CS-119 Lab #7

Expected Learning Objectives

- Creating applications in Python.
- Creating class models
- Creating user defined classes
- More experience with Python code in general.

Overview

This lab provides an opportunity to apply the concepts and principles presented in PLD chapter 7. Primarily, we will be focusing on creating user defined classes.

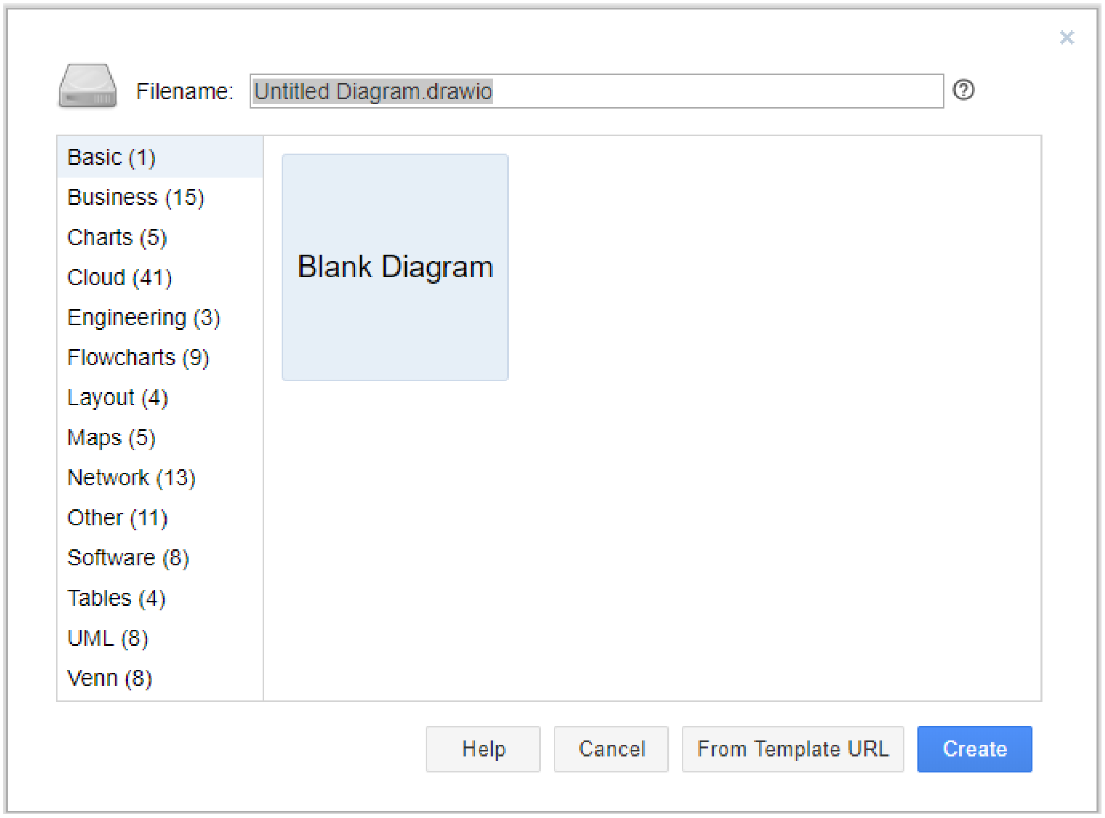
Since we're spending 2 weeks on this chapter, there are 2 guided exercises (1 for each week) to give some hands on experience creating user defined classes in Python. We will create class models and then code the classes in Python as well as implement a simple *test fixture* application where we can make sure the class functions as specified and coded. In real world software development, it is a common practice to test custom classes this way before integrating into a large scale project because it makes testing and debugging a whole lot easier. Once you complete the guided exercises, you will get a couple user defined classes to code on your own.

Exercise 1 Magazine Subscription class

This first exercise will be for a magazine subscription class based on exercise 7 on page 292 in the book. To keep things quick and simple, there are just a few properties given in the exercise. In a real world application, there would probably be quite a few more. When designing user defined classes, UML class models are used to define the class. We'll create these using Draw.io. We'll cover the Unified Modeling Language (UML) in more detail in chapter 11.

Creating the Class Model

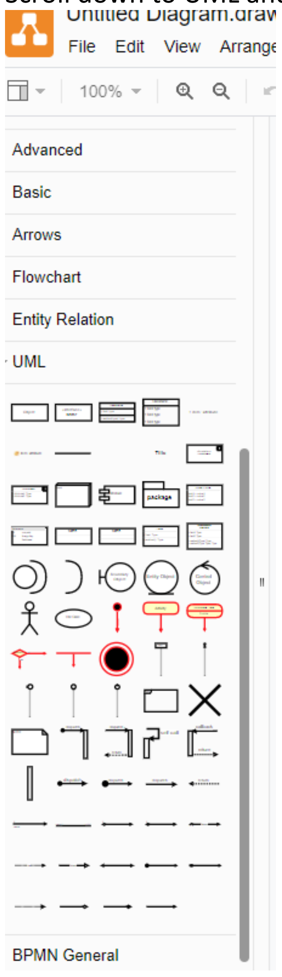
Navigate a web browser to draw.io, select Blank Diagram and click the Create button. You can save to a file later.



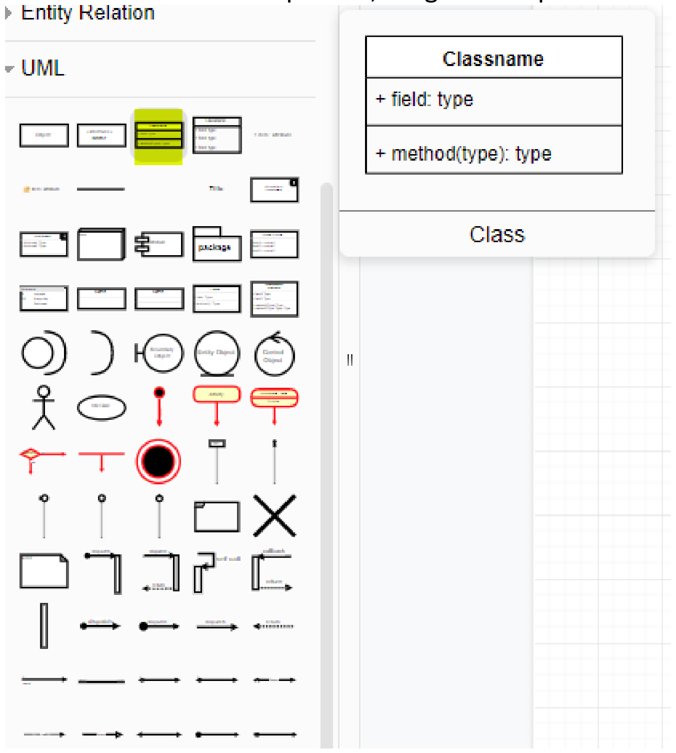
Filename: Untitled Diagram.drawio

Basic (1)
Business (15)
Charts (5)
Cloud (41)
Engineering (3)
Flowcharts (9)
Layout (4)
Maps (5)
Network (13)
Other (11)
Software (8)
Tables (4)
UML (8)
Venn (8)

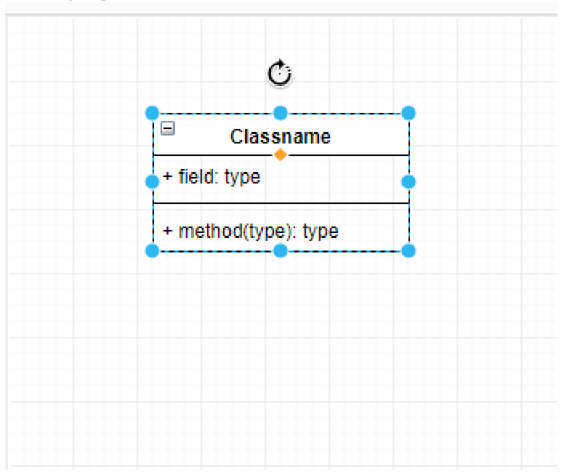
Scroll down to UML and click the arrow to expand the palette.



Locate Class in the tool palette, drag and drop a class template onto the page.



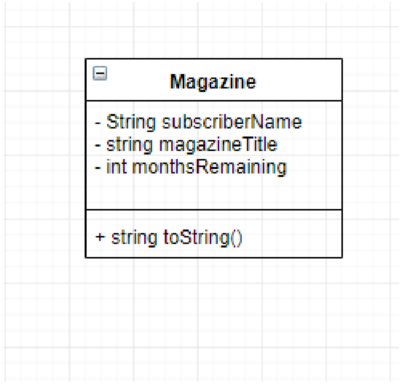
Your page now looks like this:



Quick review (text book page 262): Notice that a class is split into 3 compartments. The top compartment is the class name, the middle compartment contains the attributes or properties, and the bottom contains the methods or operations. Click inside each compartment and edit as needed. We'll shorten the class name to Magazine to save some typing. For attributes and methods, a minus (-) prefix means that member is private and a plus (+) indicates public. Attributes are normally declared private and methods are normally public unless they are used only in the internal workings of the class. Optionally, you can specify the data type as shown in the sample Magazine class below. We'll have a method called toString() that will return a formatted string of the subscriber name, magazine title and months remaining. The book also shows the getters and setters in the methods compartment. Some software developers omit these in a class model because it's a given they exist and, if there's a large number, they can make a class model very large and unwieldy in a hurry. Here's our completed class. If you want to add the getter and setter methods in, that's perfectly fine.

You can click on each individual compartment and make larger or smaller as needed.

Your completed Magazine class should look (roughly) something like this:



Be sure to save your work. If do a file save as, you can save the file for editing later. You may add the class models for the other exercises in this lab to this page. Be sure to save to your local drive or online cloud storage if you have that. Note the various save options available with draw.io.

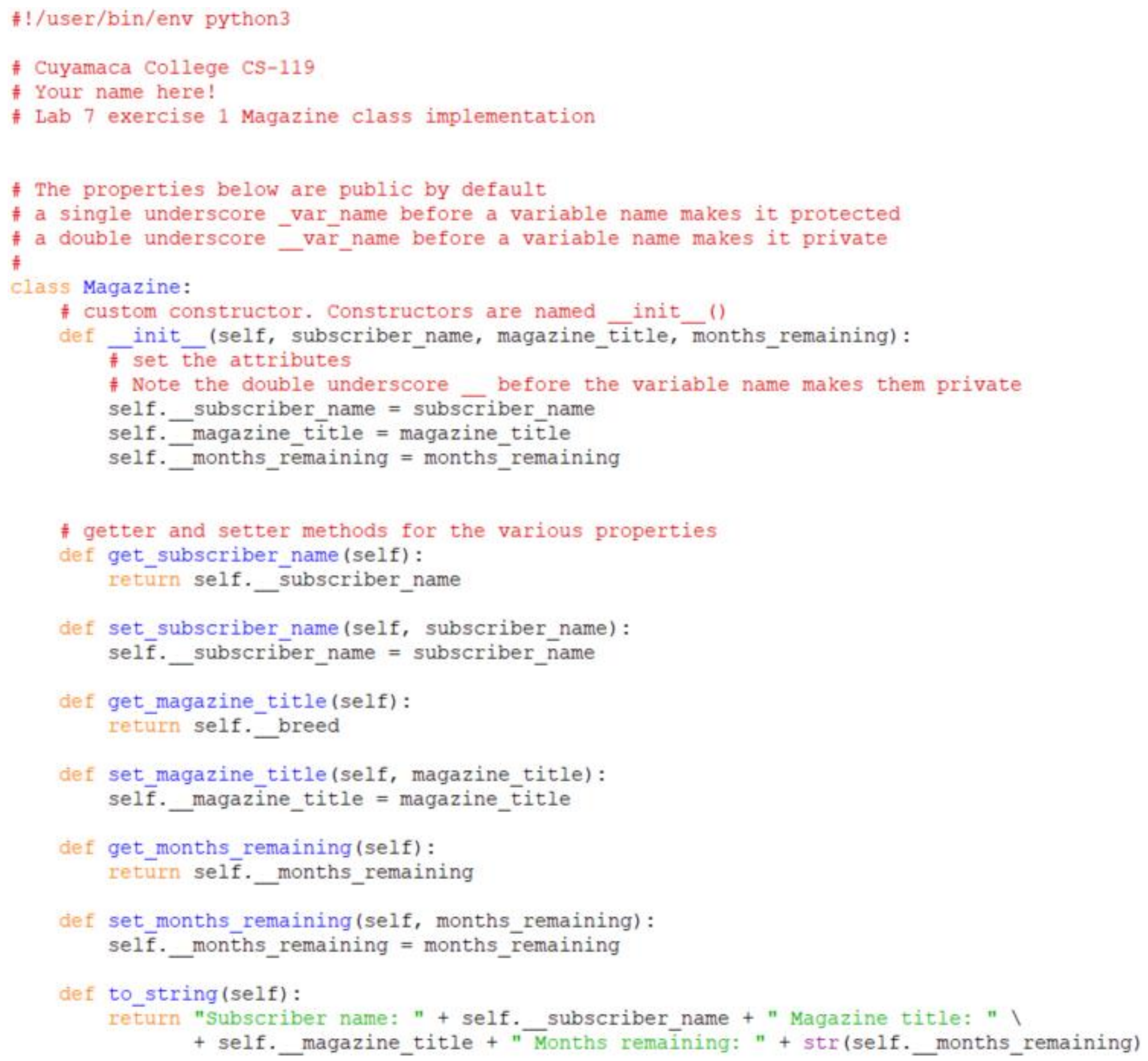
Once you have completed class models for all the lab exercises, export to either a PDF or JPEG file. Be sure to save it in your Lab 7 folder.

Python Code for the Magazine Exercise

Create a new file in the Python IDLE and save it as Magazine.

When coding your class in Python, use your class model as a “cheat sheet”. This is why we did all the work of creating it! Keep in mind some of the Python demo code in the lecture notes. Is there something there you could “borrow” from? You are given the code for this exercise.

Note the use of the *this* reference throughout the code although it is referred to as *self* in Python.



```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 7 exercise 1 Magazine class implementation


# The properties below are public by default
# a single underscore _var_name before a variable name makes it protected
# a double underscore __var_name before a variable name makes it private
#
class Magazine:
    # custom constructor. Constructors are named __init__()
    def __init__(self, subscriber_name, magazine_title, months_remaining):
        # set the attributes
        # Note the double underscore __ before the variable name makes them private
        self.__subscriber_name = subscriber_name
        self.__magazine_title = magazine_title
        self.__months_remaining = months_remaining


    # getter and setter methods for the various properties
    def get_subscriber_name(self):
        return self.__subscriber_name

    def set_subscriber_name(self, subscriber_name):
        self.__subscriber_name = subscriber_name

    def get_magazine_title(self):
        return self.__breed

    def set_magazine_title(self, magazine_title):
        self.__magazine_title = magazine_title

    def get_months_remaining(self):
        return self.__months_remaining

    def set_months_remaining(self, months_remaining):
        self.__months_remaining = months_remaining

    def to_string(self):
        return "Subscriber name: " + self.__subscriber_name + " Magazine title: " \
            + self.__magazine_title + " Months remaining: " + str(self.__months_remaining)
```

Code a “Test Fixture” for the Magazine Class

The next step is to code a test fixture where we can create an instance, set the various properties and make sure things work as expected. You are given the code for this.

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 7 exercise 1 Magazine class test fixture

# import the Magazine class
from Magazine import Magazine

subscriber_name = input("Enter subscriber name: ")
magazine_title = input("Enter magazine title: ")
months_remaining = int(input("Enter months remaining: "))

# create an instance of a Magazine
my_magazine = Magazine(subscriber_name, magazine_title, months_remaining)

# invoke the to_string() method and display everything
print(my_magazine.to_string())
```

Be sure to test your application.

```
====== RESTART: C:\Cuyamaca\2019FallCS119\LabsDev\Lab07\MagazineTest.py ======
Enter subscriber name: John Smith
Enter magazine title: PC Week
Enter months remaining: 6
Subscriber name: John Smith Magazine title: PC Week Months remaining: 6
>>>
```

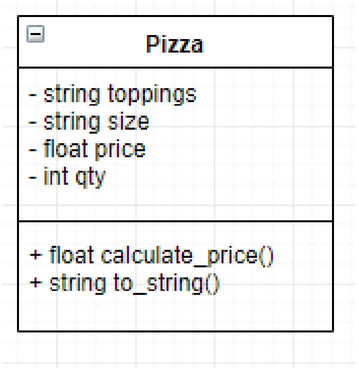
Pizza Demo

This case study is based on exercise 8 on page 292 in the text. To prepare you for an upcoming lab exercise, we'll add a quantity property in addition to the toppings, size and price properties given in the text. Instead of a numeric size (i.e. 12 inches, etc.), we'll go with small, medium and large for sizes so size will be a string value instead of a numeric type. We'll also add a calculatePrice() instance method. Another thing to notice is that price will be a read-only property. When we set the size property, we will look up and set the price.

We'll go through all the same steps as the magazine exercise although things won't be quite as detailed this time around.

Pizza Class Model

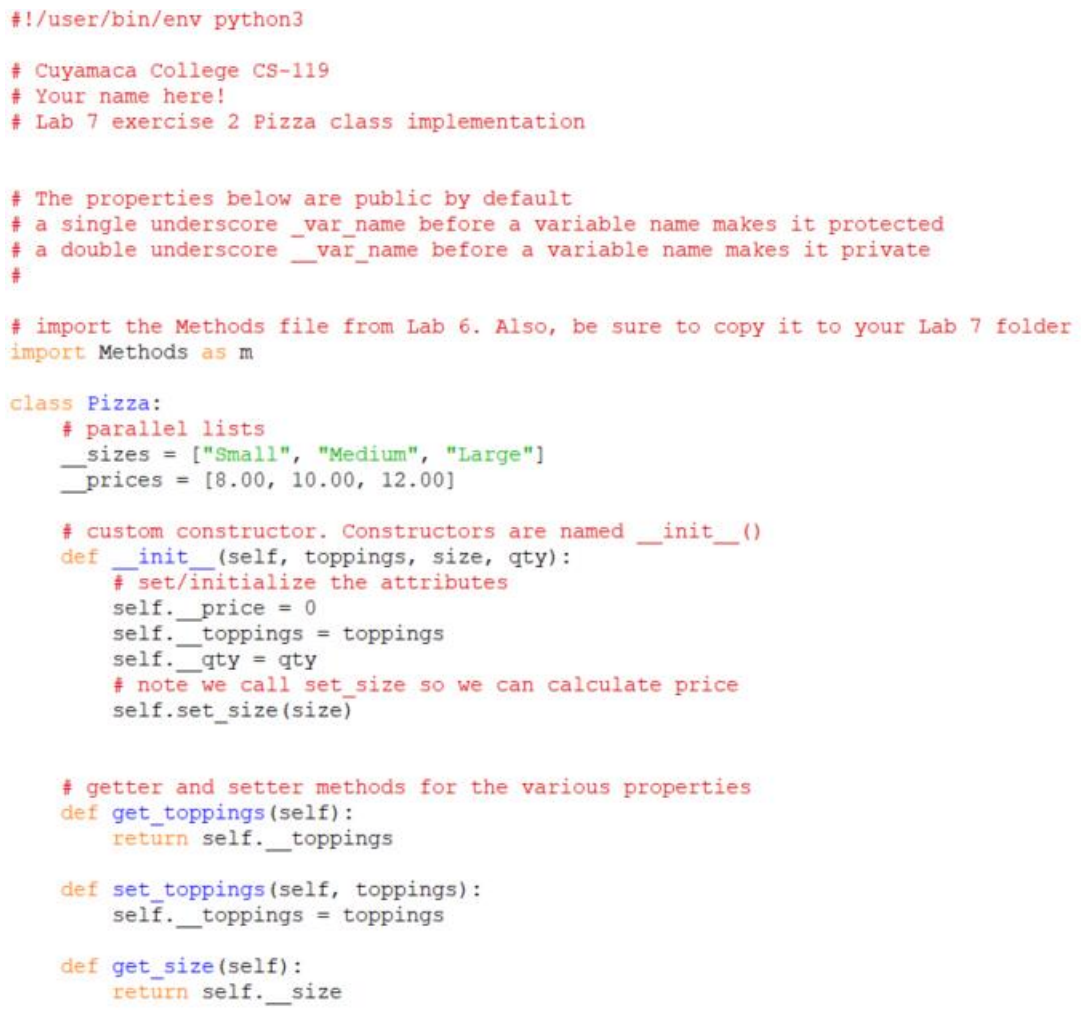
Using draw.io, open the class model you completed for the previous exercise. Add a Pizza class to the page. Our class model should look something like this:



A helpful embellishment to our methods is the return type of each method.

Coding the Pizza Class

Create a new Python file called Pizza and a new class called Pizza. Refer to the last exercise if you need detailed instructions on how to create a class, implement the properties, constructors, getters/setters, etc. Note the validation code added to setQty() and how setSize() also sets the price. Head's up, there are a few twists this time. The first one is that you'll need to copy the Methods.py file you created in Lab 6. You will be reusing the find_string() method. Import it to your Pizza.py file.

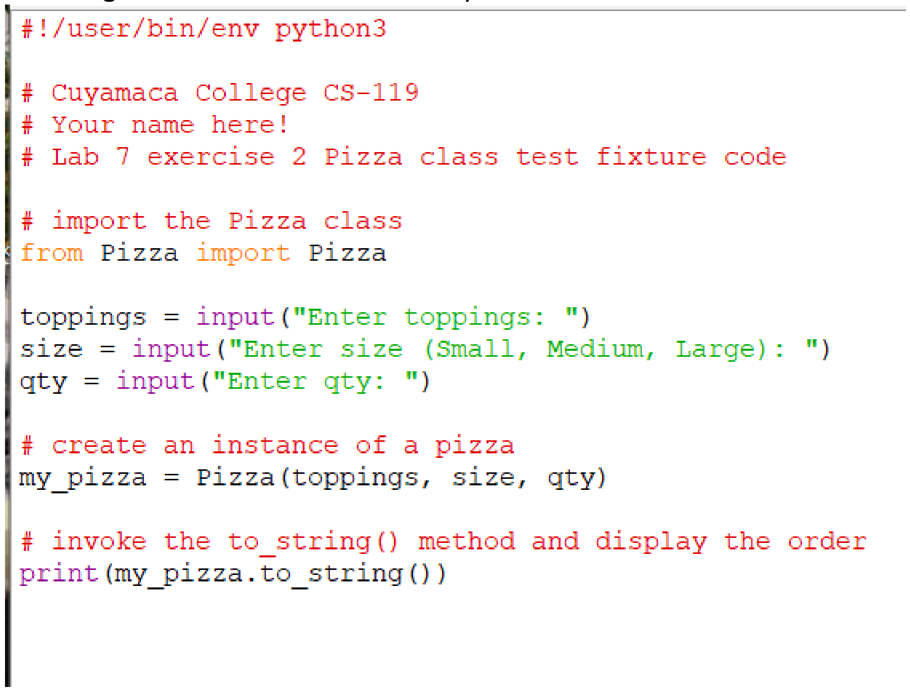


```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 7 exercise 2 Pizza class implementation


# The properties below are public by default
# a single underscore _var_name before a variable name makes it protected
# a double underscore __var_name before a variable name makes it private
#

# import the Methods file from Lab 6. Also, be sure to copy it to your Lab 7 folder
import Methods as m

class Pizza:
    # parallel lists
    __sizes = ["Small", "Medium", "Large"]
    __prices = [8.00, 10.00, 12.00]

    # custom constructor. Constructors are named __init__()
    def __init__(self, toppings, size, qty):
        # set/initialize the attributes
        self.__price = 0
        self.__toppings = toppings
        self.__qty = qty
        # note we call set_size so we can calculate price
        self.set_size(size)


    # getter and setter methods for the various properties
    def get_toppings(self):
        return self.__toppings

    def set_toppings(self, toppings):
        self.__toppings = toppings

    def get_size(self):
        return self.__size
```

```python
def set_size(self, size):
    # Use our find_string() method from lab 6
    idx = m.find_string(self.__sizes, size)
    if idx >= 0:
        self.__price = self.__prices[idx]
        self.__size = size
    else:
        # Not found - default to small. An arguably better approach would
        # be to throw an exception. We'll visit that topic in chapter 10...
        self.__size = __sizes[0]
        self.__price = __prices[0]
# end set_size()

def get_qty(self):
    return self.__qty

# Why we have setter methods: It doesn't make sense to have a negative quantity
# "Sanity check" the quantity and force a negative value to 0
def set_qty(self, qty):
    if qty < 0:
        qty = 0
    self.__qty = qty

def get_price(self):
    return self.__price

# instance methods
def calculate_price(self):
    return self.__price * float(self.__qty)


def to_string(self):
    # calculate an extended price which will be our total
    ext_price = self.calculate_price()
    return "Toppings: " + self.__toppings + " Size: " \
           + self.__size + " Qty: " + str(self.__qty  \
           + " Total price: $" + str(ext_price))
```

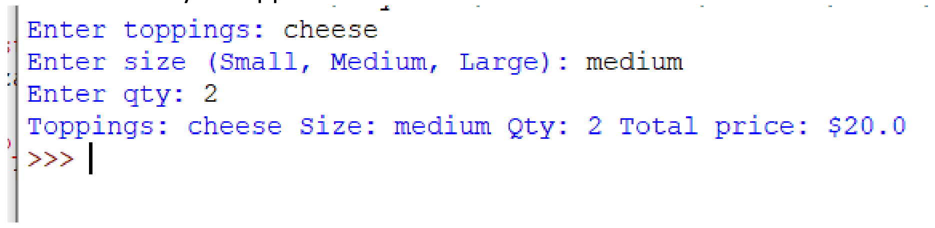
The Test Fixture

You are given the code. Notice it is very similar to the last exercise.



```python
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 7 exercise 2 Pizza class test fixture code

# import the Pizza class
from Pizza import Pizza

toppings = input("Enter toppings: ")
size = input("Enter size (Small, Medium, Large): ")
qty = input("Enter qty: ")

# create an instance of a pizza
my_pizza = Pizza(toppings, size, qty)

# invoke the to_string() method and display the order
print(my_pizza.to_string())
```

Be sure to test your application.



```
Enter toppings: cheese
Enter size (Small, Medium, Large): medium
Enter qty: 2
Toppings: cheese Size: medium Qty: 2 Total price: $20.0
>>> 
```

Exercise 3 Checking Account

In this exercise, you will design and build an application that allows the user to enter information for a checking account and then display it in. To do this, you will create a new class called CheckingAccount that will contain properties for the account number, customer name, customer address and initial balance.

1. Complete a class model in draw.io. You may add the CheckingAccount class model to the class models you did in exercises 1 and 2.
2. Create a Python code file in your Lab 7 folder called CheckingAccount.py
3. Code an appropriate constructor, getter methods, setter methods, the to_string() method, debit() and credit() methods. There is code you can “borrow” from the first 2 exercises.
4. The to_string() method must return a formatted string with the values for all the properties above.
5. Create a new Python code file called CheckingAccountTest.py. This will be the test fixture application.
6. Your test fixture code must have prompts for account number, customer name, address and initial balance. Additionally, you must have a prompt to debit an amount and a prompt to credit the account. Use the to_string() method to display the account information.
7. Be sure to test your work.

Exercise 4

This exercise is very similar to the first exercise and the pizza demo. You will create a custom class called Burger that will have the following properties: Customer name, order No., burger type, quantity ordered and price. You will implement get() and set() methods for these along with a constructor.

1. Complete a class model in draw.io. You may add your class model for this exercise to the class models you created in the previous exercises.

2. Create a new Python file and class called Burger. Review the steps and code in the previous exercises, if needed.

3. Implement an appropriate constructor, getter methods and setter methods.

4. Price is computed based on the type of burger ordered and then multipled by the quantity ordered. There is no sales tax. Burger types and prices are in the table below. There is room to be creative if you want to use some favorites of your own.

Burger Type	**Price**
Regular	$1.75 each
Cheese	$2.00 each
Double meat	$3.50 each
Double meat, bacon & cheddar	$4.50 each

5. Implement a public method called toString() that will return a string containing customer name, order No., type, qty, and total price.

6. Create a new Python code file called BurgerTest.py. It must prompt for customer name, order No., burger type, and quantity ordered.

7. Test your application. Make sure it calculates properly for each burger type. Test with quantities greater than 1 to make sure the extended price (price * qty) is calculating properly.

Grading Criteria:

Deliverable	Points	Breakdown
Ex.1 Magazine Class model	3	Completed, all properties and methods in the model.
Magazine class code	3	All properties, constructors and methods implemented per lab handout.
Magazine class test fixture	3	Coded, prompts are clear, appropriate variable names, data types, use of comments
Magazine run	3	Produces correct output
Ex.2 Pizza Class model	3	Completed, all properties and methods in the model.
Pizza class code	3	All properties, constructors and methods implemented per lab handout.
Pizza class test fixture	3	Coded, prompts are clear, appropriate variable names, data types, use of comments
Pizza run	3	Produces correct output
Ex.3 Checking Account Class model	3	Completed, all properties and methods in the model.
Checking account class code	4	All properties, constructors and methods implemented per lab handout.
Checking account test fixture	3	Coded, prompts are clear, appropriate variable names, data types, use of comments
Checking account run	3	Produces correct output
Ex.4 Burger Class model	3	Completed, all properties and methods in the model.
Burger class code	4	All properties, constructors and methods implemented per lab handout.
Burger test fixture	3	Coded, prompts are clear, appropriate variable names, data types, use of comments
Burger run	3	Produces correct output
Lab Total	**50**	