1. **(Question 1 for odd roll nos)**: Consider the following python program to compute multiplication of two numbers. Assume that the python script is saved as “multiply.py”. If you run the code using “python multiply.py”, you will see that the code gets an error that tells that the number of arguments does not match while calling the ‘multiply’ function. This is easy to understand as the calls in line 6 and 7 provide 3 and 4 numbers as arguments with the intention to get the product of 3 and 4 passed-in numbers respectively. Modify the function “multiply” such that all the code that is written under the block “if __name__ == '__main__':” runs without encountering any error. Note that as an example, we have called the multiply function with a max of 4 numbers, but your modified code must be able to handle arbitrary number of passed-in numbers. You are allowed to modify ONLY the function definition and function body of ‘multiply’ without modifying anything below “if __name__ == '__main__':”.
   Submission instruction: You should write a complete “multiply.py” file.

multiply.py

```
1  def multiply(x, y):
2      print(x * y)
3
4  if __name__ == '__main__':
5      multiply(3, 4)
6      multiply(3, 4, 5)
7      multiply(3, 4, 5, 6)
```

   Solution:

```
1  def multiply(*args):
2      ret = 1
3      for i in args:
4          ret *= i
5      print(ret)
6
7  if __name__ == '__main__':
8      multiply(3, 4)
9      multiply(3, 4, 5)
10     multiply(3, 4, 5, 6)
```

2. **(Question 2 for odd roll nos)**: Suppose you are a developer of a software firm and use a thirdparty library. Let us take “library_odd.py” file as one such library which you will be importing for writing a code for your firm. Suppose, you are writing your code in “user_odd.py”. You will notice that two very important lines of code in “library_odd.py” is commented out. This is to mimic a very common bug encountered in practice – i.e., the thirdparty library developer forgot to implement the ‘foo’ method that was supposed to be shipped with the library file. The

problem is that you don't have access to the library file. Your task is to add suitable code in "user_odd.py" so that you can catch this exception. The requirement is that you have to catch it before the object 'd1' of the 'Derived' class is even created, i.e., you need to catch it before "d1 = Derived()" statement is executed. The above statement means all your modification should come after "if __name__ == '__main__':" line and before "d1 = Derived()" statement. **HINT**: Think in terms of assert statements.
Submission instruction: You should write a complete "user_odd.py" file.

library_odd.py

```
class Base:
    pass
    # def foo(self):
    #     return 'foo'
```

user_odd.py

```
from library_odd import Base

class Derived(Base):
    def bar(self):
        return self.foo()

if __name__ == '__main__':
    d1 = Derived()
    d1.bar()
```

Solution:

user_odd_solution.py

```
from library_odd import Base

class Derived(Base):
    def bar(self):
        return self.foo()

if __name__ == '__main__':
    assert hasattr(Base, 'foo'), "\'foo\' not implemented in Base"
    d1 = Derived()
    d1.bar()
```

3. **(Question 3 for odd roll nos)**: In this assignment, you will examine the power of numpy in avoiding loops for simple addition. With the help of the following starter code (screenshot given), perform the following.

a. Generate 100,000 random numbers and store in a list 'a'.
b. Add 1 to each number in list 'a' by using list comprehension. Store the resulting list in variable 'b'.
c. Use numpy to create an array 'c' from 'a'.
d. Use numpy to add 1 to each number in the array 'c'. Store the resulting list in variable 'd'.
e. Use the imported library 'datetime' to print the times taken to perform step 'b' and 'd'. You should print the times in milliseconds. Look at the documentation of the python datetime library to use it properly: https://docs.python.org/3/library/datetime.html

```
import numpy as np
from datetime import datetime
```

Solution:

sum.py

```
1  import numpy as np
2  from datetime import datetime
3
4  # Generate 100,000 random numbers and store in a list 'a'
5  a = list(np.random.rand(100000))
6
7  startTime = datetime.now()
8  # Add 1 to each number in list a by using a list comprehension.
9  # Store the resulting list in variable 'b'
10 b = [x+1 for x in a]
11 timetaken = datetime.now() - startTime
12 print("The time taken is {} milliseconds".format(timetaken.microseconds/1000.0))
13
14 # Use numpy to create an array 'c' from 'a'
15 c = np.array(a)
16
17 startTime = datetime.now()
18 # Use numpy to add 1 to each number in the array 'c'.
19 # Store the resulting list in variable 'd'
20 d = c + 1
21 timetaken = datetime.now() - startTime
22 print("The time taken is {} milliseconds".format(timetaken.microseconds/1000.0))
```

4. **(Question 1 for even roll nos):** Consider the following python program to compute product of all the numbers in a list. Assume that the python script is saved as “product.py”. If you run the code using “python product.py”, you will see that the code gets an error that tells that the number of arguments does not match while calling the ‘product’ function. This is easy to understand as the call in line 10 provides 3 numbers as arguments instead of a list of numbers. Modify the function “product” such that it can take in any number of numbers as arguments, that is, the function “product” should not only work for a “list”. However, you are given the following constraints.
   i. The function body is not allowed to change, that is – only line 1 of the script is allowed to get changed.
   ii. You can change the calling of the function where the python list ‘num’ is passed that is line number 9.
   WHAT THE ABOVE TWO CONSTRAINTS MEAN IS THAT YOU ARE ALLOWED TO CHANGED ONLY TWO LINES IN THE CODE – LINE 1 AND LINE 9. **HINT**: You need to think in terms of unpacking.
   Submission instruction: You should write a complete “product.py” file.

product.py

```python
def product(numbers):
    ret = 1
    for i in numbers:
        ret *= i
    print(ret)

if __name__ == '__main__':
    num = [3, 4, 5]
    product(num)
    product(3, 4, 5)
```

Solution:

```python
def product(*numbers):
    ret = 1
    for i in numbers:
        ret *= i
    print(ret)

if __name__ == '__main__':
    num = [3, 4, 5]
    product(*num)
    product(3, 4, 5)
```

5. **(Question 2 for even roll nos)**: Suppose you are a developer of a software firm and use a thirdparty library. Let us take "library_even.py" file as one such library which you will be importing for writing a code for your firm. Suppose, you are writing your code in "user_even.py". Your task is to add suitable code in "user_even.py" so that your code runs error free. Note that you are ONLY allowed to modify the definition of the derived class in "user_even.py". The above statement means you are not allowed to change anything after the line "if __name__ == '__main__':". Also note than you DO NOT have any access to "library_even.py".
Submission instruction: You should write a complete "user_even.py" file.

library_even.py

```python
class Base:
    def foo(self):
        return self.bar()
```

user_even.py

```python
from library_even import Base

class Derived(Base):
    pass

if __name__ == '__main__':
    d1 = Derived()
    d1.foo()
```

Solution:

user_even_solution.py

```python
from library_even import Base

class Derived(Base):
    def bar(self):
        pass

if __name__ == '__main__':
    d1 = Derived()
    d1.foo()
```

6. **(Question 3 for even roll nos)**: In this assignment, you will examine the power of numpy in avoiding loops for simple squaring. With the help of the following starter code (screenshot given), perform the following.

a. Generate 100,000 random numbers and store in a list ‘a’.
b. Square each number in list ‘a’ by using list comprehension. Store the resulting list in variable 'b'.
c. Use numpy to create an array 'c' from 'a'.
d. Use numpy to square each number in the array 'c'. Store the resulting list in variable 'd'.
e. Use the imported library ‘datetime’ to print the times taken to perform step ‘b’ and ‘d’. You should print the times in milliseconds. Look at the documentation of the python datetime library to use it properly: https://docs.python.org/3/library/datetime.html

```python
import numpy as np
from datetime import datetime
```

Solution:

square.py

```python
import numpy as np
from datetime import datetime

# Generate 100,000 random numbers and store in a list 'a'
a = list(np.random.rand(100000))

startTime = datetime.now()
# Square each number in list a by using a list comprehension.
# Store the resulting list in variable 'b'
b = [x*x for x in a]
timetaken = datetime.now() - startTime
print("The time taken is {} milliseconds".format(timetaken.microseconds/1000.0))

# Use numpy to create an array 'c' from 'a'
c = np.array(a)

startTime = datetime.now()
# Use numpy to add 1 to each number in the array 'c'.
# Store the resulting list in variable 'd'
d = np.multiply(c,c)
timetaken = datetime.now() - startTime
print("The time taken is {} milliseconds".format(timetaken.microseconds/1000.0))
```

```python
class Rect:
    def __init__(self,a: tuple,b: tuple):
        self.p1 = a
        self.p2 = b

    def area(self):
        return abs( (self.p1[0] - self.p2[0]) * (self.p1[1] - self.p2[1]) )

    def __le__(self, other):
        return (min(self.p1[0],self.p2[0]) <= min(other.p1[0],other.p2[0])) \
               and (min(self.p1[1],self.p2[1]) <= min(other.p1[1],other.p2[1]))

a = Rect( (13,13), (45,45) )
b = Rect( (14,13), (22,33) )
print(a <= b)
print(a >= b)
```