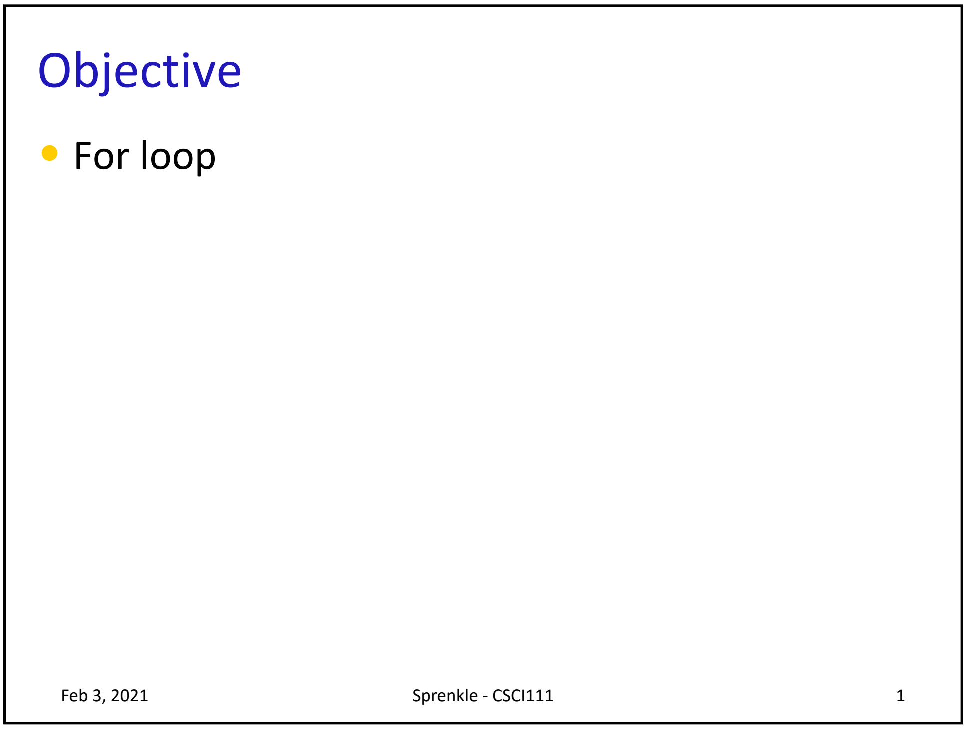
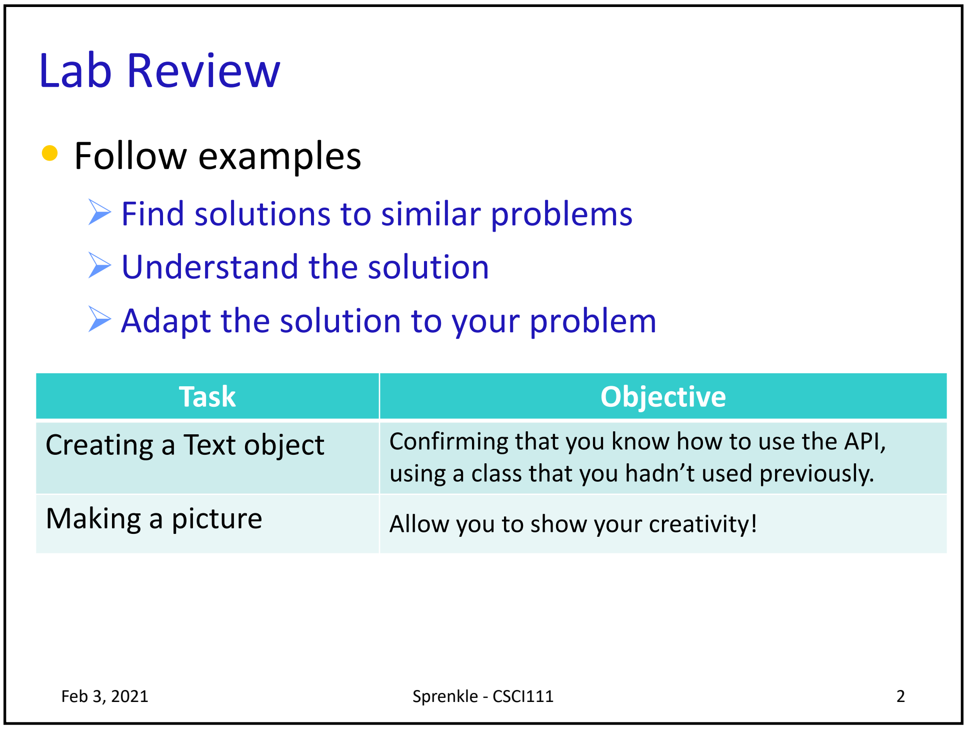


# Objective

- For loop



# Lab Review

- Follow examples
  - Find solutions to similar problems
  - Understand the solution
  - Adapt the solution to your problem

| Task | Objective |
|---|---|
| Creating a Text object | Confirming that you know how to use the API, using a class that you hadn't used previously. |
| Making a picture | Allow you to show your creativity! |

## Recommendations

- Review the slides, example programs, and/or textbook every day to review what we discussed
  - ➢ This problem made sense in class… Does it still make sense?
- Practice a problem every day
  - ➢ I rarely use problems from the text book so they’re good practice
- Ask questions
- “sense of accomplishment after lab”

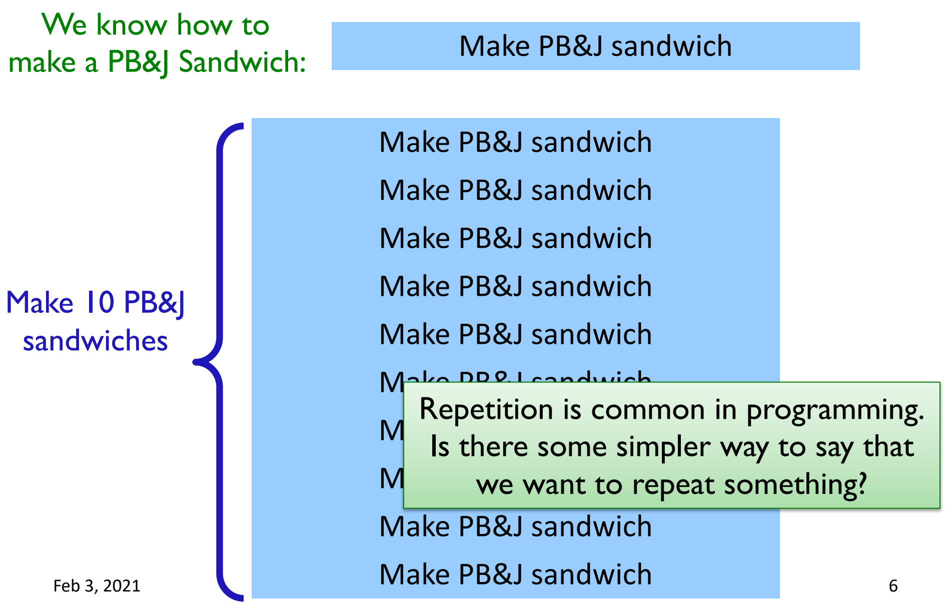
## FOR LOOPS

# Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Super Power: Superhuman Speed

# Looping/Repetition



We know how to make a PB&J Sandwich:

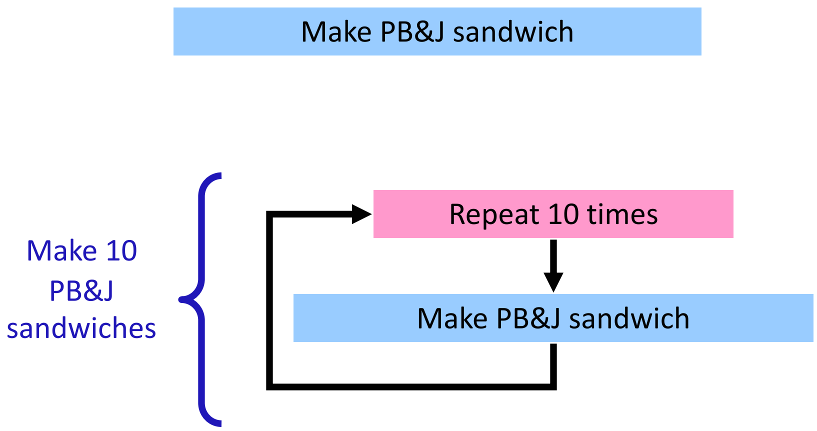
Make PB&J sandwich

Make 10 PB&J sandwiches

Make PB&J sandwich
Make PB&J sandwich
Make PB&J sandwich
Make PB&J sandwich
Make PB&J sandwich
Make PB&J sandwich
M
M
Make PB&J sandwich
Make PB&J sandwich

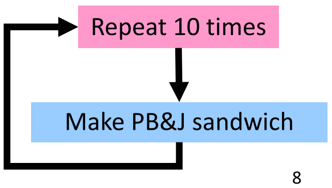
Repetition is common in programming. Is there some simpler way to say that we want to repeat something?

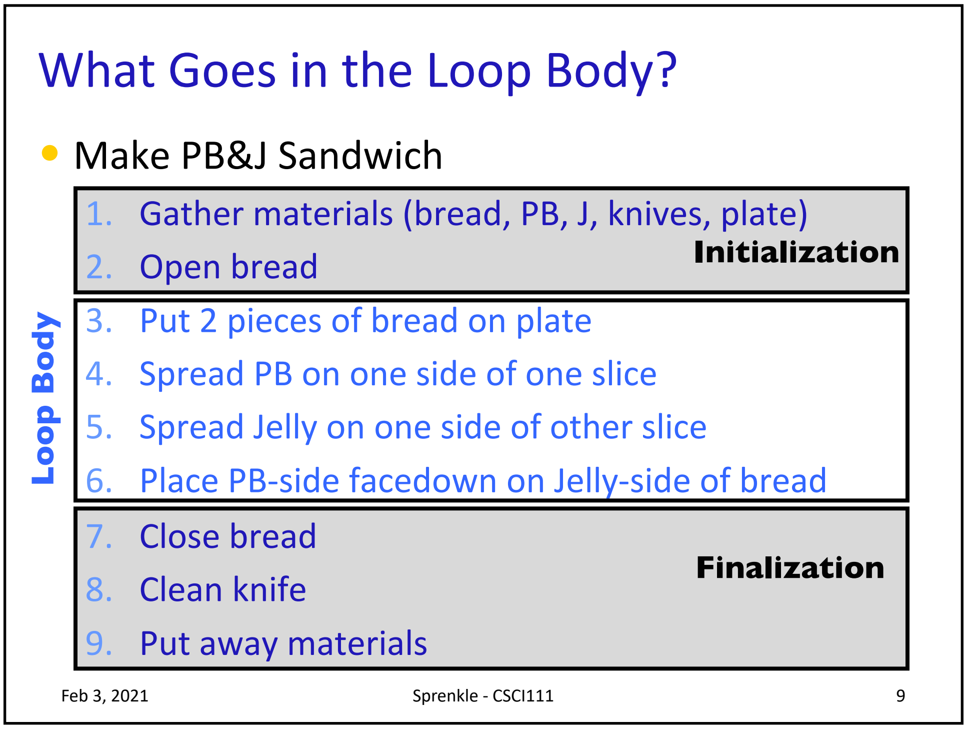
# Looping/Repetition



# What Goes in the Loop Body?

- Make PB&J Sandwich
  1. Gather materials (bread, PB, J, knives, plate)
  2. Open bread
  3. Put 2 pieces of bread on plate
  4. Spread PB on one side of one slice
  5. Spread Jelly on one side of other slice
  6. Place PB-side facedown on Jelly-side of bread
  7. Close bread
  8. Clean knife
  9. Put away materials





## What Goes in the Loop Body?

- Make PB&J Sandwich

Initialization

1. Gather materials (bread, PB, J, knives, plate)
2. Open bread

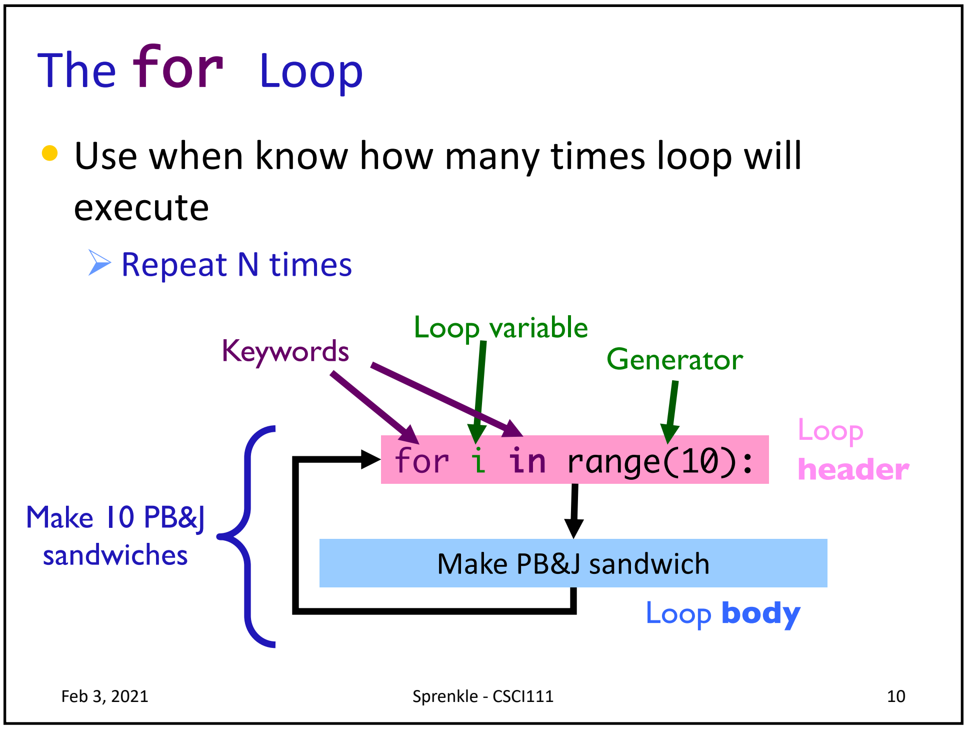
Loop Body

3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread

Finalization

7. Close bread
8. Clean knife
9. Put away materials

9



## The for Loop

- Use when know how many times loop will execute
  - Repeat N times

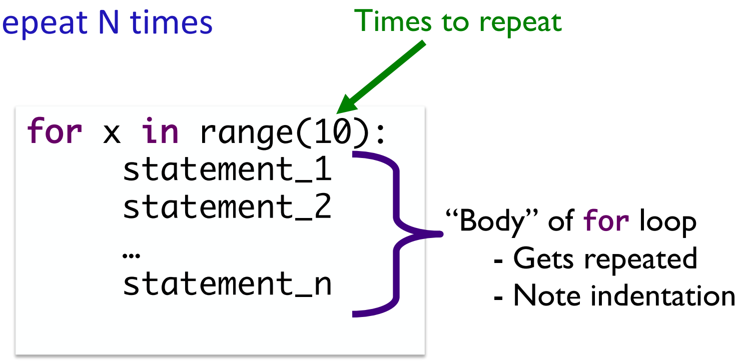
```
for i in range(10):
```

Make PB&J sandwich

10

## for Loop Syntax and Semantics

- Use when know how many times loop will execute
  - Repeat N times

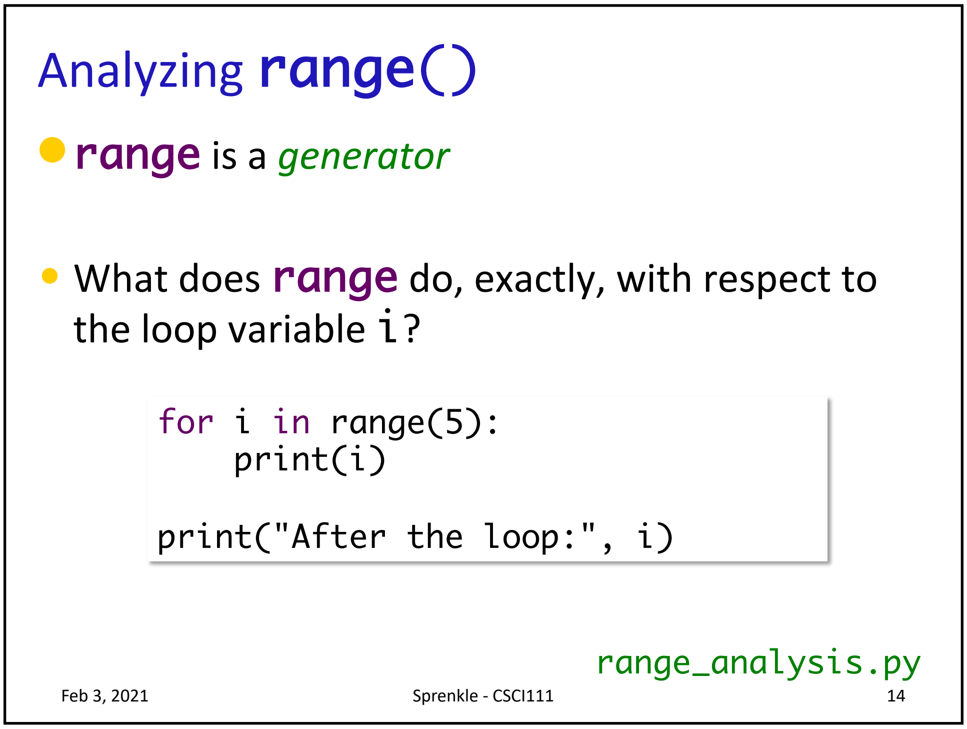


```
for x in range(10):
     statement_1
     statement_2
     …
     statement_n
```

## Modify the Move Circle Program

- Allow the user to click to move the circle 3 times
- Process: Figure out
  - What needs to be repeated? → for statement
  - How many times does it need to be repeated? → body of for loop

`circle_move_repeat.py`



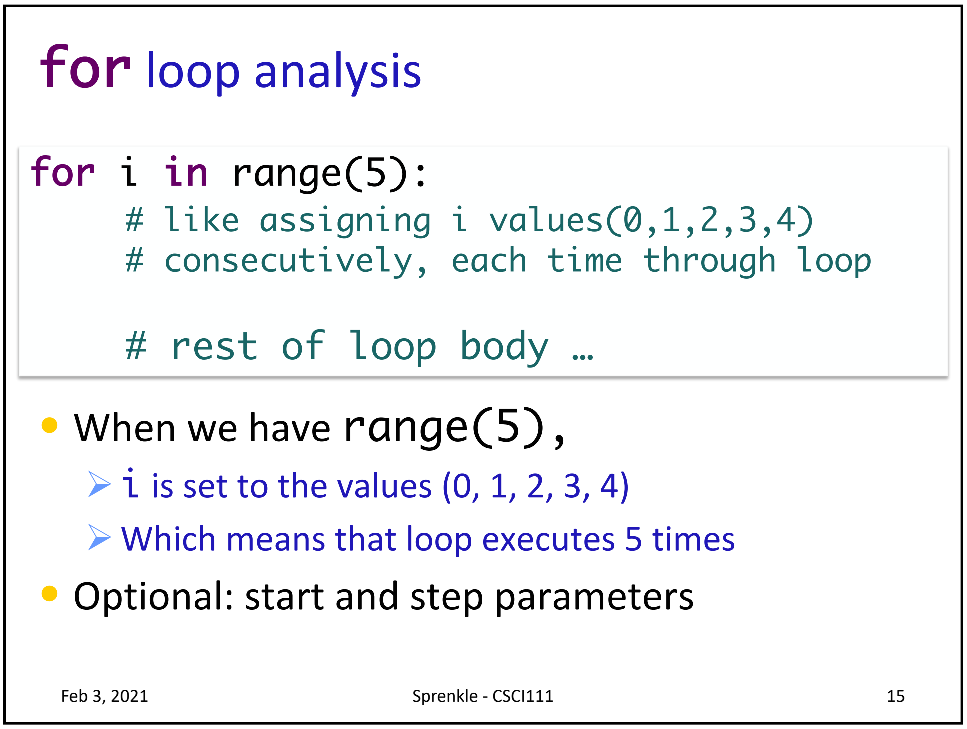
## Analyzing range()

- **range** is a *generator*
- What does **range** do, exactly, with respect to the loop variable `i`?

```
for i in range(5):
    print(i)

print("After the loop:", i)
```

range_analysis.py

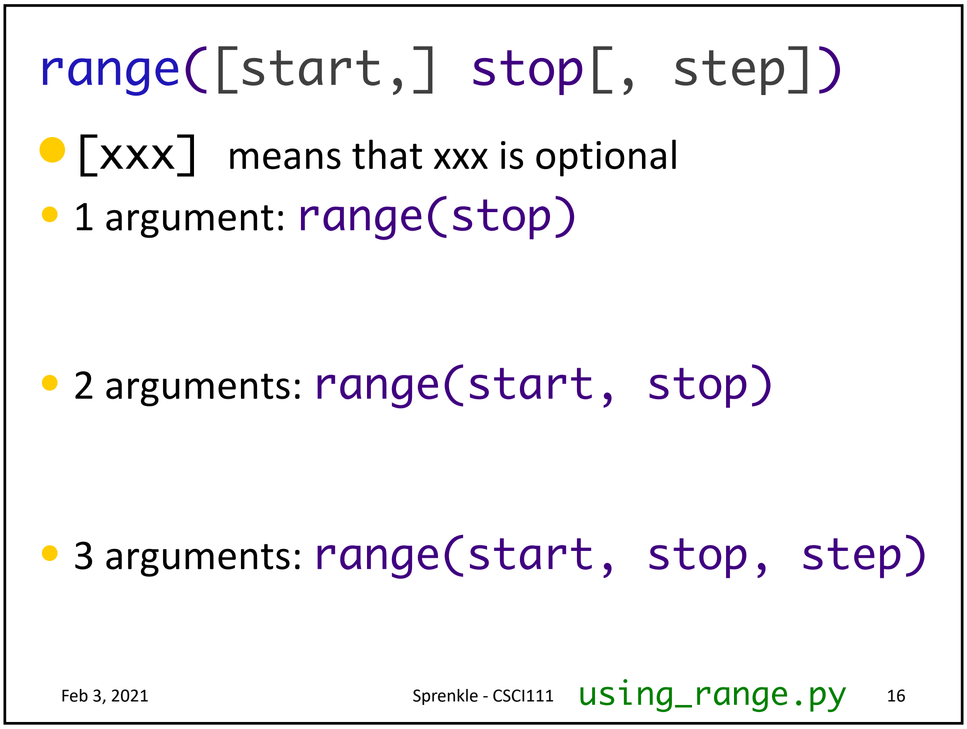


## for loop analysis

```
for i in range(5):
    # like assigning i values(0,1,2,3,4)
    # consecutively, each time through loop

    # rest of loop body …
```

- When we have `range(5),`
  - `i` is set to the values (0, 1, 2, 3, 4)
  - Which means that loop executes 5 times
- Optional: start and step parameters

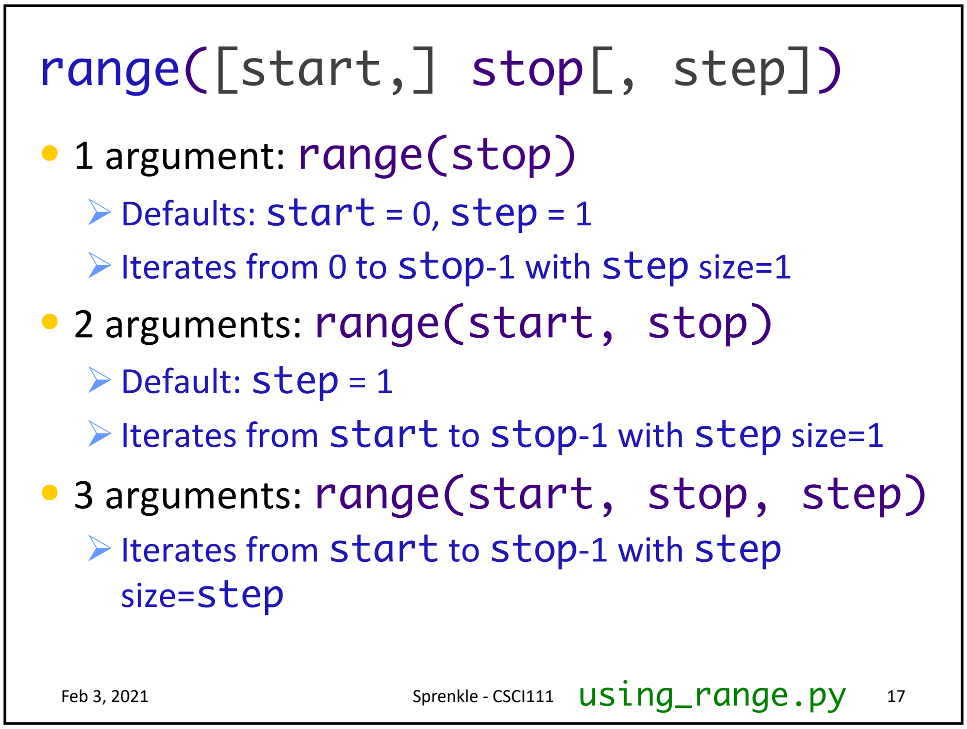


## range([start,] stop[, step])

- [xxx] means that xxx is optional
- 1 argument: range(stop)
- 2 arguments: range(start, stop)
- 3 arguments: range(start, stop, step)

using_range.py

16

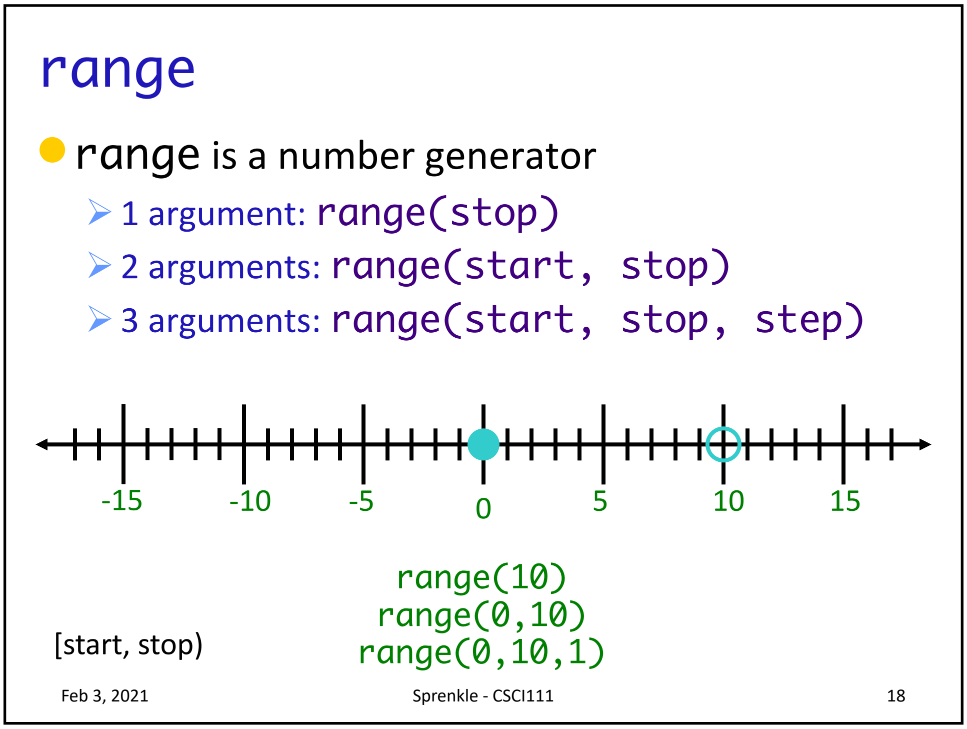


## range([start,] stop[, step])

- 1 argument: range(stop)
  - Defaults: start = 0, step = 1
  - Iterates from 0 to stop-1 with step size=1
- 2 arguments: range(start, stop)
  - Default: step = 1
  - Iterates from start to stop-1 with step size=1
- 3 arguments: range(start, stop, step)
  - Iterates from start to stop-1 with step size=step

using_range.py

17

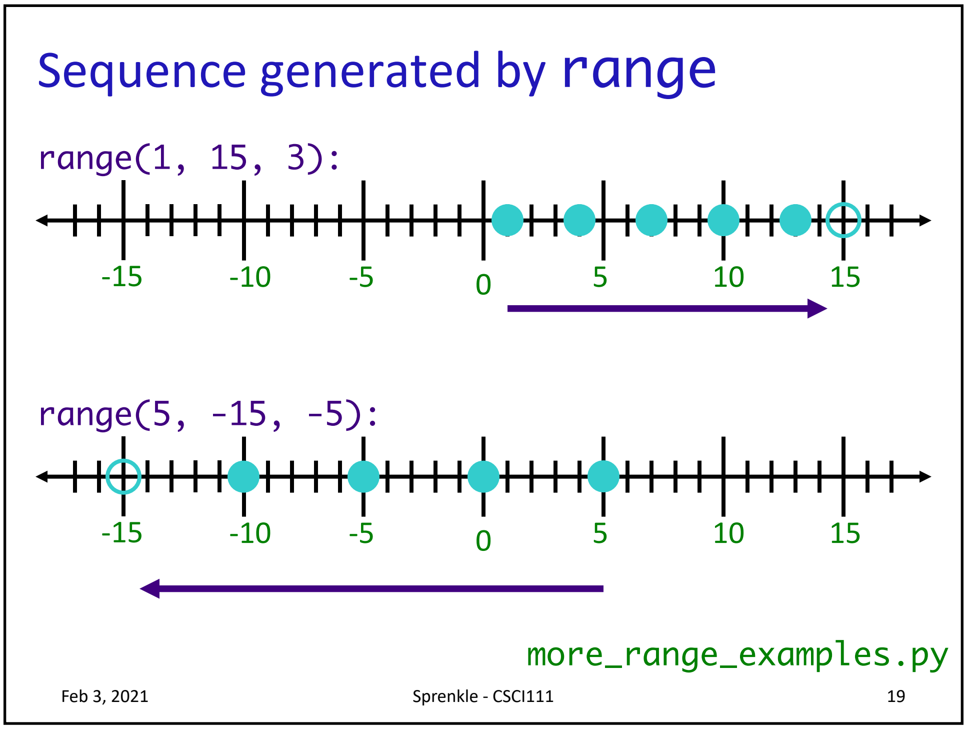


# range

- `range` is a number generator
  - 1 argument: `range(stop)`
  - 2 arguments: `range(start, stop)`
  - 3 arguments: `range(start, stop, step)`

```
range(10)
range(0,10)
range(0,10,1)
```

[start, stop)

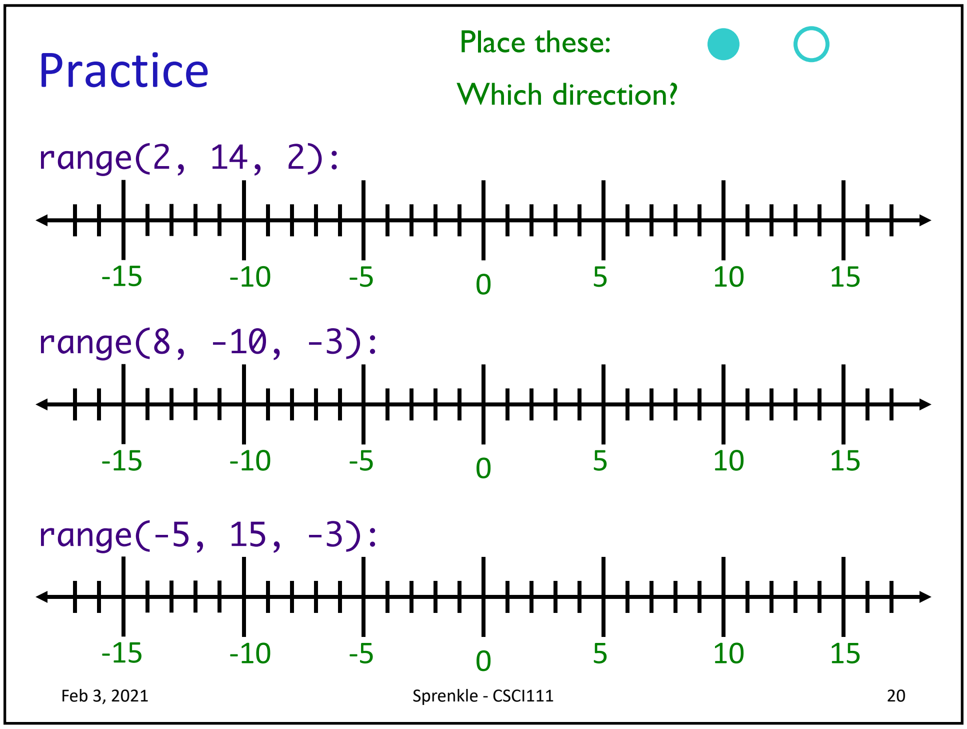


# Sequence generated by range

`range(1, 15, 3):`

`range(5, -15, -5):`

more_range_examples.py



# Practice

Place these: ● ○

Which direction?

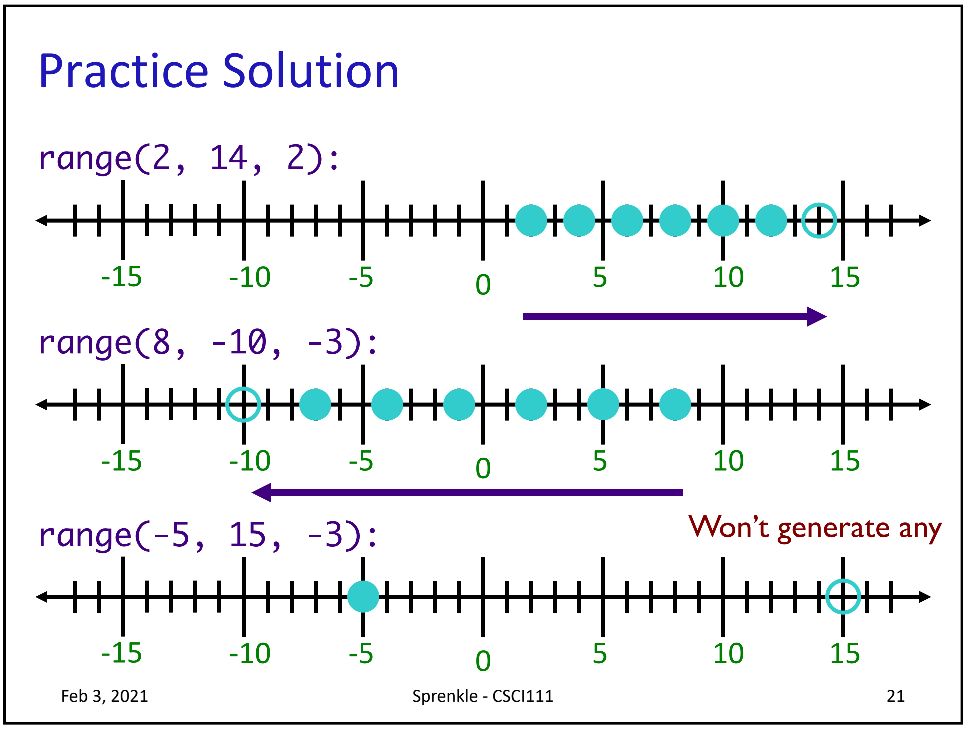
`range(2, 14, 2):`

`range(8, -10, -3):`

`range(-5, 15, -3):`

Feb 3, 2021 Sprenkle - CSCI111 20

20



# Practice Solution

`range(2, 14, 2):`

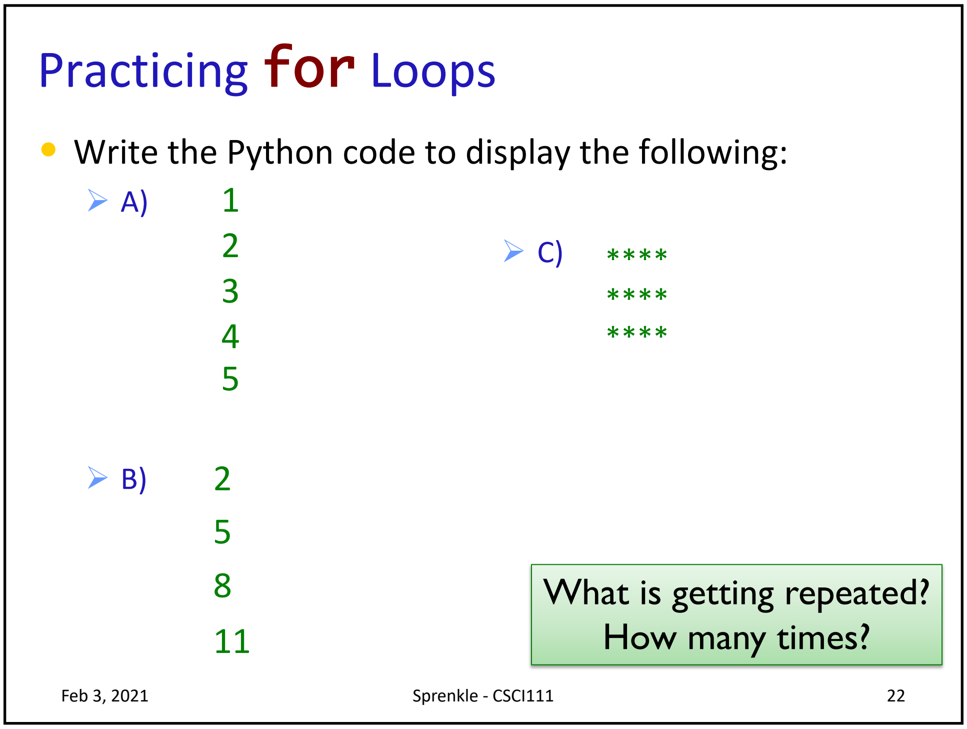
`range(8, -10, -3):`

`range(-5, 15, -3):`

Won’t generate any

Feb 3, 2021 Sprenkle - CSCI111 21

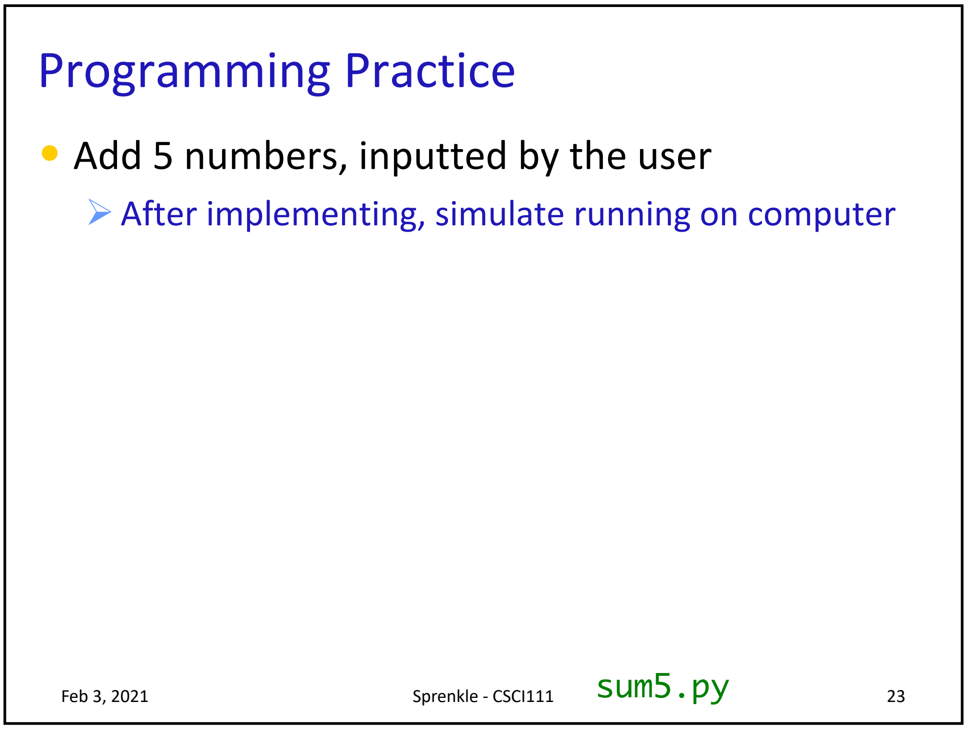
21



## Practicing for Loops

- Write the Python code to display the following:
  - A)
    ```
    1
    2
    3
    4
    5
    ```
  - B)
    ```
    2
    5
    8
    11
    ```
  - C)
    ```
    ****
    ****
    ****
    ```

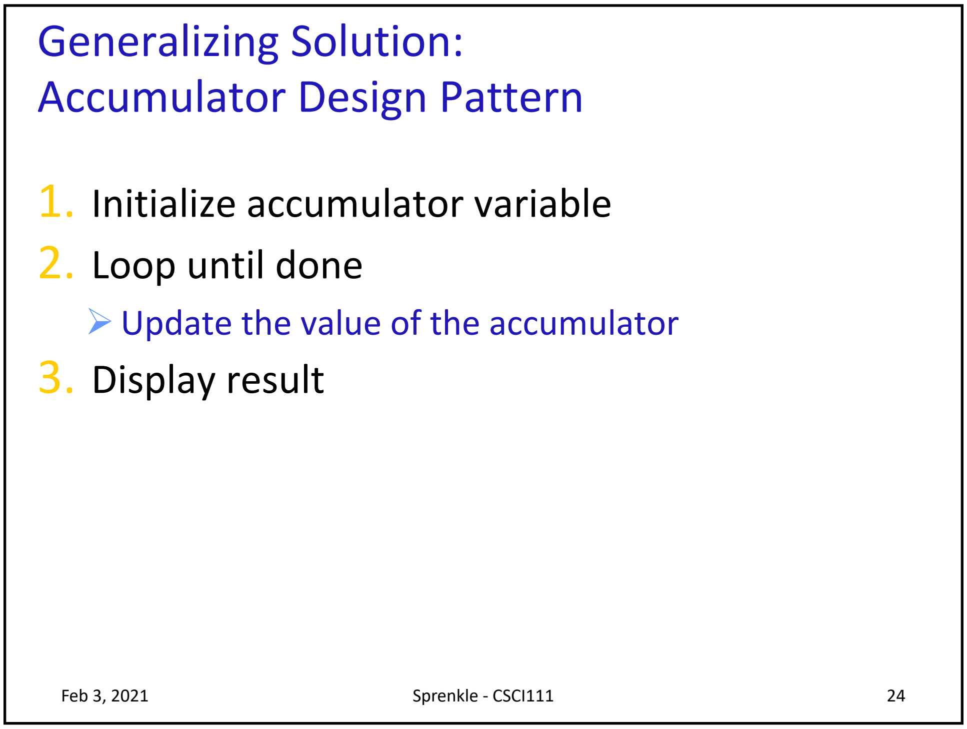
What is getting repeated?
How many times?



## Programming Practice

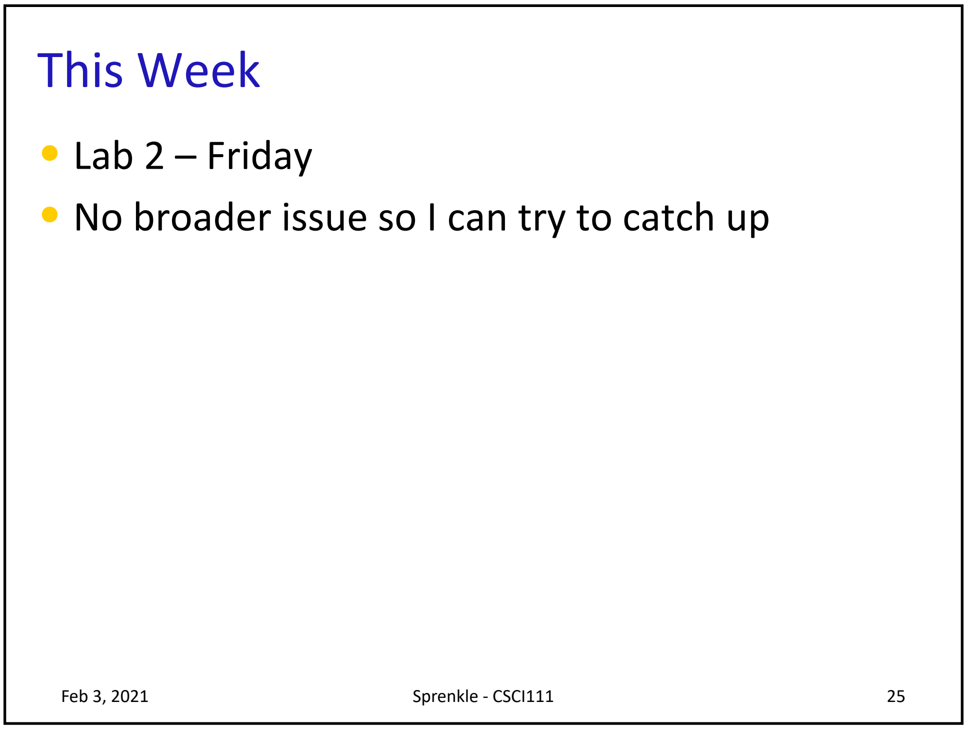
- Add 5 numbers, inputted by the user
  - After implementing, simulate running on computer

`sum5.py`



## Generalizing Solution: Accumulator Design Pattern

1. Initialize accumulator variable
2. Loop until done
   - ➢ Update the value of the accumulator
3. Display result



## This Week

- Lab 2 – Friday
- No broader issue so I can try to catch up