Objectives

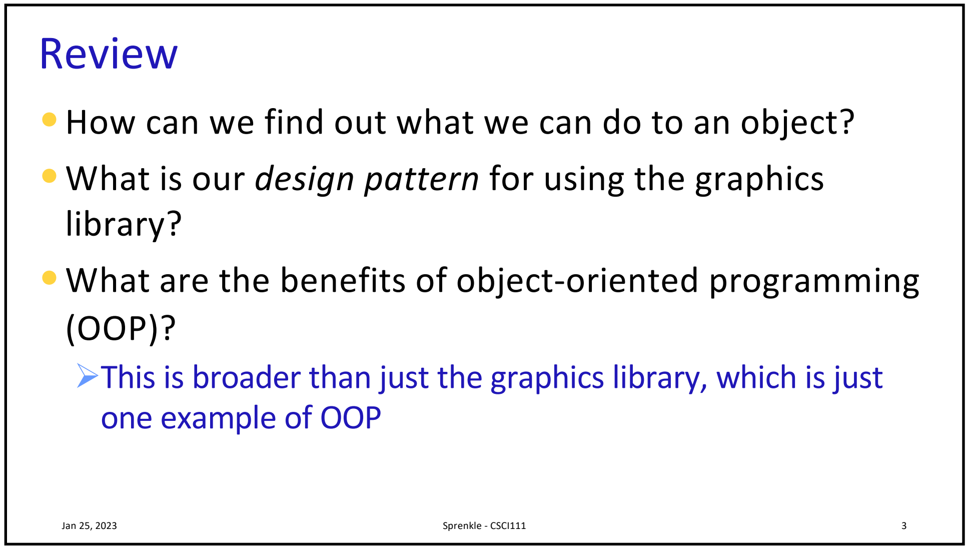
- For Loops

Lab Review

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

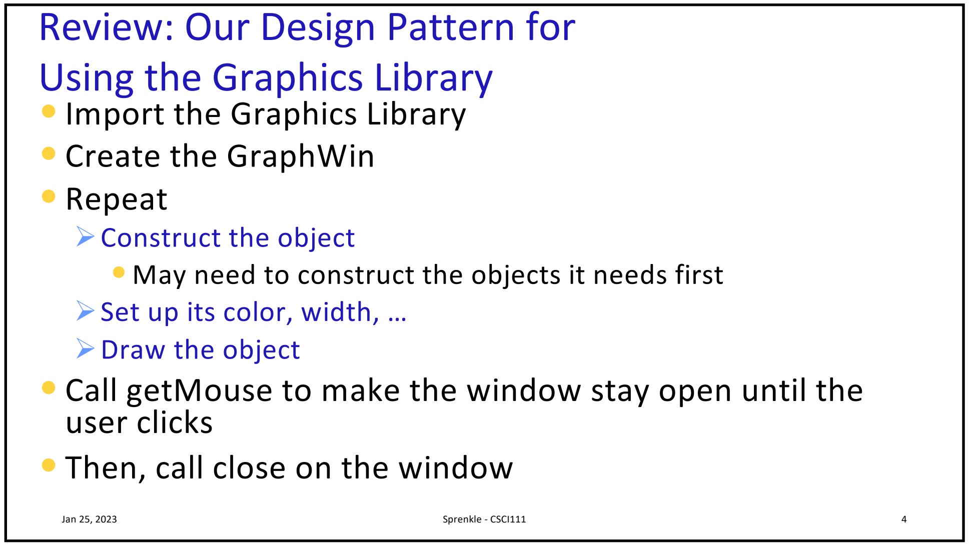
Task	Objective
Creating snowperson	Using an API to solve a new problem
Making a picture	Allow you to show your creativity!

- Celebrate your successes!



Review

- How can we find out what we can do to an object?
- What is our *design pattern* for using the graphics library?
- What are the benefits of object-oriented programming (OOP)?
  - This is broader than just the graphics library, which is just one example of OOP



Review: Our Design Pattern for Using the Graphics Library

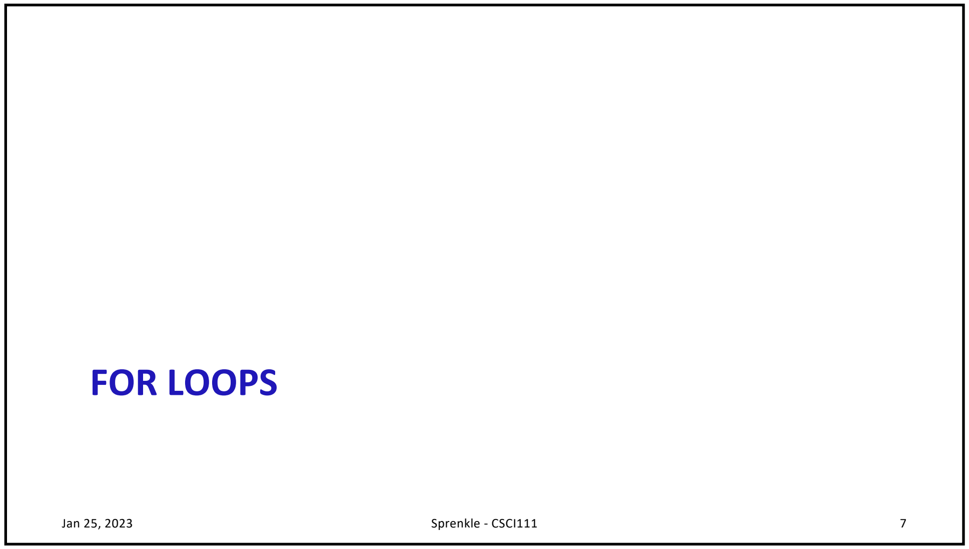
- Import the Graphics Library
- Create the GraphWin
- Repeat
  - Construct the object
    - May need to construct the objects it needs first
  - Set up its color, width, …
  - Draw the object
- Call getMouse to make the window stay open until the user clicks
- Then, call close on the window

Benefits of Object-Oriented Programming

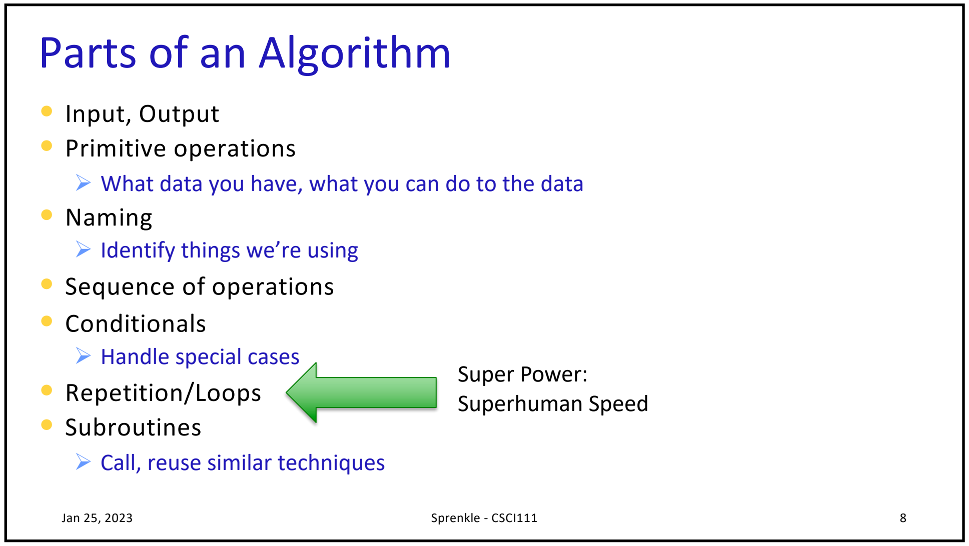
- **Abstraction**
  - ➢Hides details of underlying implementation
  - ➢Easier to change implementation
- Collects related data/methods together
  - ➢Easier to reason about data
- Less code in main program
  - ➢Our program code is relatively simple

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”



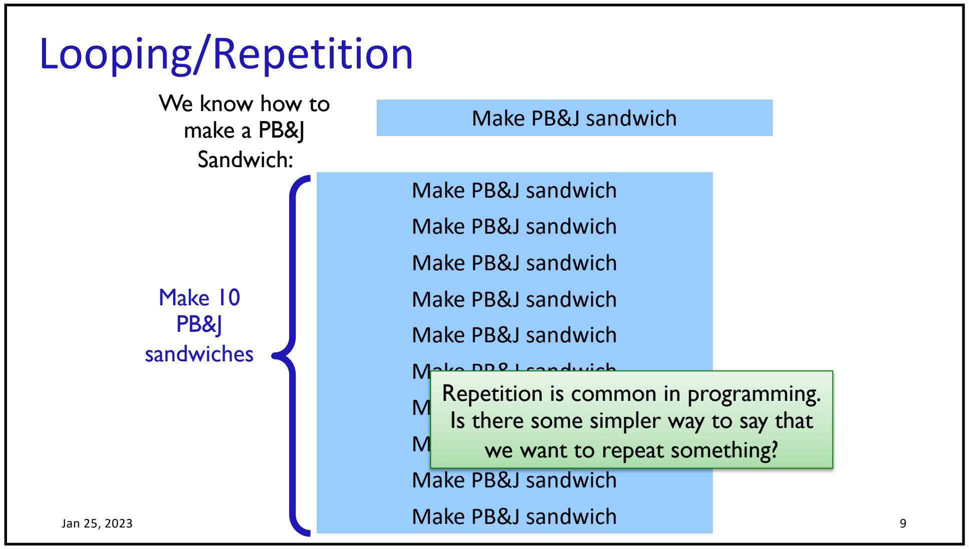
FOR LOOPS



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

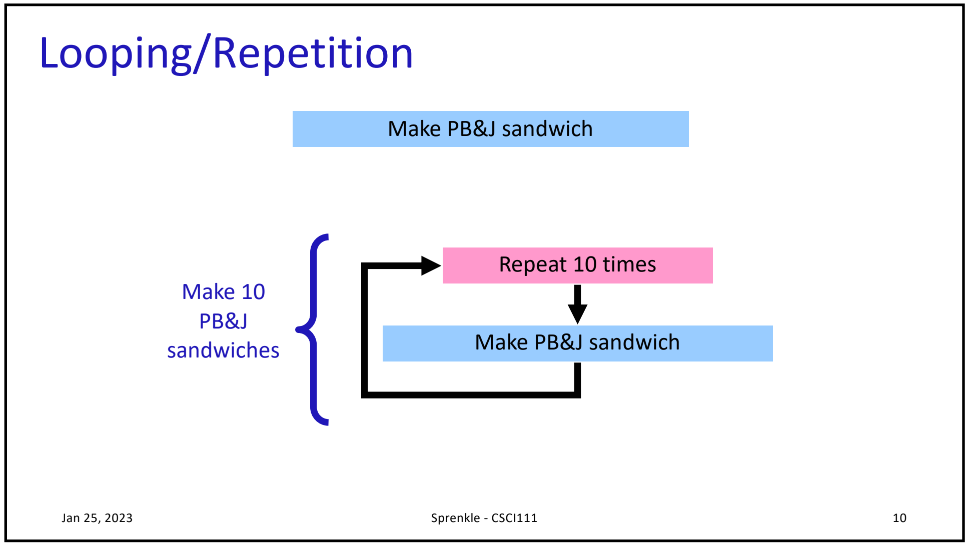
We know how to make a PB&J Sandwich:

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?



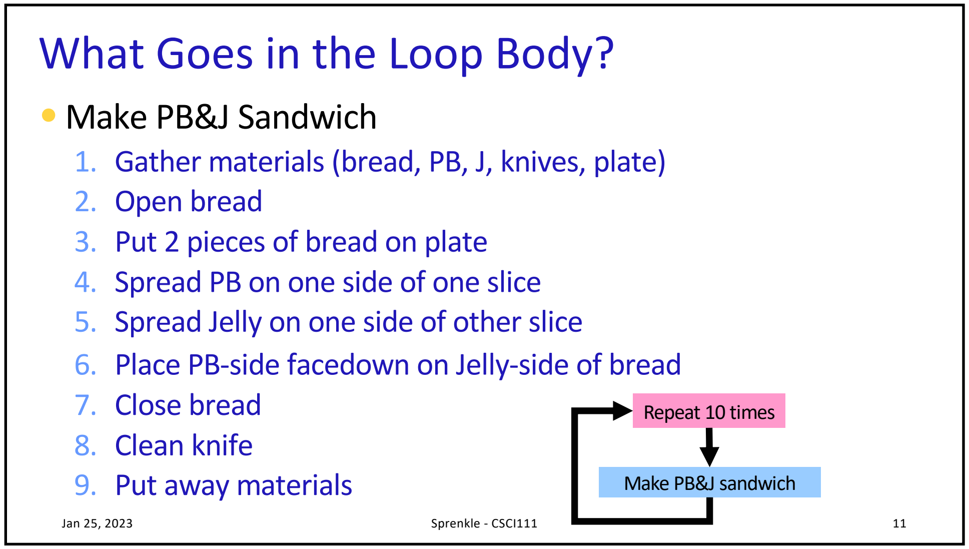
Looping/Repetition

Make PB&J sandwich

Make 10 PB&J sandwiches

Repeat 10 times

Make PB&J sandwich

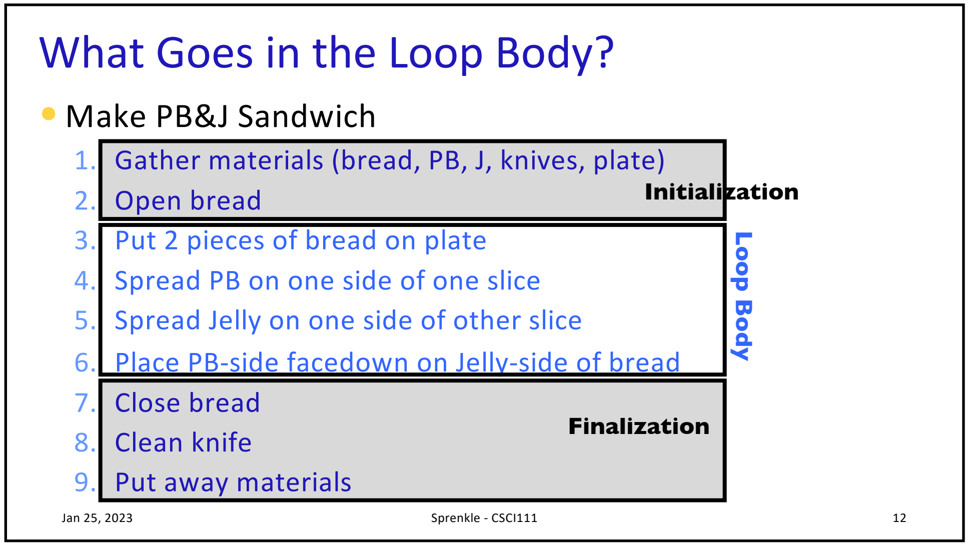


What Goes in the Loop Body?

- 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

Repeat 10 times

Make PB&J sandwich



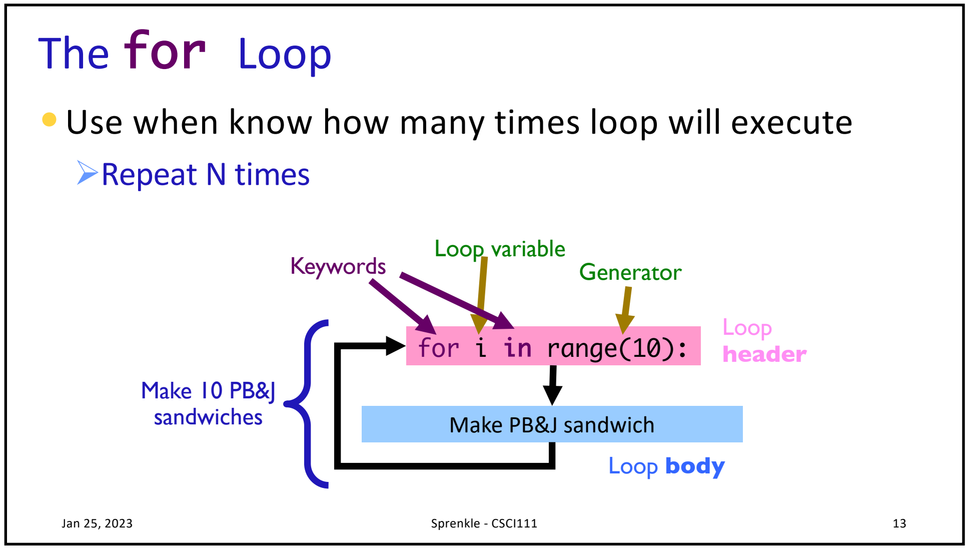
What Goes in the Loop Body?

- 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

Initialization

Loop Body

Finalization



The for Loop

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

Keywords

Loop variable

Generator

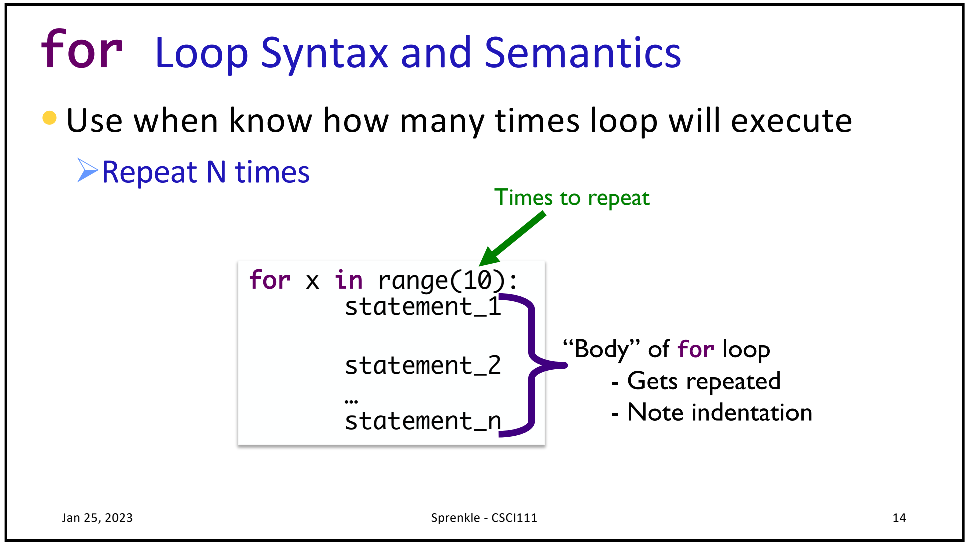
```
for i in range(10):
```

Loop **header**

Make 10 PB&J sandwiches

Make PB&J sandwich

Loop **body**



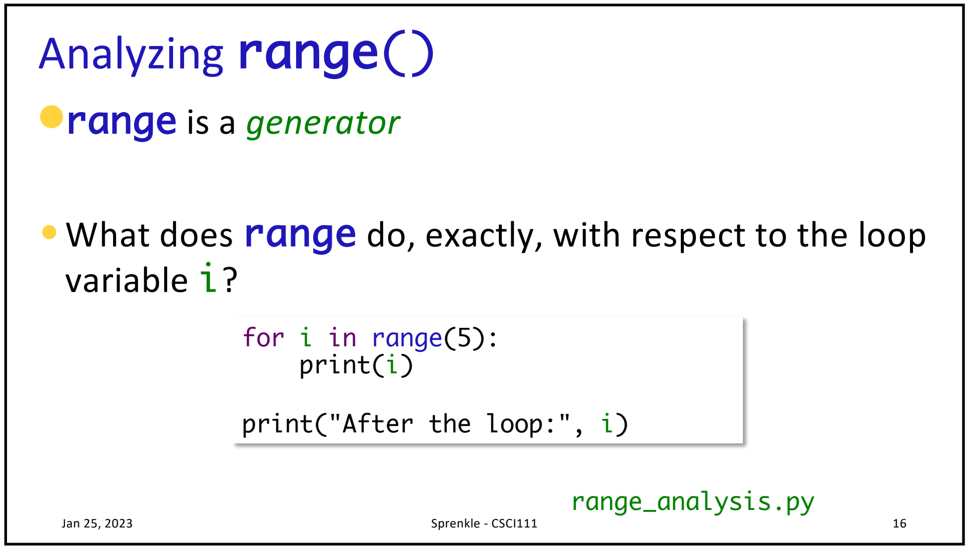
for Loop Syntax and Semantics

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

Times to repeat

```
for x in range(10):
        statement_1

        statement_2
        …
        statement_n
```

“Body” of for loop
- Gets repeated
- Note indentation



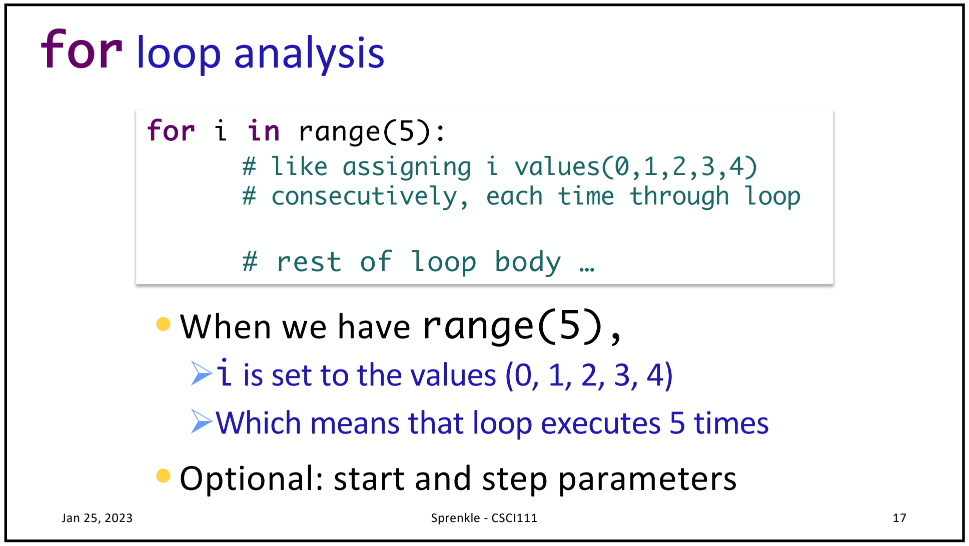
Analyzing range()

- **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)
```

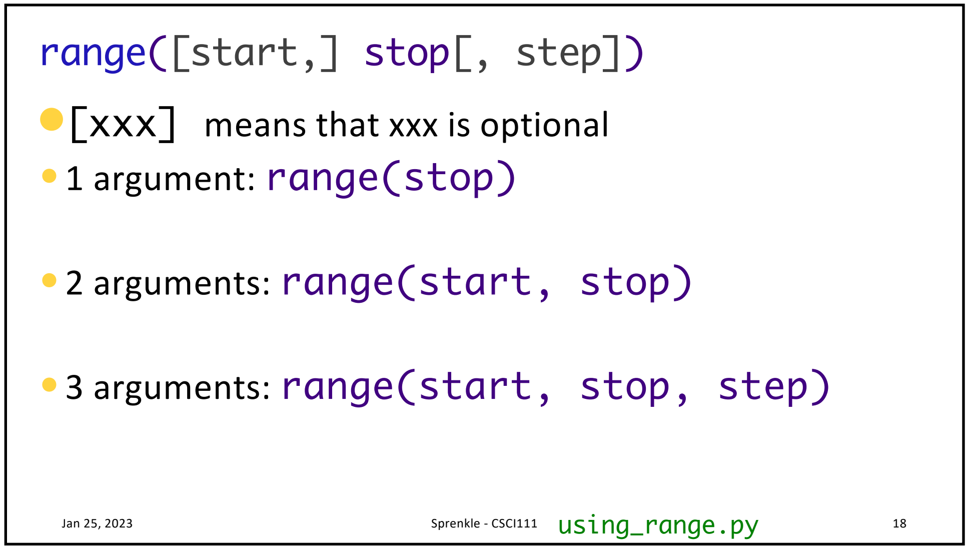
range_analysis.py



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 …
```

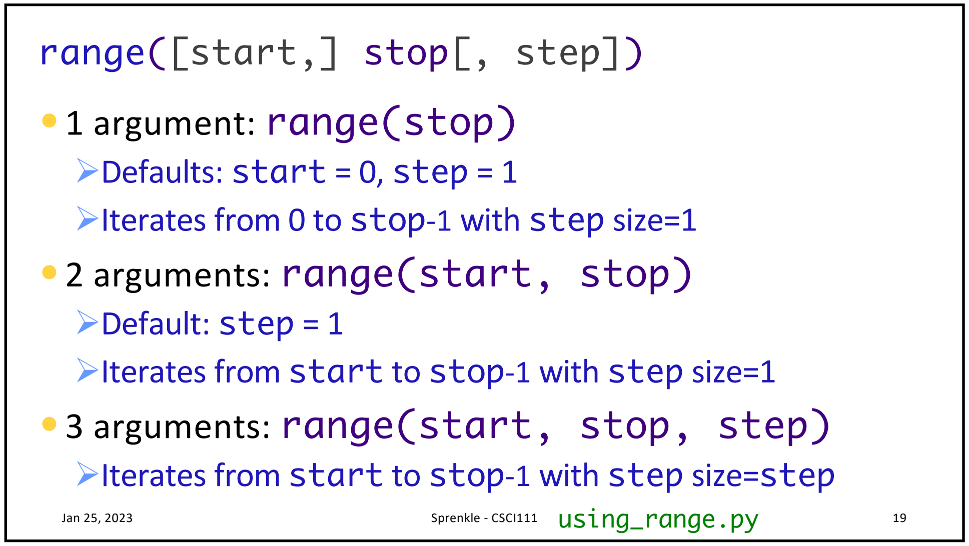
- When we have range(5),
  - i is set to the values (0, 1, 2, 3, 4)
  - Which means that loop executes 5 times
- Optional: start and step parameters



`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)`

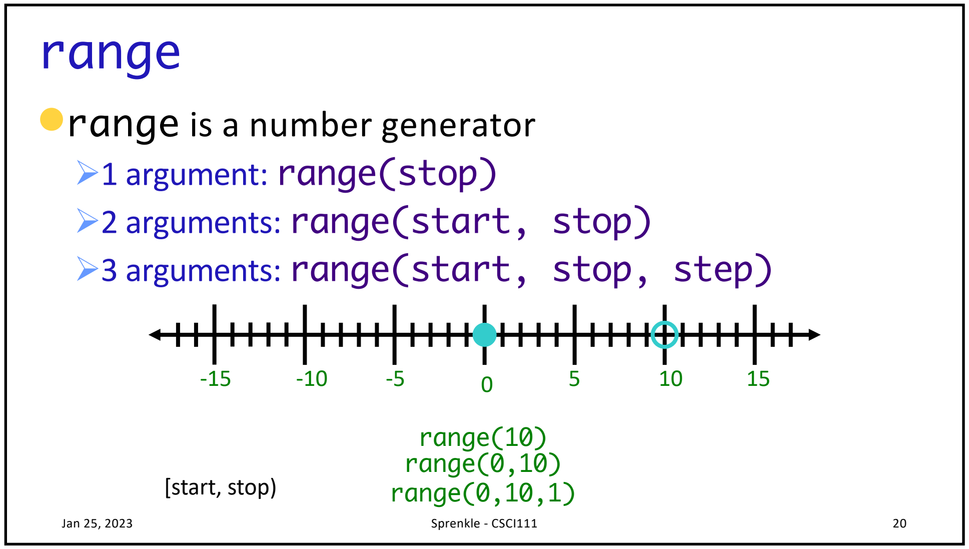
`using_range.py`



`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`



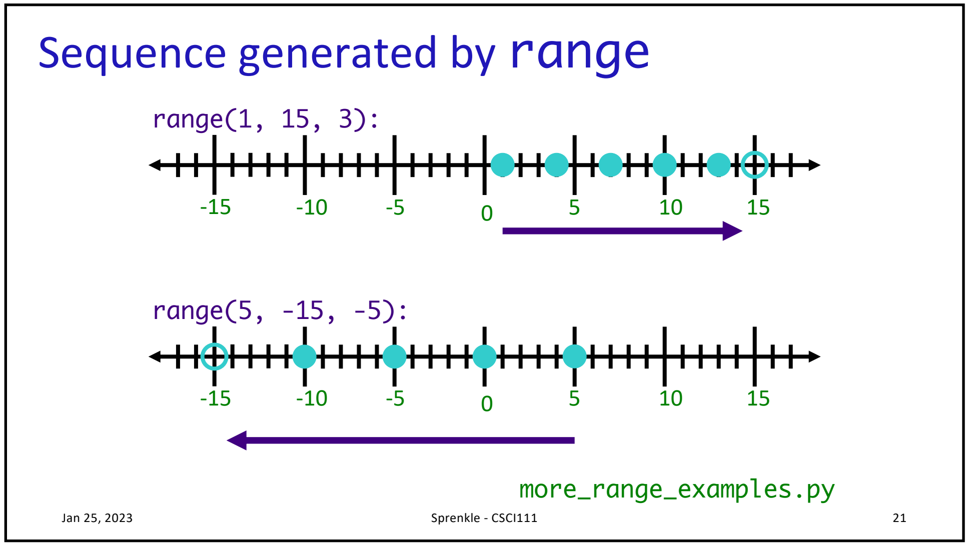
range

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

-15
-10
-5
0
5
10
15

[start, stop)

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



Sequence generated by range

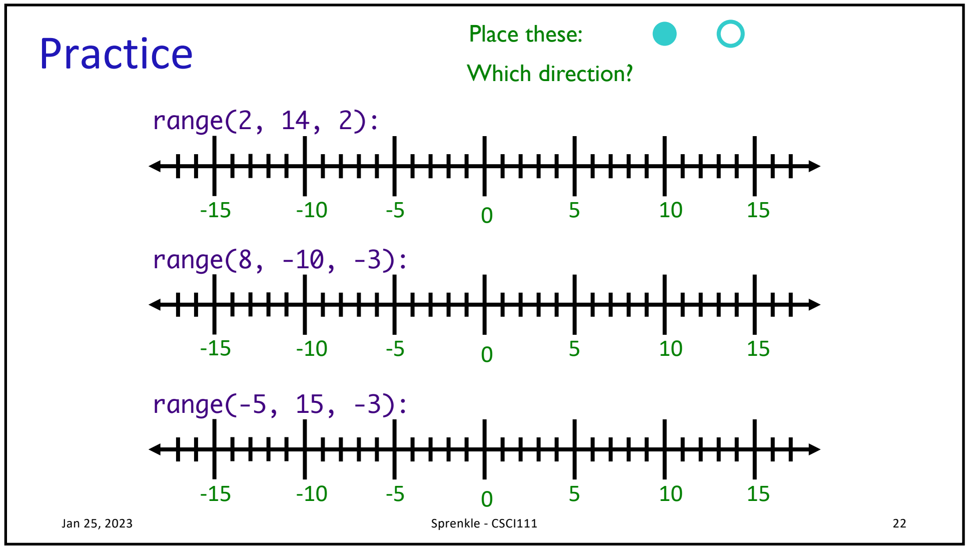
range(1, 15, 3):

-15
-10
-5
0
5
10
15

range(5, -15, -5):

-15
-10
-5
0
5
10
15

more_range_examples.py



Practice

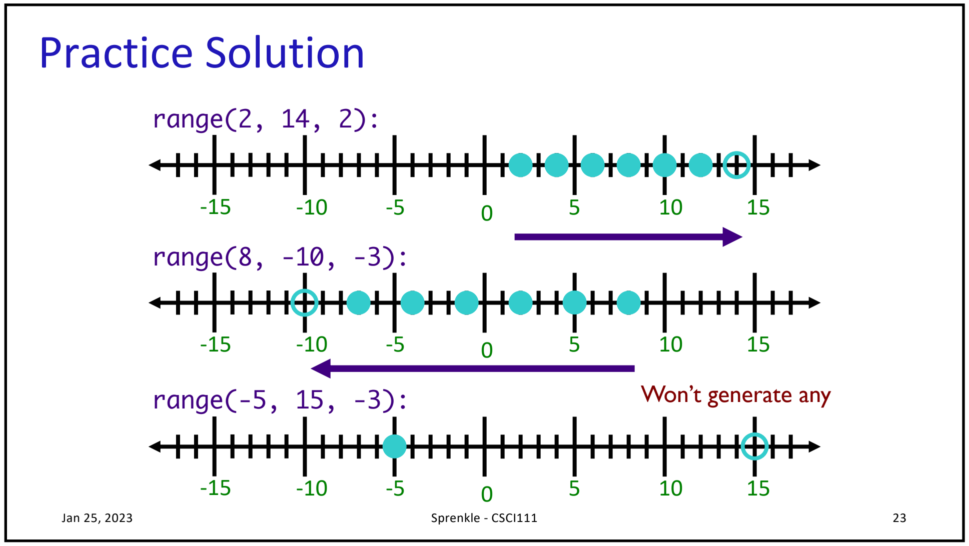
Place these: ● ○

Which direction?

`range(2, 14, 2):`

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

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



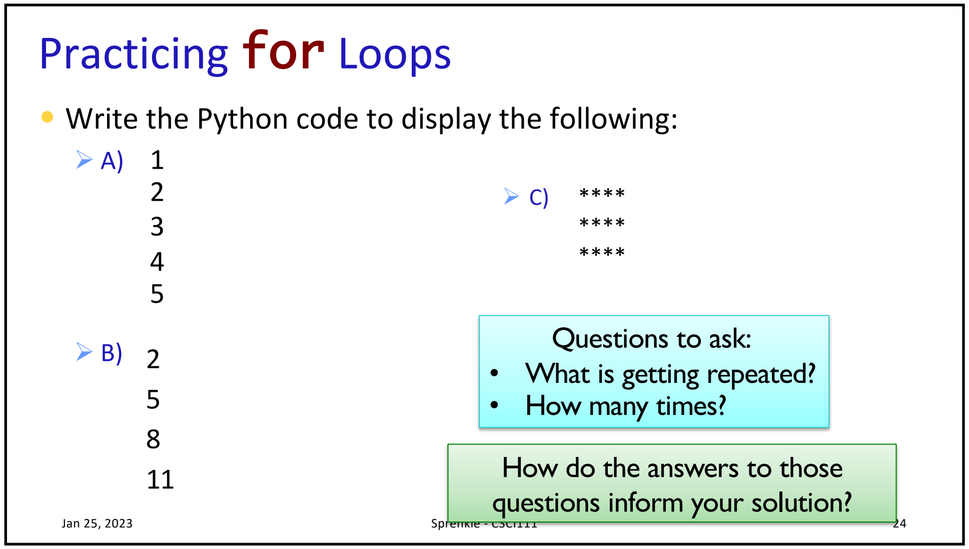
Practice Solution

`range(2, 14, 2):`

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

Won't generate any

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



Practicing for Loops

- Write the Python code to display the following:
  - A)

```
1
2
3
4
5
```

  - B)

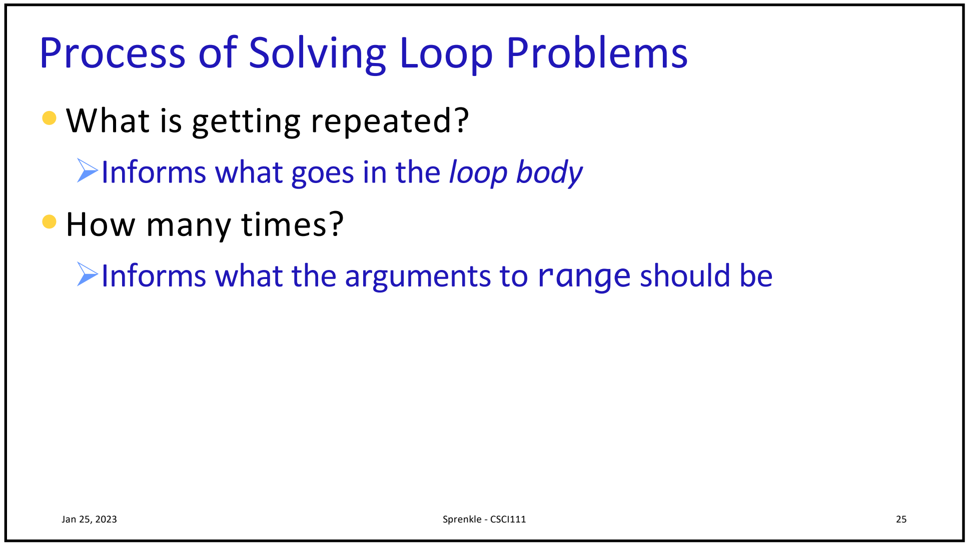
```
2

5

8

11
```

  - C)

```
****
****
****
```

Questions to ask:
- What is getting repeated?
- How many times?

How do the answers to those questions inform your solution?

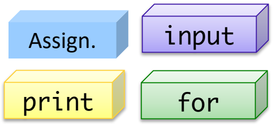


Process of Solving Loop Problems

- What is getting repeated?
  - Informs what goes in the *loop body*
- How many times?
  - Informs what the arguments to `range` should be

Programming Building Blocks

- Adding to your tool set!
- We can combine them to create more complex programs
  - ➢Solutions to problems



Programming Practice

- Add 5 numbers, inputted by the user
  - ➢We could have implemented this program yesterday, BUT we want to apply what we learned today.
- Consider what program *should* do – example behavior
- After implementing, simulate running on computer
  - ➢You can pretend to be the computer

`sum5.py`

Generalizing Solution: Accumulator Design Pattern

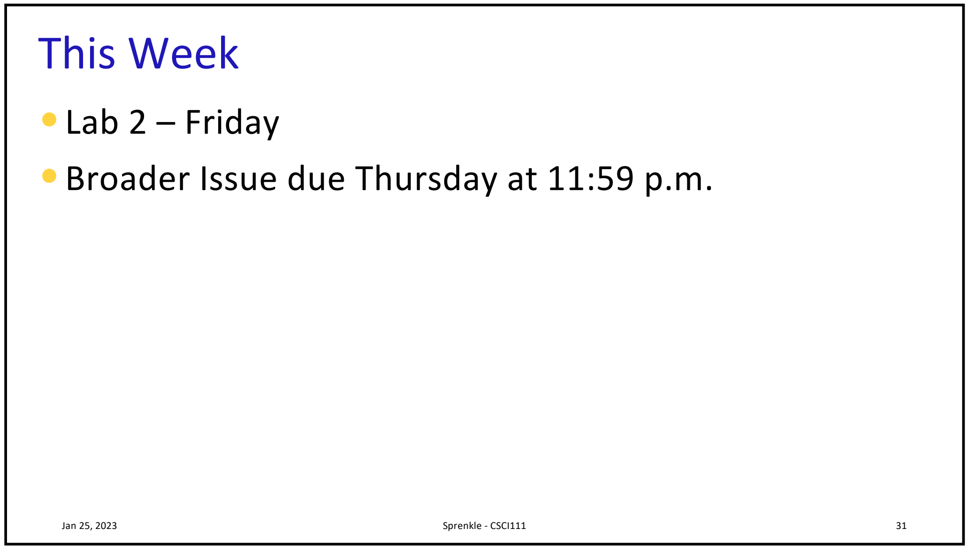
1. Initialize accumulator variable
2. Loop until done
   - Update the value of the accumulator
3. Display result

Discussion: Programming Practice

- Problem: Add 5 numbers, inputted by the user

- We could have implemented this program last week
  - 5 separate input statements, add up the numbers
- Consider how much easier this program is to change if we want a different number of numbers added up

`sum_nums.py`



This Week

- Lab 2 – Friday
- Broader Issue due Thursday at 11:59 p.m.