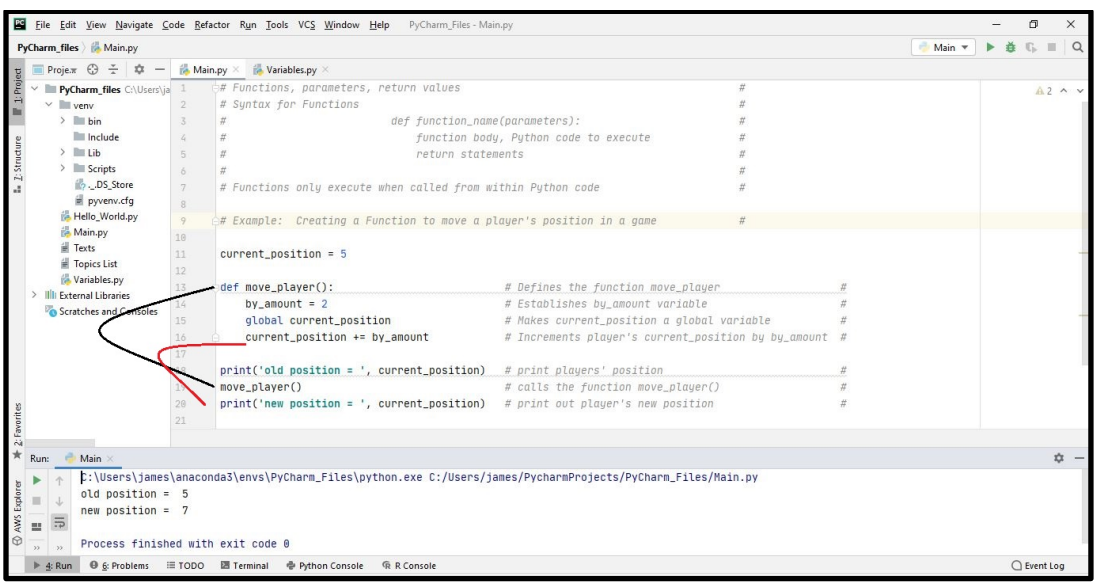
Functions, Classes, & Objects

Python Built-in and User Defined Functions:

1) A function is a block of organized, reusable code that can perform a single, related action. Functions provide better modularity for your application and a high degree of code reusability. Python gives you many built-in functions like print(). For a complete listing of Python built-in functions, please refer to the URL: https://docs.python.org/3/library/functions.html In Python, you can also create your own user defined function. You can define functions to provide the required functionality. Here are simple rules to define a function in Python:
   a) Function blocks begin with the keyword def followed by the function name and parentheses ( ).
   b) Any input parameters or arguments should be placed within the parentheses. You can also define parameters inside these parentheses.
   c) A colon (:) terminates the function name and begins the indented code block.
   d) The first statement of a function code block can be *an optional statement - the documentation string of the function or docstring.*
   e) An optional statement is a return [expression] that exits a function, while optionally passing back an expression to the caller. A return statement with no arguments is the same as return None. Not having a return [expression] simply returns control to the caller.
   f) Syntax:

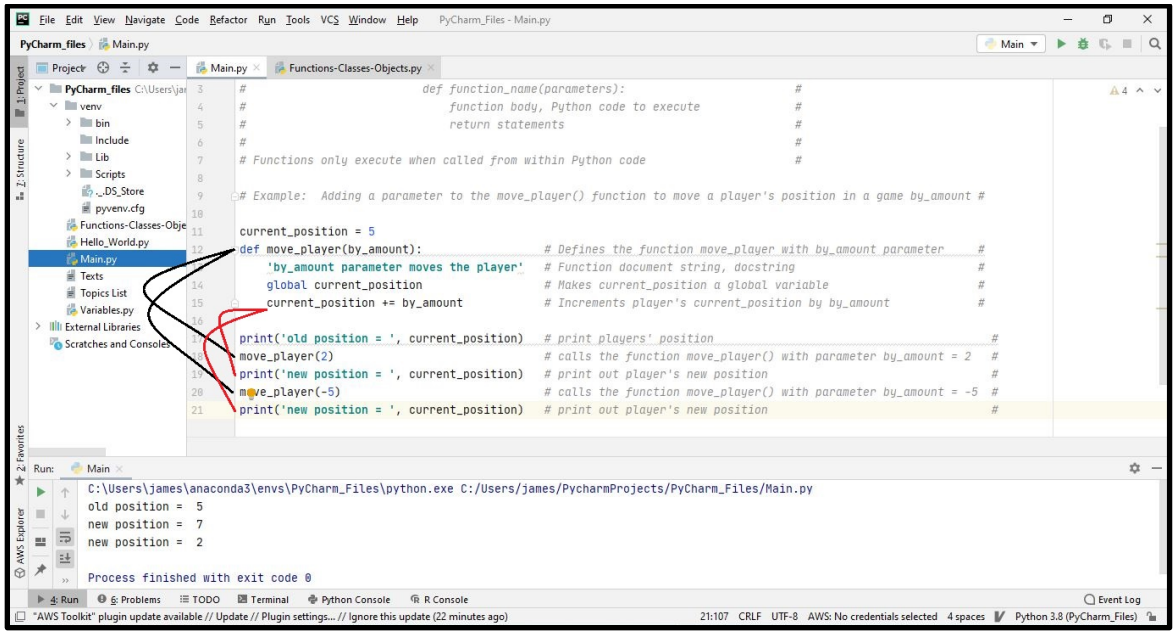
```
def function_name( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

   g) Example of a function move_player() that moves a player's position in a game:

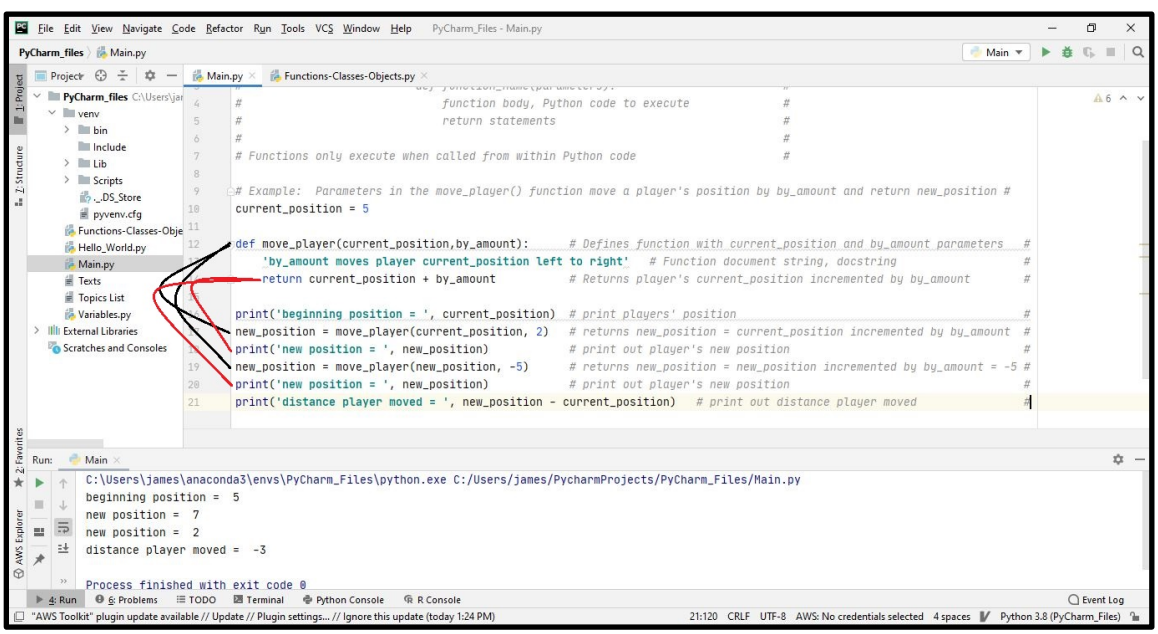


The image above can be viewed at 400% magnification for details of the screen shots.

2) In the prior move_player() function example, the by_amount parameter was static at by_amount = 2. However, the move_player() function should be able to move players left or right by more or less than 2. The following example shows how to use by_amount as a parameter of the move_player() function in order to move players' positions left or right:

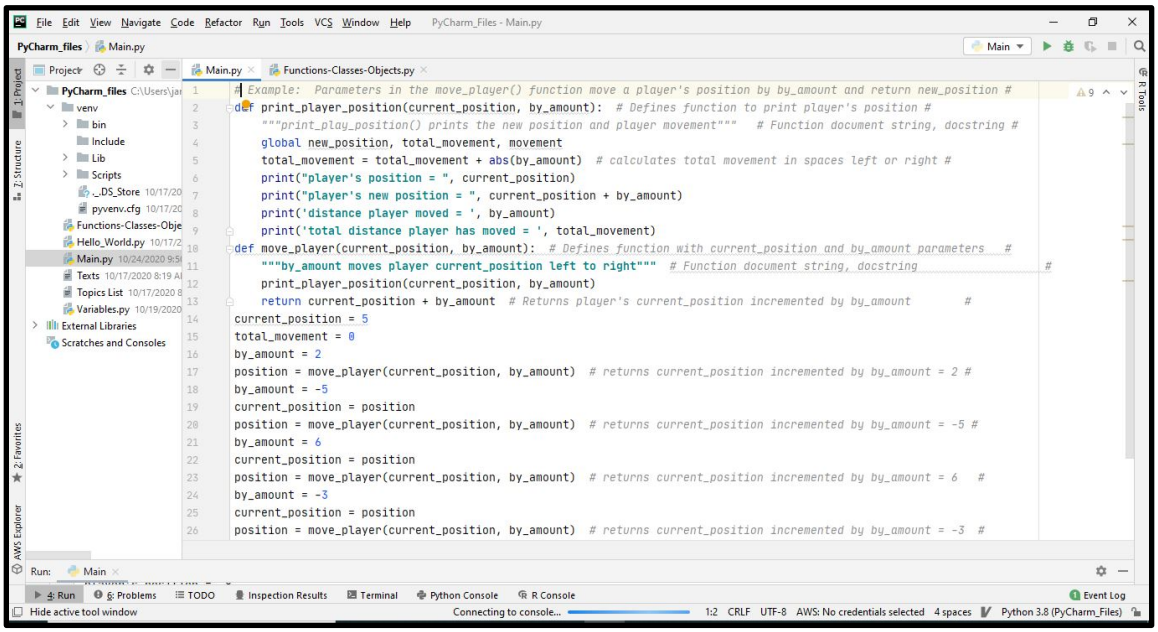


3) In the prior player_move() function examples, we kept up with a current_position variable that moved a player to the new position by_amount. We can have the player_move() function pass the position and by_amount as parameters and then return the new_position value. The following example shows how to use position and by_amount parameters in the move_player() function in return the players' positions left or right:

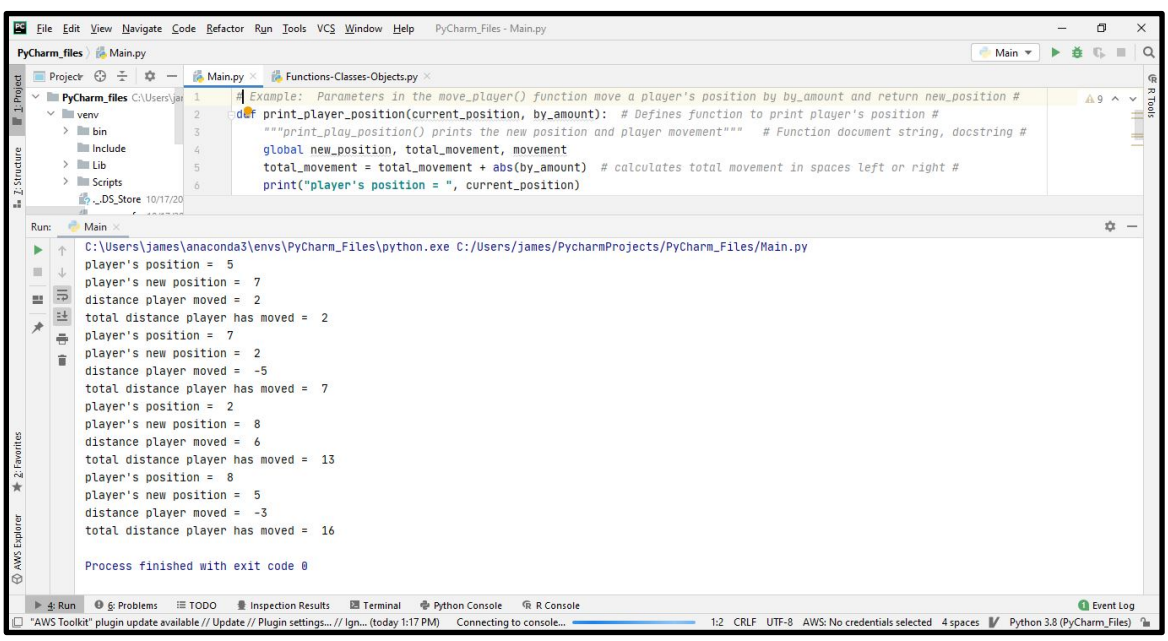


4) In the prior example, we defined a function() with two parameters that returned the new position of a player in a game. It should be noted that we can nest user defined functions within other user defined functions. The following example uses the same move_player() function from the prior example and includes a *nested defined function* print_player_position() that prints the player's position, new position, movement, and total movement. The PyCharm screenshots below have the Python code in the first and the executed Python code in the second.

Python Code Example:



Python Code Executed Example:



5) **Exercise deliverable:** Construct two examples of user_defined functions to perform a task and then execute the function in Python code. The first user-defined function example should have a parameter and return a value. The second user-defined function example should have a user-defined function nested within a user-defined function. Take a PyCharm screenshot of each user-defined function example.

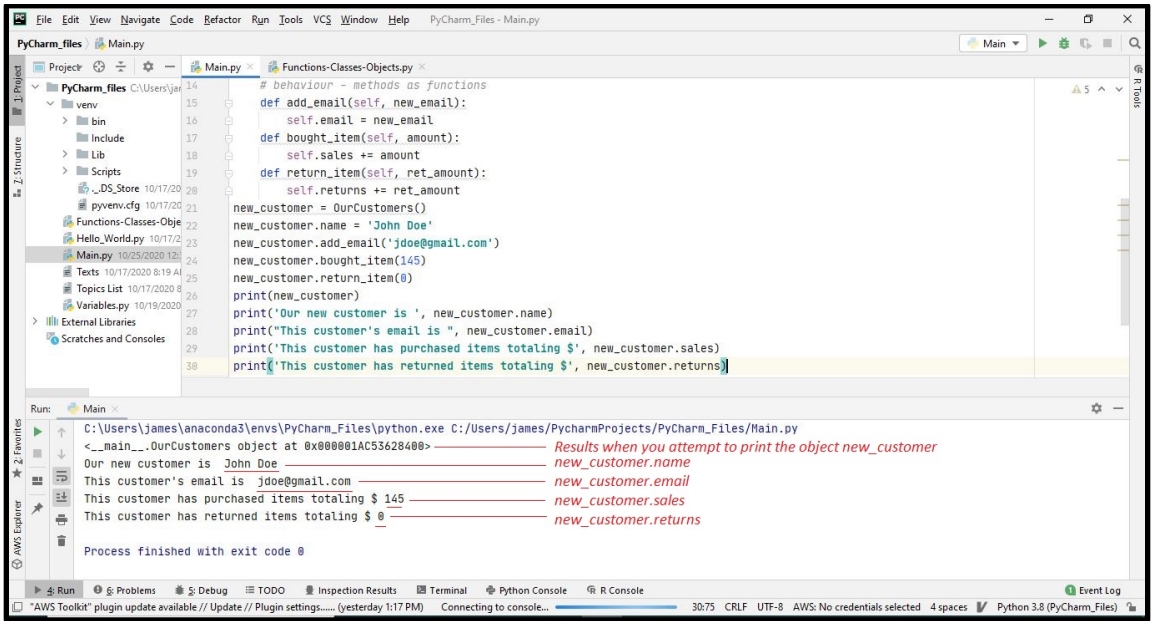
Fundamentals of Classes & Objects in Python:

1) Classes are blueprints of objects. A class is code representation of an object that identifies state and behavior. We will not delve into sub-classes and super-classes in MIS 502, but sub-classes and super-classes do exist in Python. An object is anything in Python code that has state and behavior, so an object will have attributes, behavior, and a state that we want to monitor. For our example, we will create an OurCustomers class. OurCustomers have attributes of name, email, sales, and returns. We have to have a customer name for an OurCustomers instance. OurCustomers have functions of add_email(), bought_item(), and return_item(). Review the following Python code:

```
# Classes and Objects   #
class OurCustomers:
    # attributes - representations of the class stored in variables #
    #(e.g., strings, dictionaries, arrays, or objects) #
    name = ''
    email = ''
    sales = 0
    returns = 0
    # initializer - ways to create new instances of an object #
    def __int__(self, name):
        self.name = name
        self.email = email
        self.sales = sales
        self.returns = returns
    # behaviour - methods as functions #
    def add_email(self, new_email):
        self.email = new_email
    def bought_item(self, amount):
        self.sales += amount
    def return_item(self, ret_amount):
        self.returns += ret_amount
new_customer = OurCustomers()
new_customer.name = 'John Doe'
new_customer.add_email('jdoe@gmail.com')
new_customer.bought_item(145)
new_customer.return_item(0)
print(new_customer)
print('Our new customer is ', new_customer.name)
print("This customer's email is ", new_customer.email)
print('This customer has purchased items totaling $', new_customer.sales)
print('This customer has returned items totaling $', new_customer.returns)
```

Note that in creating an instance of OurCustomers in the Python code above, we added the attributes or field variables using the class.attribute syntax. For example, we added the new_customer.name as 'John Doe.' We used functions to add the customer's email, attributed sales, and dollar value of item returns. The following PyCharm screenshot has the Python ode above executed for the

new_customer.name = 'John Doe.' Try copying the Python code above into your PyCharm main.py file and execute it. Your results should be the same as the following screenshot.



2) **Exercise deliverable:** Construct two examples of classes' and objects' Python code and then execute them. The first class and object example should use the Python code above and change the instance attribute values of the object new_customer as well as add a new attribute to the OurCustomers class that also has a behavior to manipulate the attribute. For the second class and object example, create a new class with attributes, initializer, behaviors, and associated object instance. Take a PyCharm screenshot of each class and object example.

Deliverables from Exercises Above:

Take the noted deliverables (i.e., PyCharm screenshots) from each exercise above (i.e., total of 4 screenshots), insert the screenshots into a MS Word document as you label each screenshot deliverable from the corresponding exercise, save the document as one PDF, and submit the PDF to the Tutorial Six Canvas Assignment uplink having the following qualities:

a. Your screenshot for each of the two exercises above is included.
b. Each screenshot is labeled appropriately for each exercise.
   i. Python User-defined Functions ..... 2 screenshots
   ii. Classes & Objects in Python ..... 2 screenshots
   Total PyCharm screenshots to submit ..... 4 screenshots
c. All screenshots are legible output for each PyCharm exercise deliverable.