



Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 4: Loops and the **turtle**

Goals for today

- Explain when loops are used.
- Describe the execution of a `for` loop.
- Use loops for more complex `turtle` drawings.
- Practice implementing loops and nested loops.

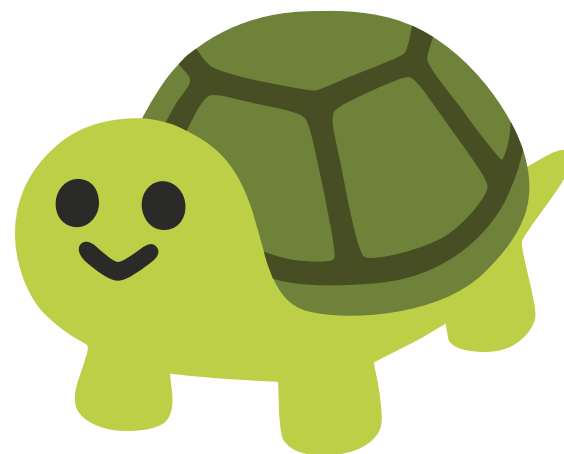
To follow along: create a `class04.py` script where we will write code together.
(in your `cs146` folder)

First, another note about the modulus operator (`%`):

`r = a % b` means `r = a - b * math.floor(a / b)`
(`type(r)` depends on `type(a)` and `type(b)`)

The **turtle** module allows us to make drawings with **Python**!

- **forward**, **backward**: move forward/backward by a certain number of units.
- **left**, **right**: turn left/right by a certain number of degrees.
- **speed**: set drawing speed (0 is no animation, 1 is slowest, 10 is fast).
- **shape**: set the shape of the turtle (e.g. **shape('turtle')** makes it a true turtle).
- **stamp**: stamps the turtle at the current location.
- **setpos(x, y)**: moves the turtle to the location with coordinates **x**, **y**.
- **penup/pendown**: lifts or puts down the pen to control whether drawing occurs.
- **pencolor**: sets the drawing color.
- **fillcolor**: sets the color to used when filling between **begin_fill** and **end_fill**
- **begin_fill/end_fill**: delimits filling a shape.
- **circle**: draws a circle.



Example turtle program to draw a square.

```
1 from turtle import *
2
3 def draw_square():
4     forward(100)
5     right(90)
6     forward(100)
7     right(90)
8     forward(100)
9     right(90)
10    forward(100)
11    right(90)
12
13 draw_square()
```

Hmm. This doesn't look very DRY. How can we make this better?

Anatomy of a **for**-loop.

```
1 for iteration_variable in sequence:
2     # do something repeatedly, maybe using iteration_variable
```

Example:

```
1 def print_loop(n):
2     """ Print numbers from 0 until (but not including) n """
3     print("Begin list of numbers")
4     for i in range(n):
5         print(i)
6     print("End list of numbers")
7
8 print_loop(5)
```

`range(0, 10)` creates a sequence of integers starting at 0 up to 9 (i.e. up to but not including 10).

```
1 from turtle import *
2
3 def draw_square():
4     for i in range(4):
5         forward(100)
6         right(90)
```

Question 1: What will this code print?

```
1 sum = 0
2 for i in range(10):
3     sum = sum + 1
4 print(sum)
```

A. 0

B. 9

C. 10

D. 45

E. 55

Question 2: What will this code print?

```
1 sum = 0
2 for i in range(10):
3     sum = sum + i
4 print(sum)
```

- A. 0
- B. 9
- C. 10
- D. 45
- E. 55

Example: drawing a spiral.

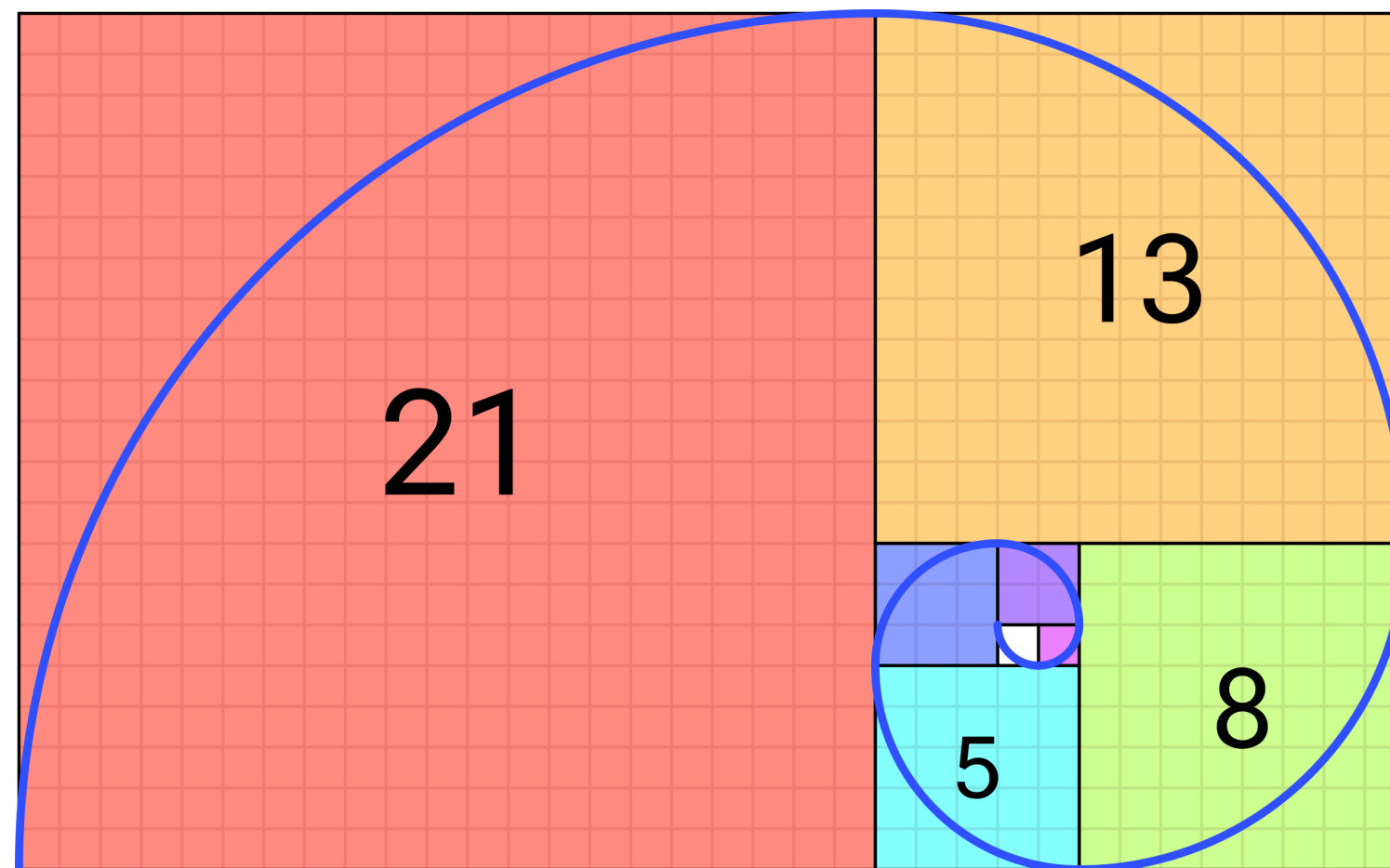
```
1 # A good example is side = 200 and angle = 89
2 def spiral(sides, angle):
3     """ Draw spiral with sides steps and update of angle """
4     for i in range(sides):
5         forward(i*5)
6         right(angle)
```

Example: wandering **turtle**.

```
1 def walk(num_steps, step_size):  
2     """ Random walk with num_steps steps of step_size """  
3     for i in range(num_steps):  
4         angle = randint(-90, 90)  
5         right(angle)  
6         forward(step_size)
```

More interesting shapes: Fibonacci spiral.

```
1 >>> import turtle
2 >>> help(turtle.circle)
3 >>> turtle.circle(100, 90) # draws a quarter circle with radius of 100
```



First try to find a pattern in the radius (hint: they are Fibonacci numbers).
How will you compute these numbers with a **for**-loop?

Possible solution to `golden_spiral`.

```
1 def golden_spiral(radius, segments):
2     """
3     Draw a Fibonacci spiral using Turtle. turtle package must be imported into namespace.
4
5     Args:
6         radius: Starting radius of the spiral
7         segments: Number of quarter circle segments to draw after initial quarter circle.
8                   Must be >= 2.
9
10    Returns:
11        None
12    """
13    a = radius
14    circle(a, 90)
15    b = radius
16    circle(b, 90)
17    # range can take two arguments, start and stop. We set the start at two since the first
18    # segments are already drawn
19    for i in range(2, segments):
20        c = a + b
21        circle(c, 90)
22        # Prepare for the next iteration of the loop
23        a = b
24        b = c
```

Note that we can also *nest* **for** loops!

How can we **print** these patterns?

```
% % % %  
% % % %  
% % % %  
% % % %  
% % % %  
% % % %
```

```
@  
@ @  
@ @ @
```

```
1 def print_rect(width, height, char):  
2     """Print an ASCII rectangle  
3  
4     Args:  
5         width: number of columns of rectangle  
6         height: number of rows of rectangle  
7         char: Character to print  
8     """  
9     for i in range(height):  
10         for j in range(width):  
11             print(char, end=" ")  
12         print() # new line at end of the row
```

```
1 def print_triangle(size, char):  
2     """Print an ASCII lower triangle  
3  
4     Args:  
5         size: number of rows/columns  
6         char: Character to print  
7     """  
8     for i in range(size):  
9         for j in range(i+1):  
10             print(char, end=" ")  
11         print()
```

Summary

- `for`-loops are good for repeating blocks of code.
- We used `range` to generate a *sequence* of numbers.
Remember that `range(n)` creates numbers from `0` to `n - 1`.
- Next week: other types of sequences.
- **Office hours:**
Monday 10 - 11am, Tuesday 11am - 11:30am (my office, room 219) and Thursdays 2:30 - 4:30pm (room 224).
- Other help hours: see `go/smith` (Smith Gakuya, ASI) and `go/cshelp`.
- Quiz 2 (15 minutes) and Lab 2 on Friday.
- PA1 (initial) submission due tomorrow.

What I did this summer CS seminar (Part 2) on Friday at 12:30pm (with pizza)!