Objectives

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

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

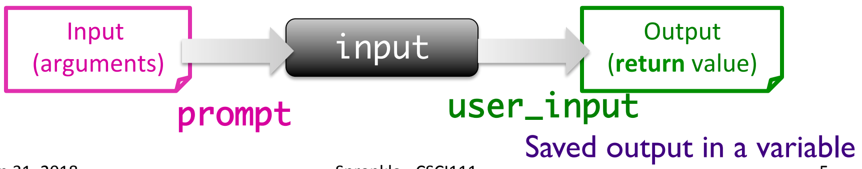
Functions

- 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**!

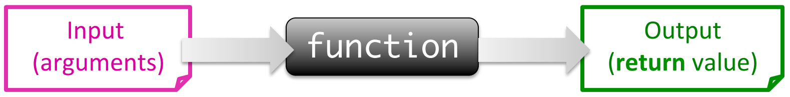
Review: Functions

- 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)`



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



Writing a Function

- We want a function that moves a circle to a new location
- Recall:

```
# create the circle in the center of the window and draw it
midPoint = Point(win.getWidth()/2, win.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(win)

# get where the user clicked
userClicked = win.getMouse()

# Move the circle to where the user clicks
centerPoint = myCircle.getCenter()

dx = userClicked.getX() - centerPoint.getX()
dy = userClicked.getY() - centerPoint.getY()

myCircle.move(dx,dy)
```

Make a Function to Do That

Parameters/inputs:

The circle to move

The point to move the circle to

```
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)
```

Make a Function to Do That

Keyword *Function Name* *Input Name/ Parameter*

```
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)
```

Function header

Function documentation

Body (or function definition)

Defining a Function

- Gives a name to some code that you'd like to be able to call again
- Analogy:
  - Defining a function: saving name, phone number, etc. in your contacts
  - Calling a function: calling that number

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.

Formal

Actual

Defined: `def round(x, n) :`

Use: `roundCelc = round(celcTemp, 3)`

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

Calling the Function

```
# create the circle in the center of the window and draw it
midPoint = Point(win.getWidth()/2, win.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(win)

# get where the user clicked
userClicked = win.getMouse()

moveCircle( myCircle, userClicked )
```

The circle to move   The point to move the circle to

Compare the code…

`circleShiftWithFunction.py`

Writing a Function

- Let's look at one more example
- I want a function that averages two numbers

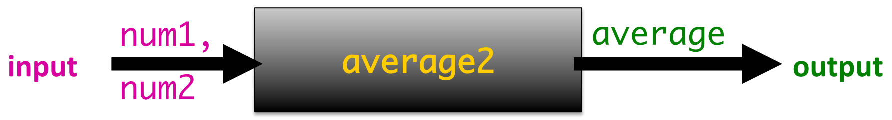
• What is the input to this function?
• What is the output from this function?

Writing a Function

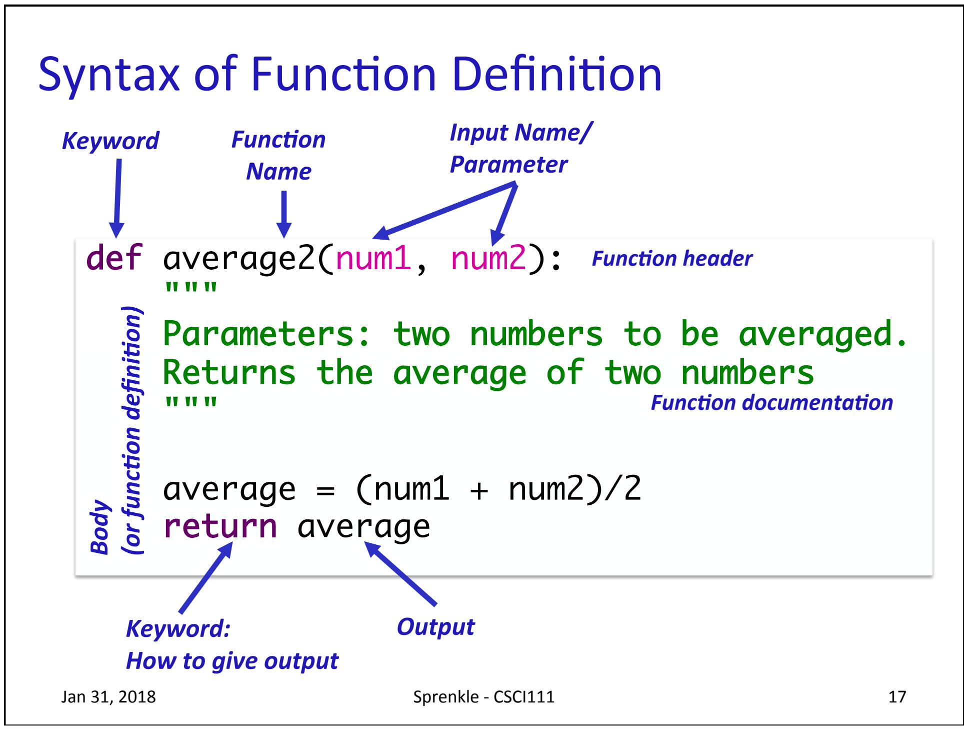
- I want a function that averages two numbers
- What is the input to this function?
  - The two numbers
- What is the output to this function?
  - The average of those two numbers, as a float

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



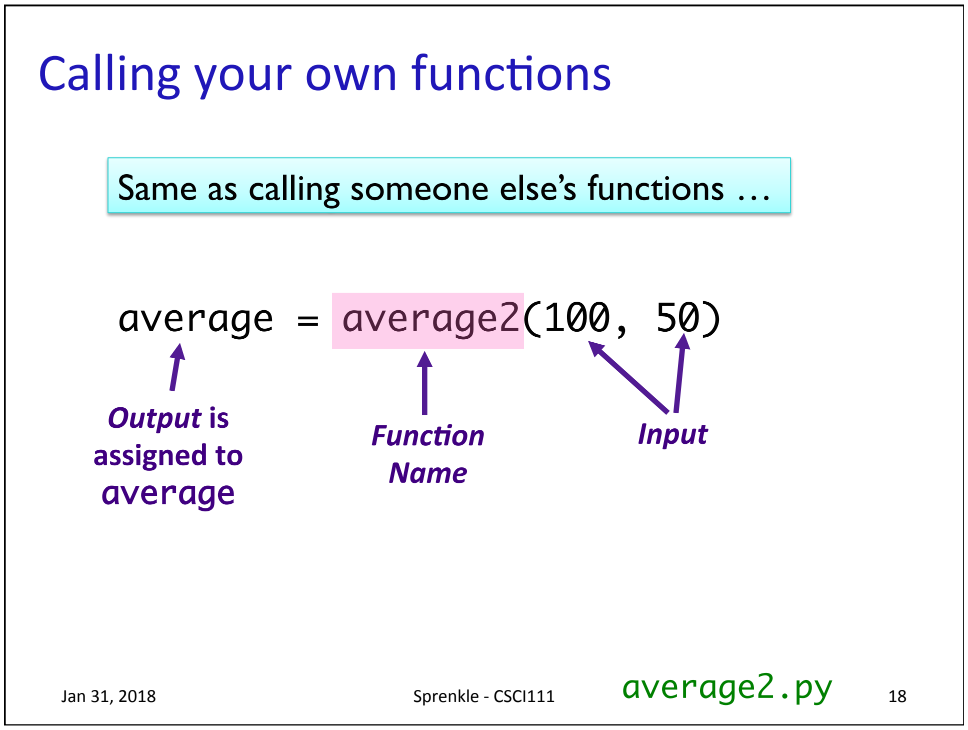
- Input: the two numbers
- Output: the average of two numbers



Syntax of Function Definition

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

    average = (num1 + num2)/2
    return average
```

Keyword
Function Name
Input Name/ Parameter
Function header
Function documentation
Body (or function definition)
Keyword: How to give output
Output



Calling your own functions

Same as calling someone else's functions …

```
average = average2(100, 50)
```

Output is assigned to average
Function Name
Input

average2.py

Functions: Similarity to Math

- In math, a function definition looks like:

$$\boxed{f(x) = x^2 + 2}$$

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

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 and Graphics objects are mutable and have different rules

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

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
```

average2.py

PROGRAM ORGANIZATION

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***

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():
```

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

Function Input and Output

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

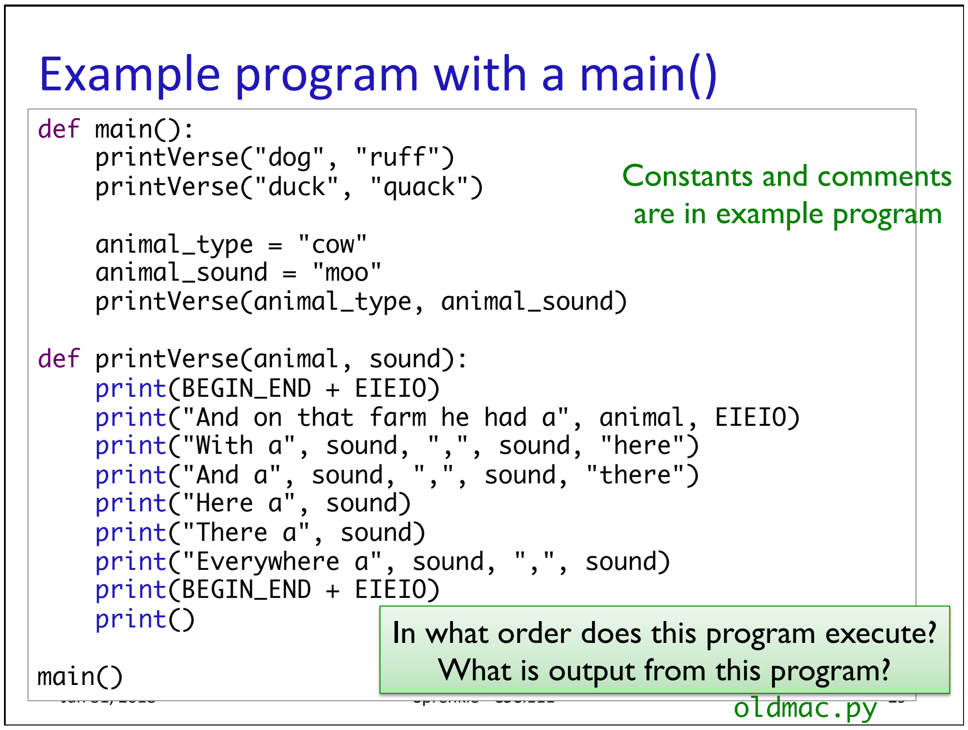
```python
def printVerse(animal, sound):
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound,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

Function Input and Output

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

```python
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



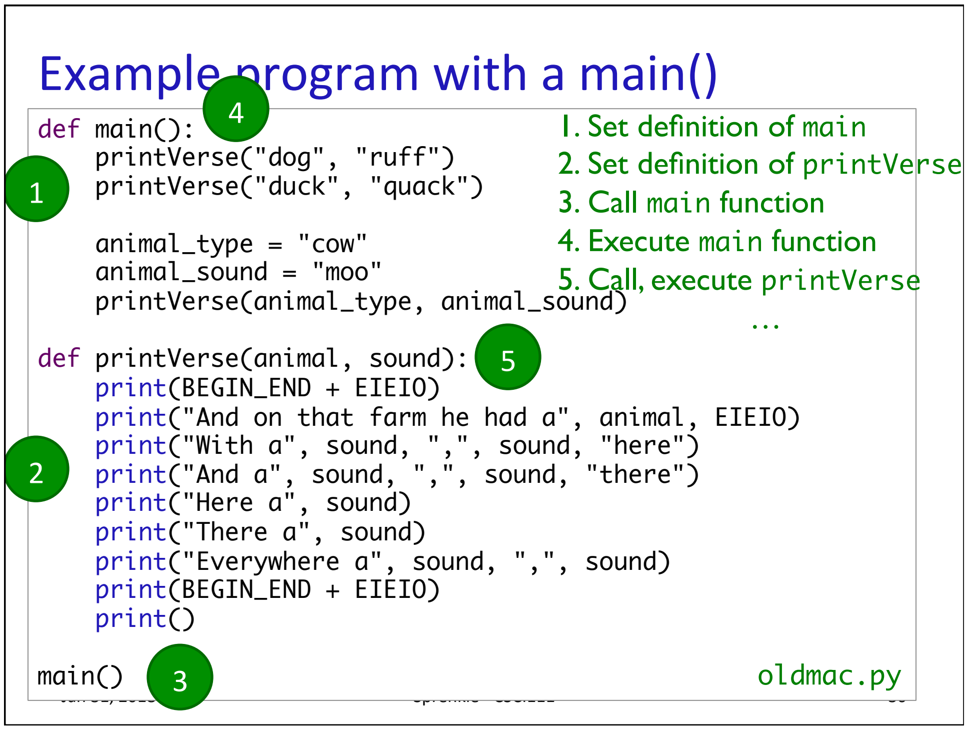
Example program with a main()

```
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 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()

```
def main():   4
    printVerse("dog", "ruff")
1   printVerse("duck", "quack")

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

def printVerse(animal, sound):   5
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
2   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()   3
```

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

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

Looking Ahead

- BI – Net Neutrality
- Lab 3 due Friday
- Exam next Friday
    - Prep document up soon