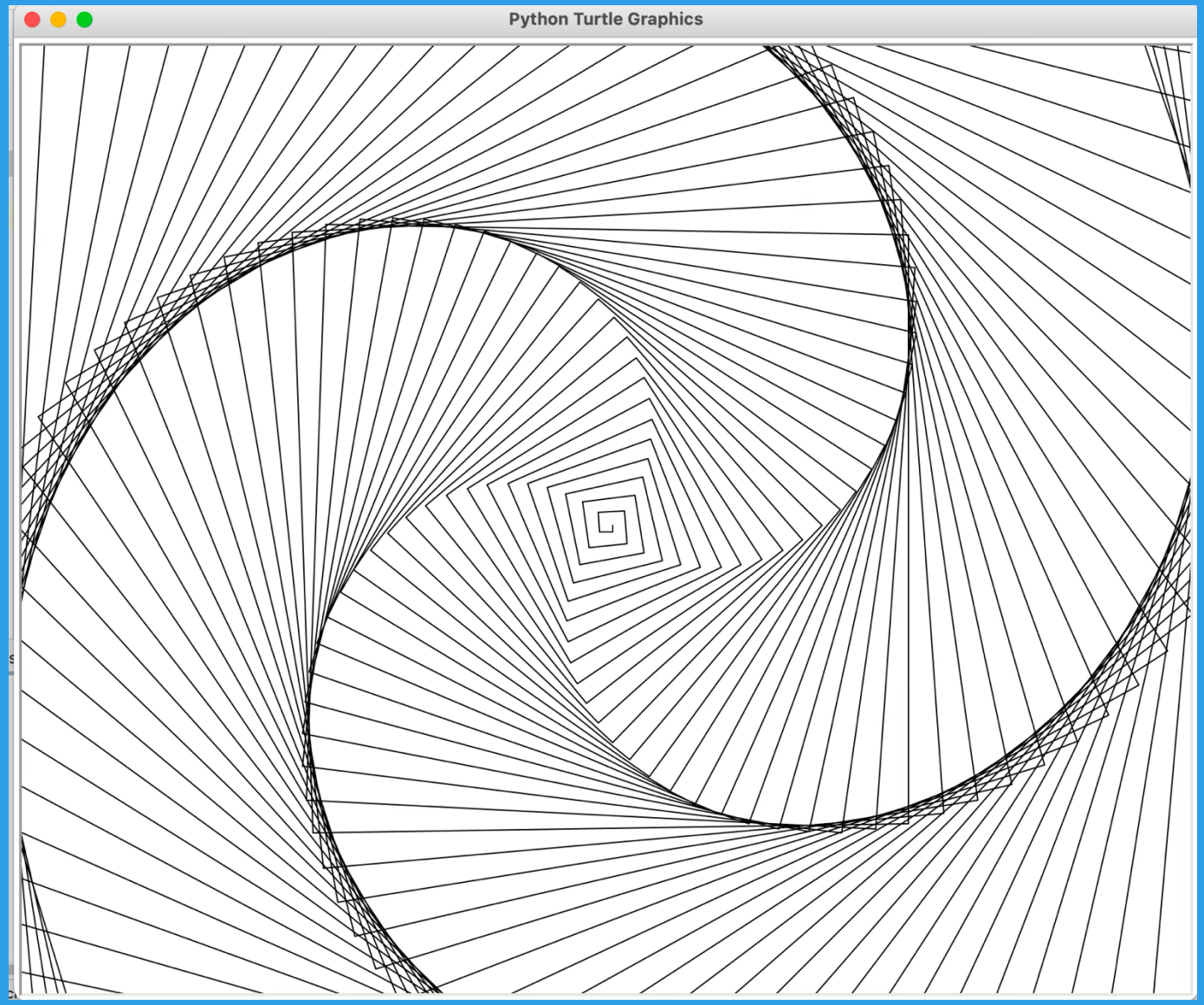
Mathematical Foundations



Turtle graphics

CS51 – Fall 2025

Python modules

- **Module**: a collection of functions and variables allowing us to share code with others.
- For example, there is a module called `math` that has many math functions you might want.
  - We can look at the documentation for this module by going to https://docs.python.org/3/ and searching `math` which will lead you to https://docs.python.org/3/library/math.html#module-math
- Some examples of useful functions and variables in the `math` module:
  - Variations of logs with different bases (e.g., `log`, `log2`, `log10`)
  - `sqrt` for calculating the square root
  - trigonometric functions (e.g., `cos`, `sin`, `tan`)
  - Constants (e.g., `pi`, `e`, etc.)

Importing modules

- If we want to use a module, we need to tell the program to "import" it in our program.
- Code to import all functions and variables from `math` module:
  - `from math import *`
- This statement has multiple components:
  - `from` is a keyword,
  - `math` is the name of the module,
  - `import` loads the module into our program,
  - `*` means everything, i.e. all functions and variables included in the `math` module.
- Just replace `math` with the name of the module you want to import.

random module

- The random module generates pseudo-random numbers.
- If you want truly random numbers, check out http://www.random.org/

Useful random functions

- random() - returns a random float between 0 and 1.
- uniform(a, b) - returns a random float between a and b.
- randint(a, b) - returns a random integer between a and b.

Importing only one function

- Let's say we only want to use the randint function.
- Rather than importing everything (*) we will be specific:
  - from random import randint

- What would the following piece of code do?

```
for i in range(100):
    print(randint(0,10))
```

- Prints 100 random integers between 0 and 10 (inclusive)

turtle module

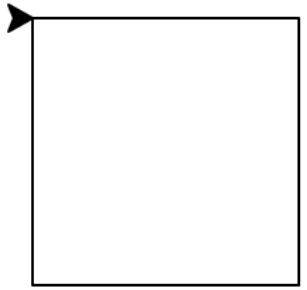
- The turtle module controls the movements of a "turtle" (really, an arrow), such as:
  - forward(distance): Move the turtle forward by the specified distance.
  - backward(distance): Move the turtle backward by distance, opposite to the direction the turtle is headed without changing the turtle's heading.
  - right(angle): Turn the turtle right by angle units.
  - left(angle): Turn turtle left by angle units.
  - ...and many others. Check the documentation and more practice in the upcoming assignment.
- As the turtle moves, it draws a line, so by controlling it we can draw on the screen!

Open VS Code

- Go to the Python shell at the bottom
- python3
- >>> from turtle import *

Let’s move our turtle

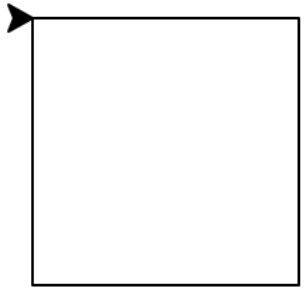
- How would you create a square by 20 units?
  - forward(20)
  - right(90)
  - forward(20)
  - right(90)
  - forward(20)
  - right(90)
  - forward(20)



Let’s move our turtle

- Packaging our square drawing code into a function:

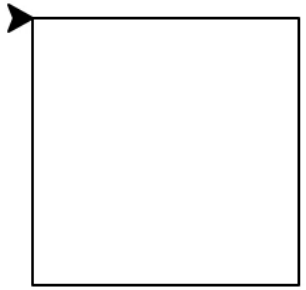
```
def square(length):
    forward(length)
    right(90)
    forward(length)
    right(90)
     forward(length)
     right(90)
     forward(length)
square(100)
```



Let’s move our turtle

- A less verbose function to draw a square:

```
def iterative_square(length):
    for i in range(4):
        forward(length)
        right(90)
iterative_square(20)
```

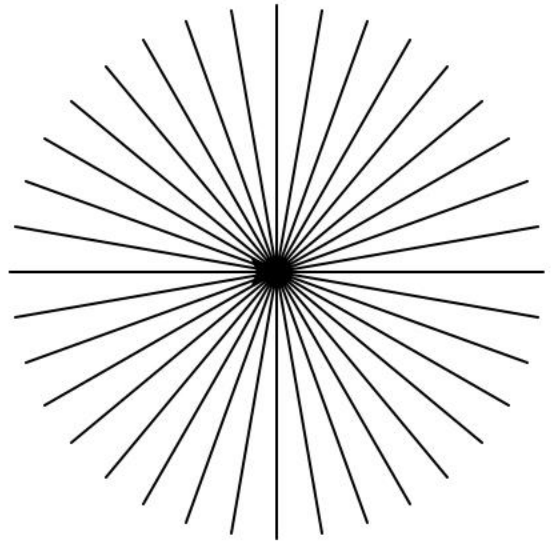


mystery1 function

```
def mystery1():
    for i in range(36):
        forward(100)
        backward(100)
        right(10)
```

simple_star function

```
def simple_star():
    for i in range(36):
        forward(100)
        backward(100)
        right(10)
```

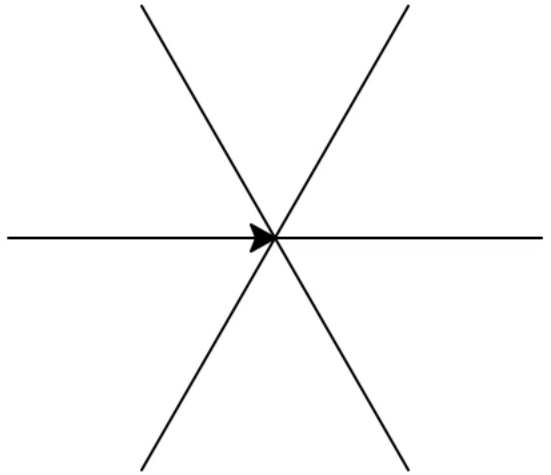


mystery2 function

```
def mystery_2(par1, par2):
    var = 360 / par2
    for i in range(par2):
        forward(par1)
        backward(par1)
        right(var)
```

A star with a different number of spokes

```
def asterisk_star(length, spokes):
    angle = 360 / spokes
    for i in range(spokes):
        forward(length)
        backward(length)
        right(angle)
```

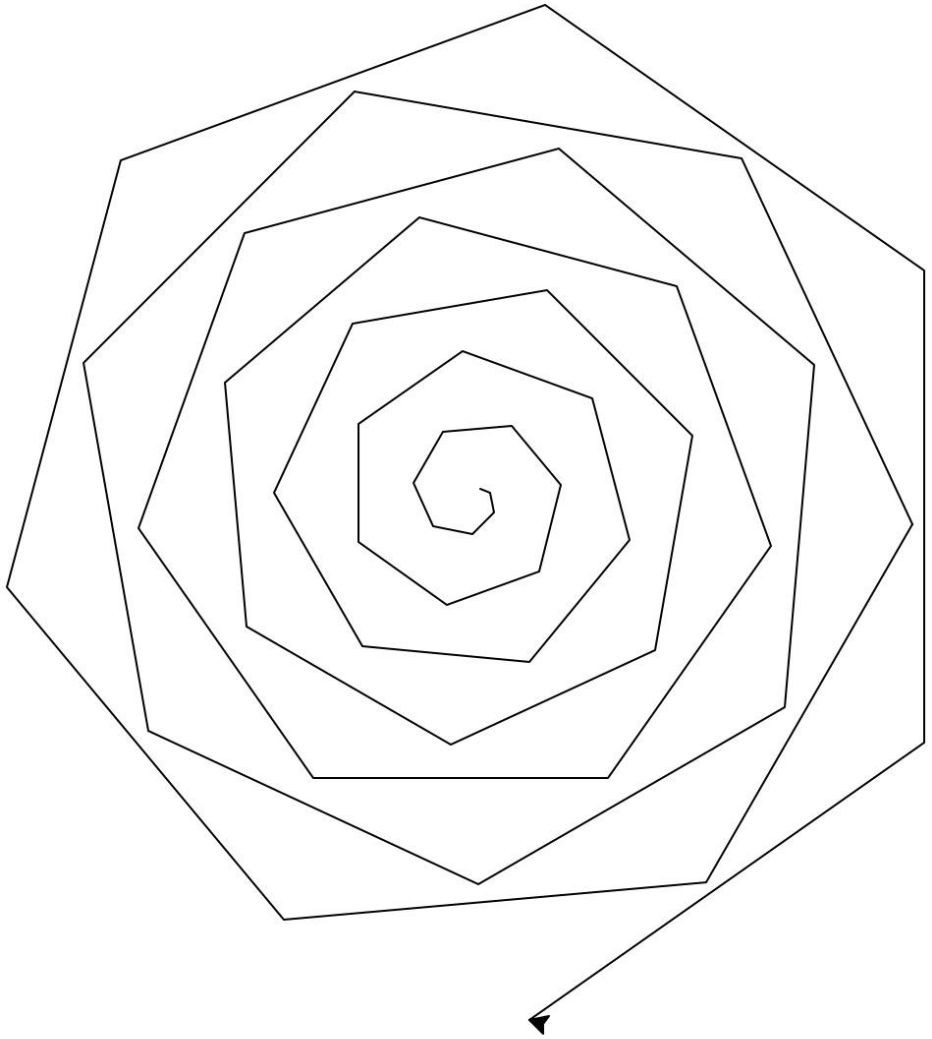


mystery3 function

```
def mystery3():
    for i in range(50):
        forward(i * 5)
        right(55)
```

simple_spiral function

```
def simple_spiral():
    for i in range(50):
        forward(i * 5)
        right(55)
```

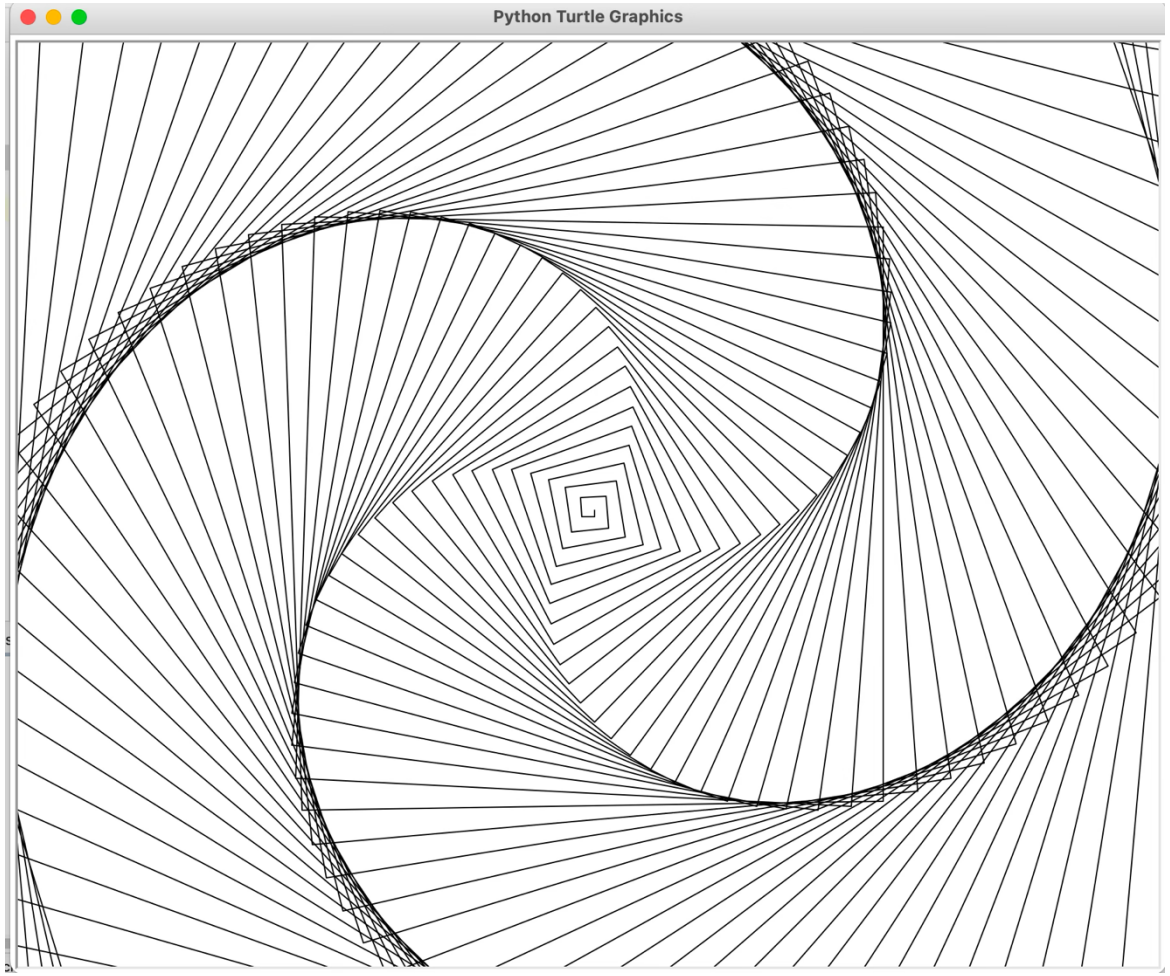


mystery4 function

```
def mystery4(par1, par2):
    for i in range(par1):
        forward(i * 5)
        right(par2)
```

spiral function

```
def spiral(sides, angle):
    for i in range(sides):
        forward(i * 5)
        right(angle)
```



mystery5 function

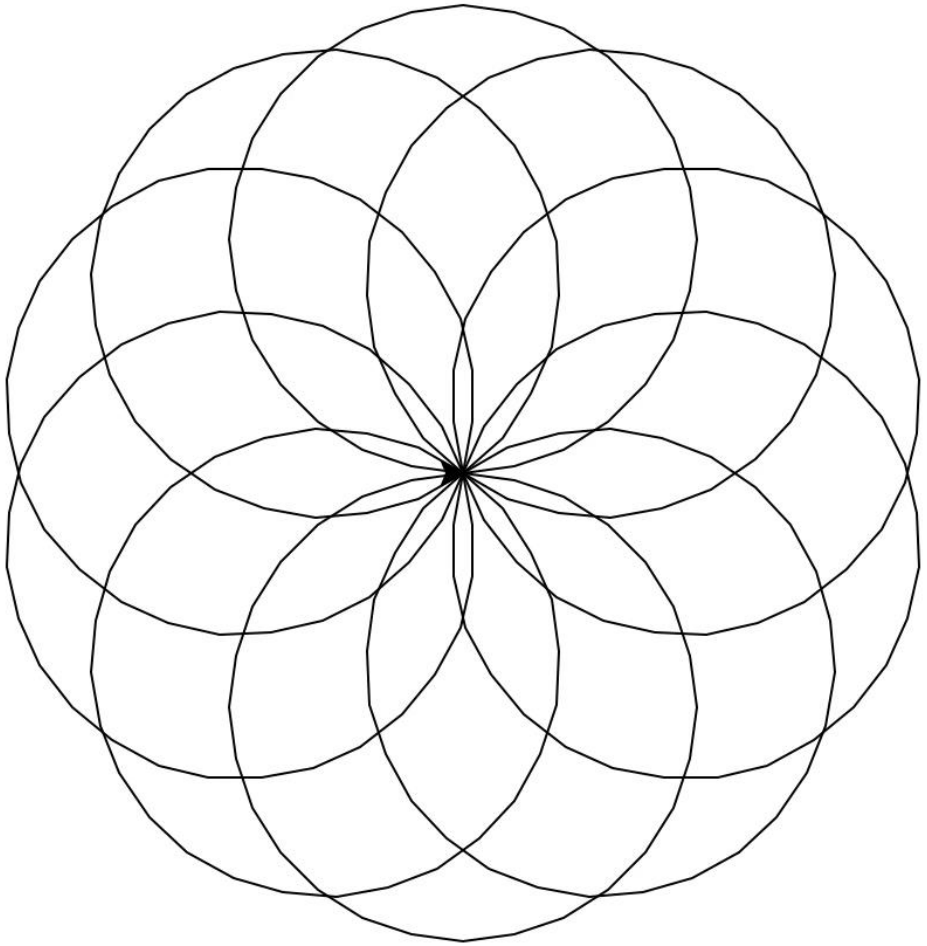
```
def mystery5(par1, par2):
    var = 360 / par2


    for i in range(par2):
        circle(par1)
        right(var)
```

rotating_circles function

```
def rotating_circles(radius, num):
    angle = 360 / num

    for i in range(num):
        circle(radius)
        right(angle)
```

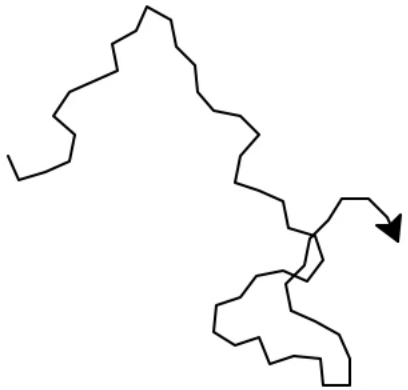


mystery6 function

```
def mystery6 (par1, par2):
    for i in range(par1):
        var = randint(-90, 90)
        right(var)
        forward(par2)
```

walk function

```
def walk(num_steps, step_size):
    for i in range(num_steps):
        angle = randint(-90, 90)
        right(angle)
        forward(step_size)
```

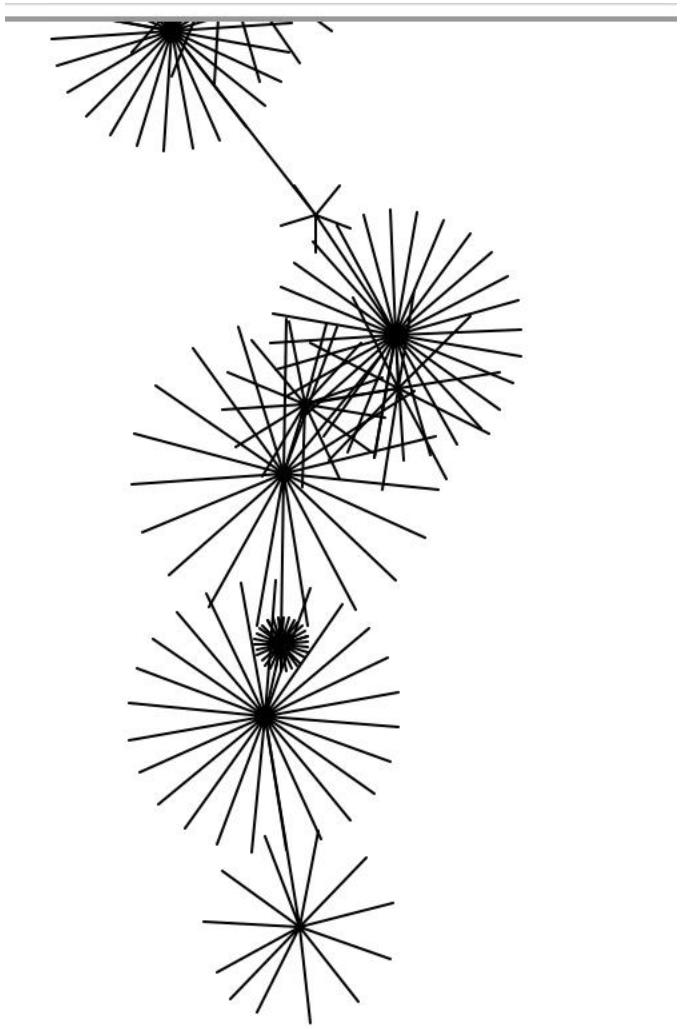


mystery7 function

```
def mystery7():
    for i in range(10):
        var1 = randint(5, 30)
        var2 = randint(10, 60)
        var3 = randint(-90, 90)
        var4 = randint(20, 100)
        right(var3)
        forward(var4)
        asterisk_star(var2, var1)
```

pretty_picture function

```
def pretty_picture():
    for i in range(10):
        spokes = randint(5, 30)
        length = randint(10, 60)
        angle = randint(-90, 90)
        move = randint(20, 100)
        right(angle)
        forward(move)
        asterisk_star(length, spokes)
```

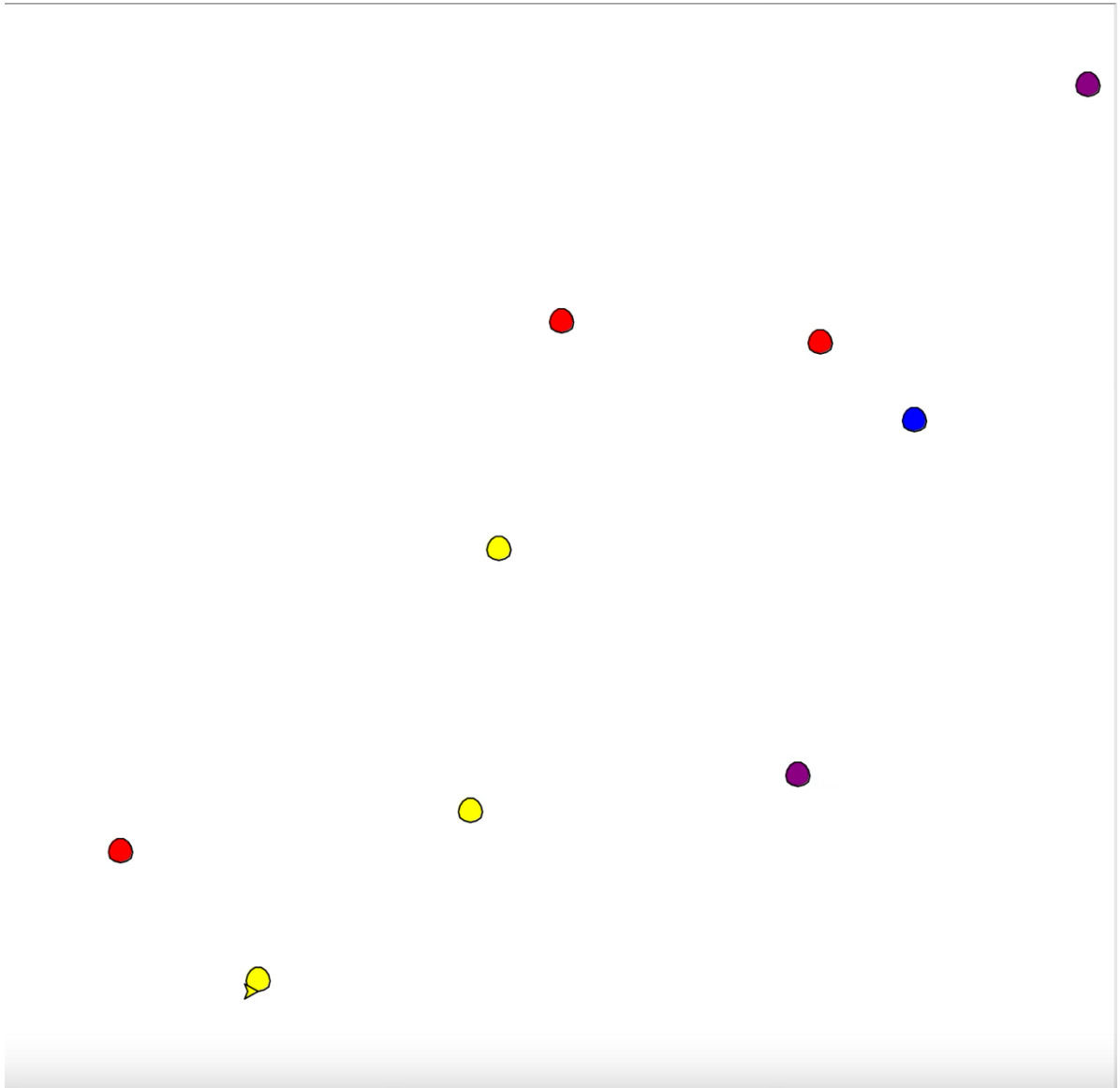


setcolor_random function

```
def setcolor_random():
    color = randint(1, 4)
    if color == 1:
        fillcolor("blue")
    elif color == 2:
        fillcolor("purple")
    elif color == 3:
        fillcolor("red")
    else:  # color == 4
        fillcolor("yellow")
```

add_circles function

```
def add_circles(number):
    x_range = int(window_width() / 2)
    y_range = int(window_height() / 2)
    for i in range(number):
        x = randint(-x_range, x_range)
        y = randint(-y_range, y_range)
        setcolor_random()
        pu()
        goto(x, y)
        pd()
        begin_fill()
        circle(8)
        end_fill()
```



Revisiting creating a spiral

```
def spiral(length, levels):
    if levels == 0:
        dot()
    else:
        forward(length)
        left(30)
        spiral(0.95 * length, levels – 1)
```

Revisiting spiral function

- Let's assume I called spiral(80, 50)
- When does it stop?
  - When levels = 0.
    - We put a dot there to make it explicit.
- Repeat 50 times:
  - forward length
  - left 30
  - reduce length by 5%

```
def spiral(length, levels):
    if levels == 0:
        dot()
    else:
        forward(length)
        left(30)
        spiral(0.95 * length, levels – 1)
```

Revisiting spiral function

- What if we wanted to end up back at the starting point, but not pick the pen up?
- We could trace our steps backwards.

```
def spiral(length, levels):
    if levels == 0:
        dot()
    else:
        forward(length)
        left(30)
        spiral(0.95 * length, levels – 1)
        right(30)
        backward(length)
```

broccoli_demo function

1. Define what the header function is:
   - broccoli(x, y, length, angle)
2. Define the recursive case:
   - broccoli is a line with three other broccolis at the end:
     - one directly straight out
     - one 20 degrees to the left
     - one 20 degrees to the right
   - the three other broccolis should be smaller/shorter than the current

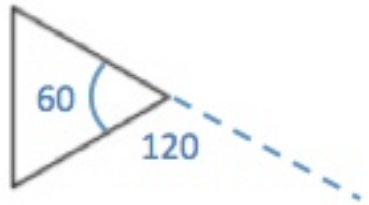
broccoli_demo function

3. Define the base case:
   - in each case, the length of the broccoli to be drawn gets shorter.
   - We stop at `length < 10` and place a yellow dot
4. put it all together!
   - Check the base case first:
     - if `length < 10`
       - Draw a yellow dot.
   - Otherwise:
     - draw three smaller broccolis at different angles.

- `new_x` and `new_y` are the ending coordinates of the line being drawn. We save them because after the first recursive call to broccoli the turtle won't be in the same place.

Practice time

- Write a function called `triangle` that draws an equilateral triangle (i.e., a triangle with all three sides the same length).
- Your function should take three parameters: the x and y coordinates where it should be drawn, and the length of the side of the triangle.
- The triangle should be drawn so that the left edge of the triangle is vertical.

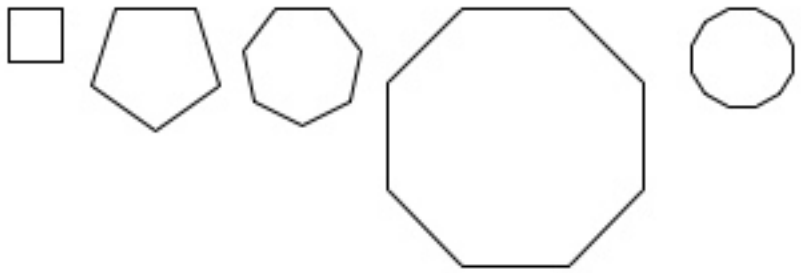


Practice time

```
def triangle(x, y, size):
    pu()
    goto(x,y)
    setheading(-30)
    pd()
    forward(size)
    right(120)
    forward(size)
    right(120)
    forward(size)
```

Practice time

- Write a function called `polygon` that draws a polygon.
- An n-sided polygon has n equal-length edges, and the angle between a straight line drawn from one side and the adjacent side is 360/n.
- Your function should take 4 parameters: the x and y location of the polygon, the number of sides, and the length of each side.
- The top edge of your polygon should always be perfectly horizontal.
- Again, the x and y coordinates may indicate any point on the polygon, so pick whatever seems easiest. But you can't just hard-code the number of sides. Instead, you'll have to use a loop.



Answer

```
def polygon(x, y, sides, size):
    pu()
    goto(x,y)
    setheading(0)
    angle = 360.0/sides
    pd()
    for i in range(sides):
        forward(size)
        right(angle)
```

Code

turtle-examples.py

turtle-recursion.py