Functions Cont'd + Midterm Review

Modules + Mock Midterm

Emily Zhao
CSCI-UA-002

Agenda

- Functions (continued)
- Mock Midterm
- Midterm Survey

Functions pt. 2

Modules

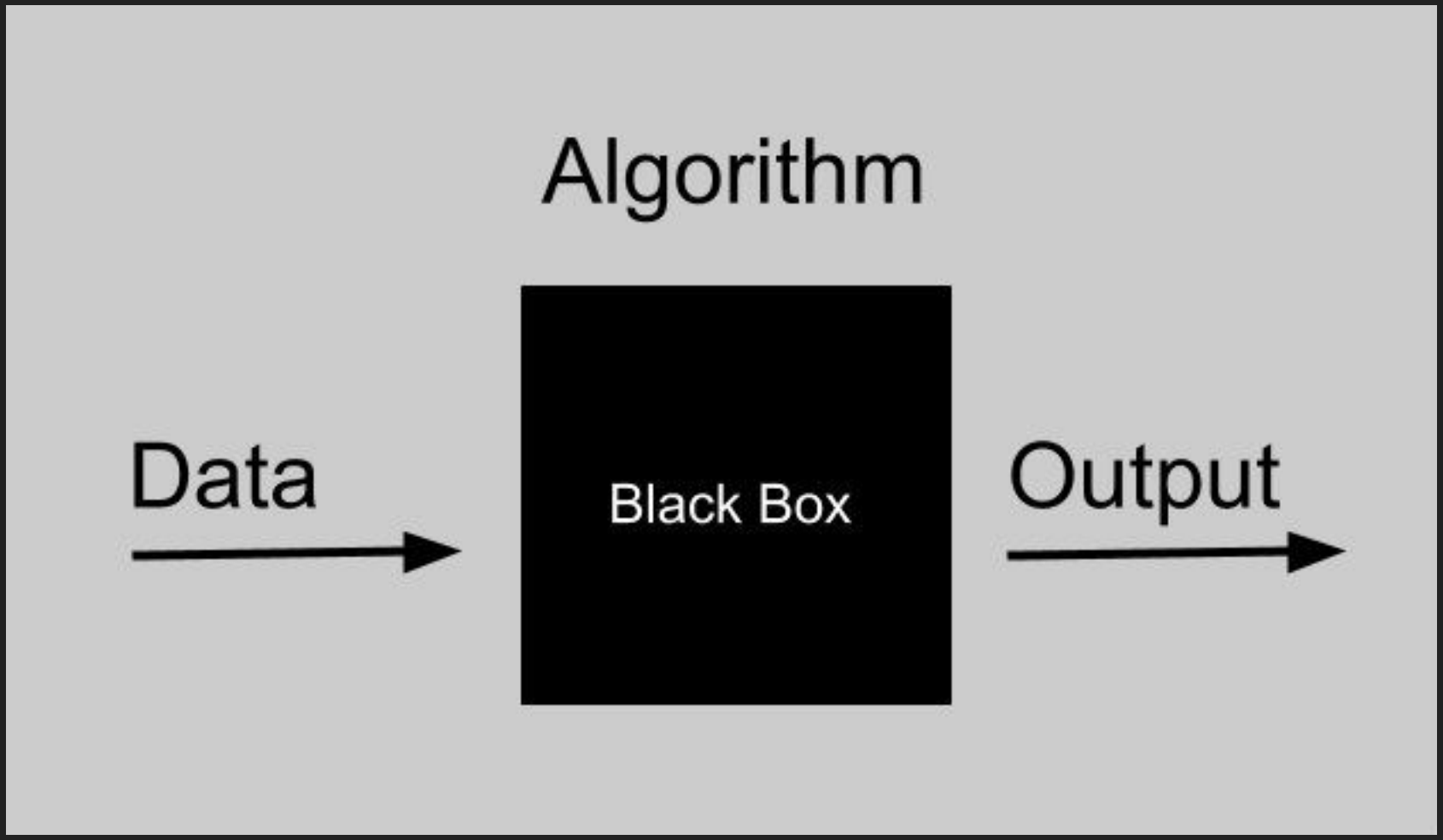
- All programming languages come pre-packaged with a standard library of functions that are designed to make your job as a programmer easier
- Some of these functions are built right into the “core” of Python (print, input, range, etc)
- Other more specialized functions are stored in a series of files called “modules” that Python can access upon request by using the “import” statement
  - `import random`
  - `import time`

Modules

- The import statement tells Python to load the functions that exist within a specific module into memory and make them available in your code
- Because you don’t see the inner workings of a function inside a module we sometimes call them “black boxes”
- A “black box” describes a mechanism that accepts input, performs an operation that can’t be seen using that input, and produces some kind of output


Input
Function
(does something)
Output

“Black Box” model



```
random.randint(0,10)
```

Input

```
0, 10
```

→

randint()

Algorithm: *randomly generates a number between 0 and 10*

→

Output

```
8
```

More information about a module

- To see information about a module, you can do the following in IDLE:
  - `help("modulename")`

- The help() function takes one argument (a string that represents the name of the module) and returns the user manual for that module

```
>>> help(print)
...

Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

IPO Notation

- As you start writing more advanced functions you should think about documenting them based on their Input, Processing and Output (IPO)

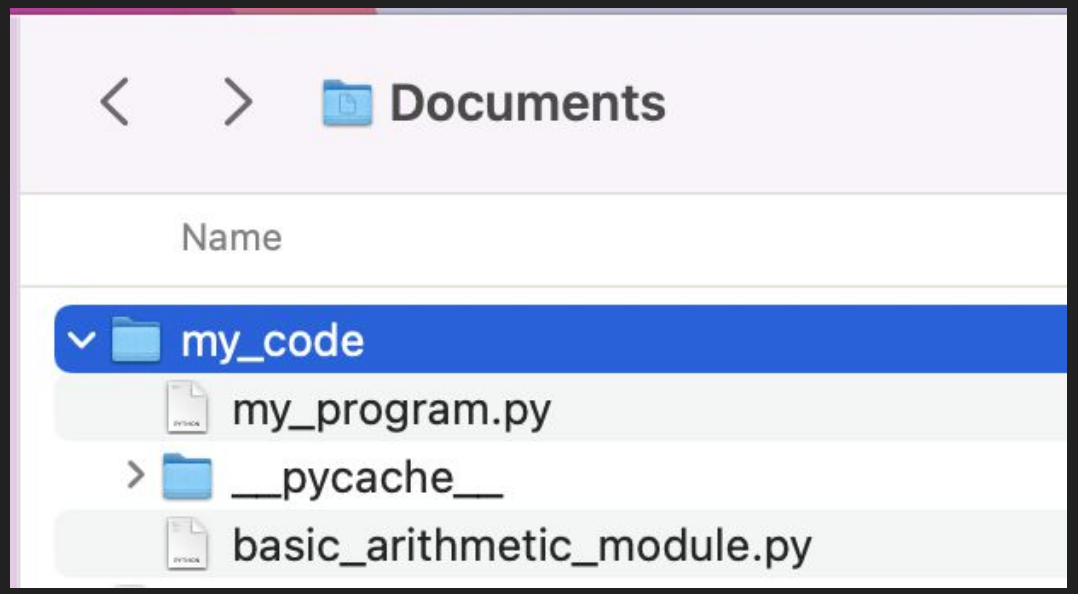
- Example:

```
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing:  combines the two integers
# output: returns the combined value

def add_ages(age1, age2):
    sum = age1+age2
    return sum
```

```python
def add(a, b):
    '''
    input: takes in two integers
    processing: adds a and b together
    output: returns the sum
    '''
    c = a + b
    return c
```

```
>>> help(add)
...
Help on function add in module __main__:

add(a, b)
    input: takes in two integers
    processing: adds a and b together
    output: returns the sum
```

How to write and use your own module



Your module.py and your program.py **<u>need to be in the same folder</u>!**

— I have placed them both in a folder called my_code

Note: a folder called _pycache_ will appear after you run your program. Do not worry about it, but don't delete it!

As you can see, both **file paths** point to the same location/folder: /Users/emilyzhao/Documents/my_code/

basic_arithmetic.py - /Users/emilyzhao/Documents/my_code/basic_arithmetic.py (3.10.7)

```python
# Basic Arithmetic Module
# Written by Emily

def add(a, b):
    '''
    input: takes in two integers
    processing: adds a and b together
    output: returns the sum
    '''
    c = a + b
    return c

def subtract(a, b):
    '''
    input: takes in two integers
    processing: subtracts b from a
    output: returns the difference
    '''
    c = a - b
    return c
```

my_program.py - /Users/emilyzhao/Documents/my_code/my_program.py (3.10.7)

```python
# My Program

# That uses Basic Arithmetic Module
import basic_arithmetic_module as basic_math

# dot call of function from module
# [name of module].[function in module]

sum1 = basic_math.add(1, 2) # returns 3
sum2 = basic_math.add(sum1, 7) # returns 10

print(sum2) # prints 10
```

How to write and use your own module

1. Create a new folder
2. Write your module and make sure you save it inside the folder you just created
3. All subsequent programs you write that use your module should also exist in that same folder
    a. You can use your file explorer to check to see if/place your files in the same folder
    b. You can also check the file paths at the top of your program to see if they exist in the same folder

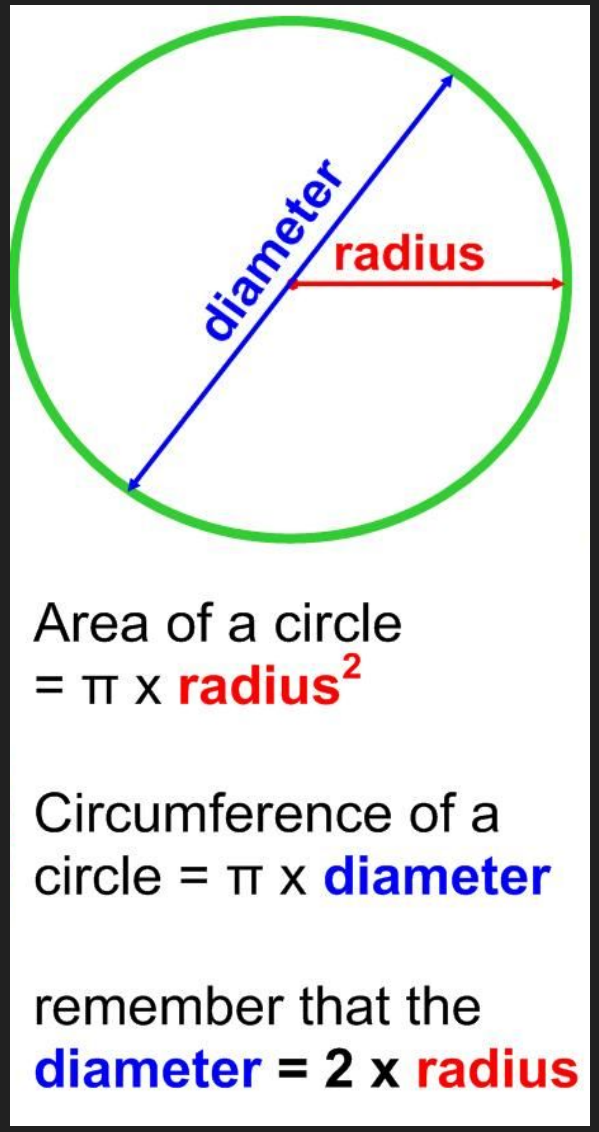
Programming Challenge

Create a module called “geometry_helper”

Write two functions in this module:

- Area of circle
- Perimeter of circle

Each of these functions will accept one argument (a radius) and will print out the result to the user.



week06_challenge

Name

- __pycache__
- circle_functions.py
- my_program.py

circle_functions.py - /Users/emilyzhao/Documents/week06_challenge/circle_functions.py (3.10.7)

```python
PI = 3.141592

def getArea(r):
    '''
    input: radius
    processing: calculates the area of a circle
    output: returns the area
    '''
    return PI * (r ** 2)

def getPerimeter(r):
    '''
    input: radius
    processing: calculates the area of a circle
    output: returns the area
    '''
    return 2 * PI * r
```

my_program.py - /Users/emilyzhao/Documents/week06_challenge/my_program.py (3.10.7)

```python
import circle_functions as c

radius = 5
area = c.getArea(radius)
perimeter = c.getPerimeter(radius)

print("Area:", area)
print("Perimeter:", perimeter)
```

Mock Midterm

Midterm Survey

pollev.com/emilyzhao

Topics Covered (Modules 1-6):

Basic Programming Mechanics

- Functions
  - What is a function?
  - How to call a function
  - Arguments
  - Return Values
- Commenting your code
- Variables
  - What is a variable?
  - Creating variables
  - Using variables in expressions
  - Naming rules
- Reading input from the keyboard with the input() function

Math Expressions

- Math operators (+, -, /, //, *)
- Writing math expressions
- Evaluating math expressions
- Storing & printing the results of math expressions
- Difference between the two division operators (/ and //)
- Order of operations in math expressions
- The exponent operator (**)
- The modulo operator (%)

Data Types

- What is a data type?
- Strings
- Numeric data types
  - Integers (int)
  - Floating point numbers (float)
- Mixed type expressions
- Data type conversion
  - Using the float() and int() function to convert strings into numbers
  - User input & data types (converting strings to floats / ints for calculation purposes)
- The Boolean data type
- Boolean variables

Output with the print() function

- General use of the print function and its default behavior
  - Unlimited arguments
  - Spaces inserted between arguments
  - Line break after each call to the function
- Customizing line endings (end=”)
- Customizing argument separators (sep=”)
- Escape characters (\n, \t, etc.)

Basic String Manipulation

- Combining two strings (concatenation) – "+" operator
- Multiplying a string (repetition) – "*" operator
- Formatting numbers using the format() function
  - Formatting Strings – width, left align, right align, center align
  - Formatting Integers – width, left align, right align, center align
  - Formatting Floats – width, left align, right align, center align, # of decimal places, "," separator
- Case manipulation using str.lower() and str.upper()
- Calculating string length using the len() function

Selection Statements

- The structure of an IF statement (IF keyword, condition, colon, indentation)
- Writing a condition for an IF statement
- Boolean operators (<, >, ==, !=, >=, <=)
- Comparing numeric values using Boolean expressions
- Comparing string values using Boolean expressions
- Using the IF-ELSE statement
- Nesting decision structures (IF statements inside other IF statements)
- The IF-ELIF-ELSE statement
- Logical operators (and, or, not)

Condition Controlled Loops

- The structure of a "while" loop
- Mechanics & how they work
- Setting up conditions for a while loop
- Infinite loops and how to work with them
- Sentinels (defining a value that the user enters that causes the loop to end)
- Input validation loops (asking the user to continually enter a value until that value matches some condition)
- Setting up and using accumulator variables
- Self referential assignment statements (i.e. counter = counter + 1)
- Augmented assignment operators (i.e. counter += 1)

The Range Function

- mechanics and how the function works
- creating simple ranges (i.e. range(5))
- creating ranges with defined start and end points (i.e. range(3,10))
- creating ranges with a step value (i.e. range(5,50,5))
- creating ranges that count backwards (i.e. range(50,5,-5))
- user controlled ranges (i.e. range(1, somevariable))

Functions

- mechanics and how functions work
- function definitions
- arguments
- return values
- calling a function
- local variables (variables that are defined inside a function and can only be accessed inside that function)
- passing arguments to your own functions
- passing multiple arguments to your own functions
- global variables (variables created outside a function that can be accessed by any part of your program)
- making changes to global variables inside a function using the ‘global’ keyword
- writing a value returning function (i.e. using the ‘return’ keyword to send a result from your function to the part of your program that called that function)
- returning multiple values from a function
- Input, Processing & Output notation

Miscellaneous Concepts

- Generating random numbers
- Errors & error types
- Debugging strategies
- Pseudocoding

Homework

Have a good Spring Break!

Thursday, March 23

- Midterm Exam
- Assignment #6 Due