

## Objectives

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

## Emailed Bugs

Sold for \$4.00. \$0.00 over your max bid. There are plenty of similar items up for grabs!

You can expect an e-mail from [REDACTED] approximately 24 hours before your trip departure time, but feel free to shoot [REDACTED] an email at [[TripLeaderEmails]] with any additional questions you might have. And remember - if for any reason you can no longer go, log in to remove yourself from the trip online. No-shows are hard on the trip planning process. If we won't be seeing you, please let us know!

## Review

- What are benefits of functions?
- What are benefits of modules?
- How do we use code in a module in our code?
  - (two ways)
- How do we add animation to our graphics programs?

Looking behind the curtain...

## DEFINING OUR OWN FUNCTIONS

## Functions

- We've used functions
  - Built-in functions: `input`, `eval`
  - Functions from modules, e.g., `math` and `random`
- Benefits
  - Reuse, reduce code
  - Easier to read, write (because of *abstraction*)

Today, we'll learn how to  
**define our own functions!**

Oct 2, 2017

Sprenkle - CSCI111

5

## Review: Functions

- Function is a **black box**
  - Implementation doesn't matter
  - Only care that function generates appropriate output, given appropriate input
- Example:
  - Didn't care how `input` function was implemented
  - Use: `user_input = input(prompt)`



Oct 2, 2017

Sprenkle - CSCI111

6

## Creating Functions

- A function can have
  - 0 or more inputs
  - 0 or 1 outputs
- When we define a function, we know its inputs and if it has output



Oct 2, 2017

Sprenkle - CSCI111

7

## 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

Oct 2, 2017

Sprenkle - CSCI111

8

## Writing a Function

- I want a function that averages two numbers

- What is the input to this function?
- What is the output to this function?

## Writing a Function

- I want a function that averages two numbers
- What is the input to this function?
  - The two numbers
- What is the output to this function?
  - The average of those two numbers, as a float

These are key questions to ask yourself when designing your own functions.

- Inputs: What are the parameters?
- Output: What is getting returned?

## Averaging Two Numbers



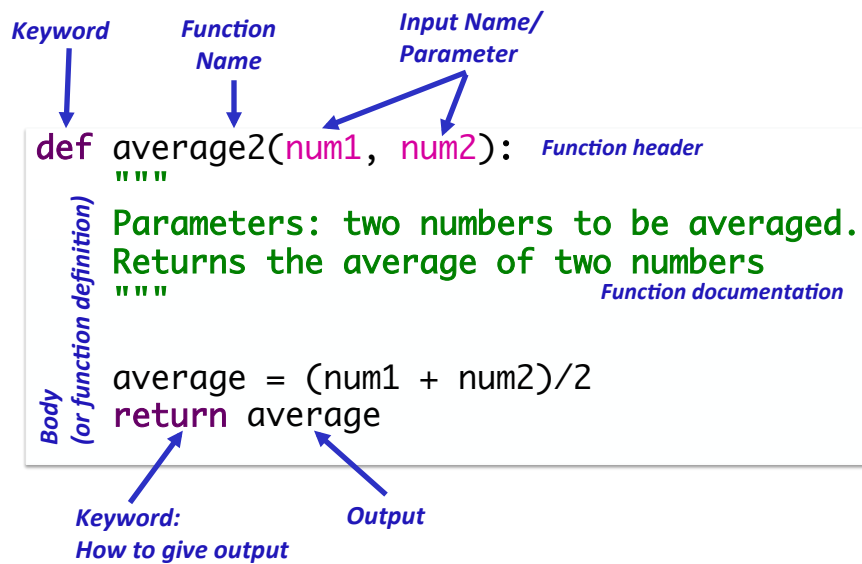
- **Input:** the two numbers
- **Output:** the average of two numbers

Oct 2, 2017

Sprenkle - CSCI111

11

## Syntax of Function Definition



Oct 2, 2017

Sprenkle - CSCI111

12

## Calling your own functions

Same as calling someone else's functions ...

average = average2(100, 50)

↑  
*Output is assigned to average*

↑  
*Function Name*

↑  
*Input*

Oct 2, 2017

Sprenkle - CSCI111

average2.py

13

## 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

Oct 2, 2017

Sprenkle - CSCI111

14

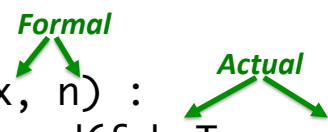
## Parameters

- The **inputs** to a function are called **parameters** or **arguments**, depending on the context
- When **calling**/using functions, arguments must appear in same order as in the function header
  - Example: `round(x, n)`
    - `x` is the float to round
    - `n` is int of decimal places to round `x` to

## Parameters

- **Formal Parameters** are the variables named in the function definition
- **Actual Parameters** or **Arguments** are the variables or literals that really get used when the function is called.

Defined: `def round(x, n) :`  
Use: `roundCelsius = round(fahrTemp, 3)`



Formal & actual parameters must match in **order**, **number**, and **type**!



## Passing Parameters

- Only **copies** of the actual parameters are given to the function for **immutable** data types
  - Immutable types: most of what we've talked about so far
    - Strings, integers, floats
  - The actual parameters in the *calling* code **do not** change
- (Lists are mutable and have different rules)

Oct 2, 2017

Sprenkle - CSCI111

17

## 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

Oct 2, 2017

Sprenkle - CSCI111

18

## CONTROL FLOW WITH FUNCTIONS

Oct 2, 2017

Sprenkle - CSCI111

19

### 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
```

Oct 2, 2017

Sprenkle - CSCI111

20

## Flow of Control

```
def max(num1, num2):  
    result = 0  
    if num1 >= num2:  
        result = num1  
    else:  
        result = num2  
    return result  
  
x = 12  
y = eval(input("Enter a number: "))  
z = max(x, y)  
print("The max is", z)
```

Oct 2, 2017

Sprenkle - CSCI111

flow\_example.py 21

## Flow of Control

```
def max(num1, num2):  
    result = 0  
    if num1 >= num2:  
        result = num1  
    else:  
        result = num2  
    return result
```

What does this function do?

### Function definitions:

- Save functions for later use, nothing executed
- Similar to adding a contact into your phone book  
→ not actually calling

```
x = 12 ← Program starts "doing stuff"  
y = float(input("Enter a number: "))  
z = max(x, y)  
print("The max is", z)
```

Oct 2, 2017

Sprenkle - CSCI111

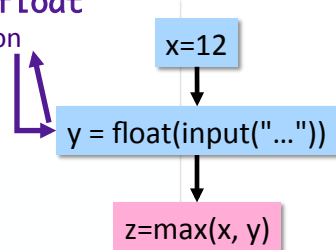
22

## Flow of Control

```
def max(num1, num2):
    result = 0
    if num1 >= num2:
        result = num1
    else:
        result = num2
    return result
```

```
x = 12
y = float(input("Enter a number: "))
z = max(x, y)
print("The max is", z)
```

To **input** and  
then **float**  
function



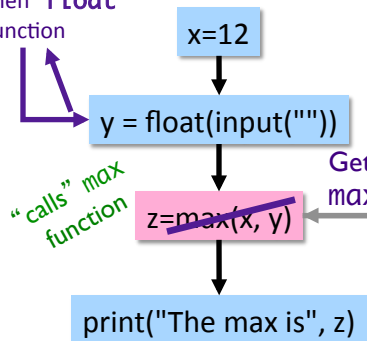
Oct 2, 2017

Sprenkle - CSCI111

23

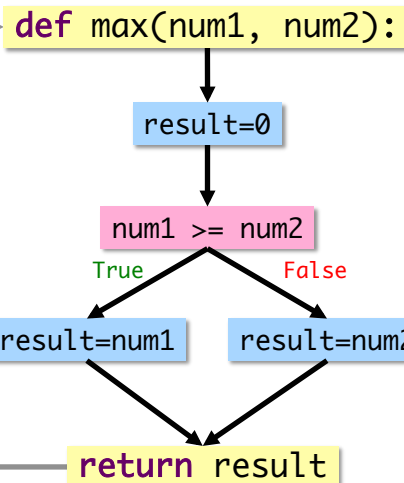
## Flow of Control

To **input** and  
then **float**  
function



```
def max(num1, num2):
    result = 0
    if num1 >= num2:
        result = num1
    else:
        result = num2
    return result
```

num1 is set to value of x  
num2 is set to value of y



return to caller

Oct 2, 2017

Sprenkle - CSCI111

24

## Flow of Control: Using **return**

Is this implementation of the function correct?

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    else:  
        return num2
```

Oct 2, 2017

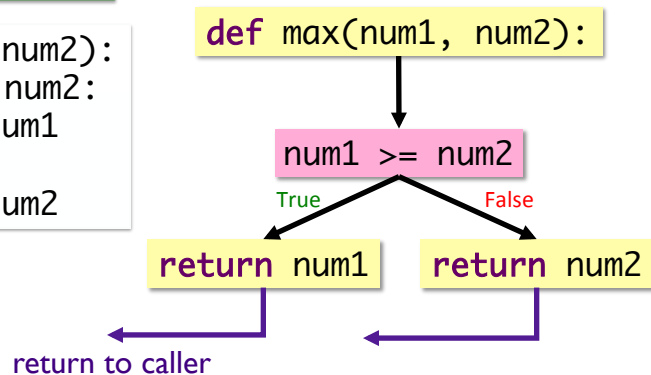
Sprenkle - CSCI111

25

## Flow of Control: Using **return**

Is this implementation of the function correct?

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    else:  
        return num2
```



Oct 2, 2017

Sprenkle - CSCI111

26

## Flow of Control: Using **return**

Is this implementation of the function correct?

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    return num2
```

Oct 2, 2017

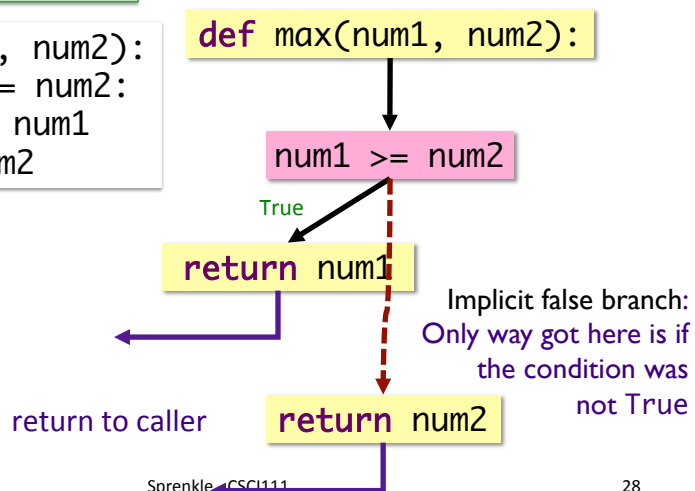
Sprenkle - CSCI111

27

## Flow of Control: Using **return**

Is this implementation of the function correct?

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    return num2
```



Oct 2, 2017

Sprenkle - CSCI111

28

## Function Input and Output

```
BEGIN_END = "Old McDonald had a farm"  
EIEIO = ", E-I-E-I-O"
```

- What does this function do?
- Identify function's input and 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()
```

Oct 2, 2017

Sprenkle - CSCI111

29

## 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
```

Oct 2, 2017

Sprenkle - CSCI111

30

# PROGRAM ORGANIZATION

Oct 2, 2017

Sprenkle - CSCI111

31

## 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*

Oct 2, 2017

Sprenkle - CSCI111

32



## 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():
```

Oct 2, 2017

Sprenkle - CSCI111

33

## 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

Oct 2, 2017

Sprenkle - CSCI111

34

## Example program with a main()

```
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, 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()

```
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. Set definition of main  
2. Set definition of 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

Oct 2, 2017

Sprenkle - CSCI111

37

## Looking Ahead

- Lab tomorrow
  - Bring exam questions
- Wednesday: Work period
- Exam on Friday – Prep Document on Course Schedule page
  - Very short answer
  - Short answer
  - Writing code

Oct 2, 2017

Sprenkle - CSCI111

38