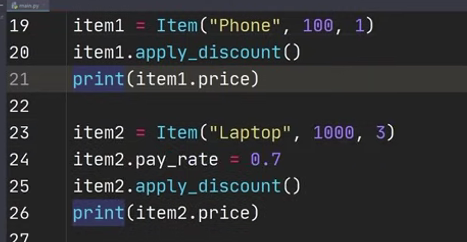


Above, accessing attributes by class level and instance level

Read well,

Dynamic pay\_rate based on class level, and instance level, if we didn’t specify the pay\_rate, it take from class level. If we specify the pay\_rate in the instance level, it will use that pay\_rate. For that, we have to change self.pay\_rate

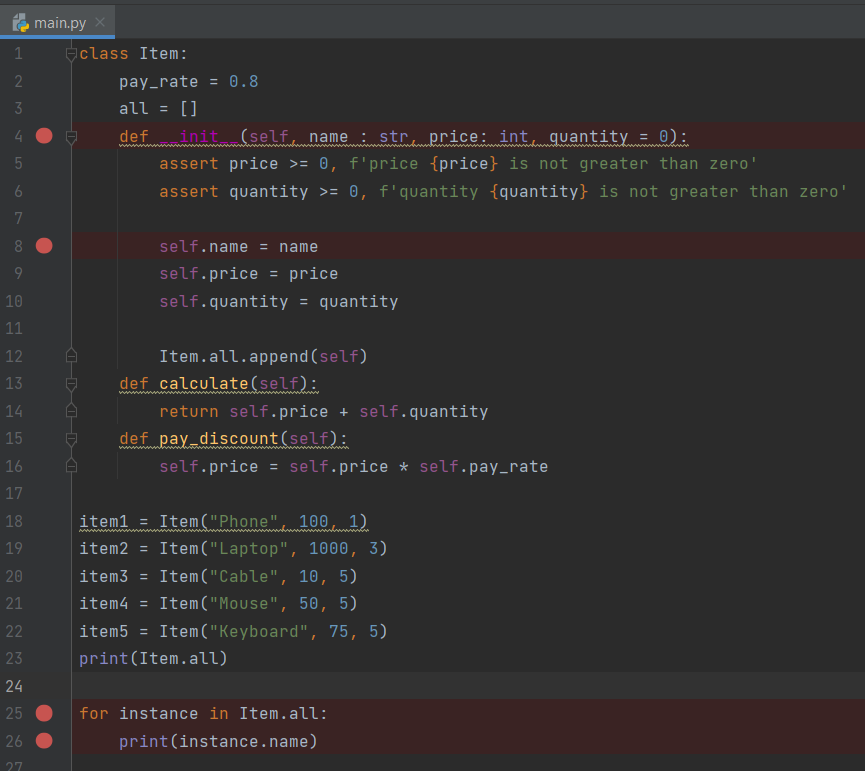




Above #1 get the pay\_rate from class level

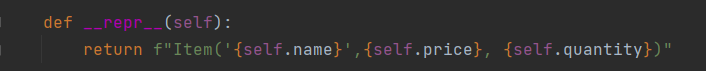
#2 get the pay\_rate in the assigned instance level

We can create a list and access the all instance names, price, and quantity



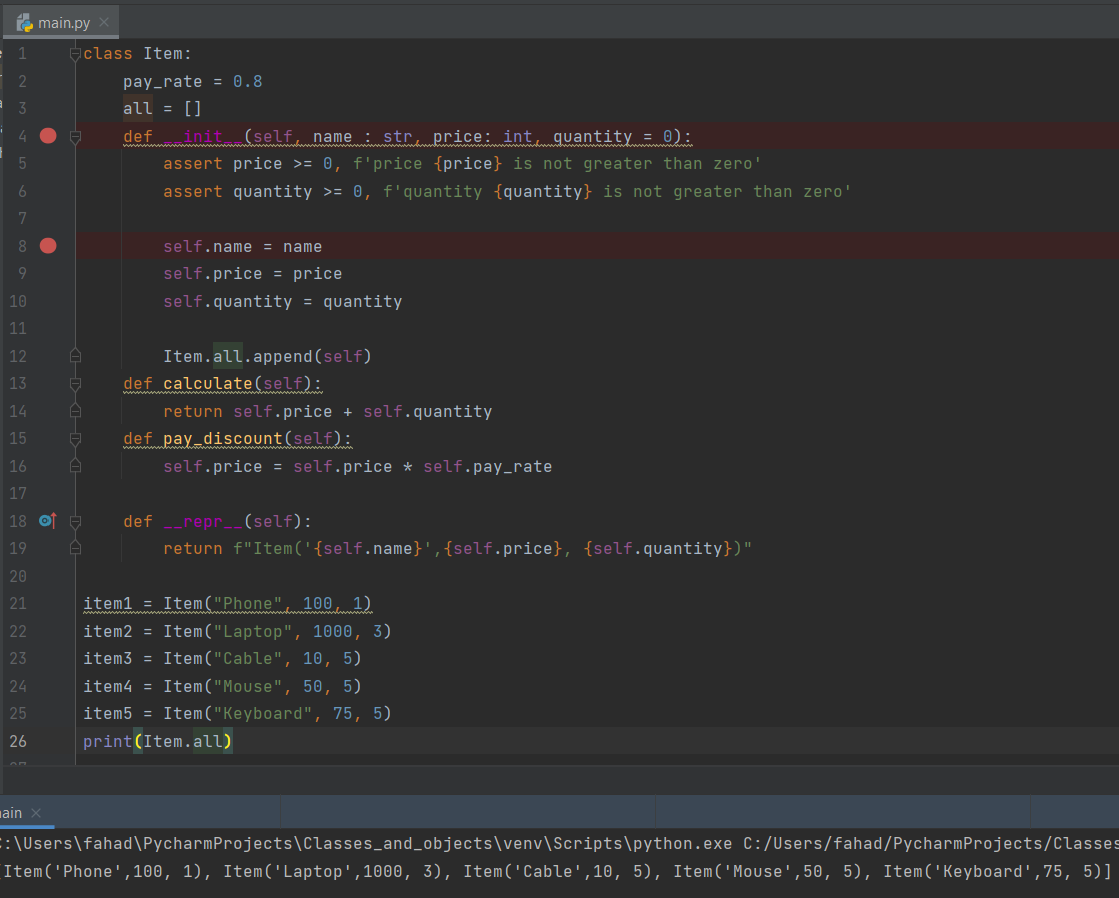
Normally if we print the instance, it will print in the object format, to print as a normal format we use \_\_repr\_\_ magic keyword,

\_\_repr\_\_ > representation of instance

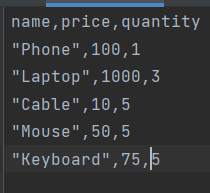


Print(item.all)

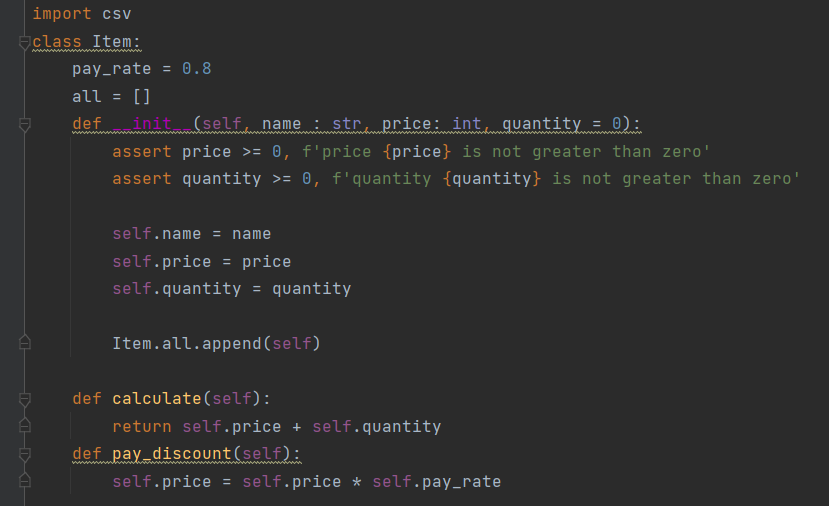




We can use csv – comma separated value files to import and use this each line as instance,



The @classmethod, used to directly call the class inside, and get the data for instances,

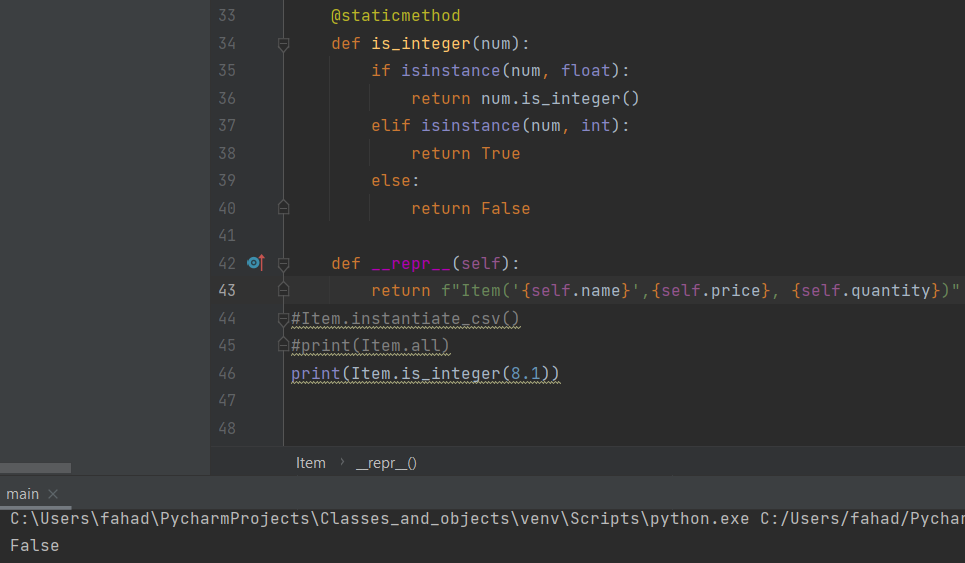


Get the csv data > convert dictionary > put it into the list

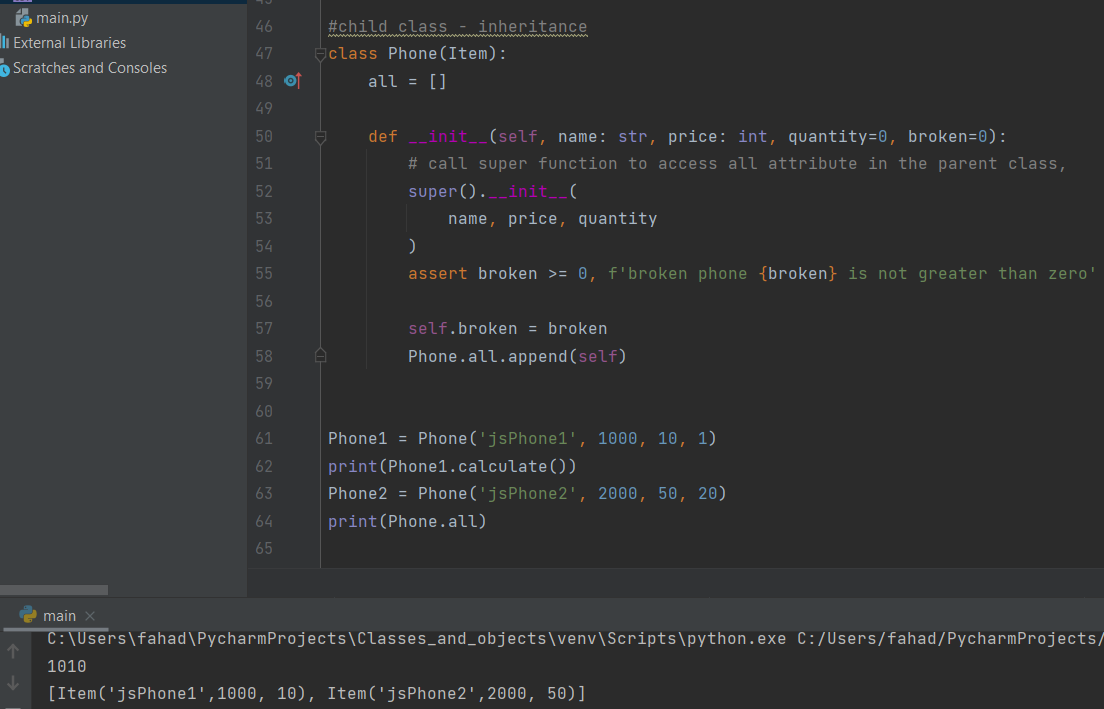


@staticmethod – act as a normal function (used as check if num is integer or not), still releated to class. We call this method using class object.

@classmethod – act as class function (used to send parameters from csv, json, yaml, etc to call a instance). We also call this method using class object.

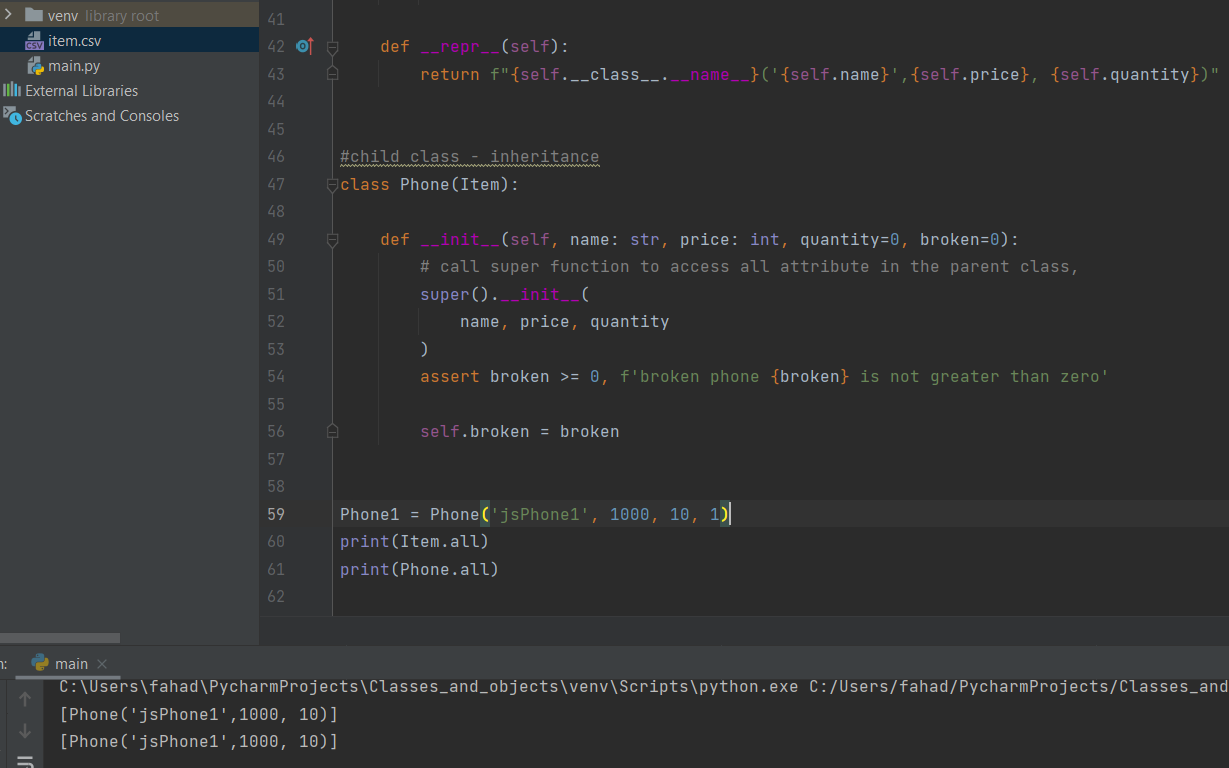


Inheritance – inherit from parent class, super() – access attribute from parent class, we can use functions from parent class, and also created new variable broken phone



Here, in the \_\_repr\_\_, we used generic (automatically give the class name based on the instance)

And we also don’t add all attribute in the child class, here we removed the ‘all’ attribute in the child class. Accessed from parent class.



Encapsulation: getters and setters

Definition : If we having the self.name from csv files for the first time when initialization, if user try to override the name with the another name,

For example: Item1 = Item(“jsphone”, 100, 3)

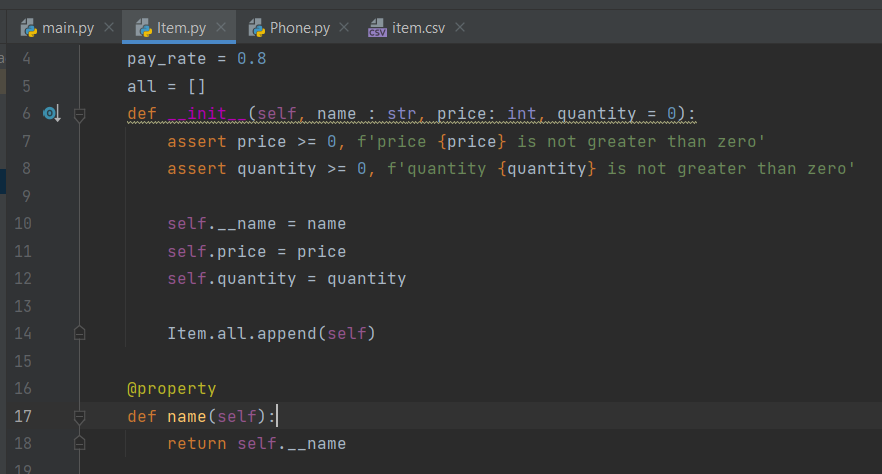
Item1.name = “othername”

Preventing the user by doing this called encapsulation(within the setters), it can be achieved by using @property decorator (read only attribute decorator)

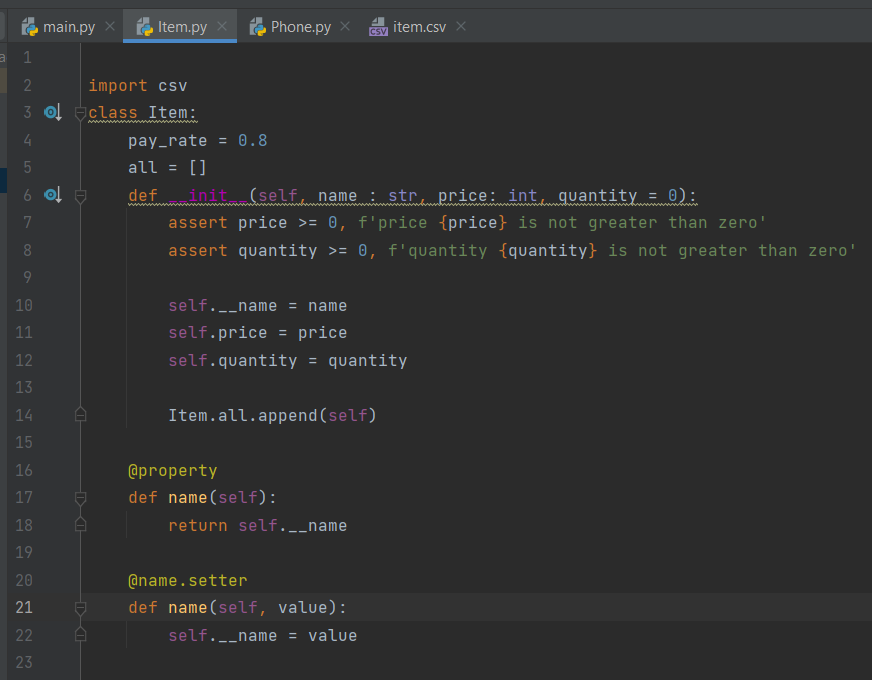
@property – user cant change the values, if user try to change it will give the error

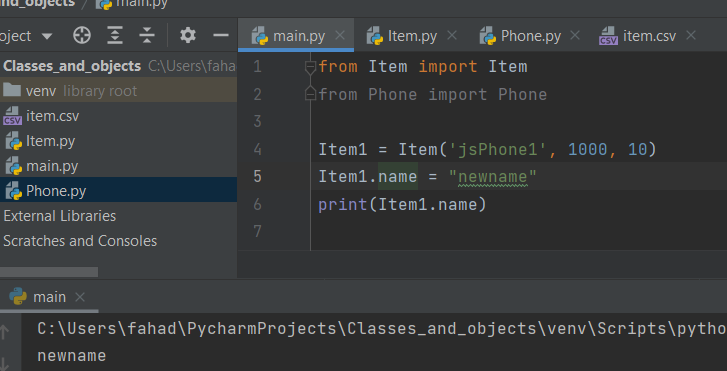
Here, If we give this \_name, the user cant change the name, but he can still access \_name variable,

To avoid this issue, we use \_\_name, user cant access this variable,



But still, if you want to change value of read only attribute, you can use @name.setter > name (will change)





Abstraction:

Definition : It is basically, not giving access from instance level (main.py)

For example: lets consider, we have send\_mail() method inside the parent class. Inside this method, we are supposed to call submethod like connect\_mail(), message\_mail() etc..

We should not give access these submethods in instance level. So, we use \_\_connect\_mail(), \_\_massage\_mail(), to not give access. Its called abstraction.

Polymorphism: more forms

Its using methods in parent class from child class.

We can give override the pay\_rate from child class for different products like laptop, phone etc…