

Objectives

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

Review

- What are benefits of functions?
- How do we add animation to our graphics programs?
- How do we create our own functions?
 - How do we indicate that our function requires input?
 - How do we say that our function has output?
- How do we call a function we created?

Function Definition Example without Output

Keyword → **def** *Function Name* → **moveCircle** *Input Name/Parameter* → **circle, newCenter** *Function header*

```
def moveCircle( circle, newCenter ):
    """
    Move the given Circle circle to be centered
    at the Point newCenter
    """
    centerPoint = circle.getCenter()

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

    circle.move(diffInX, diffInY)
```

Body (or function definition)

Jan 30, 2019

Sprenkle - CSCI111

3

Function Definition with Output

Keyword → **def** *Function Name* → **average2** *Input Name/Parameter* → **num1, num2** *Function header*

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

Body (or function definition)

Keyword: How to give output → **return** *Output* → **average**

Jan 30, 2019

Sprenkle - CSCI111

4

Functions: Similarity to Math

- In math, a function definition looks like:

$$f(x) = x^2 + 2$$

- Plug values in for x
- Example:
 - $f(3) = 3^2 + 2 = 11$
 - 3 is your *input*, assigned to x
 - 11 is output

Jan 30, 2019

Sprenkle - CSCI111

5

Function Output

- When the code reaches a statement like
return x
 - The function stops executing
 - x is the **output returned** to the place where the function was called
- For functions that don't have explicit output, **return** does not have a value with it, e.g.,
return
- Optional: don't *need* to have **return**
 - Function *automatically* returns at the end

Jan 30, 2019

Sprenkle - CSCI111

7

Flow of Control

- When program calls a function, the program jumps to the function and executes it
- After executing the function, the program returns to the same place in the *calling code* where it left off

Calling code:

```
# Make conversions
dist1 = 100
miles1 = metersToMiles(dist1)
```

Value of `dist1` (100) is assigned to `meters`

```
def metersToMiles(meters) :
    M2MI=.0006215
    miles = meters * M2MI
    return miles
```

Jan 30, 2019

Sprenkle - CSCI111

`average2.py`

8

Using print vs return

- `print` is for displaying information
- Don't always want to display the output of a function
- `return` gives us more flexibility about what we do with the output from a function
- Example:

```
avg = average2(num1, num2)
print("The average is", round(avg, 2) )
```

We don't want the "raw" value from `average2` displayed when the function is called.

We want to process that value so that we only display it to two decimal places (and another place we call it, we want to round to 4 decimal places).

Jan 30

return vs print

- In general, whenever we want output from a function, we'll use **return**
 - 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.

Jan 30, 2019

Sprenkle - CSCI111

10

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")`?

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

Jan 30, 2019

Sprenkle - CSCI111

12

Words in Different Contexts

"Time flies like an arrow.
Fruit flies like bananas."
— Groucho Marx.

- Output from a **function**
 - What is **returned** from the function
 - If the function prints something, it's what the function **displays** (rather than outputs).
- Output from a **program**
 - What is displayed by the program

Jan 30, 2019

Sprenkle - CSCI111

13

PROGRAM ORGANIZATION

Jan 30, 2019

Sprenkle - CSCI111

14

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*

Jan 30, 2019

Sprenkle - CSCI111

15

Program Organization: **main** function

- In many 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():`

Jan 30, 2019

Sprenkle - CSCI111

16

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

Jan 30, 2019

Sprenkle - CSCI111

17

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

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

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

Jan 30, 2019

Sprenkle - CSCI111

20

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

Jan 30, 2019

Sprenkle - CSCI111

21

VARIABLE LIFETIMES AND SCOPE

Jan 30, 2019

Sprenkle - CSCI111

22

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()
```

Jan 30, 2019

Sprenkle - CSCI111

mystery.py

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()
```

Why can we name two different variables X?

Jan 30, 2019

Sprenkle - CSCI111

mystery.py

24

Tracing through Execution

Defines functions

```
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()
```

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

Jan 30, 2019

Sprenkle - CSCI111

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
```

main()

Memory stack

main	x	10
------	---	----

Variable names
are like first names

Function names are like last names
Define the **SCOPE** of the variable

Jan 30, 2019

Sprenkle - CSCI111

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()

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

sum Evens	limit	10
main	x	10

Jan 30, 2019

Sprenkle - CSCI111

27

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

Jan 30, 2019

Sprenkle - CSCI111

28

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	x 0 total 0 limit 10
main	x 10

Jan 30, 2019

Sprenkle - CSCI111

29

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	x 8 total 20 limit 10
main	x 10

Jan 30, 2019

Sprenkle - CSCI111

30

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()
```

Function `sumEvens` returned

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

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

Jan 30, 2019

Sprenkle - CSCI111

31

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

Jan 30, 2019

Sprenkle - CSCI111

32

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

Jan 30, 2019

Sprenkle - CSCI111

33

Writing Comments for Functions

- Good style: Each function **must** have a comment
 - Describes functionality at a high-level
 - Include the *precondition*, *postcondition*
 - Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)

Jan 30, 2019

Sprenkle - CSCI111

34

Writing Comments for Functions

- Include the function's pre- and post- conditions
- **Precondition**: Things that must be true for function to work correctly
 - E.g., num must be even
- **Postcondition**: Things that will be true when function finishes (if precondition is true)
 - E.g., the returned value is the max

Jan 30, 2019

Sprenkle - CSCI111

35

Example Comment

- Describes at high-level
- Describes parameters

```
def printVerse(animal, sound):  
    """  
    Prints a verse of Old MacDonald, plugging in the  
    animal and sound parameters (which are strings),  
    as appropriate.  
    """  
    print(BEGIN_END + EIEIO)  
    print("And on that farm he had a " + animal + EIEIO)  
    ...
```

Comment style: **Docstring**
"documentation string"

Comments from docstrings show up when you use help function

Jan 30, 2019

Sprenkle - CSCI111

36

Write the Docstring Comment for sumEvens

```
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()
```

Jan 30, 2019

Sprenkle - CSCI111

37

TESTING FUNCTIONS

Jan 30, 2019

Sprenkle - CSCI111

38

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
 - 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!

Jan 30, 2019

Sprenkle - CSCI111

39

test Module

- Not a standard module
 - Included with our textbook
 - More sophisticated testing modules but this is sufficient for us
- FUNCTIONS
 - `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.

Jan 30, 2019

Sprenkle - CSCI111

40

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
```

This is the actual result
from our function

This is what we expect the result to be

What are other good test cases?

`testSumEvens.py`

Jan 30, 2019

Sprenkle - CSCI111

41

Summary: Why Write Functions?

- Allows you to break up a hard 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

Jan 30, 2019

Sprenkle - CSCI111

42

Looking Ahead

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

Jan 30, 2019

Sprenkle - CSCI111

43