

Objective

- More `for` loop
- Using Functions

1

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 inform our solution to a loop problem?

2

Practicing for Loops

What is getting repeated?
How many times?

- A) 1
2
3
4

Tell me that you
love me more

- C) 10
9
8
7
...
1
Blast off!

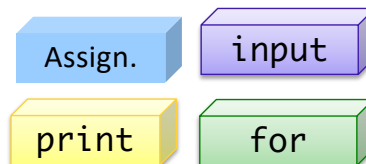
- B) I had the time of my life
And I never felt this way before
And I swear this is true
And I owe it all to you

} 3 times,
followed by Dirty bit

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
- After implementing, simulate running on computer
 - You can pretend to be the computer

Jan 28, 2022

Sprenkle - CSCI111

`sum5.py`

5

5

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

Jan 28, 2022

Sprenkle - CSCI111

`sum_nums.py`

6

6

Generalizing Solution: Accumulator Design Pattern

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

Jan 28, 2022

Sprenkle - CSCI111

7

7

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



Jan 28, 2022

Sprenkle - CSCI111

8

8

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

Jan 28, 2022

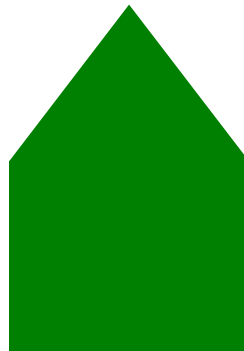
Sprenkle - CSCI111

9

9

Example

- How would you find the area of this shape?



Jan 28, 2022

Sprenkle - CSCI111

10

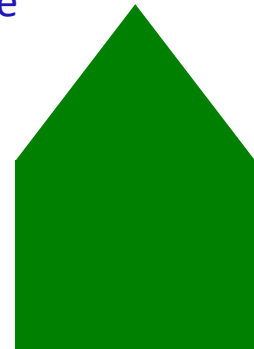
10

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



Jan 28, 2022

Sprenkle - CSCI111

11

Functions

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



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!**

Jan 28, 2022

Sprenkle - CSCI111

12

12

Functions



- Syntax:

➤ `func_name(arg0, arg1, ..., argn)`

Argument list (input)

- Depending on the function, arguments may or may not be required

➤ `[]` indicate an optional argument

- Semantics: depend on the function

Jan 28, 2022

Sprenkle - CSCI111

13

13

Built-in Functions

- Python provides some built-in functions for common tasks

Known as function's **signature**; a template for how to **call** function

● `input([prompt])`

Optional argument

➤ 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**

Jan 28, 2022

Sprenkle - CSCI111

14

14

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>

Jan 28, 2022

Sprengle - CSCI111

15

15

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`

Jan 28, 2022

Sprengle - CSCI111

print_examples.py

16

16

More Examples of Built-in Functions

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

Interpreter

Jan 28, 2022

Sprenkle - CSCI111

17

17

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)
```

Jan 28, 2022

Sprenkle - CSCI111 `function_example.py`

18

18

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

Jan 28, 2022

Sprenkle - CSCI111

19

19

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
<code>ceil(x)</code>	Return the ceiling of X as a float
<code>exp(x)</code>	Return e raised to the power of X
<code>sqrt(x)</code>	Return the square root of X

Jan 28, 2022

Sprenkle - CSCI111

20

20

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)
```

Jan 28, 2022

Sprenkle - CSCI111

21

21

Using Definitions from Modules

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

`module_example_import.py`

Jan 28, 2022

Sprenkle - CSCI111

22

22

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`

Jan 28, 2022

Sprenkle - CSCI111

23

23

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)

Jan 28, 2022

Sprenkle - CSCI111

24

24

Finding Modules To Use

- How do I know if functionality that I want already exists?
 - Python Library Reference:
<http://docs.python.org/3/library/>
- In the beginning, you may rewrite existing functionality to help you learn how it works

Jan 28, 2022

Sprenkle - CSCI111

25

25

RANDOM MODULE

Jan 28, 2022

Sprenkle - CSCI111

26

26

random module

- Python provides the **random** module to generate pseudo-random numbers
- What is “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	...
...

random module

- Why “pseudo-random”?
 - No cost-effective source of real randomness
 - Code usually doesn't *need* to be truly random
 - Can replicate the code that depends on randomness by using the seed, when appropriate

Jan 28, 2022

Sprenkle - CSCI111

29

29

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 $a \leq N \leq b$

```
import random

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

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

`random_test.py`

Jan 28, 2022

30

30

Looking Ahead

- Pre Lab 3, Lab 3 next week