Week 1 Practice Programming Assignment

Problem 1

Twin primes are pairs of prime numbers that differ by 2. For example (3, 5), (5, 7), and (11,13) are twin primes.

Write a function **Twin_Primes(n, m)** where `n` and `m` are positive integers and `n < m` , that returns all unique twin primes between n and m (both inclusive). The function returns a list of tuples and each tuple `(a,b)` represents one unique twin prime where `n <= a < b <= m`.

Sample Input 1

```
1
15
```

Output

```
[(3, 5), (5, 7), (11, 13)]
```

Sample Input 2

```
11
25
```

Output

```
[(11, 13), (17, 19)]
```

Solution

```
def prime(n):
  if n < 2:
    return False
  for i in range(2,n//2+1):
    if n%i==0:
      return False
  return True

def Twin_Primes(n, m):
  Res=[]
  for i in range(n,m-1):
    if prime(i)==True:
      if prime(i+2)==True:
        Res.append((i,i+2))
  return(Res)
```

Suffix Code(Visible)

```
n=int(input())
m=int(input())
print(sorted(Twin_Primes(n, m)))
```

Public Test Cases

Input 1

```
1
15
```

Output

```
[(3, 5), (5, 7), (11, 13)]
```

Input 2

```
11
25
```

Output

```
[(11, 13), (17, 19)]
```

Input 3

```
5
50
```

Output

```
[(5, 7), (11, 13), (17, 19), (29, 31), (41, 43)]
```

Private Test Cases

Input 1

```
1
100
```

Output

```
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
```

Input 2

```
1
10
```

Output

```
[(3, 5), (5, 7)]
```

Input 3

```
60
200
```

Output

```
[(71, 73), (101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 193), (197, 199)]
```

Input 4

```
50
73
```

Output

```
[(59, 61), (71, 73)]
```

Input 5

```
17
19
```

Output

```
[(17, 19)]
```

Problem 2

Create a **Triangle** class that accepts three side-lengths of the triangle as `a`, `b` and `c` as parameters at the time of object creation. Class **Triangle** should have the following methods:

- **Is_valid()** :- Returns `Valid` if triangle is valid otherwise returns `Invalid`.
  - A triangle is valid when the sum of its two side-length are greater than the third one. That means the triangle is valid if all three condition are satisfied:
    - a + b > c
    - a + c > b
    - b + c > a
- **Side_Classification()** :- If the triangle is invalid then return `Invalid`. Otherwise, it returns the type of triangle according to the sides of the triangle as follows:
  - Return `Equilateral` if all sides are of equal length.
  - Return `Isosceles` if any two sides are of equal length and third is different.
  - Return `Scalene` if all sides are of different lengths.
- **Angle_Classification()** :- If the triangle is invalid then return `Invalid` . Otherwise, return type of triangle using Pythagoras theorem.

  Assume `a <= b <= c`. then
  - If $a^2 + b^2 > c^2$ return `Acute`
  - If $a^2 + b^2 = c^2$ return `Right`
  - If $a^2 + b^2 < c^2$ return `Obtuse`
- **Area()** :- If the triangle is invalid then return `Invalid`. Otherwise, return the area of the triangle.
  - $Area = \sqrt{s(s-a)(s-b)(s-c)}$

    Where $s = (a+b+c)/2$

Sample Input 1

```
2
3
4
```

Output

```
Valid
Scalene
Obtuse
2.9047375096555625
```

Sample Input 2

```
10
3
5
```

Output

```
Invalid
Invalid
Invalid
Invalid
```

Solution

```
class Triangle:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def is_valid(self):
        if self.a >= self.b + self.c:
            return 'Invalid'
        if self.b >= self.a + self.c:
            return 'Invalid'
        if self.c >= self.b + self.a:
            return 'Invalid'
        return 'Valid'
    def Side_Classification(self):
        if self.is_valid() == 'Valid':
            if self.a == self.b == self.c:
                return 'Equilateral'
            elif (self.a == self.b) or (self.b == self.c) or (self.a == self.c):
                return 'Isosceles'
            else:
                return 'Scalene'
        else:
            return 'Invalid'
    def Angle_Classification(self):
        if self.is_valid() == 'Valid':
            l = [self.a**2, self.b**2, self.c**2]
            l.sort()
            if l[0] + l[1] > l[2]:
                return "Acute"
            elif l[0] + l[1] == l[2]:
                return "Right"
            elif l[0] + l[1] < l[2]:
                return "Obtuse"
        else:
            return 'Invalid'
    def Area(self):
        if self.is_valid() == 'Valid':
            s = (self.a + self.b + self.c)/2
            return (s*(s-self.a)*(s-self.b)*(s-self.c)) ** 0.5
        else:
            return 'Invalid'
```

Suffix code(visible)

```
a=int(input())
b=int(input())
c=int(input())
T=Triangle(a,b,c)
print(T.is_valid())
print(T.Side_Classification())
print(T.Angle_Classification())
print(T.Area())
```

Public Test Cases

Input 1

```
2
3
4
```

Output

```
Valid
Scalene
Obtuse
2.9047375096555625
```

Input 2

```
10
3
5
```

Output

```
Invalid
Invalid
Invalid
Invalid
```

Input 3

```
5
5
5
```

Output

```
Valid
Equilateral
Acute
10.825317547305483
```

Private Test Cases

Input 1

```
2
12
3
```

Output

```
Invalid
Invalid
Invalid
Invalid
```

Input 2

```
14
5
12
```

Output

```
Valid
Scalene
Obtuse
29.230762904857617
```

Input 3

```
32
12
21
```

Output

```
Valid
Scalene
Obtuse
61.89456761299815
```

Input 4

```
10
10
5
```

Output

```
Valid
Isosceles
Acute
24.206145913796355
```

Input 5

```
3
4
5
```

Output

```
Valid
Scalene
Right
6.0
```