

Objectives


- Defining your own functions
 - Variable Scope
 - Documentation
 - Testing

Extra Credit Opportunity

- Attend talk by Professor George Bent on Tuesday, Feb 13 at 5:30 p.m.
 - [Wilson Hall's Concert Hall](#)
- Write up on Canvas for 10 points extra credit
- Can earn up to 50 points extra credit between “special events” and CS in the news posts

Feb 12, 2024

Sprenkle - CSCI111



February 13 - March 22, 2024

Staniar Gallery Presents
**FLORENCE
AS IT WAS**

**THE DIGITAL
RECONSTRUCTION
OF A MEDIEVAL CITY**

This exhibition presents the ongoing art historical project *Florence As It Was*, which aims to digitally reconstruct the Italian city the way it appeared at the end of the fifteenth century. This exhibition features some of the 27 point clouds and 181 photogrammetry models that have been produced by David Pfaff, George Bent, Mackenzie Brooks, and a host of W&L students since the project's inception in 2016.

**Presentation by
Dr. GEORGE BENT
followed by a reception**

Tuesday, February 13
5:30pm in Wilson Hall's Concert Hall

For More Information:
540-458-8861
<https://florenceasitwas.wlu.edu>

ALL ARE WELCOME!
The exhibition, presentation, and reception are free and open to the public. The exhibition is open M-F from 9-5pm. Directions to Gallery: <https://my.wlu.edu/staniar-gallery>

Review

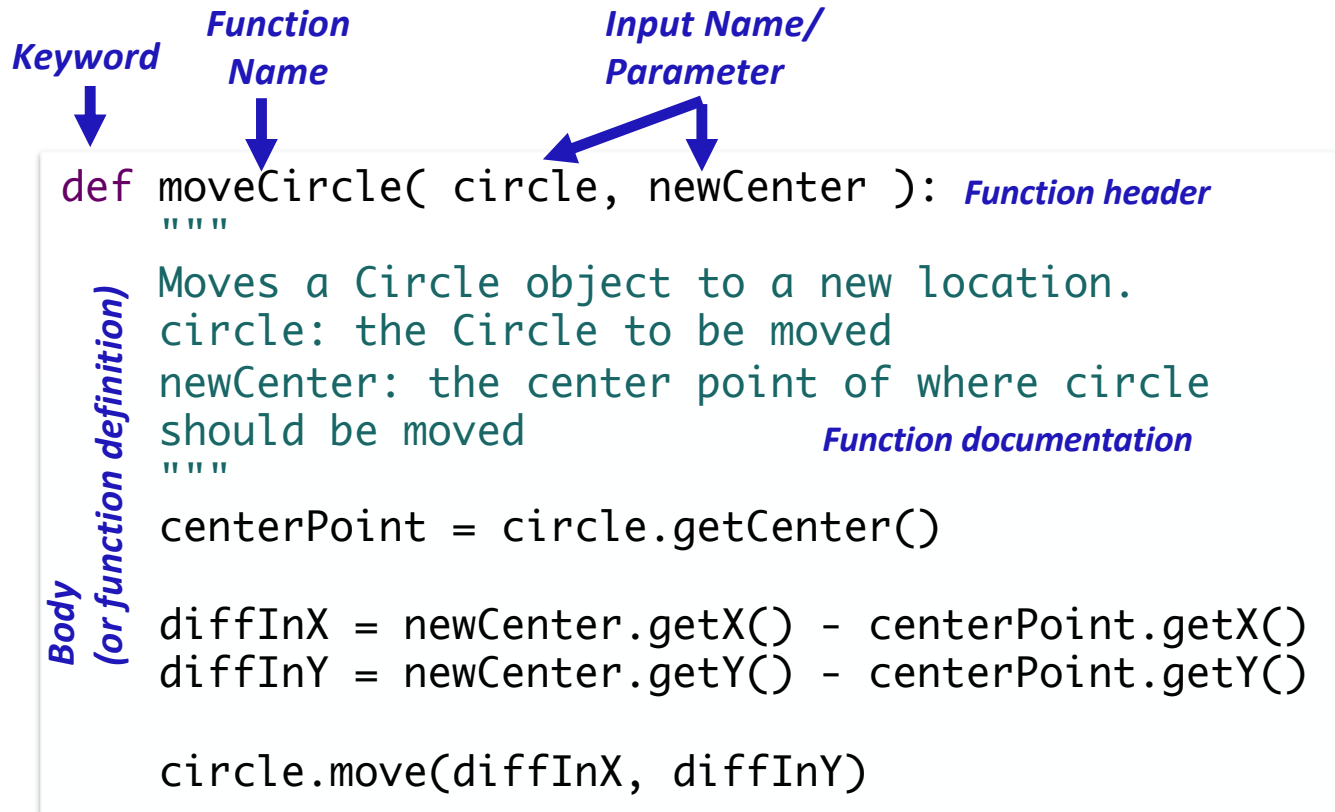
1. What are benefits of functions?
2. What is the syntax for creating our own functions?
 - How do we indicate that our function requires input?
 - How do we indicate that our function has output?
3. What's the difference between output from a function and output from a program?
4. How do we call a function we created?
5. With respect to functions, what are options for how we organize a program?

Review: Why Write Functions?

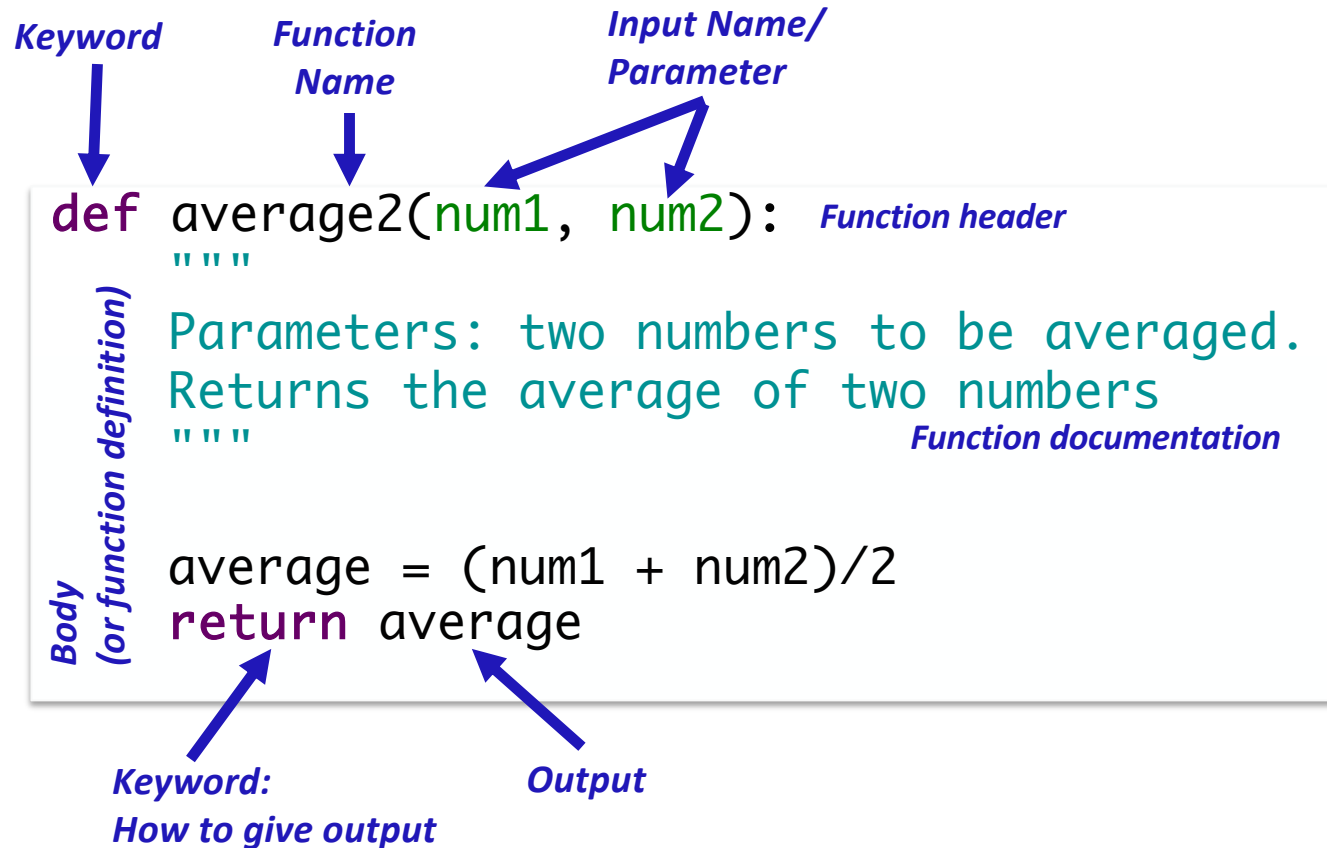
Functions do not allow you to solve any new problems, so why write them?

- 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*)

Function Definition Example without Output



Function Definition Example with Output



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.

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*

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

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

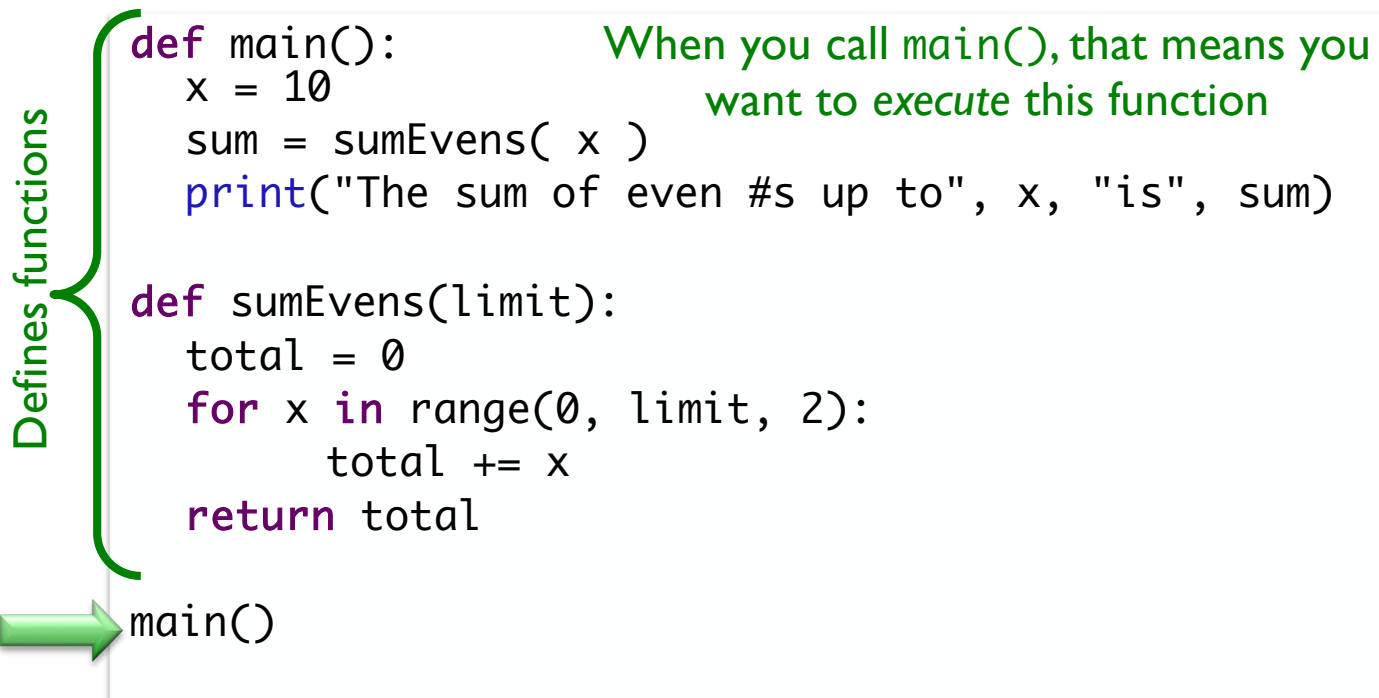
Tracing through Execution

```
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

main()
```

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  
  
main()
```

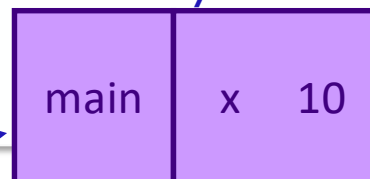
When you call `main()`, that means you want to execute this function

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

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

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

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

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

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

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

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

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()
```

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()
```

```
Enter a number to be squared: 4  
The square is 16
```

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: "))  
    square(num)  
    print("The square is", computed)  
  
def square(n):  
    computed = n * n  
    return computed  
  
main()
```


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: "))  
    square(num)  
    print("The square is", computed)  
  
def square(n):  
    computed = n * n  
    return computed  
  
main()
```

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



Practice: Trace through the Program's Execution

- One possible fix:

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

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)  
    print("The original num was", n)  
  
def square(n):  
    return n * n  
  
main()
```

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)  
    print("The original num was", n)  
  
def square(n):  
    return n * n  
  
main()
```

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

Practice: Trace through the Program's Execution

- One possible fix

```
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()
```

Synthesis: Variable Scope

- “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

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

Writing Documentation for Functions

- Good style: Each function* ***must*** have a comment that documents its use
 - `*main()` usually doesn't have a doc string because it is covered by the program's description
- Describes functionality at a high-level
 - Does *not* describe its implementation
- Focus on the interface/how to use the function:
 - 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
 - I'll show a few different ways to write the documentation

Writing Documentation for Functions

- Tell the function caller how to use the function
- 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; circle must be a Circle object
 - **Postcondition:** Things that will be true when function finishes (if precondition is true)
 - E.g., the returned value is the max; circle will be moved to the new point
- Again, the exact format matters less than the content

Example Documentation

- Describes at high-level
- Describes parameters

Note: does not discuss implementation

```
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”

When you use the `help` function, it shows the docstrings.

Another Example Comment

- Describes at high-level
- Describes parameters

Note: does not discuss implementation

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

Comment style: **Docstring**
“documentation string”

When you use the `help` function, it shows the docstrings.

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()
```

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

TESTING FUNCTIONS

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!

test Module

- Not a standard module
 - Included with our textbook
 - More sophisticated testing modules exist 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.

Example: Testing sumEvens

```
import test
...
def testSumEvens():
    actual = sumEvens( 10 )
    expected = 20
    test.testEqual( actual, expected )
    test.testEqual( sumEvens(12), 30)

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?

Exam Friday

- **Do not panic**
- In-class, on paper
 - Emphasis on critical thinking
 - Lab was to experiment and cement you're learning. Now you're ready!
- 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

Looking Ahead

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