Information Mid-Term Project Report

ID: J8240003
Name: Tausif Ibne Iqbal
Codes available on GitHub: https://github.com/Tausif30/Information

Problem 1 (Multi-perfect number)

1. I looped through all the numbers up to the given number n. If the remainder of n is 0 while divided by 0 then that number is a factor. I defined a interger sum outside the loop with a value of 0 and inside the loop I kept adding the factors to it.

```
1 usage                                                    ⚠ 12
def divisors(n):
    divisor = []
    divisor_sum = 0
    for i in range(1, n):
        if n % i == 0:
            divisor.append(i)
            divisor_sum += i
    divisor.append(n)
    divisor.sort()
    divisor_sum += n
    print("Divisors: ", ", ".join(map(str,divisor)))
    print("Divisor sum: ", divisor_sum)
    return divisor_sum

2 usages
def divisors_sum(n):
    divisor_sum = 0
    for i in range(1, n):
        if n % i == 0:
            divisor_sum += i
    divisor_sum += n
    return divisor_sum
```

2. Used the previous divisor_sum function and if the given number run inside divisor_sum function equals twice that number then that number is a P_2 number.

```
def p2(n):
    p2_num = []
    for i in range(1,n):
        a = divisors_sum(i)
        if a == 2 * i:
            p2_num.append(i)

    print("Doubly Perfect Numbers below 10000: ", ", ".join(map(str,p2_num)) )
    return p2
```

3. Used the modulus operator to see if the remainder is 0 or not, then used floor to find the quotient and thus defined the degree of the multi perfect number.

```
def pm(n):
    pm = []
    for i in range(1, n):
        divisor_sum = divisors_sum(i)

        if divisor_sum % i == 0:
            multiple = divisor_sum // i
            pm.append((i, multiple))
    print("Multiply Perfect Numbers below 50000: ", ", ".join(map(str,pm)) )
    return pm
```

4. Sorry didn't find any better algorithm. However the previous one is already O(n)

Problem 2 (Goldbach Conjecture)

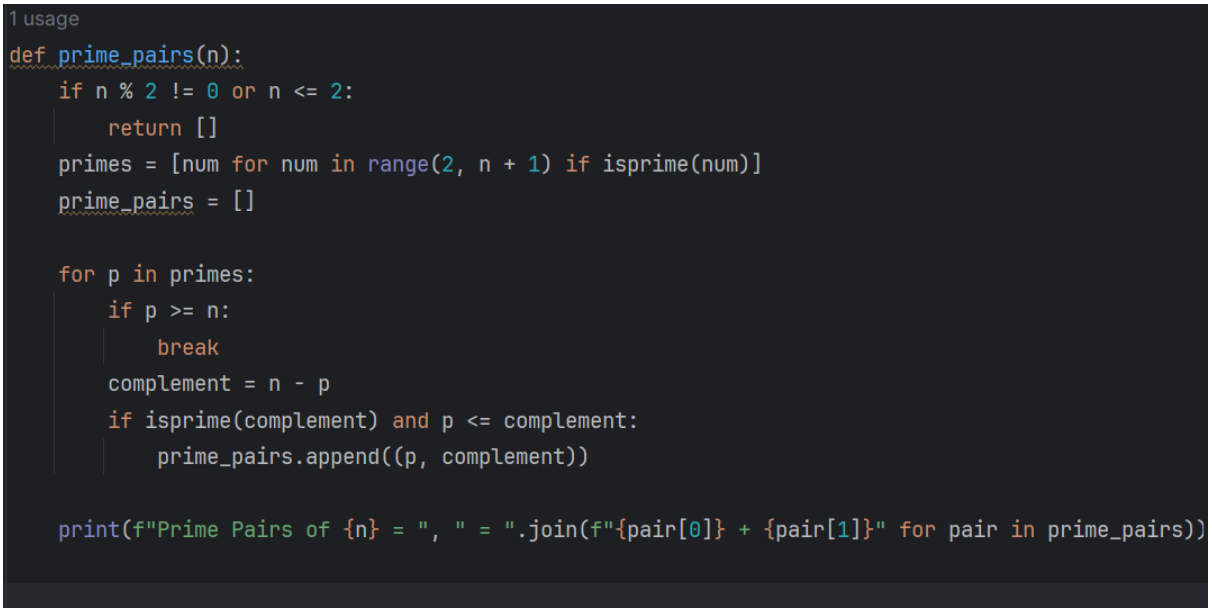
1,2. Used prime number functions made for previous homework.

```
5 usages
def isprime(n):
    for i in range(2, int(n**0.5)+1):
        if n%i==0:
            return False
    return True


1 usage
def prime_numbers(maxn):
    num = []
    for i in range(1, maxn+1):
        for j in range(2, int(i**0.5)+1):
            if i % j == 0:
                break
        else:
            num.append(str(i))
    print("Prime Numbers: ", ", ".join(num))

```

3. Ran a loop through all the numbers up to n and found the primes, p and then defined complement = n – p. If complement is also a prime number, then Goldbach Conjecture is proved true for that number. Now I just loop through all the numbers like this.

```
1 usage
def goldbach_conjecture(maxn):
    print("Checking Goldbach Conjecture for even numbers up to", maxn)
    for n in range(4, maxn + 1, 2):
        found_prime_pair = False
        for p in range(2, n):
            if not isprime(p):
                continue
            complement = n - p
            if isprime(complement):
                print(f"{n} = {p} + {complement}")
                found_prime_pair = True
                break
        if not found_prime_pair:
            print(f"Goldbach Conjecture fails for {n}")
            return False
    print("Goldbach Conjecture holds for all even numbers up to", maxn)
    return True
```

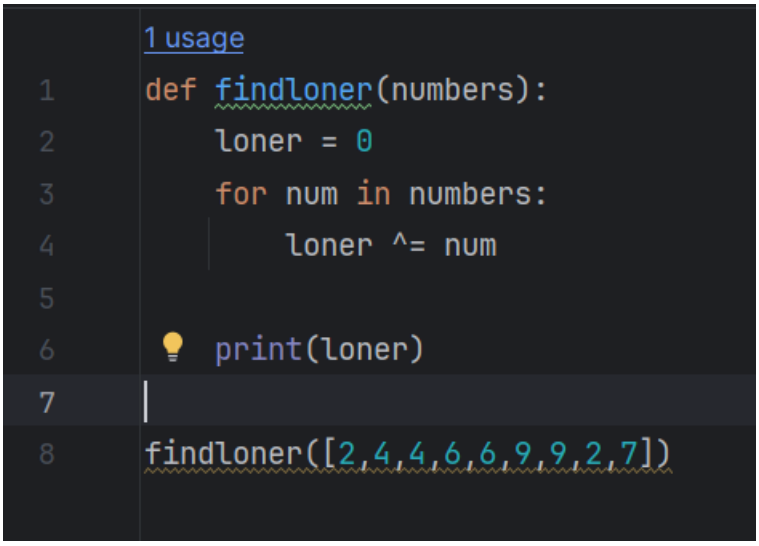
4. Same thing as the previous one. Made a prime_pair list, kept looping throughout the number, subtracting primes with n to find the complements. If complement also turns out to be a prime number, then I appended those prime pairs in the list I made.



```
1 usage
def prime_pairs(n):
    if n % 2 != 0 or n <= 2:
        return []
    primes = [num for num in range(2, n + 1) if isprime(num)]
    prime_pairs = []

    for p in primes:
        if p >= n:
            break
        complement = n - p
        if isprime(complement) and p <= complement:
            prime_pairs.append((p, complement))

    print(f"Prime Pairs of {n} = ", " = ".join(f"{pair[0]} + {pair[1]}" for pair in prime_pairs))
```

Problem 3 (Loner Number)

Defined a loner integer with initial value of 0. Then looped through the whole list of numbers in the function and used XOR operator with each of the numbers. This cancels out all the number except for the unique one.



```
1 usage
def findloner(numbers):
    loner = 0
    for num in numbers:
        loner ^= num

    print(loner)

findloner([2,4,4,6,6,9,9,2,7])
```

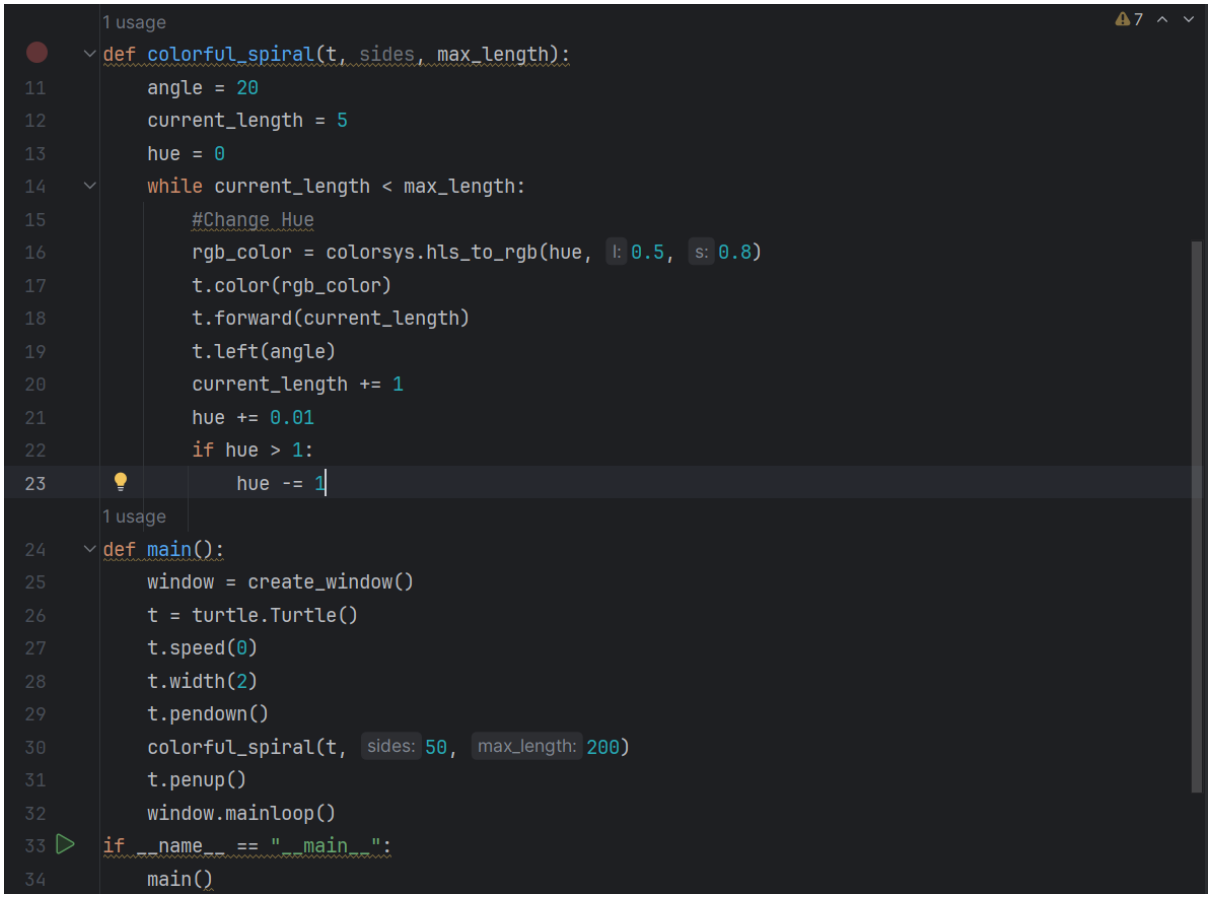
The XOR operator works as follows with the numbers.

0 ^ 2 = 2
2 ^ 4 = 6
6 ^ 4 = 2

It continues this way and at the end we get 7 as all of the others cancel each other out.

Problem 4 (Colorful Spiral)

Use the same polygon function to draw a circle except now kept changing the side length of the polygon. The polar function of a spiral is $r = a\theta$. Used this idea that the radius is continuously changing. For the color change I used HSL color and kept changing the hue of the color.



```
1 usage
def colorful_spiral(t, sides, max_length):
    angle = 20
    current_length = 5
    hue = 0
    while current_length < max_length:
        #Change Hue
        rgb_color = colorsys.hls_to_rgb(hue, l: 0.5, s: 0.8)
        t.color(rgb_color)
        t.forward(current_length)
        t.left(angle)
        current_length += 1
        hue += 0.01
        if hue > 1:
            hue -= 1
1 usage
def main():
    window = create_window()
    t = turtle.Turtle()
    t.speed(0)
    t.width(2)
    t.pendown()
    colorful_spiral(t, sides: 50, max_length: 200)
    t.penup()
    window.mainloop()
if __name__ == "__main__":
    main()
```

Problem 5 (Happy Number)

1. Turned the number into a list and then summed the square of all their digits to get the next numbers and if it converges to 1 then it's True and if we get the number itself back (forms a loop) then we stop the function and conclude False.

```python
import matplotlib.pyplot as plt

4 usages
def happy_list(n0):
    happy_list = [n0]
    while True:
        # Next Number = Sum of the digits squared
        next_number = sum(int(digit) ** 2 for digit in str(happy_list[-1]))
        # If number converges to 1 it is happy
        if next_number == 1:
            happy_list.append(next_number)
            return happy_list, True
        # If we get a loop number is unhappy
        elif next_number in happy_list:
            return happy_list, False
        else:
            happy_list.append(next_number)

print("Happy List for 7:", *happy_list(7))
print("Happy List for 17:", *happy_list(17))
```

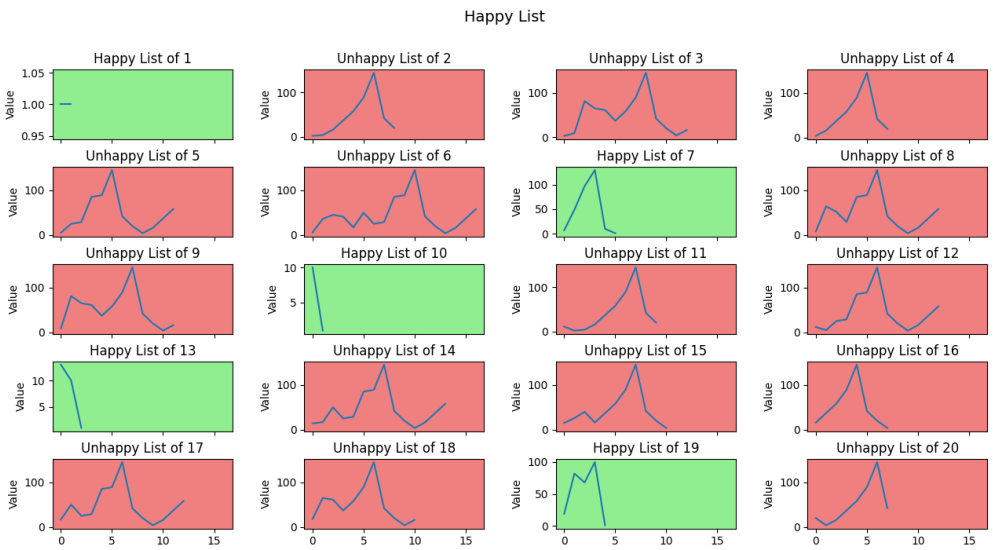
2. Used the same function to calculate the happy numbers and then add 1 to a count variable whenever we get a happy number. For happiness rate I just divided the count with the given number.

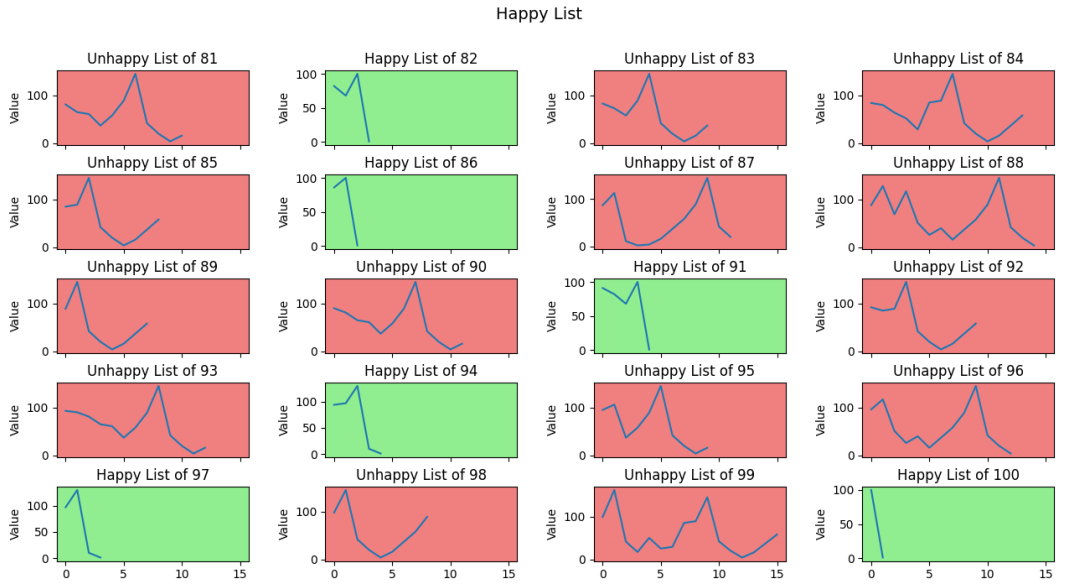
```python
1 usage
def count_happy_numbers(limit):
    happy_count = 0
    for n in range(1, limit + 1):
        i, is_happy = happy_list(n)
        if is_happy:
            happy_count += 1
    happiness_rate = (happy_count / limit) * 100
    return happy_count, happiness_rate

for limit in [100, 1000, 1000000]:
    happy_count, happiness_rate = count_happy_numbers(limit)
    print(f"Under {limit}:")
    print(f"Happy Numbers: {happy_count}")
    print(f"Happiness Rate: {happiness_rate:.2f}%")
    print("---")
```

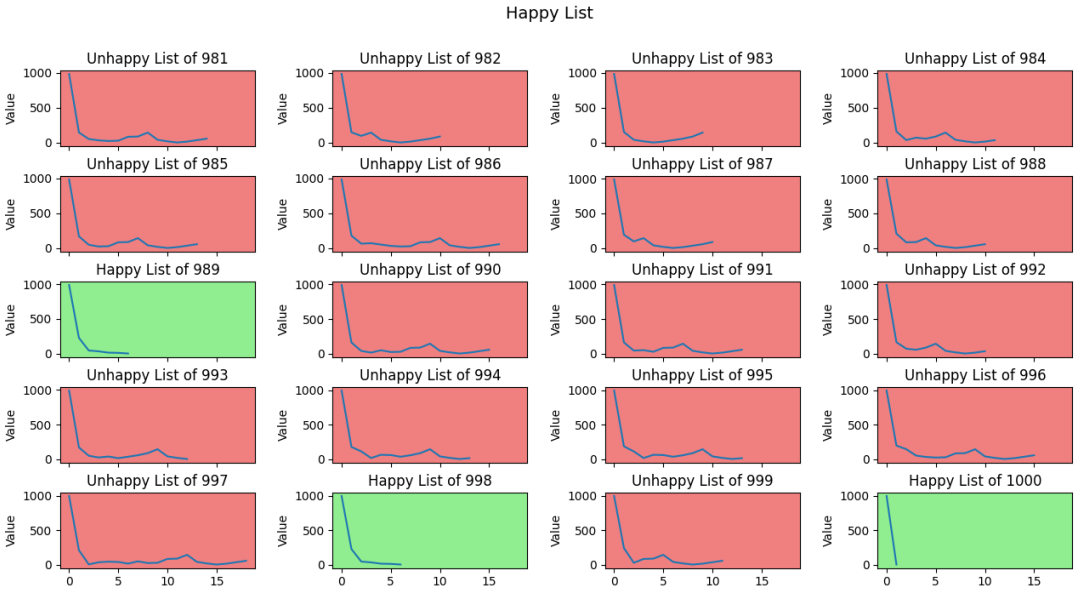
3. Used matplotlib to plot the graphs for each of the numbers.

```
1 usage
def plot_all_lists(limit):
    fig, axs = plt.subplots(nrows=5, ncols=4, figsize=(15, 15), sharex=True, gridspec_kw={'hspace':
    axs = axs.flatten() #Convert the 2D Array to 1D
    for i, n in enumerate(range(981, limit + 1)):
        if i >= len(axs):
            break
        current_list, is_happy = happy_list(n)
        axs[i].plot(current_list)
        axs[i].set_title(f"{'Happy' if is_happy else 'Unhappy'} List of {n}")
        axs[i].set_ylabel('Value')
        if is_happy:
            axs[i].set_facecolor('lightgreen')  # Green for happy numbers
        else:
            axs[i].set_facecolor('lightcoral')  # Red for unhappy numbers
    for ax in axs[i+1:]:
        ax.axis('off')
    plt.suptitle( t: f'Happy List', fontsize=14)
    plt.show()

plot_all_lists(1000)
```

It takes too much processing power to calculate all 1000 Happy numbers together and also gets messy. So, I just did 20 at a time. These are some of the plots I got.




Happy List
Unhappy List of 81
Happy List of 82
Unhappy List of 83
Unhappy List of 84
Unhappy List of 85
Happy List of 86
Unhappy List of 87
Unhappy List of 88
Unhappy List of 89
Unhappy List of 90
Happy List of 91
Unhappy List of 92
Unhappy List of 93
Happy List of 94
Unhappy List of 95
Unhappy List of 96
Happy List of 97
Unhappy List of 98
Unhappy List of 99
Happy List of 100
Value


Happy List
Unhappy List of 981
Unhappy List of 982
Unhappy List of 983
Unhappy List of 984
Unhappy List of 985
Unhappy List of 986
Unhappy List of 987
Unhappy List of 988
Happy List of 989
Unhappy List of 990
Unhappy List of 991
Unhappy List of 992
Unhappy List of 993
Unhappy List of 994
Unhappy List of 995
Unhappy List of 996
Unhappy List of 997
Happy List of 998
Unhappy List of 999
Happy List of 1000
Value

Problem 5 (Circumscribed Polygons)

1. Defined polygon function from before for drawing regular polygons. Used random, to get random side lengths and used setheading and random function together to get the random orientation part.
2. I used the circle function in turtle. The centroid of the polygon is just the average of all its vertices. I found the radius by finding the distance between the centroid and any one of the vertices of the polygon. A circumscribed circle will be touching all the sides of the polygon. So, I started drawing the circle from the first vertex specifically vertex[0]. We get the circle, but it 's still not correctly oriented. After some drawings and trying out, I found that we need to rotate the circle clockwise by a degree of (Exterior Angle + Half of Interior Angle - 90). This comes mainly from the way I drew the polygon (forward left). It will be different if someone does it differently. Now I add the random angle to the required orientation of the circle and then use it with the setheading function and we get

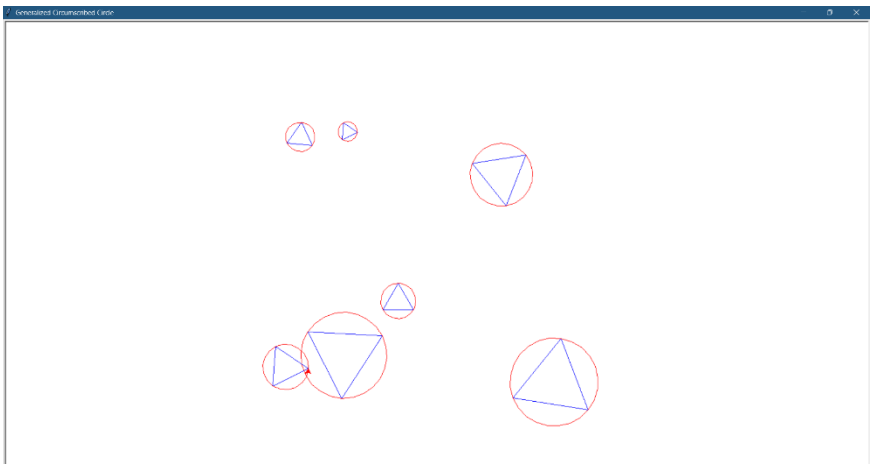
```
27

    1 usage
28  def centroid(coordinates):
29      # Finding Centroid
30      x = sum(coord[0] for coord in coordinates) / len(coordinates)
31      y = sum(coord[1] for coord in coordinates) / len(coordinates)
32      return x, y
33
34

    1 usage
35  def radius_from_centroid(coordinates):
36      # Calculate radius as distance between centroid and first vertex
37      cen_x, cen_y = centroid(coordinates)
38      x, y = coordinates[0]
39      return math.dist( __p: (x, y),  __q: (cen_x, cen_y))
40
```

perfect circumscribed circles.

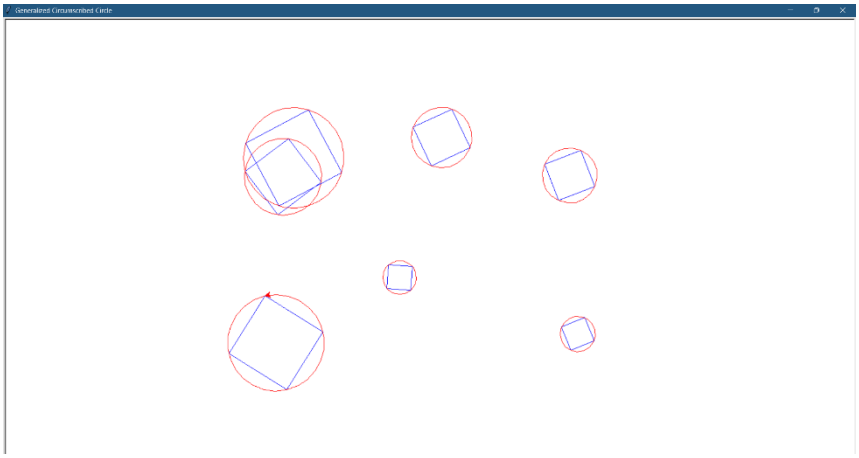
```python
def main():
    window = create_window()
    t = turtle.Turtle()
    t.speed(0)
    for i in range(3,10):
        n = 3
        t.penup()
        t.goto(random.randint(-300, b: 300), random.randint(-250, b: 250))
        t.pendown()
        random_angle = random.randint( a: 0, b: 360)
        t.setheading(random_angle)
        side_length = random.randint( a: 30, b: 150)
        t.color("blue")
        coordinates = polygon(n, t, side_length)
        x, y = coordinates[0]
        t.penup()
        radius = radius_from_centroid(coordinates)
        # Go to first vertex
        t.goto(x, y)
        # Calculate interior angle for rotation
        interior_angle = (n - 2) * 180 / n
        exterior_angle = 360/n
        rotation = exterior_angle - 90 + interior_angle/2
        t.setheading(random_angle - rotation)
        t.pendown()
        t.color("red")
```

3.

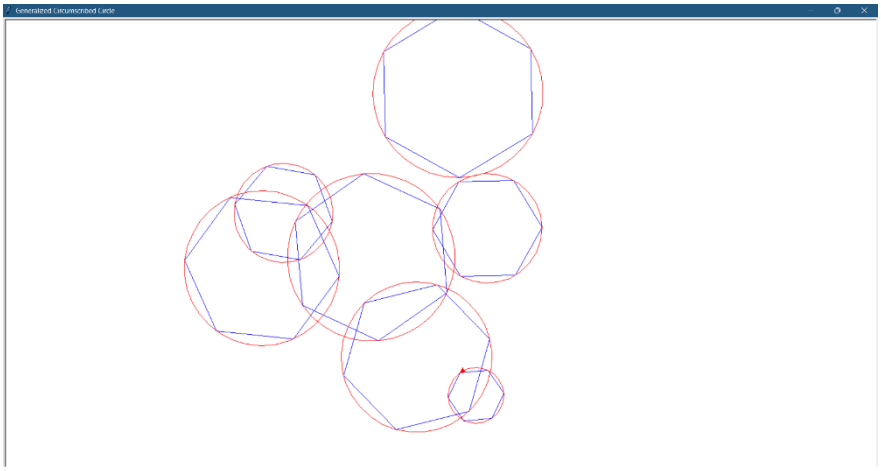
Circumscribed Triangles:



Circumscribed Squares:



Circumscribed Hexagons:



Observation: The polygon becomes closer to a circle and the area difference of the two figures gets closer to 0. This can be seen much better if we draw different polygons with circumscribed circles.