Practice Problem Set – Abstract Classes, Exceptions

1. Define a class `Polygon` to model polygons in two dimensional space having two attributes:
   `noOfSide`: stores number of sides for the polygon as an integer.
   `sideLengthList`: stores length of sides of a polygon as a list.
   Define a constructor to initialize both instance variables. Also add a method `perimeter` to find the perimeter of the polygon by adding all the side lengths stored in `sideLengthList`.
   Create property called `lengths` to manipulate `sideLengthList` instance variable. Write some test code to check the functionality of the class.

2. In problem 1, alter the `perimeter` method to work on `lengths` instead of `sideLengthList`. Does it work?

3. Define an abstract class `Polygon` to model polygons in two dimensional space having two attributes:
   `noOfSide`: stores number of sides for the polygon as an integer.
   `lengths`: stores length of sides of a polygon as a list. Make this attribute abstract by defining only a getter method with the name `lengths`. The getter method should contain only one statement: `pass`.
   Define constructor to initialize only the concrete attribute. Also add a method `perimeter` to find the perimeter of the polygon by adding all the side lengths stored in `lengths`.

4. Derive a concrete class `Triangle` from the `Polygon` class developed in problem 3. Define a constructor which sets the attributes `lengths` and `noOfSides`. `noOfSides` should be set by calling the constructor of the parent. Run the following test code to check the functionality of your code:
   ```
   base=4
   height=7
   p=Triangle([base, height, hypotenuse]) #calculate hypotenuse
   print(p.perimeter())
   ```

5. To the `Polygon` abstract class developed in problem 3, add an abstract method `area`. Add implementation of this abstract method in class `Triangle` developed in problem 4.

6. Add another subclass `Rectangle` to the Polygon abstract class of problem 3. Initialize a `Rectangle` object to any length and breadth values. Print area and perimeter of this rectangle.

7. List all local, instance and class variables for the three classes Polygon, Triangle and Rectangle, developed in problems 3, 5 and 6

8. Answer the following questions:
   a. What is the maximum number of except blocks associated with a try statement?
   b. What is the minimum number of except blocks associated with a try statement?
   c. What is the maximum number of else blocks associated with a try statement?
   d. What is the minimum number of else blocks associated with a try statement?
   e. What is the maximum number of finally blocks associated with a try statement?
   f. What is the minimum number of finally blocks associated with a try statement?

9. Wrap the following code in a try statement to defend against any exceptions it can raise. Do not use a catch-all handler.
   ```
   lst = [0, 0, 0, 0]
   with open('data.txt', 'r') as f:
      count = 0
      for line in f.readlines():
         lst[count] = int(line)
         count += 1
   ```

10. For the next set of questions show what each program will print when the user supplies the indicated input text. Write *EXCEPTION* if and when the execution will generate an exception stack trace for an uncaught exception.

a.
```
print('Begin')
x = int(input())
print(x)
print('End')
```

i. User enters 22
ii. User enters ZZ

b.
```
print('Begin')
try:
   x = int(input())
   print(x)
except ValueError:
   print('Wrong!')
print('End')
```

i. User enters 22
ii. User enters ZZ

c.
```
print('Begin')
try:
   x = int(input())
   print(x)
except IndexError:
   print('Wrong!')
print('End')
```

i. User enters 22
ii. User enters ZZ

d.
```
print('Begin')
try:
   x = int(input())
   print(x)
except Exception:
   print('Wrong!')
print('End')
```

i. User enters 22
ii. User enters ZZ

e.
```
print('Begin')
try:
   x = int(input())
   print(x)
except ValueError:
   print('Wrong!')
else:
   print('Wow')
print('End')
```

i. User enters 22

ii. User enters ZZ

f.
```
print('Begin')
try:
    x = int(input())
    print(x)
except ValueError:
    print('Wrong!')
finally:
    print('Done')
print('End')
```

i. User enters 22
ii. User enters ZZ

g.
```
print('Begin')
try:
    x = int(input())
    print(x)
except ValueError:
    print('Wrong!')
else:
    print('Wow')
finally:
    print('Done')
print('End')
```

i. User enters 22
ii. User enters ZZ

11. What is the problem with the following code?

```
try:
        f() # Function f can raise an exception
except Exception:
        print(1)
except ValueError:
        print(2)
```

12. You might have encountered some exceptions during the development of code in problems 3, 5 and 6. List all possible exceptions your code can generate, that you can think of, when these classes are used by a novice user.

13. Include code in problem 3, 5 and 6 to handle all the exceptions you can came up with in problem 12.