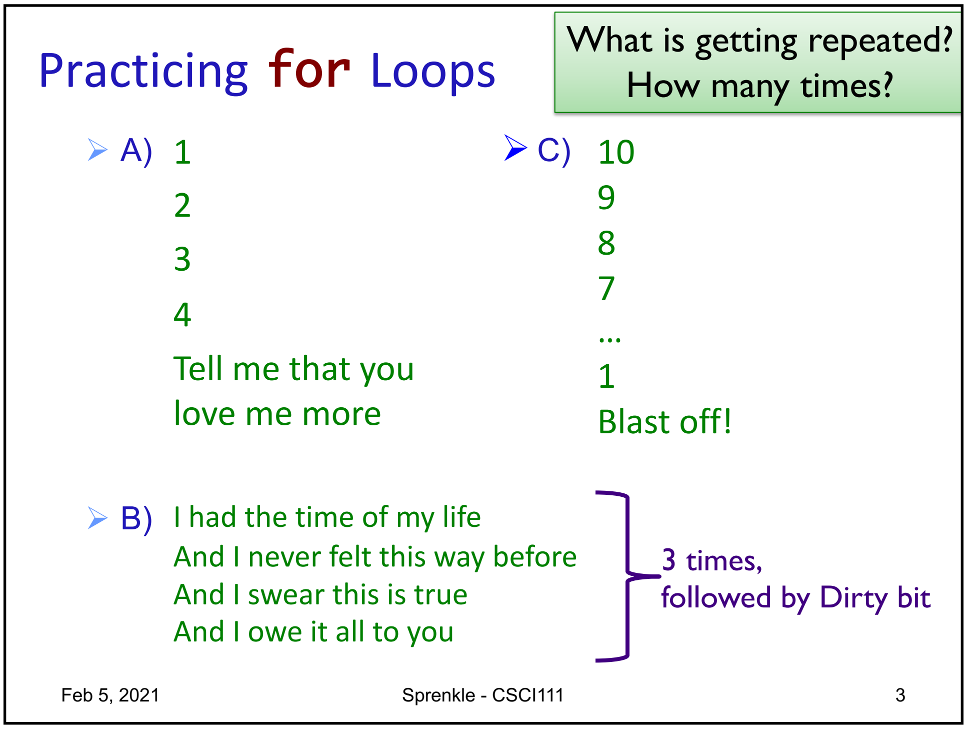
Objective

- More for loop
- Using Functions

Review

- Which lab did you submit today?
    - How many have you completed?
- What statement do we use to repeat something?
- What are the possible ways to use the `range` function?
    - What do they mean?
- When we suspect we need a loop to solve a problem, what questions should we ask?
    - How do the answers to those questions provide a process for solving loop problems?
- What is the *accumulator design pattern*?

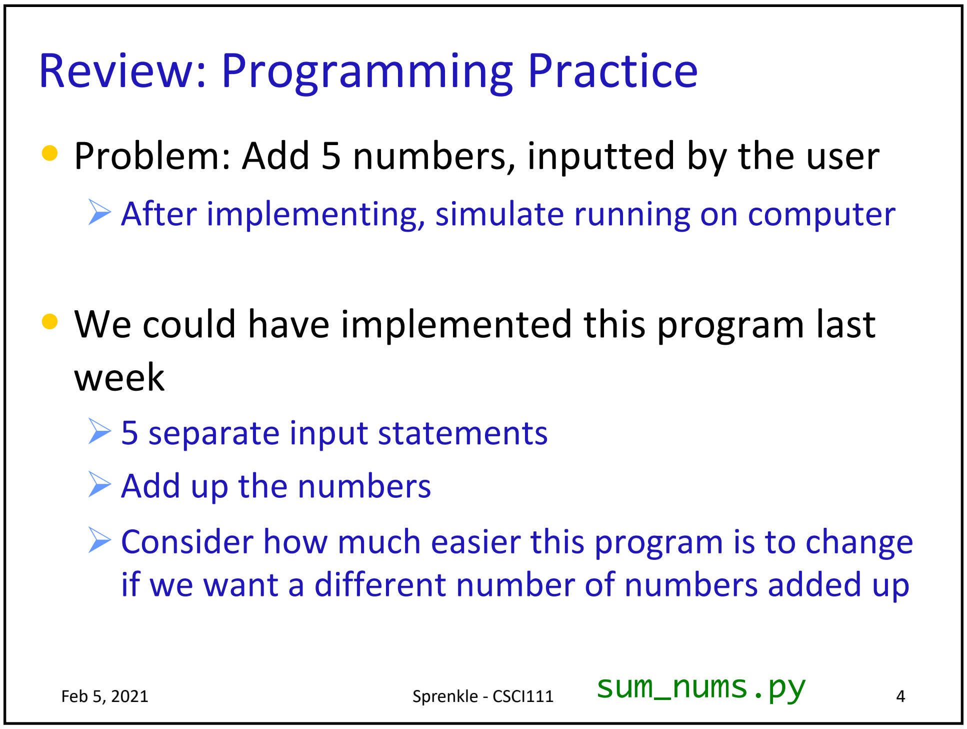


Practicing `for` Loops

What is getting repeated? How many times?

- A) 1
  2
  3
  4
  Tell me that you
  love me more
- B) I had the time of my life
  And I never felt this way before
  And I swear this is true
  And I owe it all to you

  3 times,
  followed by Dirty bit
- C) 10
  9
  8
  7
  …
  1
  Blast off!



Review: Programming Practice

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

Review: Accumulator Design Pattern

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

`sum_nums.py`

Parts of an Algorithm

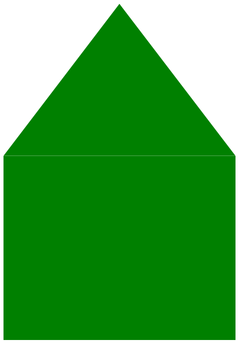
- Input, Output
- Primitive operations
  - ➢ What data you have, what you can do to the data
- Naming
  - ➢ Identify things we’re using
- Sequence of operations
- Conditionals
  - ➢ Handle special cases
- Repetition/Loops
- Subroutines
  - ➢ **Call**, reuse similar techniques

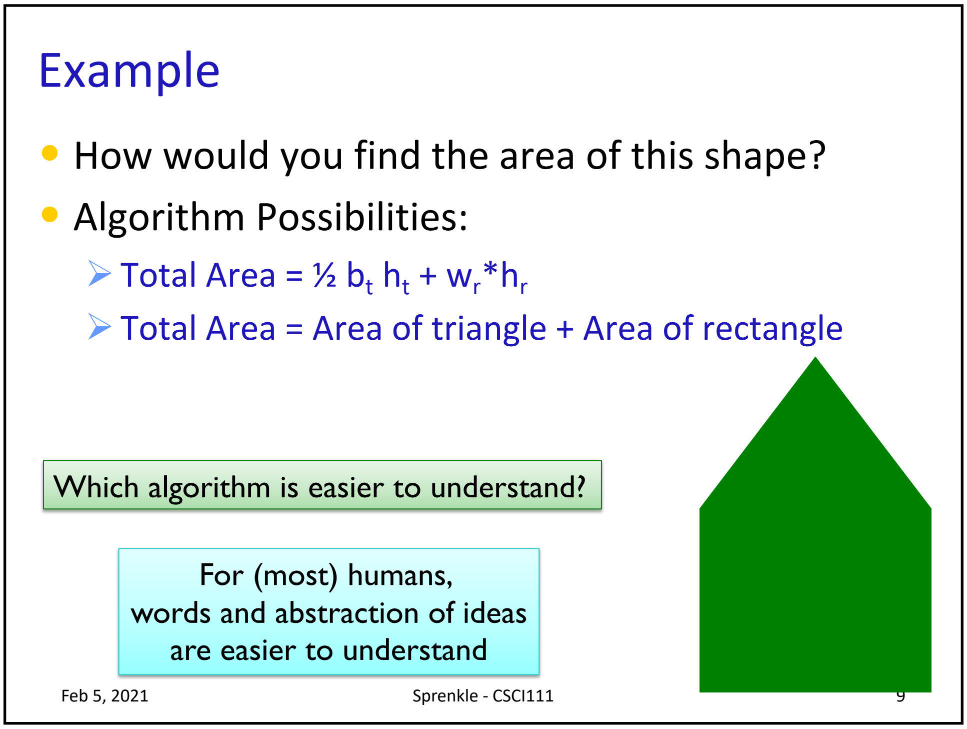
Motivating Functions

- PB&J: spreading PB, spreading jelly
    - Similar processes
    - Want to do many times
    - Rather than saying “move the knife back and forth, condiment side down, against the bread until you get X inches of …”, say “spread”
- Benefits
    - Reuse, reduce code
    - Breaks problems into more manageable pieces
    - Easier to read, write

Example

- How would you find the area of this shape?



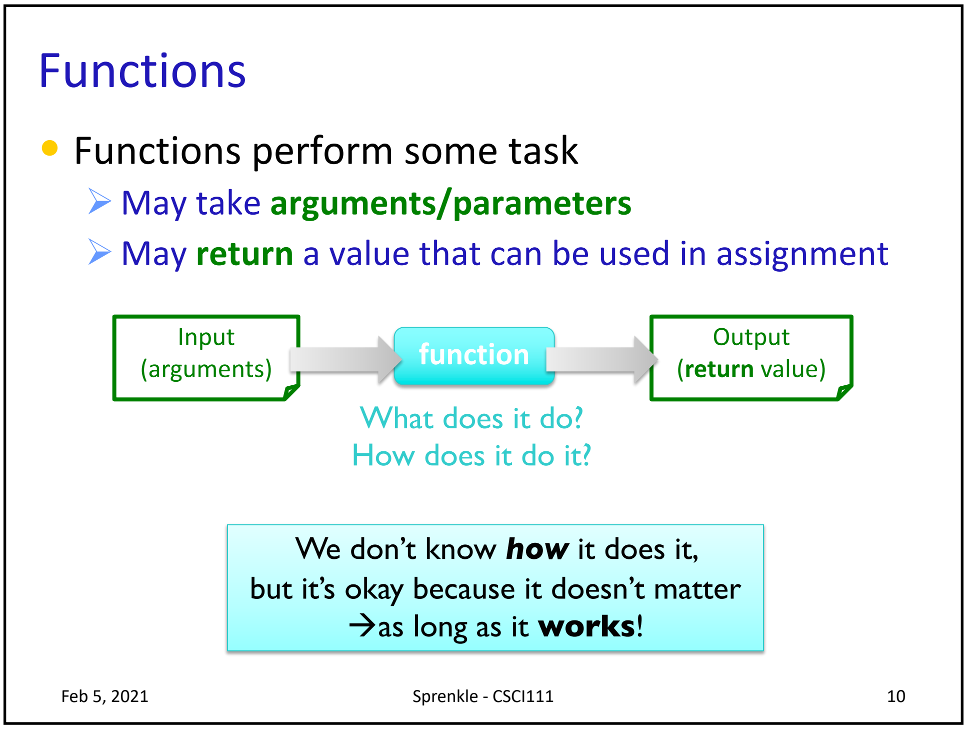


Example

- How would you find the area of this shape?
- Algorithm Possibilities:
  - Total Area = $\frac{1}{2} b_t h_t + w_r * h_r$
  - Total Area = Area of triangle + Area of rectangle

Which algorithm is easier to understand?

For (most) humans, words and abstraction of ideas are easier to understand



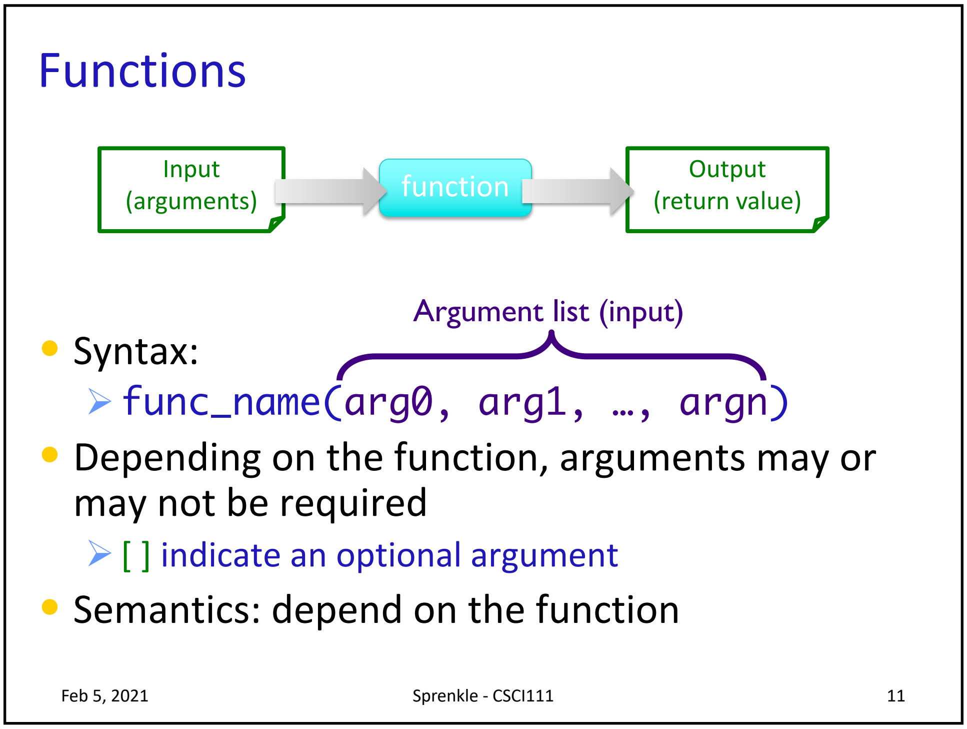
Functions

- Functions perform some task
  - May take **arguments/parameters**
  - May **return** a value that can be used in assignment

Input (arguments) → function → Output (**return** value)

What does it do?
How does it do it?

We don't know ***how*** it does it,
but it's okay because it doesn't matter
→as long as it **works**!

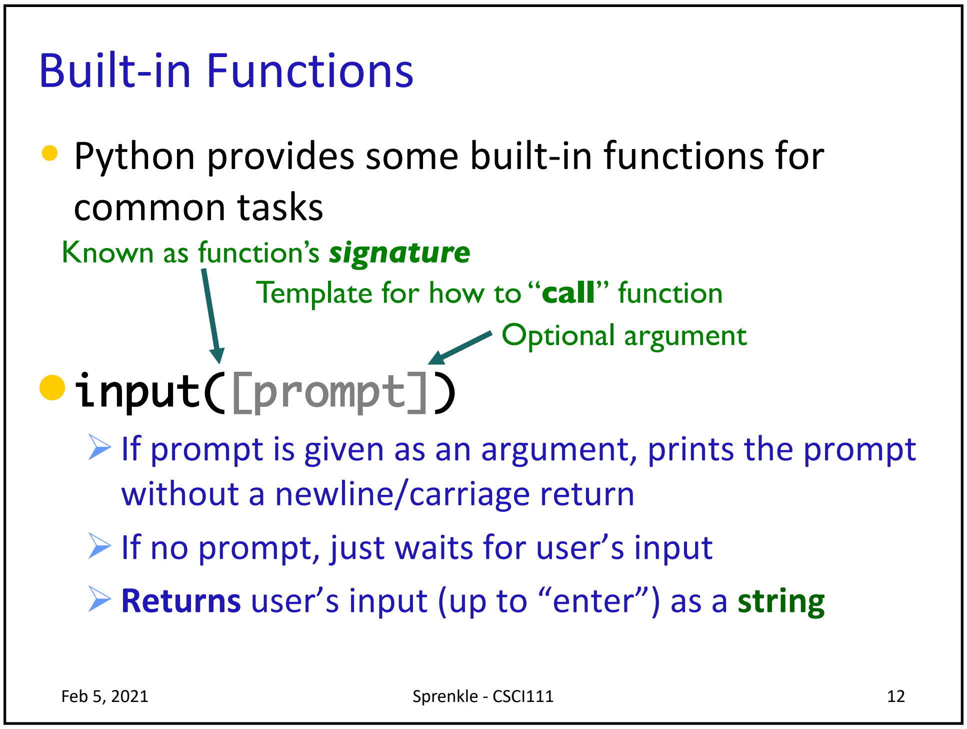


Functions

Input (arguments) → function → Output (return value)

- Syntax:
  - `func_name(arg0, arg1, …, argn)`

    Argument list (input): `arg0, arg1, …, argn`
- Depending on the function, arguments may or may not be required
  - [ ] indicate an optional argument
- Semantics: depend on the function



Built-in Functions

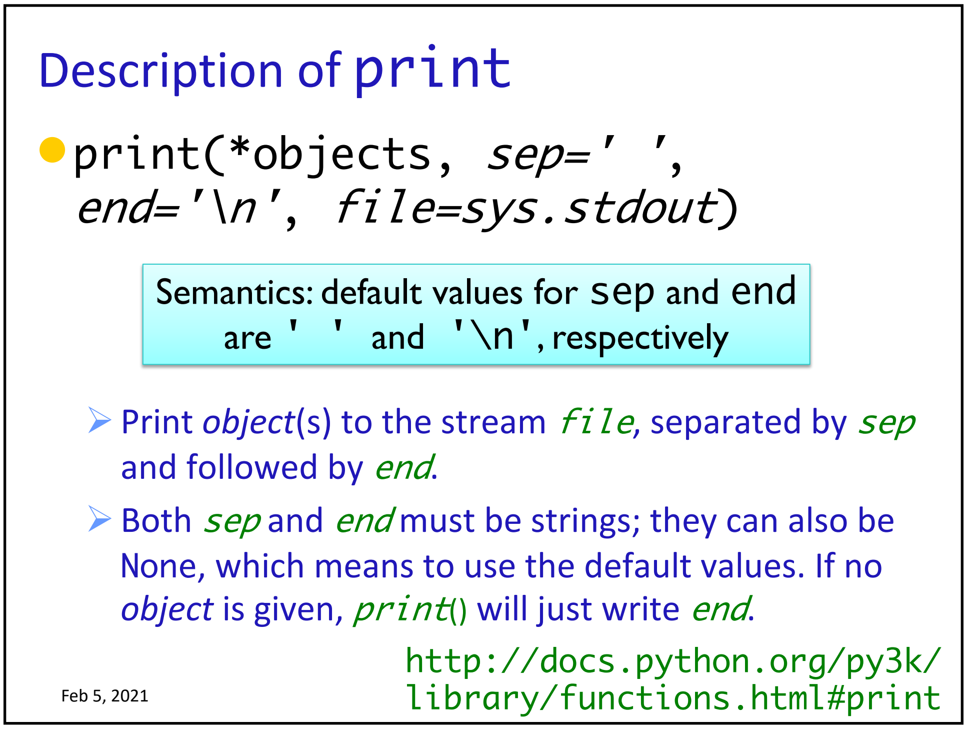
- Python provides some built-in functions for common tasks

Known as function's ***signature***

Template for how to "**call**" function

Optional argument

- `input([prompt])`
  - If prompt is given as an argument, prints the prompt without a newline/carriage return
  - If no prompt, just waits for user's input
  - **Returns** user's input (up to "enter") as a **string**



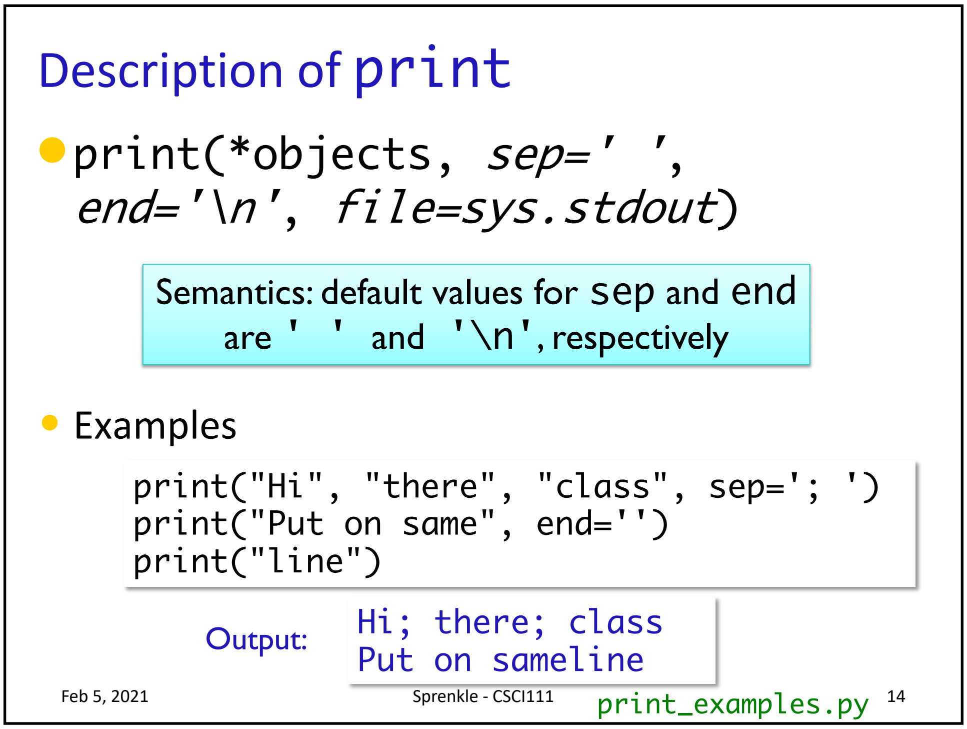
Description of `print`

- `print(*objects, sep=' ', end='\n', file=sys.stdout)`

Semantics: default values for `sep` and `end` are `' '` and `'\n'`, respectively

- Print *object*(s) to the stream *`file`*, separated by *sep* and followed by *end*.
- Both *sep* and *end* must be strings; they can also be None, which means to use the default values. If no *object* is given, *`print`*() will just write *end*.

http://docs.python.org/py3k/library/functions.html#print



Description of `print`

- `print(*objects, sep=' ', end='\n', file=sys.stdout)`

Semantics: default values for `sep` and `end` are `' '` and `'\n'`, respectively

- Examples

```
print("Hi", "there", "class", sep='; ')
print("Put on same", end='')
print("line")
```

Output:

```
Hi; there; class
Put on sameline
```

print_examples.py

More Examples of Built-in Functions

Function Signature	Description
`round(x[,n])`	*Return* the `float` **x** rounded to **n** digits after the decimal point If no **n**, round to nearest `int`
`abs(x)`	*Returns* the absolute value of **x**
`type(x)`	*Return* the type of **x**
`pow(x, y)`	*Returns* x^y

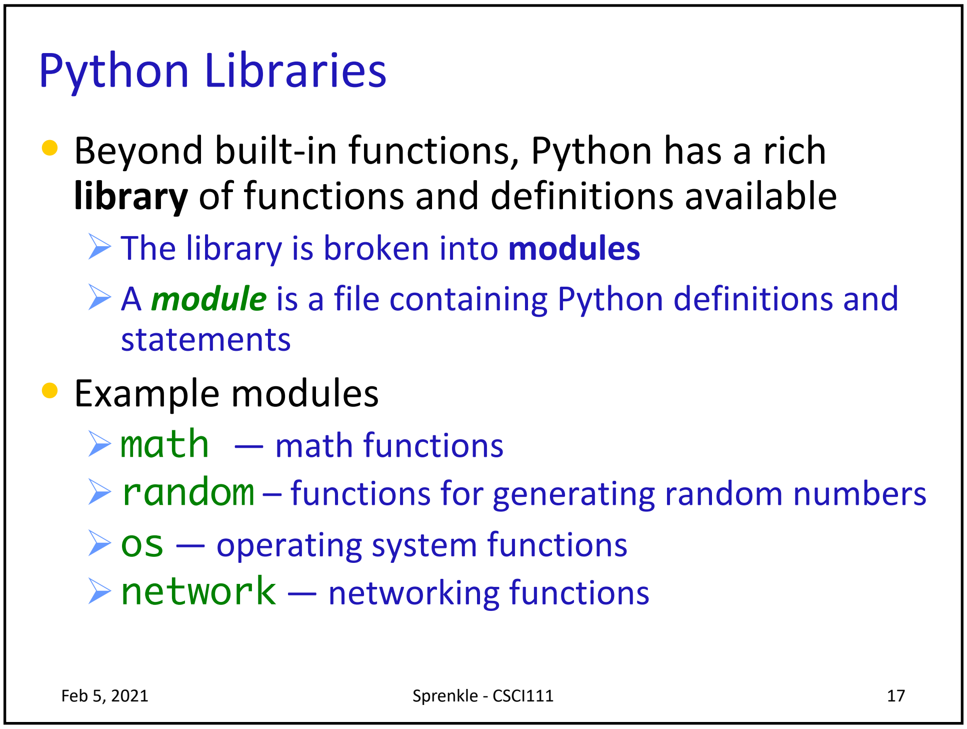
Interpreter

Using Functions

- Example use: Alternative to exponentiation
  - Objective: compute -3^2
  - Python alternatives:
    - **pow**(-3, 2)
    - (-3) ** 2
- We often use functions in assignment statements
  - Function does something
  - Save the *output* of function (i.e., what is ***returned*** in a variable)

```
roundedX = round(x)
```

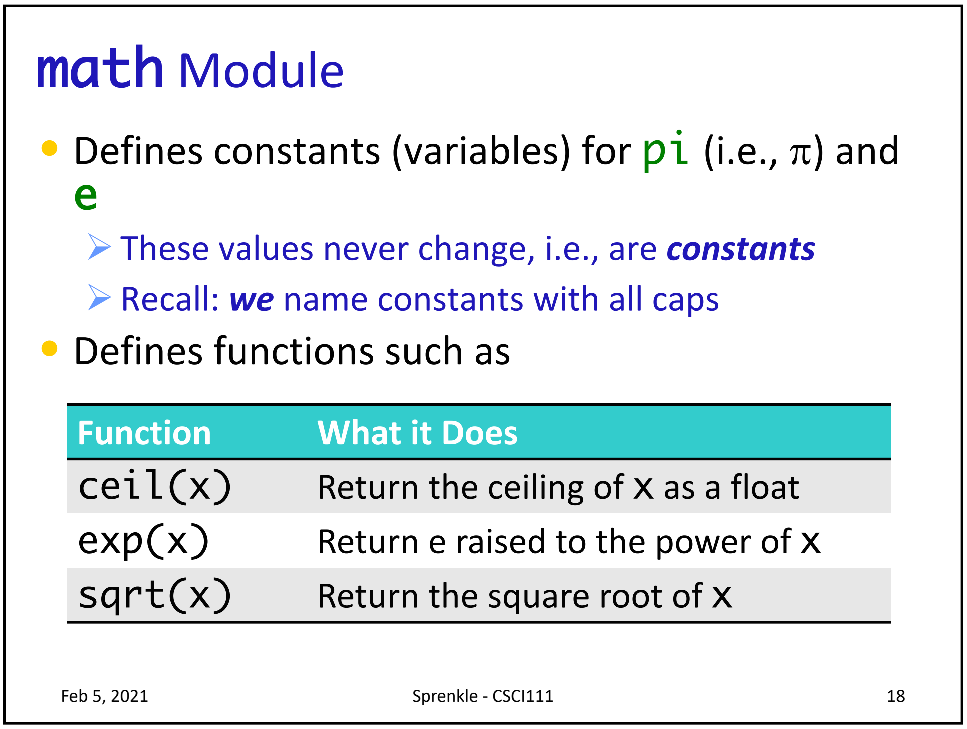
`function_example.py`



Python Libraries

- Beyond built-in functions, Python has a rich **library** of functions and definitions available
  - The library is broken into **modules**
  - A ***module*** is a file containing Python definitions and statements
- Example modules
  - `math` — math functions
  - `random` – functions for generating random numbers
  - `os` — operating system functions
  - `network` — networking functions

17

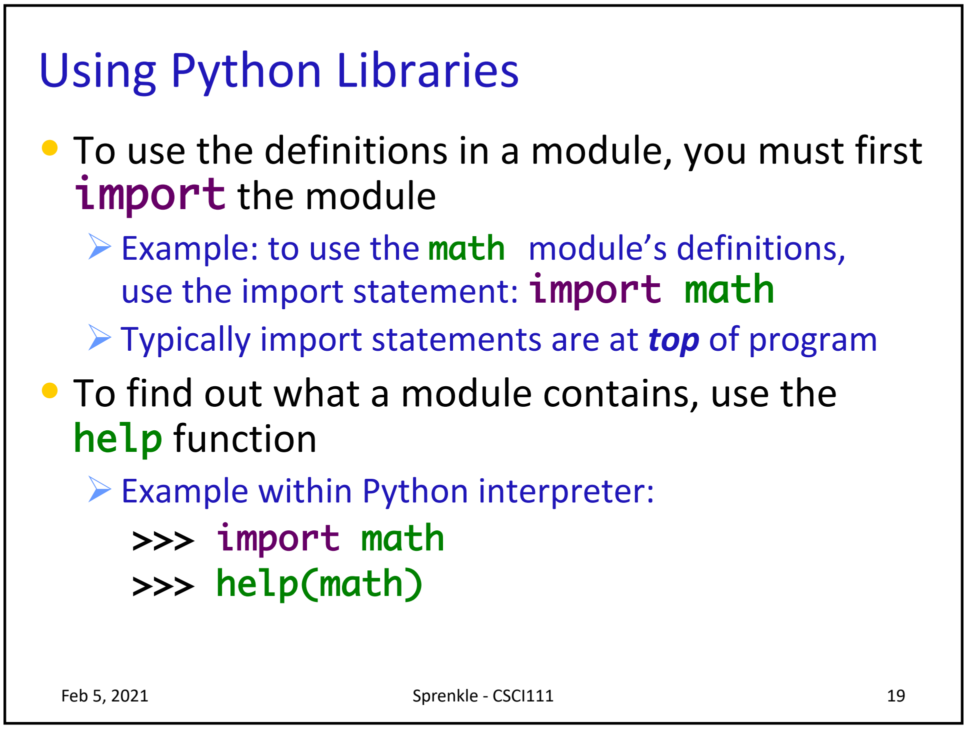


`math` Module

- Defines constants (variables) for `pi` (i.e., π) and `e`
  - These values never change, i.e., are ***constants***
  - Recall: ***we*** name constants with all caps
- Defines functions such as

Function	What it Does
`ceil(x)`	Return the ceiling of `x` as a float
`exp(x)`	Return e raised to the power of `x`
`sqrt(x)`	Return the square root of `x`

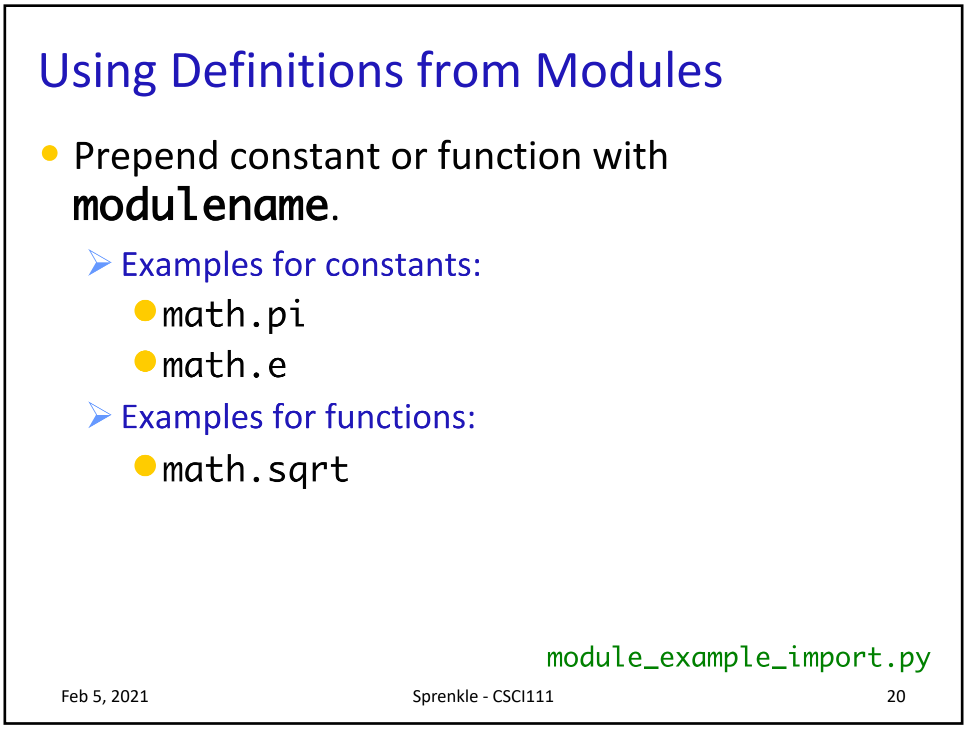
18



Using Python Libraries

- To use the definitions in a module, you must first **`import`** the module
  - Example: to use the **`math`** module's definitions, use the import statement: **`import math`**
  - Typically import statements are at ***top*** of program
- To find out what a module contains, use the **`help`** function
  - Example within Python interpreter:

```
>>> import math
>>> help(math)
```



Using Definitions from Modules

- Prepend constant or function with **`modulename.`**
  - Examples for constants:
    - `math.pi`
    - `math.e`
  - Examples for functions:
    - `math.sqrt`

`module_example_import.py`

Alternative Import Statements

```
from <module> import <defn_name>
```

- Examples:
  - `from math import pi`
    - Means “import pi from the math module”
  - `from math import *`
    - Means “import *everything* from the math module”
- With this `import` statement, don’t need to prepend module name before using functions
  - Example: `e**(1j*pi) + 1`

`module_example_from_import.py`

Benefits of Using Python Libraries/Modules

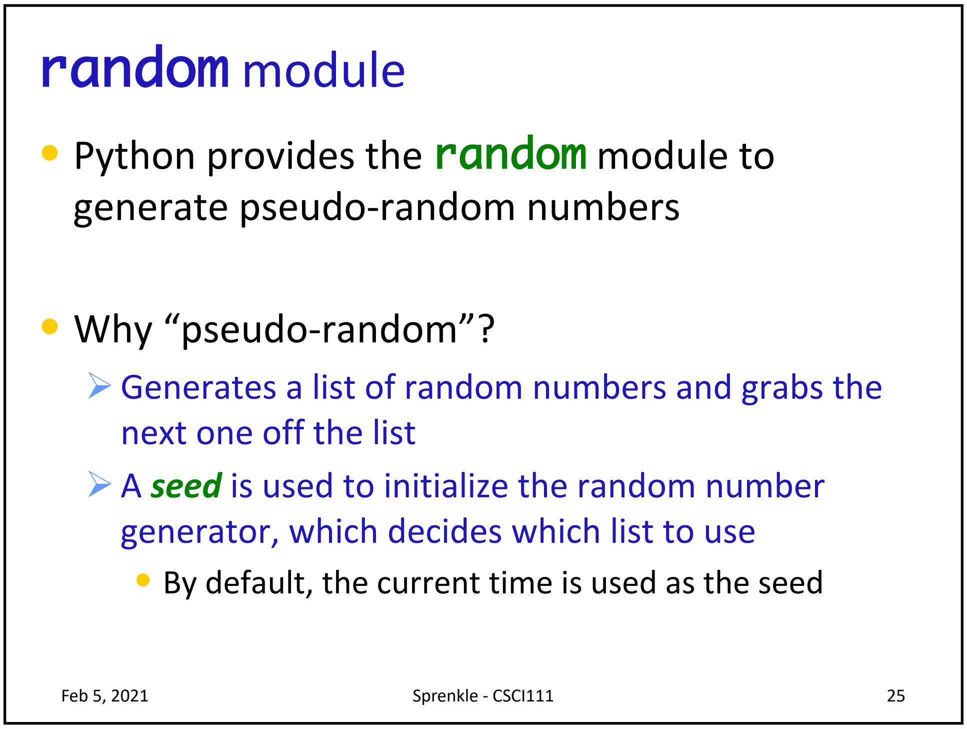
- Don’t need to rewrite code
- If it’s in a module, it is very *efficient* (in terms of computation speed and memory usage)

Finding Modules To Use

- How do I know if functionality that I want already exists?
  - Python Library Reference: `http://docs.python.org/py3k/library/`

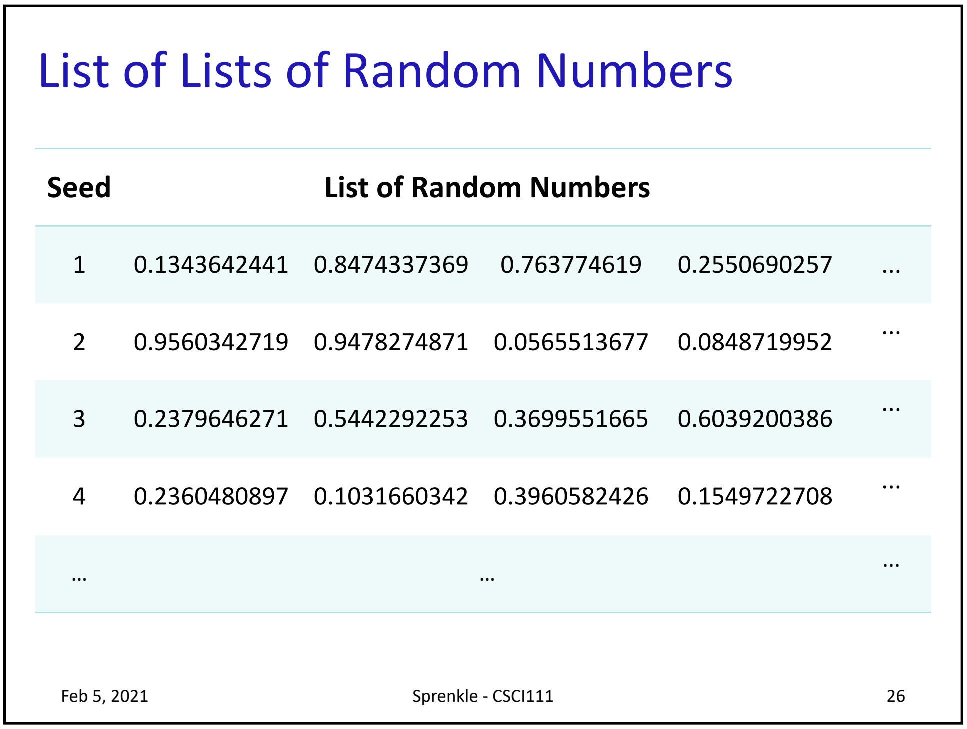
- For the most part, in the beginning you will write most of your code from scratch

RANDOM MODULE



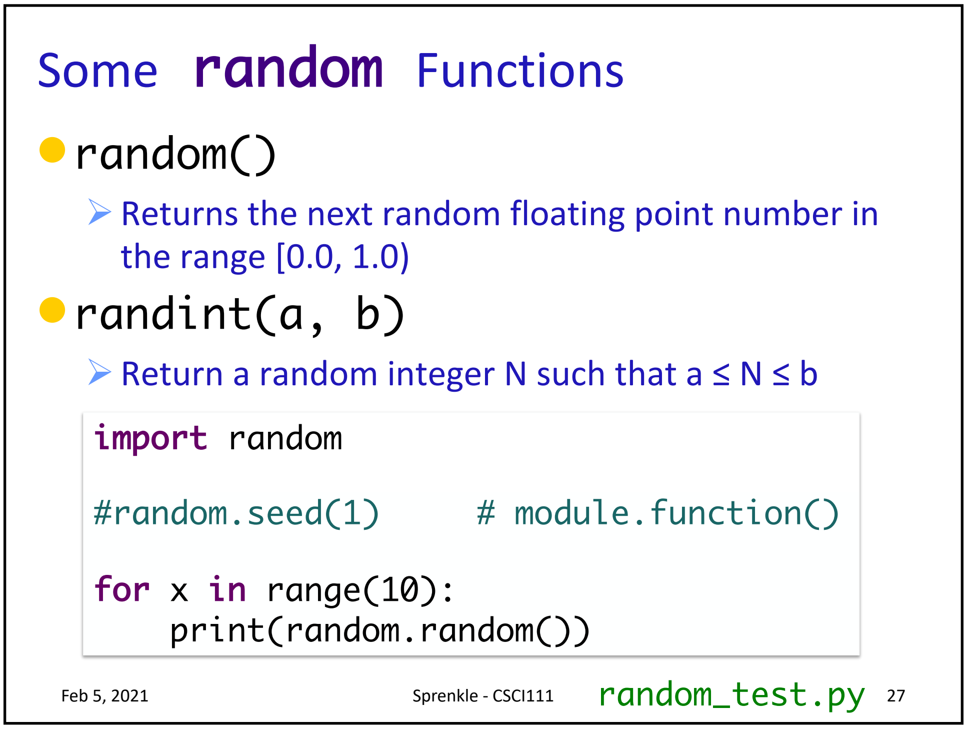
random module

- Python provides the random module to generate pseudo-random numbers
- Why “pseudo-random”?
  - Generates a list of random numbers and grabs the next one off the list
  - A ***seed*** is used to initialize the random number generator, which decides which list to use
    - By default, the current time is used as the seed



List of Lists of Random Numbers

Seed	List of Random Numbers				
1	0.1343642441	0.8474337369	0.763774619	0.2550690257	…
2	0.9560342719	0.9478274871	0.0565513677	0.0848719952	…
3	0.2379646271	0.5442292253	0.3699551665	0.6039200386	…
4	0.2360480897	0.1031660342	0.3960582426	0.1549722708	…
…	…				…



Some `random` Functions

- random()
  - Returns the next random floating point number in the range [0.0, 1.0)
- randint(a, b)
  - Return a random integer N such that a ≤ N ≤ b

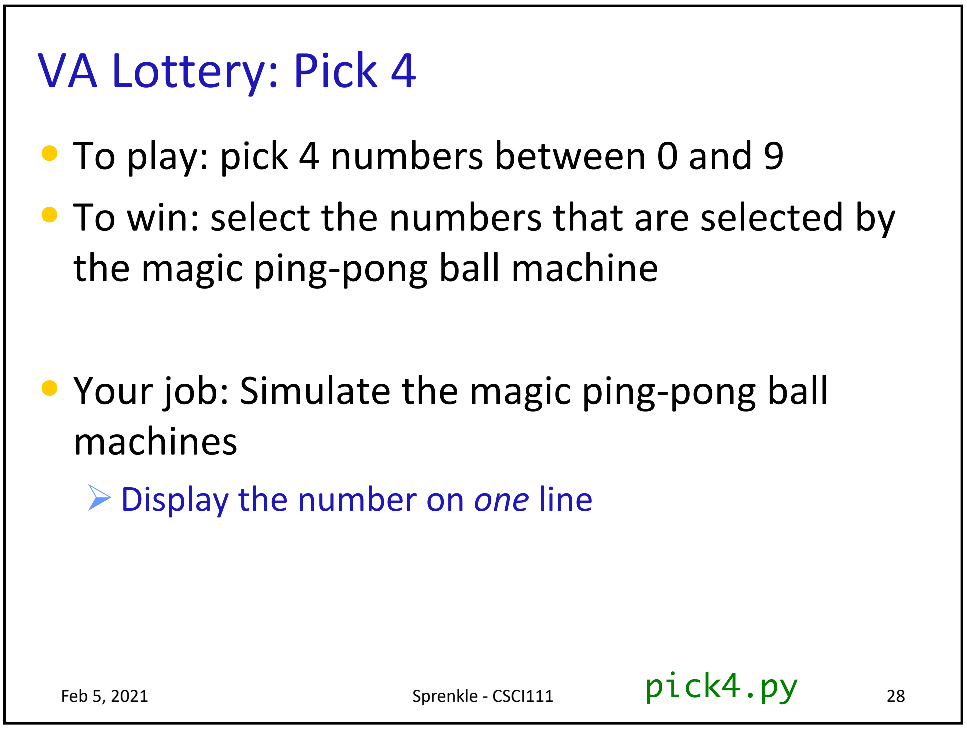
```
import random

#random.seed(1)      # module.function()

for x in range(10):
    print(random.random())
```

Feb 5, 2021 Sprenkle - CSCI111 random_test.py 27

27



VA Lottery: Pick 4

- To play: pick 4 numbers between 0 and 9
- To win: select the numbers that are selected by the magic ping-pong ball machine

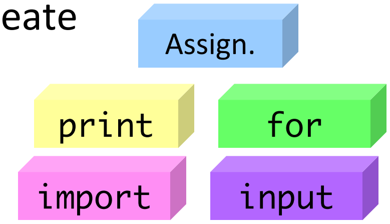
- Your job: Simulate the magic ping-pong ball machines
  - Display the number on *one* line

Feb 5, 2021 Sprenkle - CSCI111 pick4.py 28

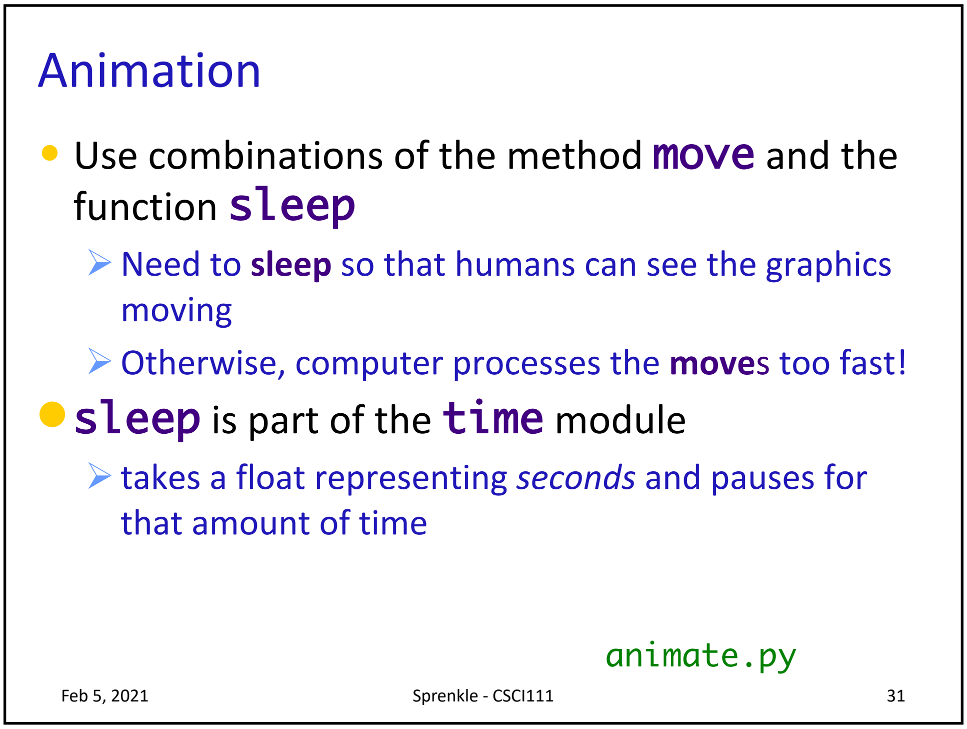
28

Programming Building Blocks

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



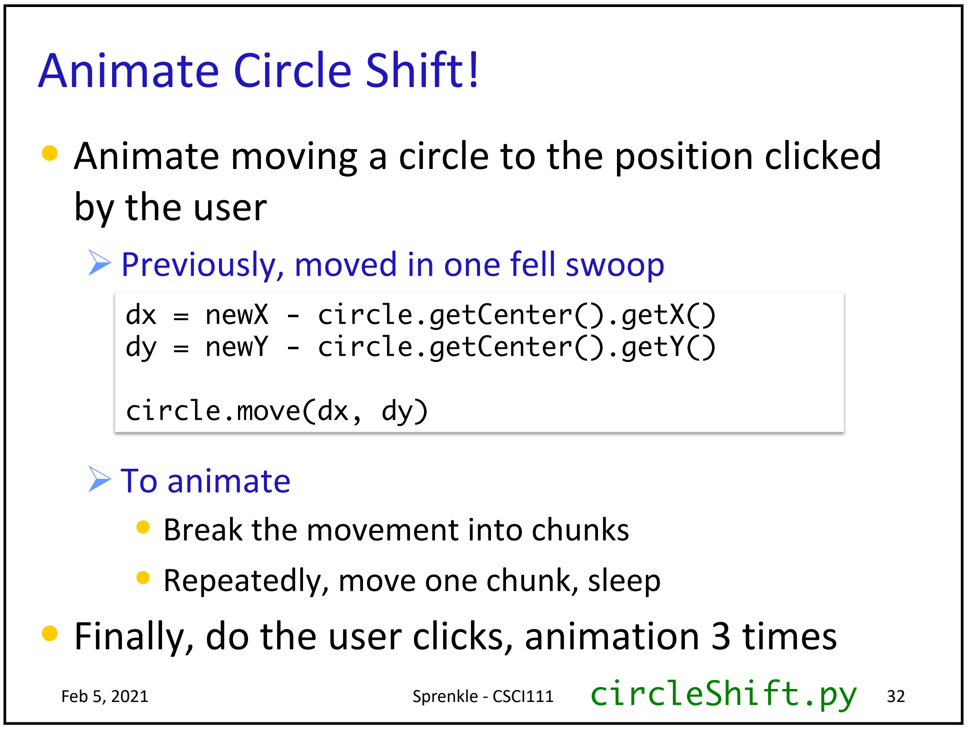
ANIMATION



Animation

- Use combinations of the method **`move`** and the function **`sleep`**
  - Need to **sleep** so that humans can see the graphics moving
  - Otherwise, computer processes the **move**s too fast!
- **`sleep`** is part of the **`time`** module
  - takes a float representing *seconds* and pauses for that amount of time

`animate.py`



Animate Circle Shift!

- Animate moving a circle to the position clicked by the user
  - Previously, moved in one fell swoop

```
dx = newX - circle.getCenter().getX()
dy = newY - circle.getCenter().getY()

circle.move(dx, dy)
```

  - To animate
    - Break the movement into chunks
    - Repeatedly, move one chunk, sleep
- Finally, do the user clicks, animation 3 times

`circleShift.py`

Looking Ahead

- Pre Lab 3, Lab 3 next week