

## Objective

- Using functions
- Animation
- Defining functions

Jan 28, 2019

Sprenkle - CSCI111

1

## Review

- What new design pattern did we discuss?
  - What are its steps?
- What is a function?
- What are some variations in how we use the print function?
- How do we use another module in our program?
  - Two ways – what are the implications of each way
- How do we find out what a module provides?
- What are two modules we discussed?

Jan 28, 2019

Sprenkle - CSCI111

2

## Review: Functions



- Syntax:
  - `func_name(arg0, arg1, ..., argn)`
- Depending on the function, arguments may or may not be required
  - `[]` indicate an optional argument
- Semantics: depend on the function

Jan 28, 2019

Sprenkle - CSCI111

3

## Review: Using Python Libraries

- To use the definitions in a module, you must first **import** the module
  - Example: to use the `math` module's definitions, use the import statement: **import math**
  - Typically import statements are at **top** of program
- To find out what a module contains, use the **help** function
  - Example within Python interpreter:

```
>>> import math
>>> help(math)
```

Jan 28, 2019

Sprenkle - CSCI111

4

## Review:

### Benefits of Using Python Libraries/Modules

- Don't need to rewrite code
- If it's in a module, it is very *efficient* (in terms of computation speed and memory usage)

Jan 28, 2019

Sprenkle - CSCI111

5

## VA Lottery: Pick 4

- To play: pick 4 numbers between 0 and 9, inclusive
- To win: select the numbers that are selected by the magic ping-pong ball machine
- Your job: Simulate the magic ping-pong ball machines
  - Display the number on one line

Jan 28, 2019

Sprenkle - CSCI111

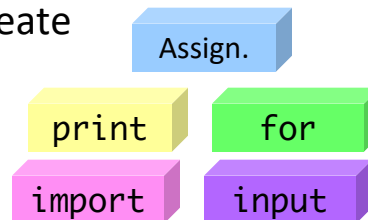
`pick4.py`

6

## Programming Building Blocks

- Adding to your tool set
- We can combine them to create more complex programs

➤ Solutions to problems



Jan 28, 2019

Sprenkle - CSCI111

7

## ANIMATION

Jan 28, 2019

Sprenkle - CSCI111

8

## Review: Circle Shift

- Move a circle to the position clicked by the user
  - Repeat three times

Jan 28, 2019

Sprenkle - CSCI111

circleShift.py 9

## Animation

- Use combinations of the method **move** and the function **sleep**
  - Need to **sleep** so that humans can see the graphics moving
  - Computer would process the **moves** too fast!
- **sleep** is part of the **time** module
  - takes a float parameter representing *seconds* and pauses for that amount of time

Jan 28, 2019

Sprenkle - CSCI111

animate.py

10

## Problem: Animate Moving to User Click

- In X steps, move from the circle's current location to the location clicked by user

Jan 28, 2019

Sprenkle - CSCI111

`circleShiftAnim.py`

11

## Examples of Animation

- From Previous Classes

Jan 28, 2019

Sprenkle - CSCI111

12

Looking behind the curtain...

## DEFINING OUR OWN FUNCTIONS

Jan 28, 2019

Sprenkle - CSCI111

13

## 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!**

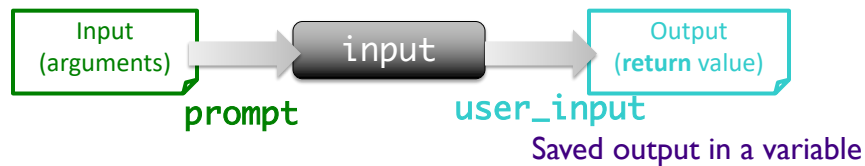
Jan 28, 2019

Sprenkle - CSCI111

14

## 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)**



Jan 28, 2019

Sprenkle - CSCI111

15

## 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**



Jan 28, 2019

Sprenkle - CSCI111

16



## 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(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)
```

## 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

```
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)
```

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

Jan 28, 2019

Sprenkle - CSCI111

19

## 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

Jan 28, 2019

Sprenkle - CSCI111

20

## 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

Jan 28, 2019

Sprenkle - CSCI111

21

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

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

Jan 28, 2019

Sprenkle - CSCI111

22

## 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

Compare the code...

`circleShiftWithFunction.py`

Jan 28, 2019

Sprenkle - CSCI111

23

## 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?

Jan 28, 2019

Sprenkle - CSCI111

24

## 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?

Jan 28, 2019

Sprenkle - CSCI111

25

## Averaging Two Numbers



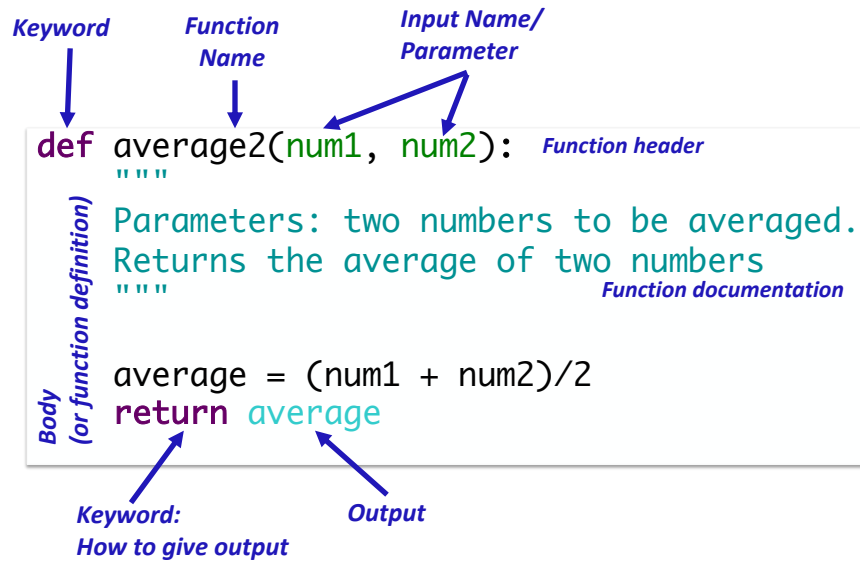
- **Input:** the two numbers
- **Output:** the average of two numbers

Jan 28, 2019

Sprenkle - CSCI111

26

## Syntax of Function Definition



The diagram illustrates the syntax of a function definition in Python. It shows a code block with the following components:

- Keyword:** `def`
- Function Name:** `average2`
- Input Name/Parameter:** `num1, num2`
- Function header:** `def average2(num1, num2):`
- Function documentation:** A multi-line string containing the text: `Parameters: two numbers to be averaged. Returns the average of two numbers`
- Body (or function definition):** The code block containing the function logic: `average = (num1 + num2)/2` and `return average`
- Keyword: How to give output:** `return`
- Output:** `average`

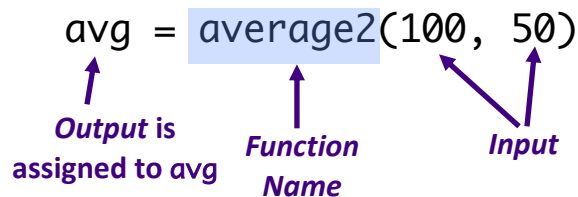
Jan 28, 2019

Sprenkle - CSCI111

27

## Calling your own functions

Same as calling someone else's functions ...



The diagram illustrates the syntax of a function call in Python. It shows a code block with the following components:

- Output is assigned to avg:** `avg =`
- Function Name:** `average2`
- Input:** `100, 50`

Jan 28, 2019

Sprenkle - CSCI111

average2.py

28

## 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

Jan 28, 2019

Sprenkle - CSCI111

29

## 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

Jan 28, 2019

Sprenkle - CSCI111

30

## 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
```

Jan 28, 2019

Sprenkle - CSCI111

*average2.py*

31

## Looking Ahead

- Pre Lab 3 - tomorrow
- Lab 3 – due Friday
- BI – due Friday

Jan 28, 2019

Sprenkle - CSCI111

32