Networking for Software Developers

# Lab 2A – Pizza Class.

We will continue working on python. Using a google colab notebook write the python statement to define a Pizza class. The test harness is provided to you by the instructor as well as the resulting output given.

**Maximum points for this exercise is 18 marks.**

## Topics covered:

* List
* Dictionary
* Class
  + Constructor
  + Property
  + Methods
* Exception

## How to do this assignment.

From the test harness and the given output, try to deduce the definition of the Pizza class. Code this class in a colab notebook and copy the test harness to a cell below. Execute the notebook and ensure that the output matches EXACTLY with the output on the following page.

You must use python f-strings for your output.

## Deductions.

Do not modify the test harness.

Do not add frivolous members or un-necessary logic.

## Documentation.

Because the code is so simple no code documentation is required, however you must put your name and the current date somewhere at the top of your code.

## How to submit this assignment.

Make the notebook shareable and submit the link to the course assessment folder.

See the folder documentation for due date.

See the course documentation for policy on deadlines.

# Specifications for the Pizza class.

The implementation of this class is almost simplistic! All of the methods (except two) are only single code statements. My class implementation is only 30 lines of code.

### Class attributes

Remember that class members do not have a "self" prefix and are accessed with the class name and the dot operator.

1. Valid sizes for pizza contained in a collection at the class level.   
   You get to decode on the type of collection to store the sizes below.  
   Valid sizes are small, medium, large and x-large.
2. Another class level collection having the prices for each valid size.   
   You get to decide on the type of collection to store the prices below.   
   Prices are small: $6.49, medium: $8.49, large: $10.49, x-large: $13.49.

### Constructor

Instance methods have an implicit first argument "self".

1. A constructor that takes a default size of medium and topping of a list of cheese that does the following:
   1. Set the first argument size to the instance attribute size. Size must be verified. This can be done by setting the size property.
   2. Creates an instance attribute a list with the second argument. If the second argument is missing, a single cheese topping is inserted in the list.

### Instance methods

Instance methods have an implicit first argument "self".

1. A method that takes an argument of type list of strings. It adds topping to the list of pizza toppings.
2. Implement the \_\_str\_\_() method to return a formatted string. Examine the output from the test harness for clues on how to implement this method.

### Instance Properties

Instance properties have an implicit first argument "self".

1. A property that returns the price of the pizza.
   1. Price is based on the size as well as the number of toppings.   
      See spec#2 for cost based on size. Each topping cost an additional $0.50 each.
2. A property that returns the size of the pizza.
3. A property that sets the size of the pizza.
   1. Size must be verified. A ValueError exception is raised if the size is invalid.

### Instance Attributes

All of the following Instance attributes are initialized in the constructor from the values of the argument. Notice the \_\_ prefix to make is private.

1. \_\_size is a str that stores the size of this object. This is initialized in the constructor. It is mutated by the property in #8. It is returned in #7. It is used in #6 to calculate the cost of the pizza.
2. \_\_toppings a list of string that represents the toppings for this object. This is initialized in the constructor. It is mutated in the add() method and is used in #6 to calculate the cost of the pizza.

# The test harness.

You may not change the test harness.

print(f'Creating a default pizza')

p = Pizza()

print(p)

toppings = 'cheese olive'.split()

print(f'\nAdding topping: {toppings}')

p.add(toppings=toppings)

print(p)

print(f'\nCreating a new pizza')

p = Pizza('large', 'cheese pepper'.split())

print(p)

toppings = ['pineapple', 'mushroom']

print(f'\nAdding topping: {toppings}')

p.add(toppings)

print(p)

size = 'x-large'

p.size = size

print(f'\nChanging order size to {size}')

print(p)

size = 'gigantic'

print(f'\nChanging order size to {size}')

try:

p.size = size

except ValueError as err:

print(err)

# The resulting output

Creating a default pizza

medium pizza with ['cheese'] for $8.99

Adding topping: ['cheese', 'olive']

medium pizza with ['cheese', 'cheese', 'olive'] for $9.99

Creating a new pizza

large pizza with ['cheese', 'pepper'] for $11.49

Adding topping: ['pineapple']

large pizza with ['cheese', 'pepper', 'pineapple', 'mushroom'] for $12.49

Changing order size to x-large

x-large pizza with ['cheese', 'pepper', 'pineapple', 'mushroom'] for $15.99

Changing order size to gigantic

ERROR: gigantic is not a valid size for a pizza