

Information Mid-Term Project Report

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
2
1  def divisors(n):
2      divisor = []
3      divisor_sum = 0
4      for i in range(1, n):
5          if n % i == 0:
6              divisor.append(i)
7              divisor_sum += i
8      divisor.append(n)
9      divisor.sort()
10     divisor_sum += n
11     print("Divisors: ", " ".join(map(str, divisor)))
12     print("Divisor sum: ", divisor_sum)
13     return divisor_sum
14
2  usages
15 def divisors_sum(n):
16     divisor_sum = 0
17     for i in range(1, n):
18         if n % i == 0:
19             divisor_sum += i
20     divisor_sum += n
21     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
1  def isprime(n):
2      for i in range(2, int(n**0.5)+1):
3          if n%i==0:
4              return False
5      return True
6
7
1 usage
8  def prime_numbers(maxn):
9      num = []
10     for i in range(1, maxn+1):
11         for j in range(2, int(i**0.5)+1):
12             if i % j == 0:
13                 break
14             else:
15                 num.append(str(i))
16     print("Prime Numbers: ", " ".join(num))
17
```

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
18  def goldbach_conjecture(maxn):
19      print("Checking Goldbach Conjecture for even numbers up to", maxn)
20      for n in range(4, maxn + 1, 2):
21          found_prime_pair = False
22          for p in range(2, n):
23              if not isprime(p):
24                  continue
25              complement = n - p
26              if isprime(complement):
27                  print(f"{n} = {p} + {complement}")
28                  found_prime_pair = True
29                  break
30          if not found_prime_pair:
31              print(f"Goldbach Conjecture fails for {n}")
32              return False
33      print("Goldbach Conjecture holds for all even numbers up to", maxn)
34      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
2 def findloner(numbers):
3     loner = 0
4     for num in numbers:
5         loner ^= num
6     print(loner)
7
8 findloner([2,4,4,6,6,9,9,2,7])
```

The XOR operator works as follows with the numbers.

$$0 \wedge 2 = 2$$

$$2 \wedge 4 = 6$$

$$6 \wedge 4 = 2$$

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
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

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

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.

```
1 import matplotlib.pyplot as plt
2
3 4 usages
4 def happy_list(n0):
5     happy_list = [n0]
6     while True:
7         # Next Number = Sum of the digits squared
8         next_number = sum(int(digit) ** 2 for digit in str(happy_list[-1]))
9         # If number converges to 1 it is happy
10        if next_number == 1:
11            happy_list.append(next_number)
12            return happy_list, True
13        # If we get a loop number is unhappy
14        elif next_number in happy_list:
15            return happy_list, False
16        else:
17            happy_list.append(next_number)
18
19 print("Happy List for 7:", *happy_list(7))
20 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.

```
1 usage
21 def count_happy_numbers(limit):
22     happy_count = 0
23     for n in range(1, limit + 1):
24         i, is_happy = happy_list(n)
25         if is_happy:
26             happy_count += 1
27     happiness_rate = (happy_count / limit) * 100
28     return happy_count, happiness_rate
29
30 for limit in [100, 1000, 1000000]:
31     happy_count, happiness_rate = count_happy_numbers(limit)
32     print(f"Under {limit}:")
33     print(f"Happy Numbers: {happy_count}")
34     print(f"Happiness Rate: {happiness_rate:.2f}%")
35     print("---")
```

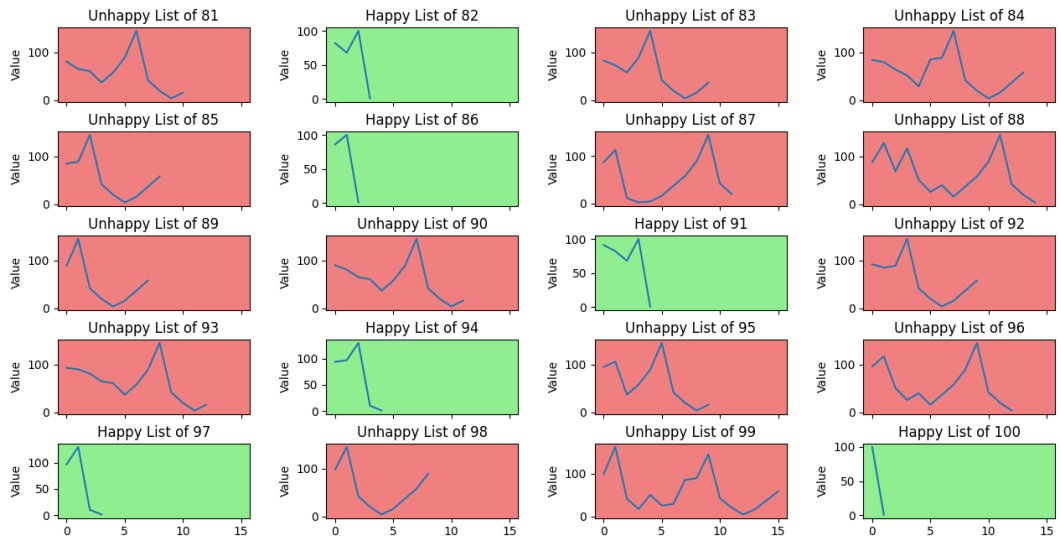

3. Used matplotlib to plot the graphs for each of the numbers.

```
1 usage
37 def plot_all_lists(limit):
38     fig, axs = plt.subplots(nrows=5, ncols=4, figsize=(15, 15), sharex=True, gridspec_kw={'hspace':
39     axs = axs.flatten() #Convert the 2D Array to 1D
40     for i, n in enumerate(range(981, limit + 1)):
41         if i >= len(axs):
42             break
43         current_list, is_happy = happy_list(n)
44         axs[i].plot(current_list)
45         axs[i].set_title(f"{'Happy' if is_happy else 'Unhappy'} List of {n}")
46         axs[i].set_ylabel('Value')
47         if is_happy:
48             axs[i].set_facecolor('lightgreen') # Green for happy numbers
49         else:
50             axs[i].set_facecolor('lightcoral') # Red for unhappy numbers
51     for ax in axs[i+1:]:
52         ax.axis('off')
53     plt.suptitle(f'Happy List', fontsize=14)
54     plt.show()
55
56 plot_all_lists(1000)
```

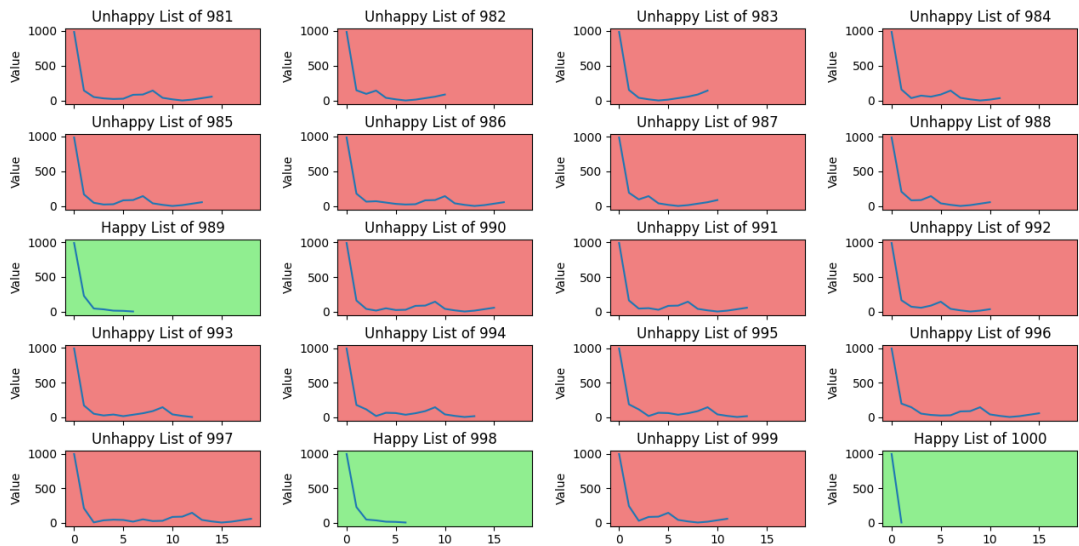
It takes too much processing power to calculate all 1000 Happy numbers together and also gets messy. So, I just did 20 at a time. These are some of the plots I got.



Happy List



Happy List



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 $(\text{Exterior Angle} + \text{Half of Interior Angle} - 90)$. 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 1 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
35 1 usage
36 1 def radius_from_centroid(coordinates):
37     # Calculate radius as distance between centroid and first vertex
38     cen_x, cen_y = centroid(coordinates)
39     x, y = coordinates[0]
40     return math.dist((x, y), (cen_x, cen_y))
```

perfect circumscribed circles.

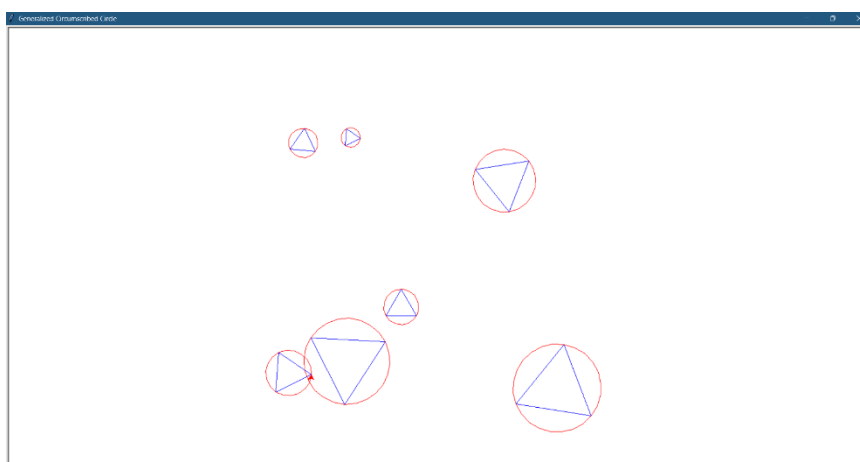
```

42 def main():
43     window = create_window()
44     t = turtle.Turtle()
45     t.speed(0)
46     for i in range(3,10):
47         n = 3
48         t.penup()
49         t.goto(random.randint(-300, 300), random.randint(-250, 250))
50         t.pendown()
51         random_angle = random.randint(0, 360)
52         t.setheading(random_angle)
53         side_length = random.randint(30, 150)
54         t.color("blue")
55         coordinates = polygon(n, t, side_length)
56         x, y = coordinates[0]
57         t.penup()
58         radius = radius_from_centroid(coordinates)
59         # Go to first vertex
60         t.goto(x, y)
61         # Calculate interior angle for rotation
62         interior_angle = (n - 2) * 180 / n
63         exterior_angle = 360/n
64         rotation = exterior_angle - 90 + interior_angle/2
65         t.setheading(random_angle - rotation)
66         t.pendown()
67         t.color("red")

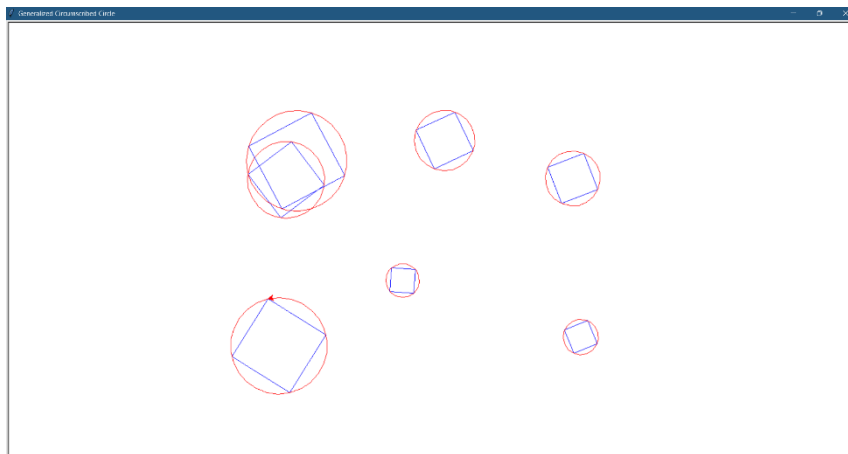
```

3.

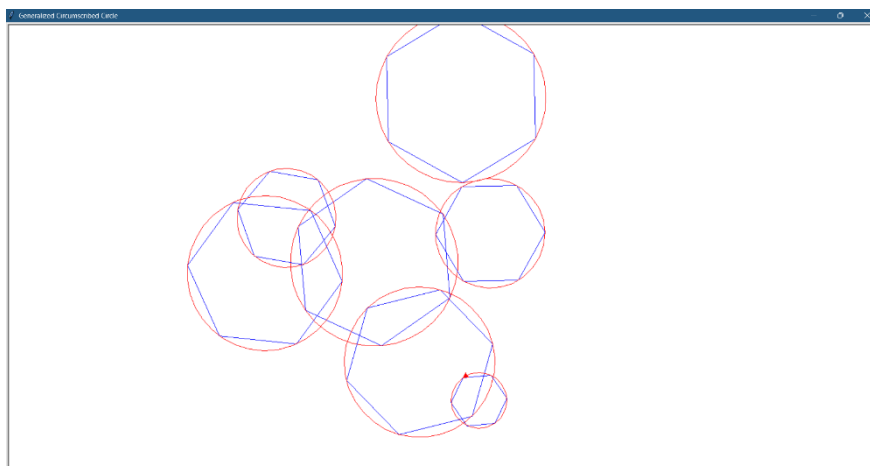
Circumscribed Triangles:



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.

