Objectives

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

1

Looking behind the curtain…

DEFINING OUR OWN FUNCTIONS

2

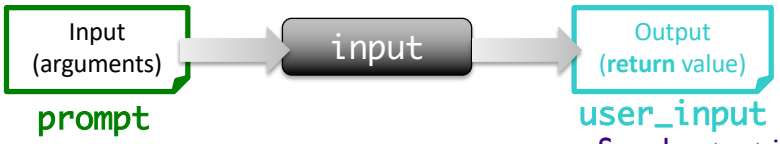
Functions

- 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

5

Writing a Function

- Goal: 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(canvas.getWidth()/2, canvas.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(canvas)

# get where the user clicked
new_point = canvas.getMouse()

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

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

myCircle.move(dx,dy)

canvas.getMouse()
```


6

A Function to Move a Circle

Inputs/Parameters:

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)
```


A Function to Move a Circle

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

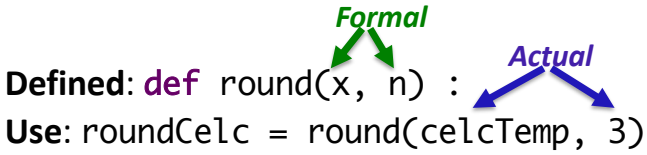
- Gives a name to some code that you'd like to be able to call again
- Analogy:
  - **Defining a function**: saving name, phone number, etc. in your contacts
  - **Calling a function**: calling that number

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 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(canvas.getWidth()/2, canvas.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(win)

# get where the user clicked
new_point = canvas.getMouse()

moveCircle( myCircle, new_point )
```

The circle to move The point to move the circle to

Same as calling someone else's functions …

Compare the code…

circleShiftWithFunction.py

A Function to Move a Circle

Note: I'm using "generic" names (e.g., circle, newCenter)
Why? A function should be general-purpose.
We want anyone who to be able to use this function for their purposes, specifically, anyone who wants to move their circle to a new spot can use this function

```
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)
```


Writing a Function

- I want a function that averages two numbers

- What is the input to this function?
- What is the output from this function?
- What should the function do?

Writing a Function

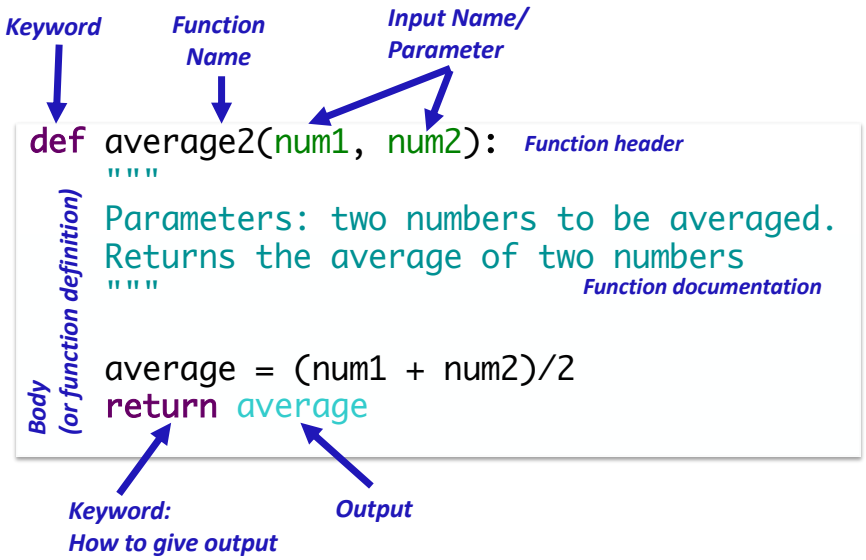
- I want a function that averages two numbers
- What is the input to this function?
  - The two numbers
- What is the output from this function?
  - The average of those two numbers, as a float

These are key questions to ask yourself when designing your own functions.
- Inputs: What are the parameters?
- Output: What is getting returned?
- Body: What does the function do?

Averaging Two Numbers

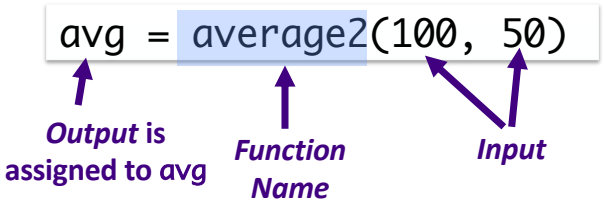
- **Input**: the two numbers
- **Output**: the average of two numbers

Syntax of Function Definition



Calling your own functions

Same as calling someone else's functions …




```
avg = average2(num1, num2)
```

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

Function Output

- When the code reaches a statement like

```
return x
```

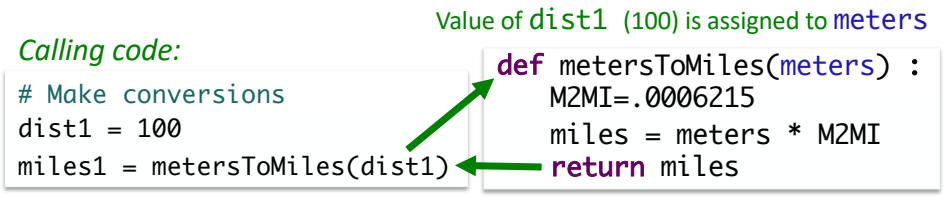
  - The function stops executing
  - x is the **output *returned*** to the place where the function was called
- For functions that don't have explicit output, `return` does not have a value with it, e.g.,

```
return
```

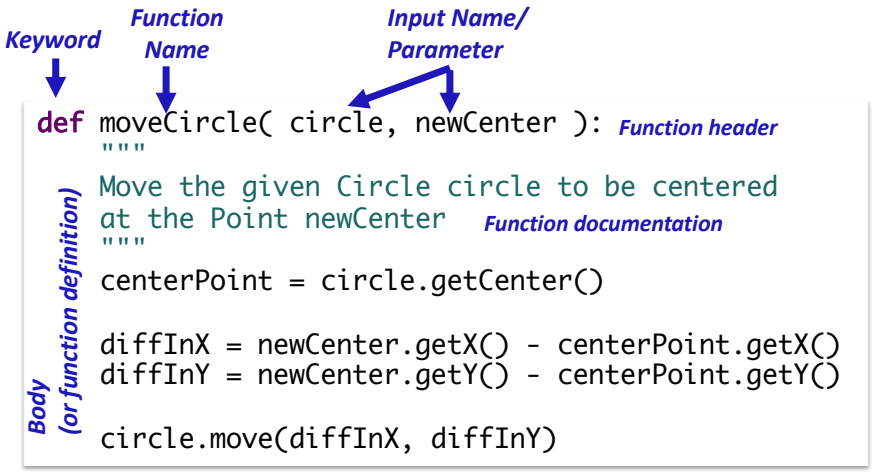
- Optional: don't *need* to have `return`
  - Function *automatically* returns at the end (like `moveCircle`)

Flow of Control

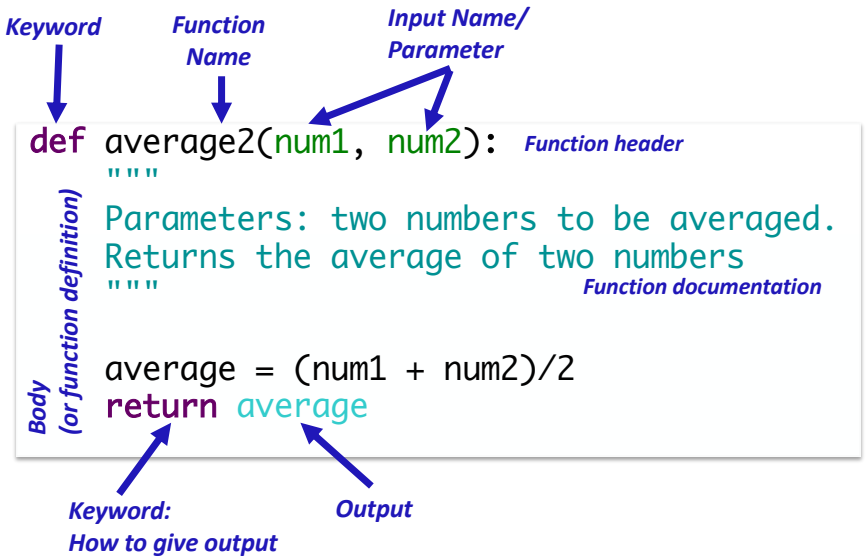
- When program calls a function, the program jumps to the function and executes it
- After executing the function, the program returns to the same place in the *calling code* where it left off



Function Definition Example without Output



Function Definition with Output



Function Input and Output

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

Constants and comments are in example program

```
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()
```

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

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. Maybe another place we call it, we want to round the result to 4 decimal places.

`return` vs `print`

`return`

- 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

`print`

- Use print for
  - Debugging your function (then remove before final submission)
    - Otherwise, unintended side effect of calling the function
  - When you have a function that is supposed to *display* something
    - Sometimes, displaying something is what you want.

With experience, you'll learn when to use each one

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

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 near 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
- `main` is a function that calls other functions
  - *Any* function can call other functions !

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()
```

4 1 5 2 3

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

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

VARIABLE LIFETIMES AND SCOPE

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()
```

To be continued…

Looking Ahead

- Lab 3 is due on Friday
- Broader Issue write up is due Thursday night