

Objectives

- Defining your own functions
 - Control flow
 - Scope, variable lifetime
 - Testing

Feb 10, 2021

Sprenkle - CSCI111

1

1

Review

- What are benefits of functions?
- How do we create our own functions?
 - How do we indicate that our function requires input?
 - How do we indicate that our function has output?
- What's the difference between output from a function and output from a program?
- How do we call a function we created?

Feb 10, 2021

Sprenkle - CSCI111

2

2

Function Definition Example without Output

Keyword **Function Name** **Input Name/Parameter**

```
def moveCircle( circle, newCenter ): Function header
    """
    Moves a Circle object to a new location.
    circle: the Circle to be moved
    newCenter: the center point of where circle
    should be moved
    """ Function documentation
    centerPoint = circle.getCenter()

    diffInX = newCenter.getX() - centerPoint.getX()
    diffInY = newCenter.getY() - centerPoint.getY()

    circle.move(diffInX, diffInY)
```

Body (or function definition)

Feb 10, 2021

Sprenkle - CSCI111

3

3

Function Definition Example with Output

Keyword **Function Name** **Input Name/Parameter**

```
def average2(num1, num2): Function header
    """
    Parameters: two numbers to be averaged.
    Returns the average of two numbers
    """ Function documentation
    average = (num1 + num2)/2
    return average
```

Body (or function definition)

Keyword: How to give output **Output**

Feb 10, 2021

Sprenkle - CSCI111

4

4

Review: `return` vs `print`

- In general, whenever we want output from a function, we'll use `return`
 - Results in a more flexible, reusable function
 - Let whoever called the function figure out what to display
- Use `print` for
 - Debugging your function (then remove)
 - Otherwise, unintended side effect of calling the function
 - When you have a function that is supposed to display something
 - Sometimes, that is what you want.

Feb 10, 2021

Sprenkle - CSCI111

6

6

Function Input and Output

- What does this function do?
- What is its input? What is its output?

```
def printVerse(animal, sound):  
    print(BEGIN_END + EIEIO)  
    print("And on that farm he had a", animal, EIEIO)  
    print("With a", sound, ",", sound, "here")  
    print("And a", sound, ",", sound, "there")  
    print("Here a", sound)  
    print("There a", sound)  
    print("Everywhere a", sound, ",", sound)  
    print(BEGIN_END + EIEIO)  
    print()
```

Constants and comments
are in example program

What does this function do if called as `printVerse("pig", "oink")`?
As `printVerse("oink", "pig")`?

7

Function Input and Output

- 2 inputs: **animal** and **sound**
- 0 outputs
 - *Displays* something but does not **return** anything (None)

```
def printVerse(animal, sound):  
    print(BEGIN_END + EIEIO)  
    print("And on that farm he had a", animal, EIEIO)  
    print("With a", sound, ", ", sound, "here")  
    print("And a", sound, ", ", sound, "there")  
    print("Here a", sound)  
    print("There a", sound)  
    print("Everywhere a", sound, ", ", sound)  
    print(BEGIN_END + EIEIO)  
    print() ← Function exits here
```

Feb 10, 2021

Sprenkle - CSCI111

8

8

Arithmetic Example

- Our favorite expression: $i^2 + 3j - 5$

1. Define the function:
 - a. What does the function do?
 - b. What is its input?
 - c. What is its output?
2. Call the function

our_favorite_expression.py

Feb 10, 2021

Sprenkle - CSCI111

9

9

PROGRAM ORGANIZATION

Feb 10, 2021

Sprenkle - CSCI111

10

10

Where are Functions Defined?

- Functions can go inside program script
 - If no `main()` function, defined *before* use/called
 - `average2.py`
 - If `main()` function, defined anywhere in script
- Functions can go inside a separate *module*

Feb 10, 2021

Sprenkle - CSCI111

11

11

Program Organization: `main` function

- In many programming languages, you put the “driver” for your program in a `main` function
 - You can (and should) do this in Python as well
- Typically `main` functions are defined at the top of your program
 - Readers can quickly see an overview of what program does
- `main` usually takes no arguments
 - Example: `def main():`

Feb 10, 2021

Sprenkle - CSCI111

12

12

Using a `main` Function

- Call `main()` at the bottom of your program
- Side effects:
 - Do not need to define functions before `main` function
 - `main` can “see” all other functions
- Note: `main` is a function that calls other functions
 - Any function can call other functions

Feb 10, 2021

Sprenkle - CSCI111

13

13

Example program with a main() function

```
def main():
    printVerse("dog", "ruff")
    printVerse("duck", "quack")

    animal_type = "cow"
    animal_sound = "moo"
    printVerse(animal_type, animal_sound)

def printVerse(animal, sound):
    print(BEGIN_END, EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END, EIEIO)
    print()

main()
```

Constants and comments
are in example program

In what order does this program execute?
What is output from this program?

oldmac.py

14

Example program with a main() function

```
def main():
    printVerse("dog", "ruff")
    printVerse("duck", "quack")

    animal_type = "cow"
    animal_sound = "moo"
    printVerse(animal_type, animal_sound)
    ...

def printVerse(animal, sound):
    print(BEGIN_END, EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END, EIEIO)
    print()

main()
```

1. Define (store) main
2. Define (store) printVerse
3. Call main function
4. Execute main function
5. Call, execute printVerse

oldmac.py

15

Summary: Program Organization

- Larger programs require **functions** to maintain readability
 - Use **main()** and other functions to break up program into *smaller, more manageable* chunks
 - “**Abstract** away” the details
- As before, can still write smaller scripts without any functions
 - Can try out functions using smaller scripts
- Need the **main()** function when using other functions to keep “driver” at top
 - Otherwise, functions need to be defined **before** use

Feb 10, 2021

Sprenkle - CSCI111

16

16

VARIABLE LIFETIMES AND SCOPE

Feb 10, 2021

Sprenkle - CSCI111

17

17

What does this program output?

```
def main():
    x = 10
    sum = sumEvens( x )
    print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit):
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total

main()
```

Feb 10, 2021

Sprenkle - CSCI111

mystery.py

18

18

Function Variables

```
def main():
    x = 10
    sum = sumEvens( x )
    print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit):
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total

main()
```

Why can we name two different variables X?

Feb 10, 2021

Sprenkle - CSCI111

mystery.py

19

19

Tracing through Execution

```
def main():  
    x = 10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)  
  
def sumEvens(limit):  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

When you call `main()`, that means you want to execute this function

Defines functions

main()

Feb 10, 2021

Sprenkle - CSCI111

20

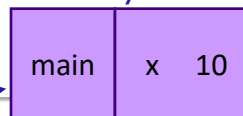
20

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)  
  
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

main()

Memory stack



Variable names are like first names

Function names are like last names

Define the **SCOPE** of the variable

Feb 10, 2021

Sprenkle - CSCI111

21

21

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)
```

```
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

Called the function **sumEvens**
Add its parameters to the stack

main()

sum Evens	limit 10
main	x 10

Feb 10, 2021

Sprenkle - CSCI111

22

22

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)
```

```
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

main()

sum Evens	total 0 limit 10
main	x 10

Feb 10, 2021

Sprenkle - CSCI111

23

23

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)  
  
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total  
  
main()
```

sum	x 0
Evens	total 0
	limit 10
main	x 10

Feb 10, 2021

Sprenkle - CSCI111

24

24

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)  
  
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total  
  
main()
```

sum	x 8
Evens	total 20
	limit 10
main	x 10

Feb 10, 2021

Sprenkle - CSCI111

25

25

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)
```

```
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

- Function `sumEvens` returned
- no longer have to keep track of its variables on stack
 - lifetime of those variables is over

main()

main	sum 20 x 10
------	----------------

Feb 10, 2021

Sprenkle - CSCI111

26

26

Function Variables

```
def main() :  
    x=10  
    sum = sumEvens( x )  
    print("The sum of even #s up to", x, "is", sum)
```

```
def sumEvens(limit) :  
    total = 0  
    for x in range(0, limit, 2):  
        total += x  
    return total
```

main()

main	sum 20 x 10
------	----------------


Feb 10, 2021

Sprenkle - CSCI111

27

27

Variable Scope

- Functions can have the same parameter and variable names as other functions
 - Need to look at the variable's **scope** to determine which one you're looking at
 - Use the **stack** to figure out which variable you're using
- Scope levels
 - **Local scope** (also called **function scope**) 
 - Can only be seen within the function
 - **Global scope** (also called **file scope**)
 - Whole program can access
 - More on these later

Feb 10, 2021

Sprenkle - CSCI111

28

28

Variable Scope

- Know "lifetime" of variable
 - Only during execution of function
 - Related to idea of "scope"
- Consider: how many functions probably use a variable like `x` or `i`? What would the impact be on our programs if all variables had global scope?
 - Example: `round(x, n)`
- In general, our only *global* variables will be constants because we don't want them to change value
 - e.g., `EIEIO`

Feb 10, 2021

Sprenkle - CSCI111

29

29

TESTING FUNCTIONS

Feb 10, 2021

Sprenkle - CSCI111

30

30

Testing Functions

- Functions make it easier for us to test our code
- We can write code to test the functions
 - Test Case:
 - Input: parameters
 - Expected Output: what we expect to be returned
 - Or if state changed as we expected
 - We can verify the function programmatically
 - “programmatically” – automatically execute test cases and verify that the actual returned result is what we expected
 - No user input required!

Feb 10, 2021

Sprenkle - CSCI111

31

31

test Module

- Not a standard module
 - Included with our textbook
 - More sophisticated testing modules but this is sufficient for us
- Function:
 - `testEqual(actual, expected)`
 - Parameters: actual and expected results for a function.
 - Displays "Pass" and returns True if the test case passes.
 - Displays error message, with expected and actual results, and returns False if test case fails.

Feb 10, 2021

Sprenkle - CSCI111

32

32

Example: Testing sumEvens

```
import test
...
def testSumEvens():
    actual = sumEvens( 10 )
    expected = 20
    test.testEqual( actual, expected )

def sumEvens(limit):
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total
```

What are other good test cases?

`testSumEvens.py`

Feb 10, 2021

Sprenkle - CSCI111

33

33

Summary: Why Write Functions?

- Allows you to break up a problem into *smaller*, more *manageable* parts
- Makes your code easier to *understand*
- Hides implementation details (*abstraction*)
 - Provides interface (input, output)
- Makes part of the code *reusable* so that you:
 - Only have to write function code once
 - Can debug it all at once
 - Isolates errors
 - Can make changes in one function (*maintainability*)

Similar to benefits of OO Programming

Feb 10, 2021

Sprenkle - CSCI111

34

34

Evolving General Design Patterns

- Former general design pattern:
 1. Optionally, get user input
 2. Do some computation
 3. Display results
- Now general design pattern:
 1. Optionally, get user input
 2. Do some computation by calling **functions**, get results
 3. Display results

Feb 10, 2021

Sprenkle - CSCI111

35

35

Looking Ahead

- BI – Google Search
- Lab 3 due Friday
- Exam next Friday
 - Prep document up soon