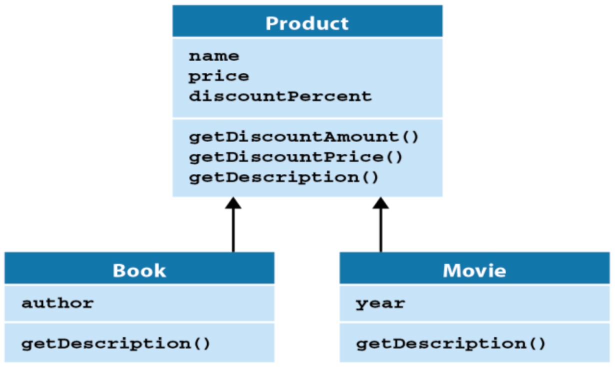
Assignment 14 – Inheritance

Inheritance

Inheritance is a fundamental concept of object-oriented programming that allows you to create new classes that's based on an existing class.

Description

- A class that another class inherits from is called a base class, parent class or superclass.
- A class that inherits from another class is called a derived class, child class or subclass.
- A subclass can add new attributes and methods. It can also override inherited methods, providing its functionality to that method.



In this image, Product is the superclass and book and movie are inheriting from product so they are the subclass(es).

So both book and movie will have the name, price and discountPercent attributes as well as the getDiscountAmount, getDiscountPrice and getDescription methods.

In addition, each of the subclasses will include their own attributes (author for book and year for move) as well as a getDescription method which will override the getDescription from the superclass.

Definition

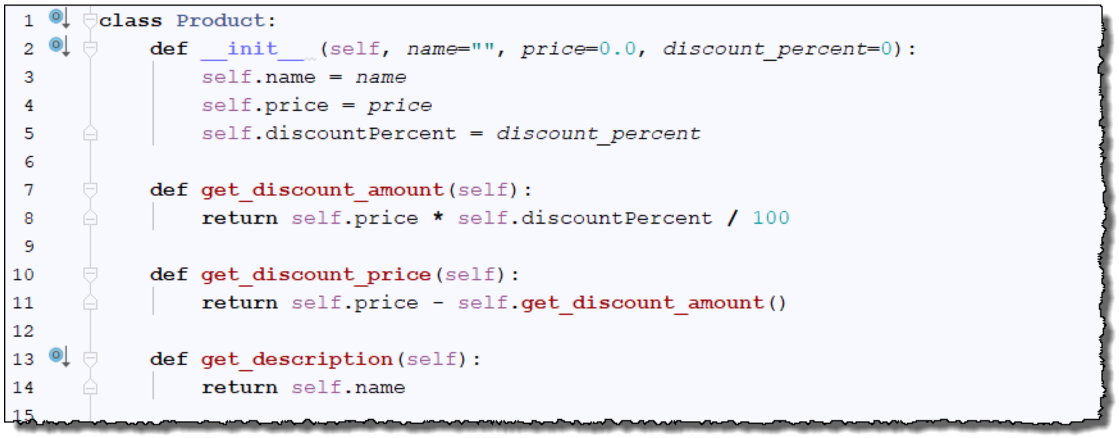
class *SubClassName*(*SuperClassName*)

Product Viewer

In this example, we'll create an object class (Products) which will be a superclass for 2 subclass objects (books & movies).

Creating a Product Class (Superclass)

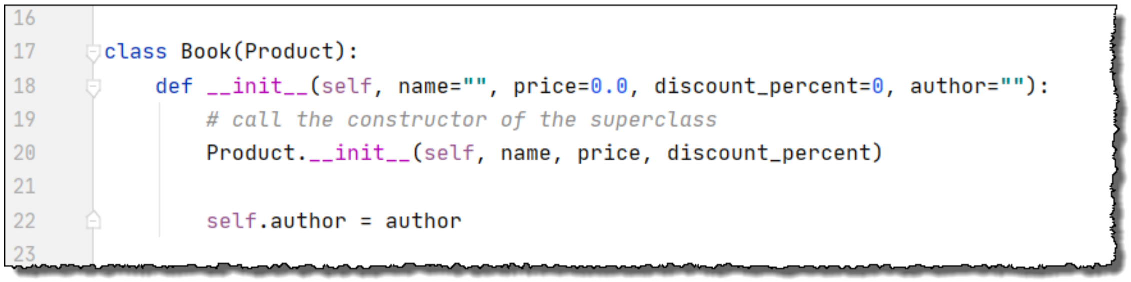
1. Create a new python file **objects.py.**
2. Add a Product class as follows:



```python
class Product:
    def __init__(self, name="", price=0.0, discount_percent=0):
        self.name = name
        self.price = price
        self.discountPercent = discount_percent

    def get_discount_amount(self):
        return self.price * self.discountPercent / 100

    def get_discount_price(self):
        return self.price - self.get_discount_amount()

    def get_description(self):
        return self.name
```

Creating a Book Class (Subclass)

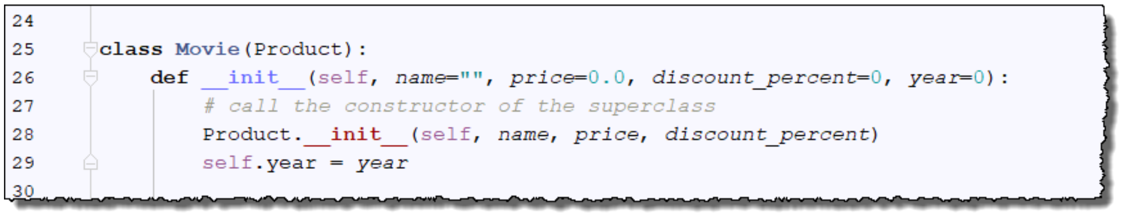
3. Now create a Book class that will inherit from the Product class
4. Code as follows:



```python

class Book(Product):
    def __init__(self, name="", price=0.0, discount_percent=0, author=""):
        # call the constructor of the superclass
        Product.__init__(self, name, price, discount_percent)

        self.author = author
```

Creating a Movie Class (Subclass)

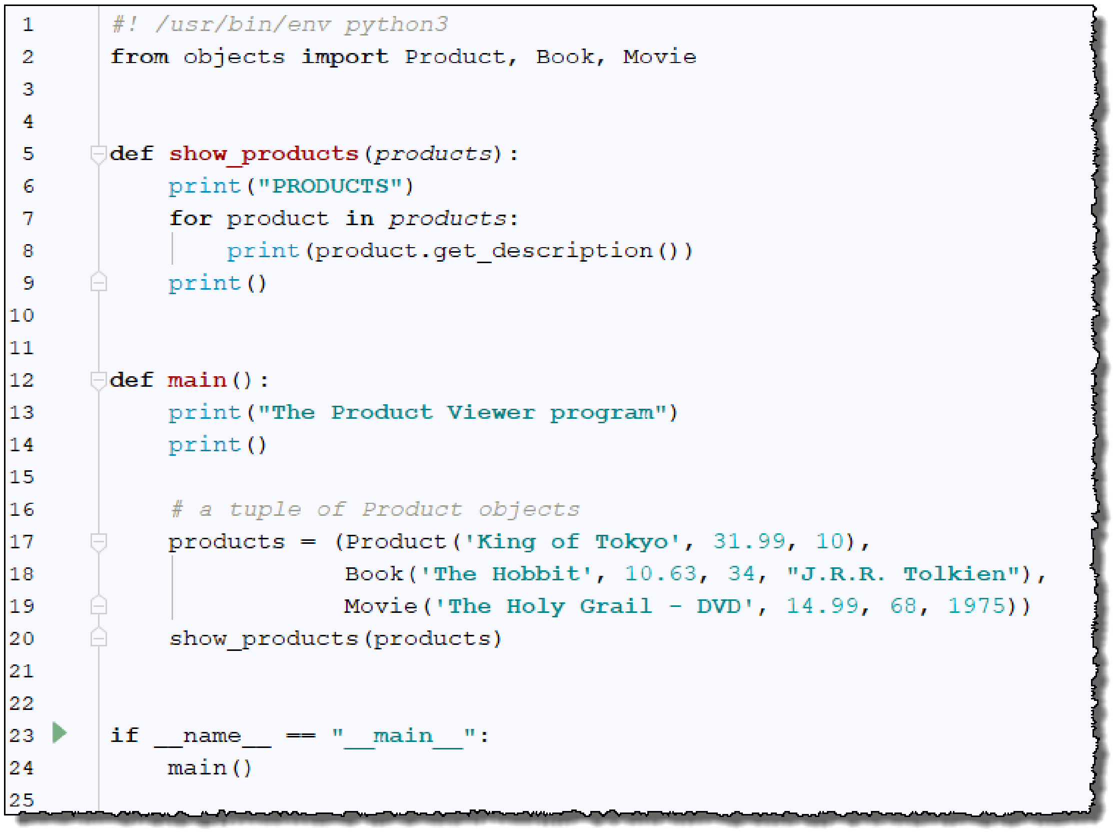
5. Finally, add a Movie class that will also inherit from the Product class as follows:



```python

class Movie(Product):
    def __init__(self, name="", price=0.0, discount_percent=0, year=0):
        # call the constructor of the superclass
        Product.__init__(self, name, price, discount_percent)
        self.year = year
```

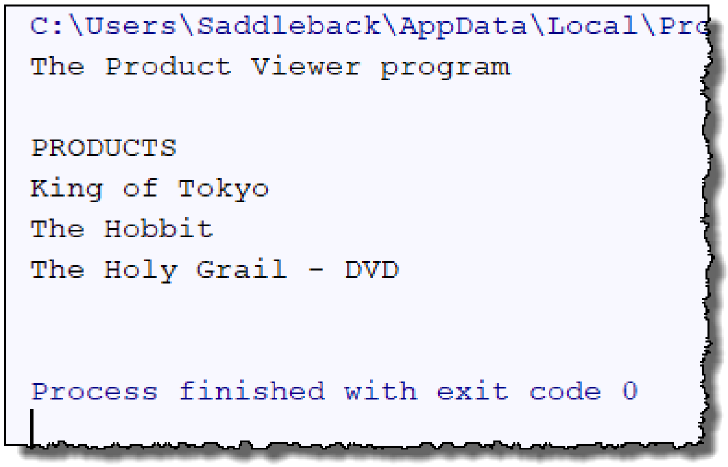
Creating a Product Viewer

6. Create a new python file **product_viewer.py.**
7. Crate the main which will call a method to list all products:



```
#! /usr/bin/env python3
from objects import Product, Book, Movie


def show_products(products):
    print("PRODUCTS")
    for product in products:
        print(product.get_description())
    print()


def main():
    print("The Product Viewer program")
    print()

    # a tuple of Product objects
    products = (Product('King of Tokyo', 31.99, 10),
                Book('The Hobbit', 10.63, 34, "J.R.R. Tolkien"),
                Movie('The Holy Grail - DVD', 14.99, 68, 1975))
    show_products(products)


if __name__ == "__main__":
    main()
```

8. Now run the application to verify the products are all listed.
   Notice when we list the classes, we use the get_description method from the superclass.
   Screen Capture #1 (4 points)



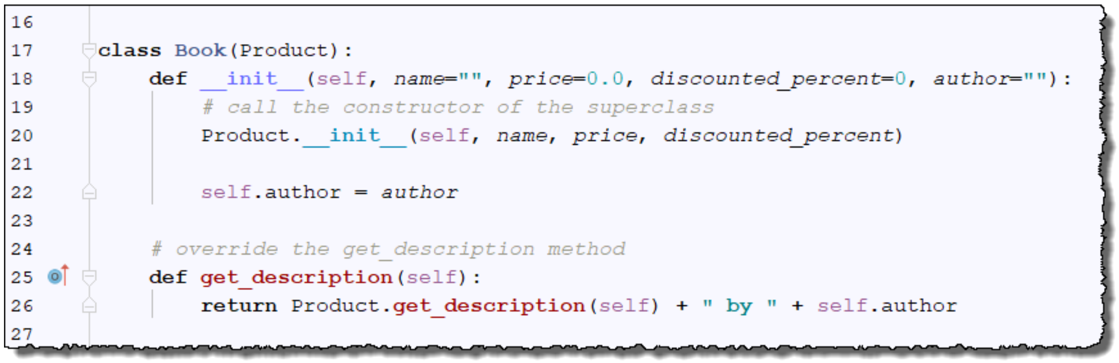
```
C:\Users\Saddleback\AppData\Local\Pro
The Product Viewer program

PRODUCTS
King of Tokyo
The Hobbit
The Holy Grail - DVD


Process finished with exit code 0
```

Polymorphism

Polymorphism lets you treat objects of different types as if they were the same object.

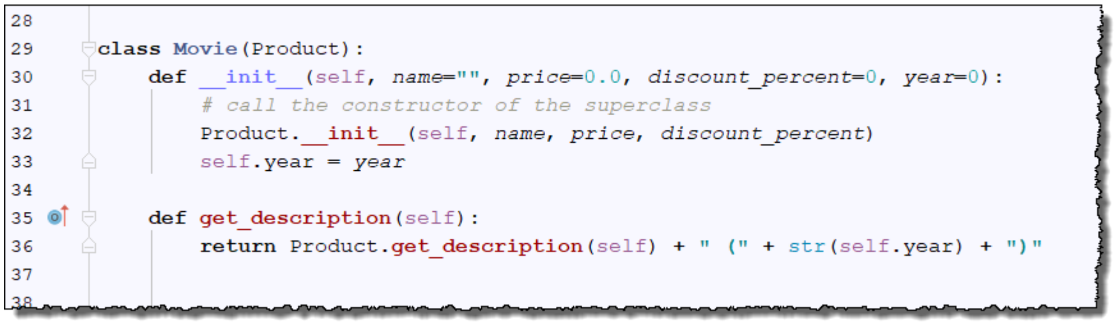
In this example, we'll add a get_description method to both the Book and Movie class where these methods will override the superclass method. So, depending on the object type, the description will be displayed differently.

9. Add the new get_description to the Book class.



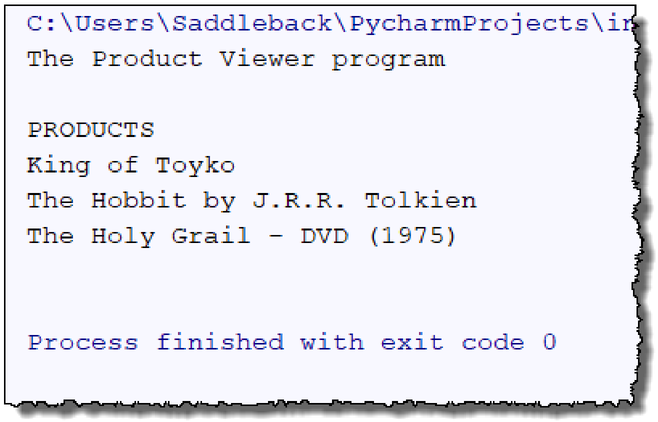
```

class Book(Product):
    def __init__(self, name="", price=0.0, discounted_percent=0, author=""):
        # call the constructor of the superclass
        Product.__init__(self, name, price, discounted_percent)

        self.author = author

    # override the get_description method
    def get_description(self):
        return Product.get_description(self) + " by " + self.author

```

10. Add the new get_description to the Movie class.



```

class Movie(Product):
    def __init__(self, name="", price=0.0, discount_percent=0, year=0):
        # call the constructor of the superclass
        Product.__init__(self, name, price, discount_percent)
        self.year = year

    def get_description(self):
        return Product.get_description(self) + " (" + str(self.year) + ")"

```

11. Once again, run the application and verify the products are all listed with the different description based on the object's type.
    Screen Capture #2 (3 points)

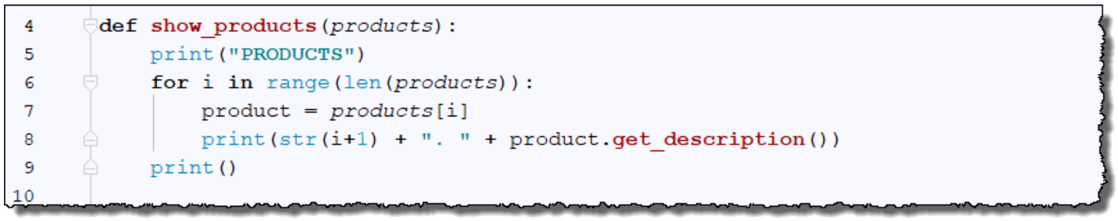


```
C:\Users\Saddleback\PycharmProjects\in
The Product Viewer program

PRODUCTS
King of Toyko
The Hobbit by J.R.R. Tolkien
The Holy Grail - DVD (1975)


Process finished with exit code 0
```

Determining Object Type

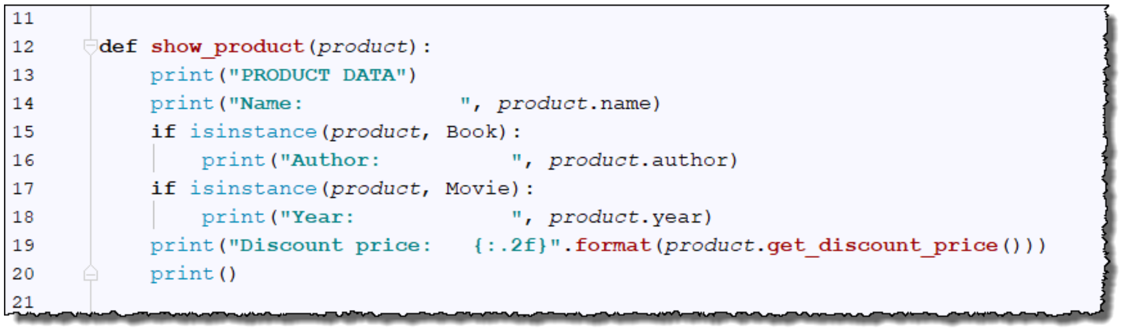
In this example, we'll list the product with a different set of data depending on the object's type (Product, Book, Movie).

12. The following changes will all take place in the product_viewer file.
13. Update the show_products to display a number, we'll use this number to select which product to display.
14. Code as follows:



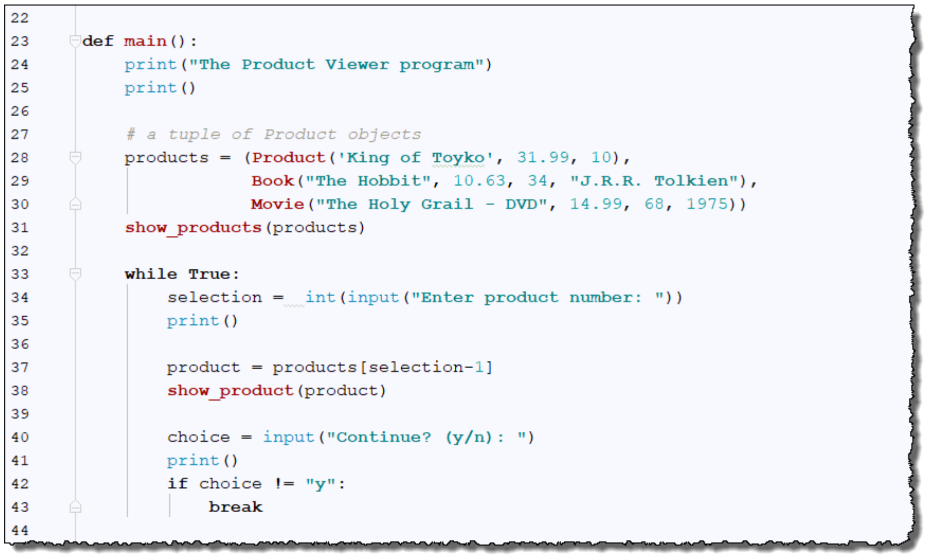
```
def show_products(products):
    print("PRODUCTS")
    for i in range(len(products)):
        product = products[i]
        print(str(i+1) + ". " + product.get_description())
    print()
```

15. Now we'll add a new method to display the item detail based on the object's type
16. Code as follows:



```

def show_product(product):
    print("PRODUCT DATA")
    print("Name:             ", product.name)
    if isinstance(product, Book):
        print("Author:           ", product.author)
    if isinstance(product, Movie):
        print("Year:             ", product.year)
    print("Discount price:    {:.2f}".format(product.get_discount_price()))
    print()

```

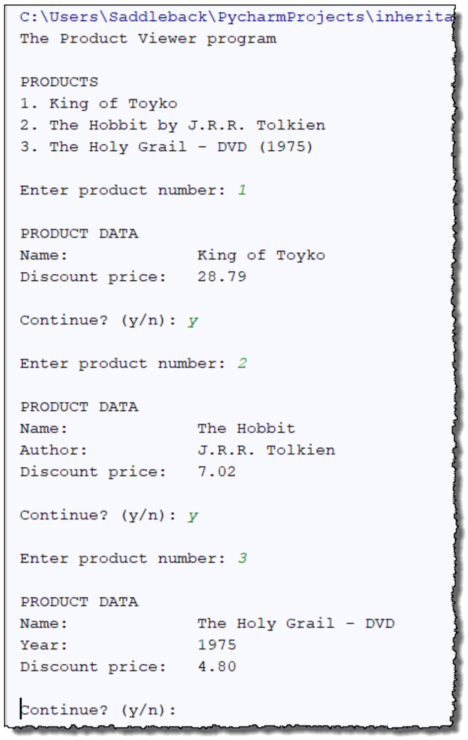
17. Update the main to prompt the user to select a product to list the product's details
18. Code as follows:



```
def main():
    print("The Product Viewer program")
    print()

    # a tuple of Product objects
    products = (Product('King of Toyko', 31.99, 10),
                Book("The Hobbit", 10.63, 34, "J.R.R. Tolkien"),
                Movie("The Holy Grail - DVD", 14.99, 68, 1975))
    show_products(products)

    while True:
        selection = int(input("Enter product number: "))
        print()

        product = products[selection-1]
        show_product(product)

        choice = input("Continue? (y/n): ")
        print()
        if choice != "y":
            break
```

19. Run the application. This time, select each product to verify each object type displays differently.

Screen Capture #3 (3 points)



```
C:\Users\Saddleback\PycharmProjects\inherita
The Product Viewer program

PRODUCTS
1. King of Toyko
2. The Hobbit by J.R.R. Tolkien
3. The Holy Grail - DVD (1975)

Enter product number: 1

PRODUCT DATA
Name:              King of Toyko
Discount price:    28.79

Continue? (y/n): y

Enter product number: 2

PRODUCT DATA
Name:              The Hobbit
Author:            J.R.R. Tolkien
Discount price:    7.02

Continue? (y/n): y

Enter product number: 3

PRODUCT DATA
Name:              The Holy Grail - DVD
Year:              1975
Discount price:    4.80

Continue? (y/n):
```

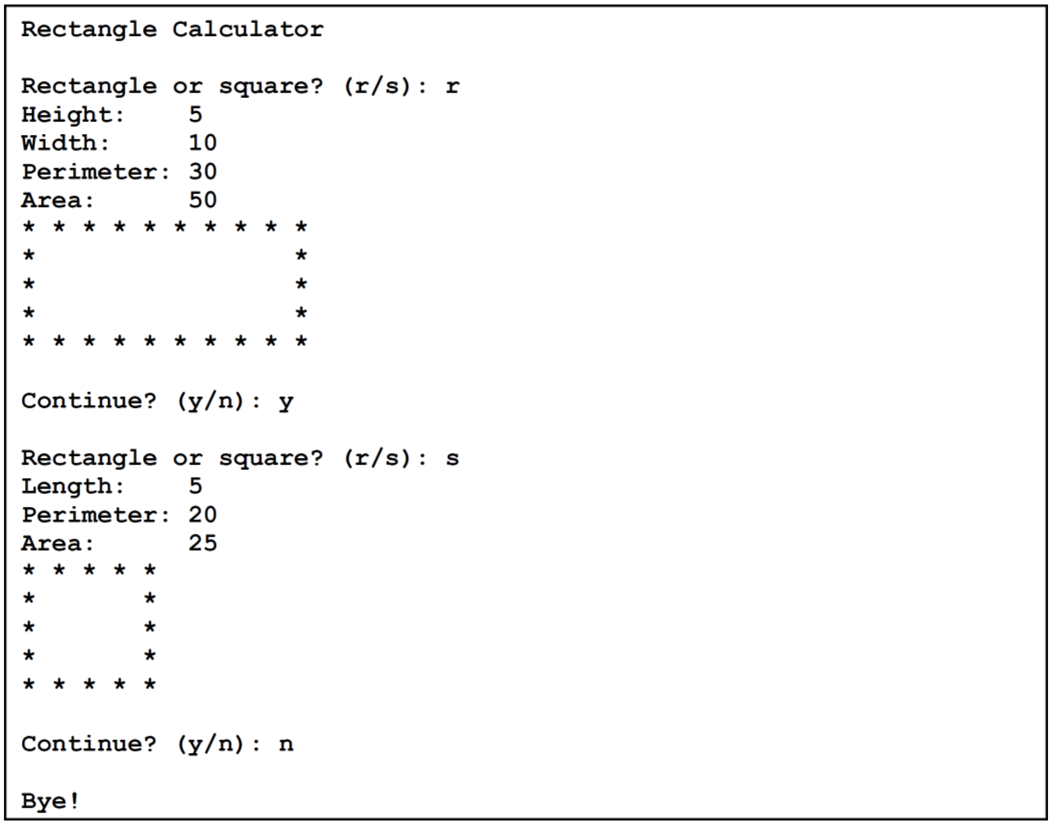
Validating Code for the objects.py & product_viewer.py. (10 points)

Extra Credit

To get full points for each extra credit, you must include screen captures of the running output as well as the python (.py) code files.

Extra Credit #1 – Rectangle or Square Calculator (+1 Extra Credit)

Create an object-oriented program that uses inheritance to perform calculations on a rectangle or a square.



```
Rectangle Calculator

Rectangle or square? (r/s): r
Height:    5
Width:     10
Perimeter: 30
Area:      50
* * * * * * * * * *
*                 *
*                 *
*                 *
* * * * * * * * * *

Continue? (y/n): y

Rectangle or square? (r/s): s
Length:    5
Perimeter: 20
Area:      25
* * * * *
*       *
*       *
*       *
* * * * *

Continue? (y/n): n

Bye!
```

Specifications:

- Use a Rectangle class that provides attributes to store the height and width of a rectangle. This class should also provide methods that calculate the perimeter and area of the rectangle. In addition, it should provide a __str__ method that returns a string representation of the rectangle.
- Use a Square class that inherits the Rectangle class. This class should set the height and the width of the Rectangle superclass to the length that's set in the Square subclass.
- The program should determine whether the user wants to enter a rectangle or a square.
- For a rectangle, the program should get the height and width from the user.
- For a square, the program should get the length of the square from the user.

Extra Credit #2 – Roshambo (+1 Extra Credit)

Create an object-oriented program for a Roshambo game where the user can choose to, compete against one of two computer players: Bart or Lisa..

```
Roshambo Game

Enter your name: Joel

Would you like to play Bart or Lisa? (b/l): b

Rock, paper, or scissors? (r/p/s): r

Joel: rock
Bart: rock
Draw!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): p

Joel: paper
Bart: rock
Joel wins!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): s

Joel: scissors
Bart: rock
Bart wins!

Play again? (y/n): n

Thanks for playing!
```

Specifications:

- Create a class named Player that provides attributes for storing the player's name and Roshambo value.
- Create a class named Bart that inherits the Player class and adds a generateRoshambo method. This method should set the Roshambo attribute to rock.
- Create a class named Lisa that inherits the Player class and adds a generateRoshambo method. This method should randomly select rock, paper, or scissors.
- The program should allow the user (Player) to play Roshambo against Bart or Lisa.
- In the game of Roshambo, rock beats scissors, paper beats rock, and scissors beats paper.