

Objectives

- Defining your own functions
 - Documentation
 - Testing
 - Development Process
- Broader Issue: Google Search

Feb 4, 2022

Sprenkle - CSCI111

1

1

Review

- What are benefits of functions?
- What is the syntax for creating 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?
- What does a variable's "scope" mean?
- With respect to functions, what are options for how we organize our program?
- How do we document a function? What should its content be?

Feb 4, 2022

Sprenkle - CSCI111

2

2

Review: 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*)

Feb 4, 2022

Sprenkle - CSCI111

3

3

Function Definition Example without Output

```
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)
```

Diagram annotations:

- Keyword**: points to `def`
- Function Name**: points to `moveCircle`
- Input Name/Parameter**: points to `circle, newCenter`
- Body (or function definition)**: points to the entire function body (documentation, calculations, and the `circle.move` call)

Feb 4, 2022

Sprenkle - CSCI111

4

4

Function Definition Example with Output

```
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
```

Keyword points to `def`

Function Name points to `average2`

Input Name/Parameter points to `num1` and `num2`

Body (or function definition) points to the entire function body

Keyword: How to give output points to `return`

Output points to `average`

Feb 4, 2022

Sprenkle - CSCI111

5

5

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 4, 2022

Sprenkle - CSCI111

6

6

Practice: Trace through the Program's Execution

- What is the output of this program?

➤ Example: user enters 4

```
def main():
    num = eval(input("Enter a number to be squared: "))
    squared = square(num)
    print("The square is", squared)

def square(n):
    return n * n

main()
```

Feb 7, 2022

Sprenkle - CSCI111

practice1.py

7

7

Practice

- What is the output of this program?

➤ Example: user enters 4

```
def main():
    num = eval(input("Enter a number to be squared: "))
    squared = square(num)
    print("The square is", squared)
    print("The original num was", num)

def square(n):
    return n * n

main()
```

Feb 7, 2022

Sprenkle - CSCI111

practice2.py

8

8

Practice

- What is the output of this program?

➤ Example: user enters 4

```
def main():
    num = eval(input("Enter a number to be squared: "))
    squared = square(num)
    print("The square is", squared)
    print("The original num was", n)

def square(n):
    return n * n

main()
```

Error! n does not have a value in function main()

Feb 7, 2022

Sprenkle - CSCI111

9

9

Review: 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 7, 2022

Sprenkle - CSCI111

10

10

Summary: 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 4, 2022

Sprenkle - CSCI111

11

11

Review: Where are Functions Defined?

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

Feb 4, 2022

Sprenkle - CSCI111

12

12

Divergence from Text Book: Conventions

Us: main at the top

- See an overview of the code (driver) at the top
- Need to scroll down to see function definitions to understand what the main does
 - Mitigated by having descriptive function names
- Need to call main at the bottom

Book: main at the bottom

- All functions have already been defined before main
- Need to scroll down to the bottom to see the driver/overview of the program
- Need to call main at the bottom

WHAT ARE CHARACTERISTICS OF A GOOD FUNCTION?

Writing a “Good” Function

- Should be an “intuitive chunk”
 - Doesn’t do too much or too little
 - If does too much, try to break into more functions
- Should be reusable
- Should have a descriptive, “action” name
- Should have a comment that tells what the function does

Feb 4, 2022

Sprenkle - CSCI111

15

15

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()
```

Feb 4, 2022

16

16

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) :
    """
    Returns the sum of even numbers from 0 up to but
    not including limit, which is an integer
    """
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total

main()
```

Many other correct doc strings

Feb 4, 2022

Sprenkle - CSCI111

17

17

Writing Documentation for Functions

- Good style: Each function (**except main**) **must** have a comment that documents its use
- Describes functionality, at a high level. `main()` is typically covered by the program's description
- Include the *precondition*, *postcondition*
- Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)
- The exact format matters less than that the content is there

Feb 4, 2022

Sprenkle - CSCI111

18

18

Practice

1. Define the function to calculate our favorite expression: $i^2 + 3j - 5$
 - a. What does the function do?
 - b. What is its input?
 - c. What is its output?
2. Test the function
3. Use the function

our_favorite_expression.py

Feb 4, 2022

Sprenkle - CSCI111

19

19

TESTING FUNCTIONS

Feb 4, 2022

Sprenkle - CSCI111

20

20

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 4, 2022

Sprenkle - CSCI111

21

21

test Module

- Not a standard module
 - Included with our textbook
 - More sophisticated testing modules but this is sufficient for us
- Function:
 - `testEqual(actual, expected[, places=5])`
 - 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 4, 2022

Sprenkle - CSCI111

22

22

Example: Testing sumEvens

```
import test
...
def testSumEvens():          This is the actual result
    actual = sumEvens( 10 ) from our function
    expected = 20 This is what we expect the result to be
    test.assertEqual( 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 4, 2022

Sprenkle - CSCI111

23

23

Exam Next Friday

- **Do not panic**
- In-class, on paper
 - Emphasis on critical thinking
- Exam Preparation Document is on course web page
- Similar problems to class and lab
 - Review questions
 - Worksheets
 - Problems
- Content: up through Tuesday's lab 4
 - Practicing what we learned Wed – Mon
- No broader issue next week

Feb 4, 2022

Sprenkle - CSCI111

24

24

Broader Issue Groups

Amanda Cassandra Jenna Lakpa Mac	Aiden Ellie Nick Renan Shelby	Cole Declan Han Mary	Ford Jack Matt Patrick
--	---	-------------------------------	---------------------------------

Feb 4, 2022

Sprenkle - CSCI111

25

25

Broader Issue: Google Search

- Why is Google search a “broader issue”?
- How does Google search work?
 - How is it tested?
- What are some ways you think searches could be improved?
 - How do you measure “improved search”?
- Will you use Google differently, now that you know how it works (kind of)?

Feb 4, 2022

Sprenkle - CSCI111

26

26

Broader Issue: Google Search

- What power do search engines have?
- Is Google search biased?

Feb 4, 2022

Sprenkle - CSCI111

27

27

Looking Ahead

- Pre-Lab due before lab on Tuesday
- Exam next Friday

Feb 4, 2022

Sprenkle - CSCI111

28

28