Objective

- Animation
- Defining functions

1

Review

- 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?
- How do we animate pictures created using the graphics library?

2

Review: Functions

- Syntax:
  - `func_name(arg0, arg1, …, argn)`

Argument list (input)

- Depending on the function, arguments may or may not be required
  - [ ] indicate an optional argument
- Semantics: depend on the function

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
  - Prepend the module to refer to parts of module, e.g., `math.sqrt(x)`
- To find out what a module contains, use the **`help`** function
  - Example within Python interpreter:

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


4

Review:
Benefits of Using Python Libraries/Modules

- Don't need to rewrite code
  - Reuse, reduce code
- Easier to read, write (because of ***abstraction***)
- If it's in a Python module, it is very *efficient* (in terms of computation speed and memory usage)

Review: 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 **move**s too fast!
- **`sleep`** is part of the **`time`** module
  - takes a `float` parameter representing *seconds* and pauses for that amount of time

`animate.py`

Animate Circle Shift!

- Animate moving a circle to the position clicked by the user
  - Previously, moved in one fell swoop

```
dx = newX - circle.getCenter().getX()
dy = newY - circle.getCenter().getY()

circle.move(dx, dy)
```

  - To animate
    - Break the movement into chunks
    - Repeatedly, move one chunk, sleep
- Finally, do the user clicks/animation 3 times

 `circleShiftAnim.py`

Looking behind the curtain…

DEFINING OUR OWN FUNCTIONS

Functions

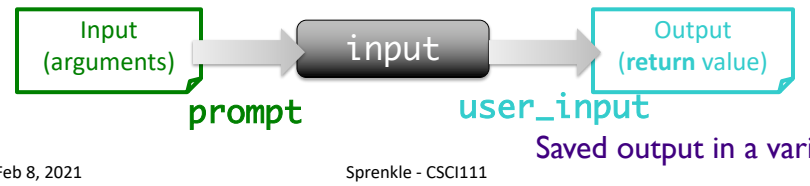
- We've used functions
  - Built-in functions: `input`, `eval`
  - Functions from modules, e.g., `math` and `random`
- Benefits
  - Reuse, reduce code
  - Easier to read, write (because of ***abstraction***, solving smaller problems)

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

9

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)`



10

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

11

Writing a Function

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

```
# create the circle in the upperleft of window and draw it
cirPoint = Point(CIRCLE_RADIUS, CIRCLE_RADIUS)
myCircle = Circle(cirPoint, CIRCLE_RADIUS)
myCircle.draw(canvas)

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

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

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

myCircle.move(dx,dy)
…
```

Process: need to identify
- What function does
- Inputs to function
- Outputs from function

12

Make a Function to Do That

Parameters/inputs:

The circle to move The point to move the circle to

```
def moveCircle( circle, newCenter ):
    """
    Moves a Circle object a new location.
    circle: the Circle to be moved
    newCenter: the center point of where circle
    should be moved
    """
    centerPoint = circle.getCenter()

    diffInX = newCenter.getX() - centerPoint.getX()
    diffInY = newCenter.getY() - centerPoint.getY()

    circle.move(diffInX, diffInY)
```


13

Make a Function to Do That

Keyword *Function Name* *Input Name/ Parameter*

```
def moveCircle( circle, newCenter ):
    """
    Moves a Circle object to a new location.
    circle: the Circle to be moved
    newCenter: the center point of where circle
    should be moved
    """
    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)

14

Defining a Function

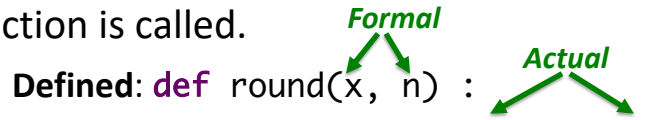
- Gives a name to some code that you'd like to be able to call again
- Analogy:
  - **Defining a function**: saving name, phone number, etc. in your contacts
  - **Calling a function**: calling that number

Parameters

- The inputs to a function are called ***parameters*** or ***arguments***, depending on the context
- When ***calling*** 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 upperleft of window and draw it
cirPoint = Point(CIRCLE_RADIUS, CIRCLE_RADIUS)
myCircle = Circle(cirPoint, CIRCLE_RADIUS)
myCircle.draw(canvas)

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

# Move the circle to where the user clicks
moveCircle( myCircle, new_point )
```

The circle to move

The point to move the circle to

Compare the code…

`circleShiftWithFunction.py`

Writing a Function

- Let's look at another 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 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? → parameters
- Output? → what (if anything) is returned

Averaging Two Numbers

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

Syntax of Function Definition

Keyword *Function Name* *Input Name/ Parameter*

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

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

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 …

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

Function's output is assigned to avg

Function Name

Input

average2.py

Functions: Similarity to Math

- In math, a function definition looks like:

$$f(x) = x^2 + 2$$

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

Function Output

- When the code reaches a statement like

```
return output
```

  - The function stops executing
  - output 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 (e.g., `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

Value of `myMiles` (100) is assigned to `meters`

Calling code:

```
# Make conversions
myDist = 100
myMiles = metersToMiles(myDist)
print("The number of miles is", miles)
```

```
def metersToMiles(meters) :
    M2MI=.0006215
    miles = meters * M2MI
    return miles
```

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

27

Using `print` vs `return`

- `print` is for displaying information
- Don't necessarily 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. Another place we call the function, we want to round to 4 decimal places.

28

`return` vs `print`

- 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
- 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.

Arithmetic Example

- Our favorite expression: $i^2 + 3j - 5$

1. Define the function:
   a. What does the function do?
   b. What is its input?
   c. What is its output?
2. Call the function

`our_favorite_expression.py`

Looking Ahead

- Pre Lab 3 – tomorrow
  - Not defining functions yet; we'll continue on that for a bit more
- Lab 3 – due Friday
- BI – due Friday
  - Google Search