CSCI-UA-0002

Intro to Computer Programming (No Prior Experience)

Module 6: Functions pt. 2

Professor Emily Zhao

Section 008	Section 012
T/R 12:30-1:45PM	T/R 4:55-6:10PM

Agenda

- Continue Functions
- Global vs Local variables
- How to make a module
- Midterm Topics Survey
- Midterm Prep Quiz
- Midterm Practice Exam

Your Questions

→ What are ***arguments*** and ***parameters***?
    → What does it mean to ***pass a value***?

→ What is `return`? How is it different from `break`? How does it differ from `print()`?

→ What is a ***global*** variable?
    → When do I use it?
    → How does it differ from a ***local*** variable?

→ Quiz questions

Functions

You can think of functions like verbs!

1) They DO things
2) They RETURN things *

The `print()` function

What it does: prints objects to the shell
What is returns: nothing

The `input()` function

What it does: asks the user for input with prompt
What is returns: the user input as a string *

*** If your function returns a value, you must store the value!**

```python
# Definining function
def add(a, b):          # a and b are "parameters"
    c = a + b           # DO: add two nums together
    return c            # RETURN: the sum of the nums

# Calling the function
result = add(3, 5)  # 3 and 5 are "arguments"
                    # they are "passed" to the function
                    # since add returns a value,
                    # we must store it in a variable
```

```
def translateRight(x, y):
    x += 1
    return x, y

x_coord, y_coord = translateRight(1, 5)

# x_coord -> 2
# y_coord -> 5
```

When you run a function, you usually want to capture the output.

You can do that by assigning what is returned to a variable, or multiple variables.

Passing Arguments to a Function

Passing Arguments to a Function

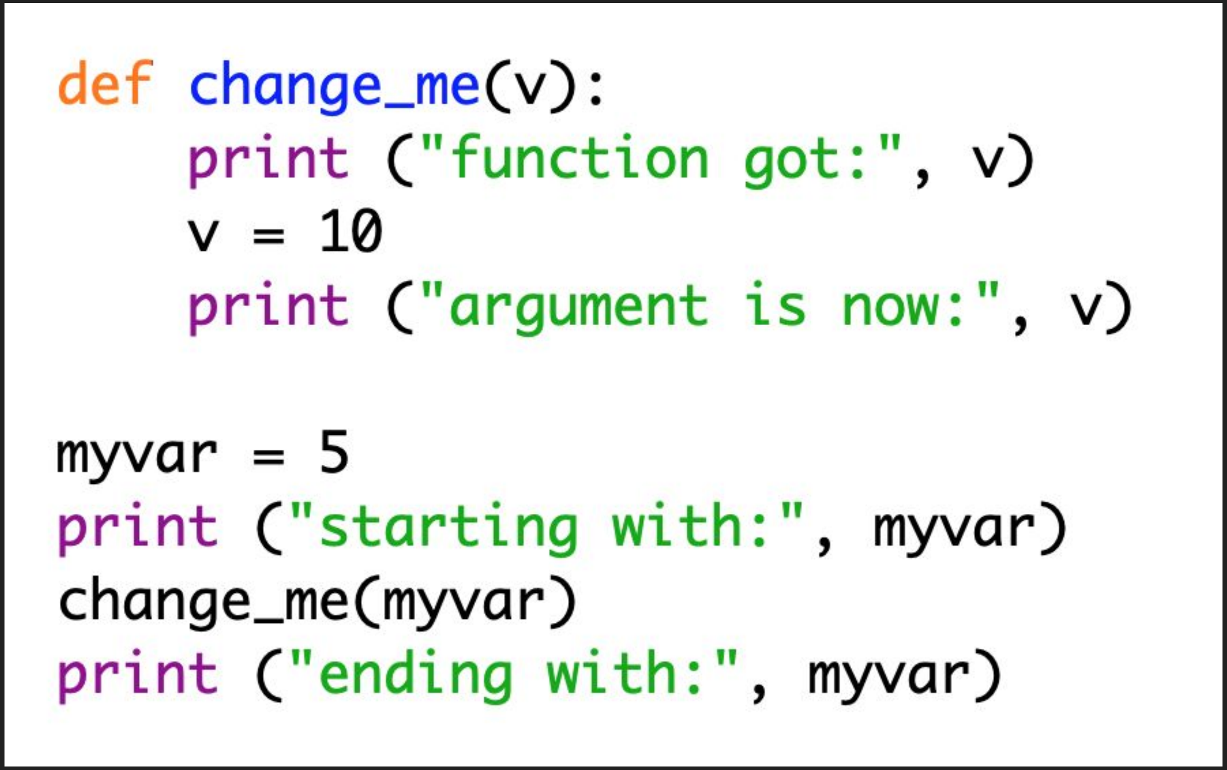
- Sometimes it's useful to not only call a function but also send it one or more pieces of data as an argument

- This process is identical to what you've been doing with the built-in functions we have studied so far

```
x = random.randint(1,5)    # send 2 integers
y = len('Emily')           # send 1 string
```

Argument Mechanics

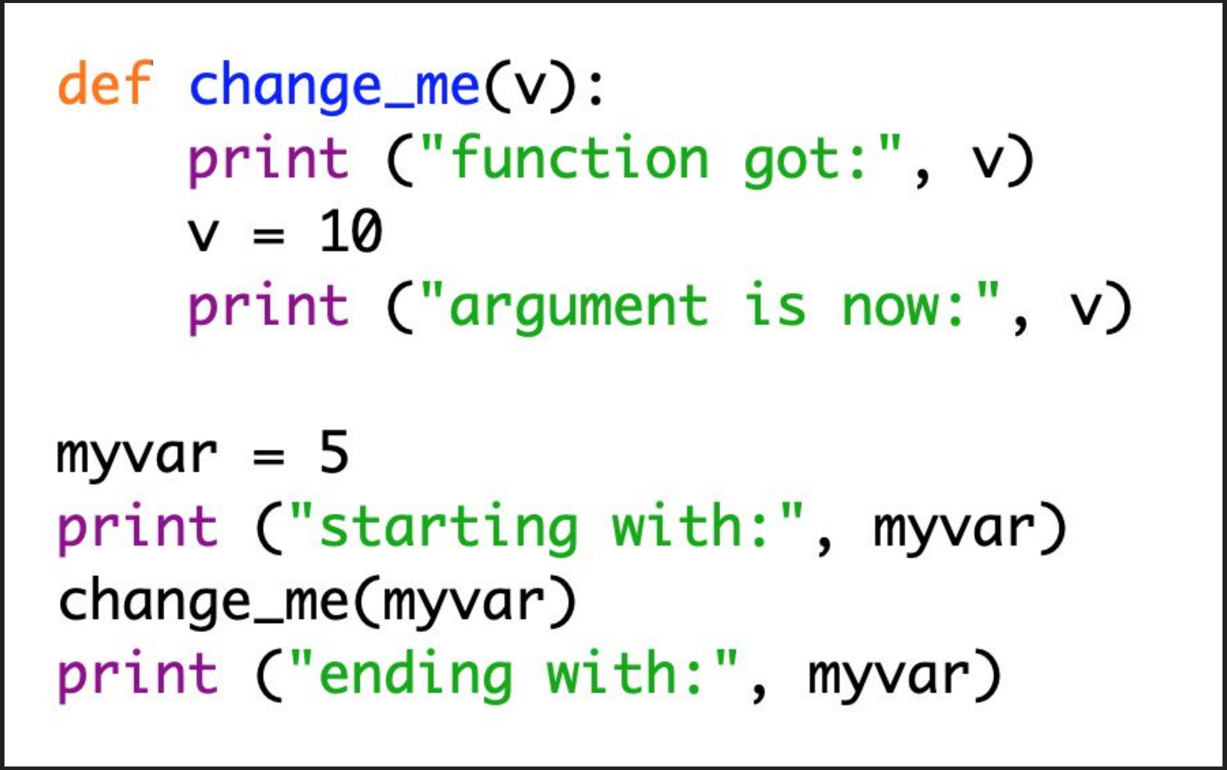
When we pass an argument to a function in Python we are actually passing it’s “value” into the function, and not an actual variable



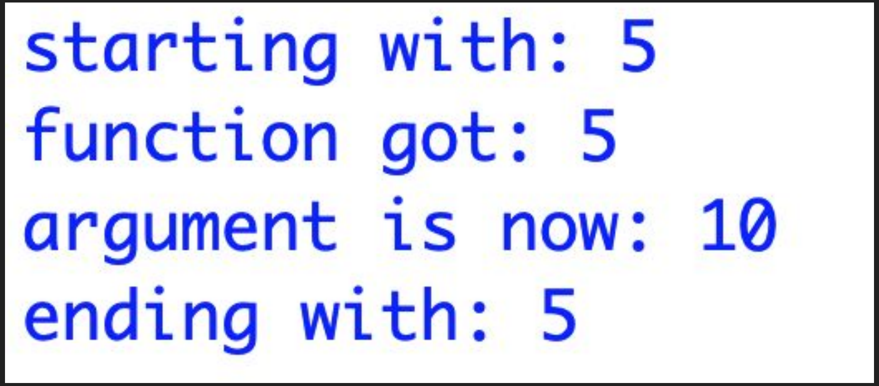
```
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
```

Argument Mechanics

When we pass an argument to a function in Python we are actually passing it’s “value” into the function, and not an actual variable

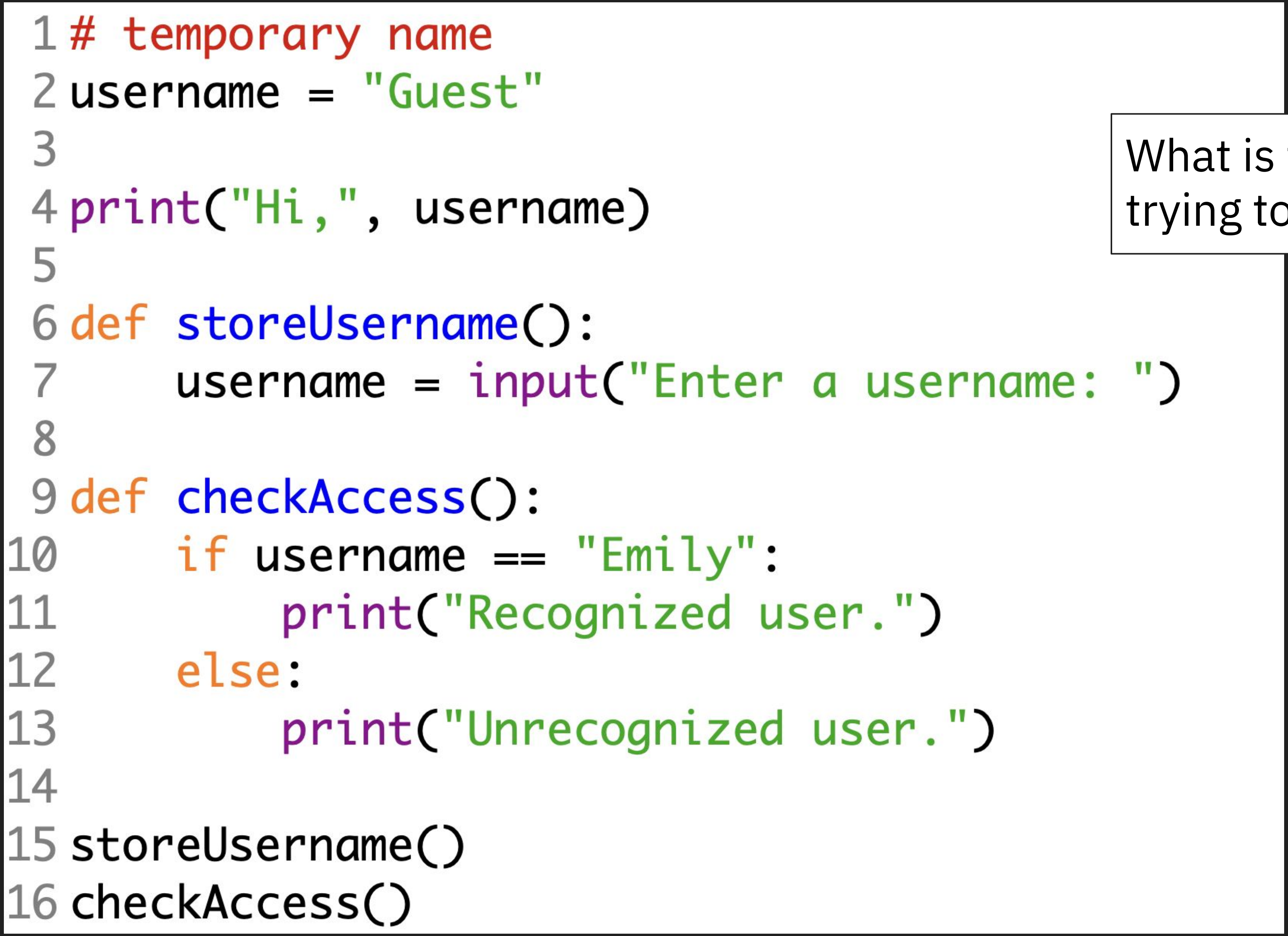


```
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
```



```
starting with: 5
function got: 5
argument is now: 10
ending with: 5
```

Local vs Global Variables

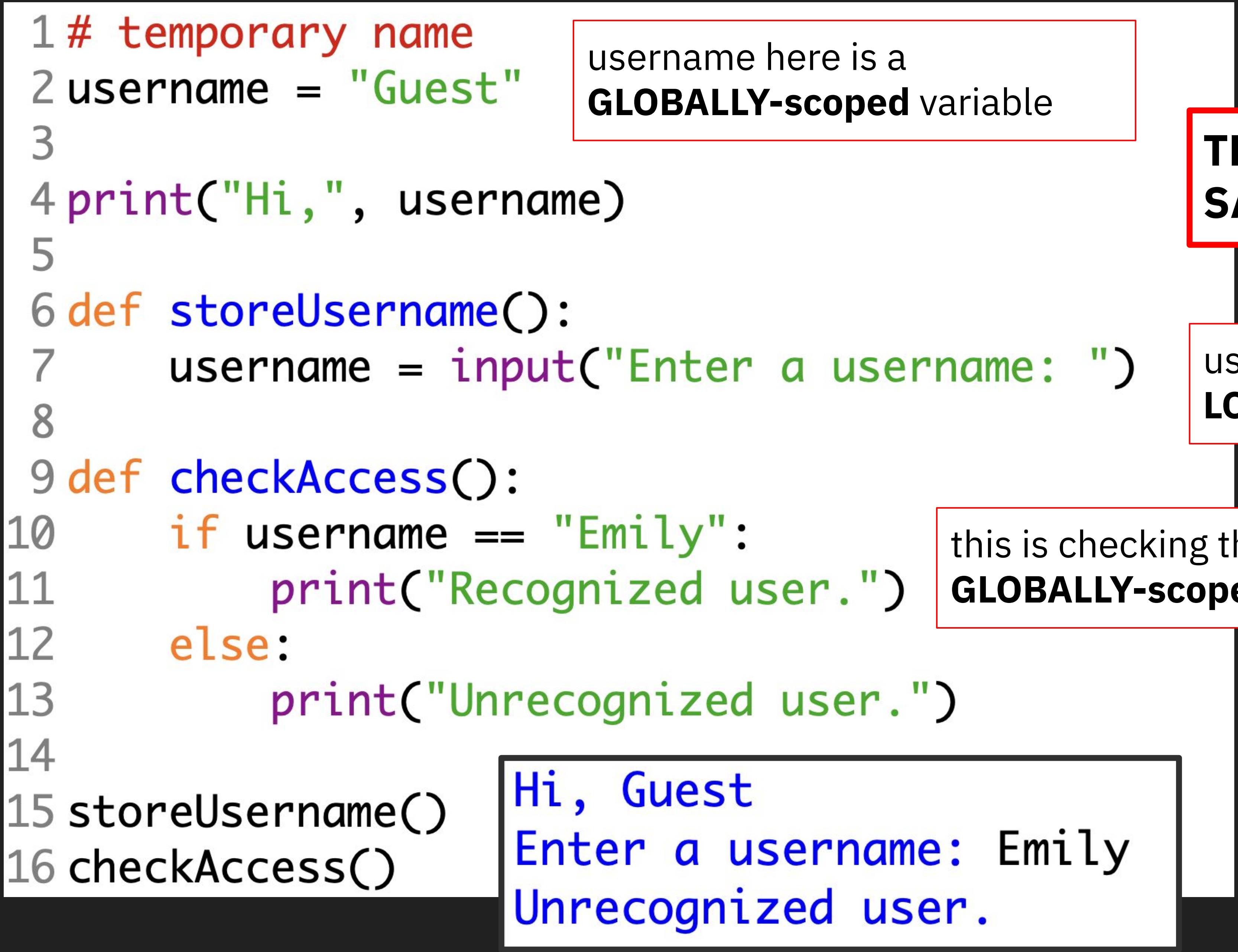


```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username: ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

What is this code trying to accomplish?

```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username: ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

```
Hi, Guest
Enter a username: Emily
Unrecognized user.
```



```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username:  ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

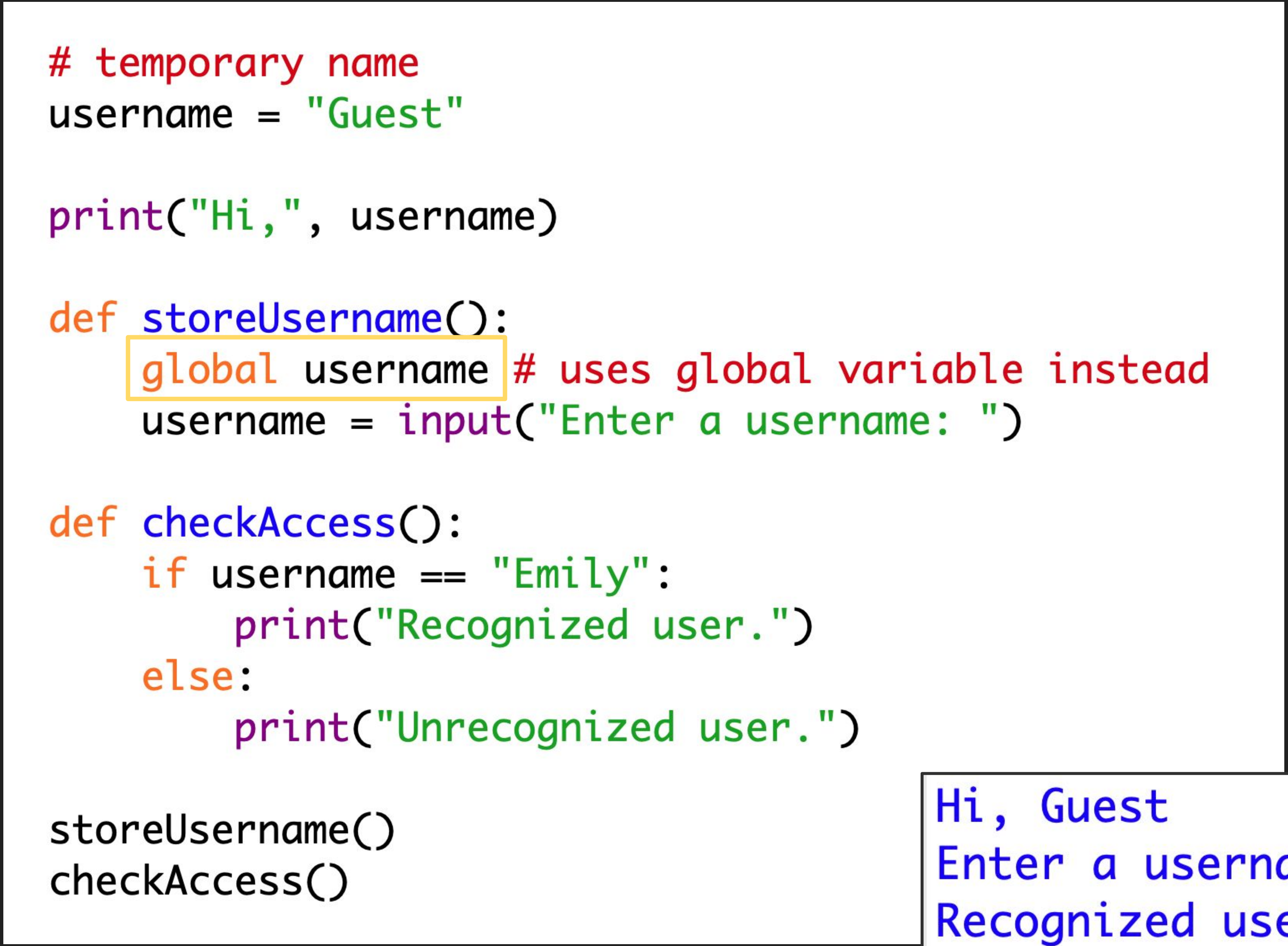
```
Hi, Guest
Enter a username: Emily
Unrecognized user.
```

username here is a **GLOBALLY-scoped** variable

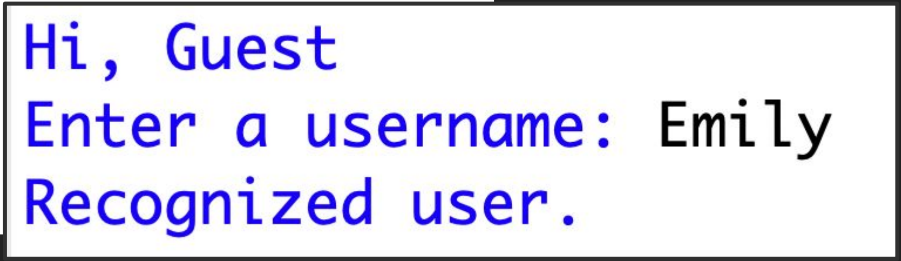
THEY ARE NOT THE SAME!

username here is a **LOCALLY-scoped** variable

this is checking the **GLOBALLY-scoped** variable



```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    global username # uses global variable instead
    username = input("Enter a username: ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

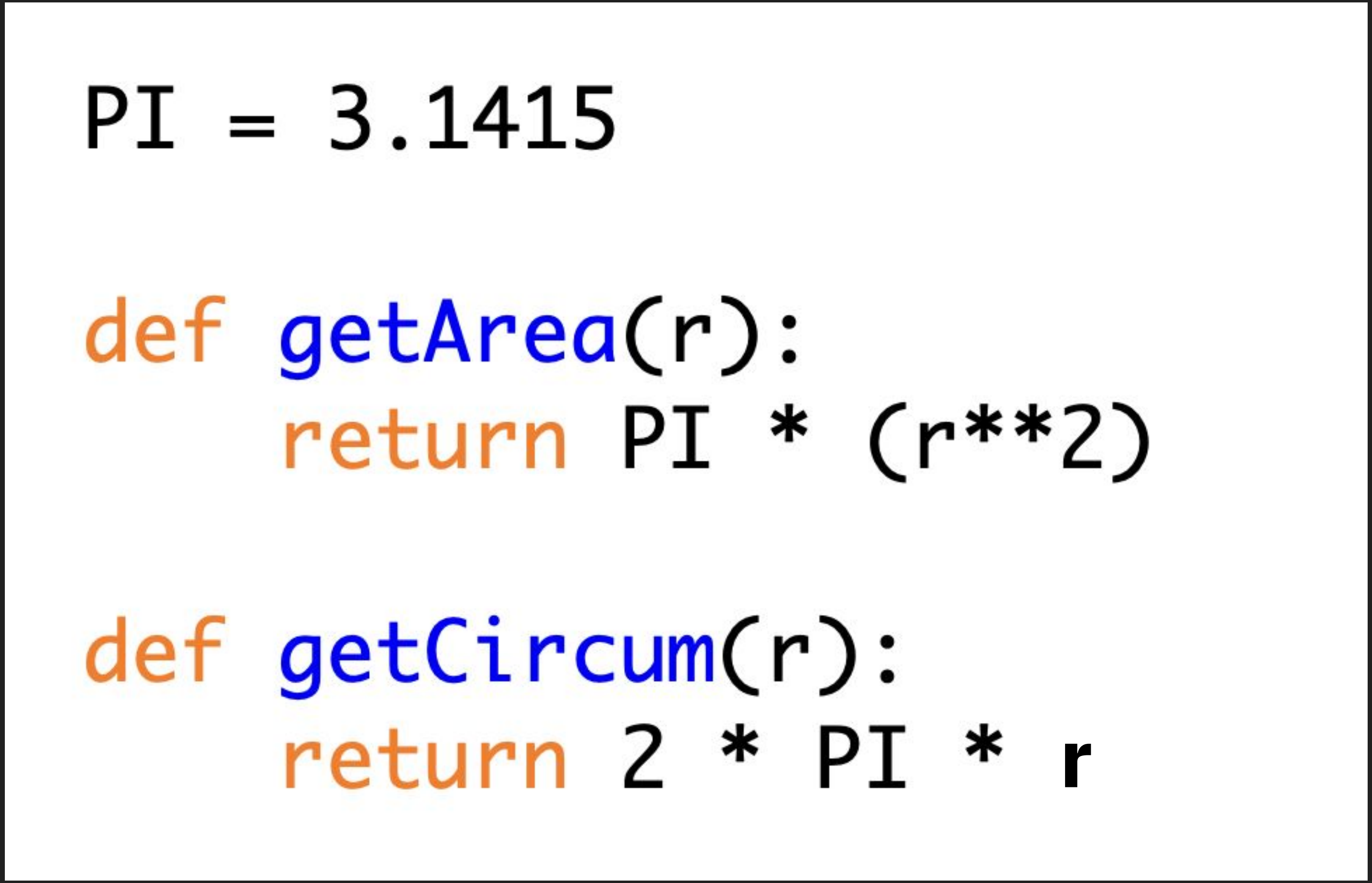


```
Hi, Guest
Enter a username: Emily
Recognized user.
```

Local vs Global Variables: Best Practices

- Keep variables as local as possible
- It makes your code more readable and easier to debug
- The best use cases for global variables are constants (variables that rarely change but appear in multiple functions)

Local vs Global Variables: Best Practices

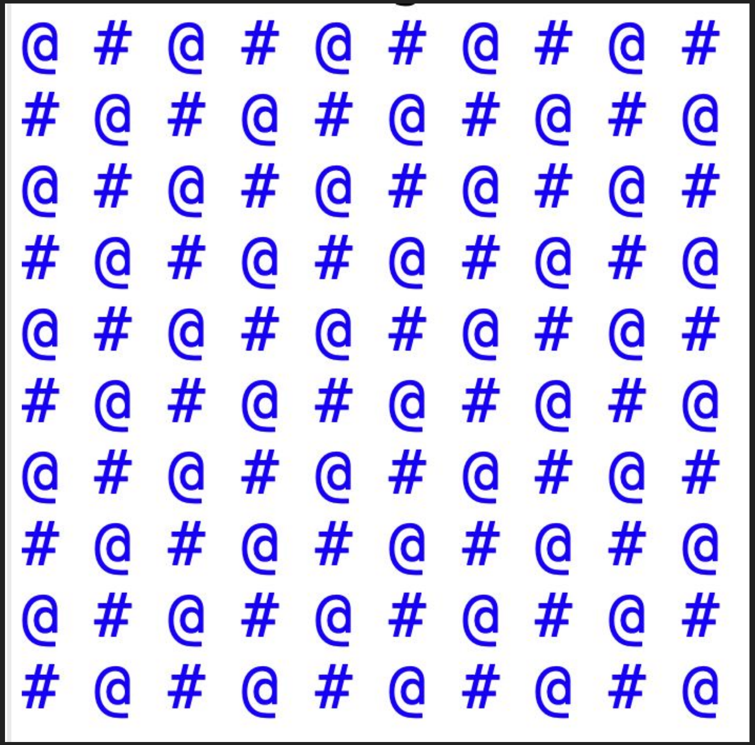


```
PI = 3.1415

def getArea(r):
    return PI * (r**2)

def getCircum(r):
    return 2 * PI * r
```

Value Returning Functions

```python
def sayHi(name):
    # sends back a string
    return "Hi, " + name + "!"

sayHi("Emily") # returns "Hi, Emily!" but isn't used
print(sayHi("Emily")) # same thing as saying print("Hi, Emily!")
```

Programming Challenge



```
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
```

Convert our earlier checkerboard code into a function that accepts three parameters – grid size, first character, second character

Solution 1: the function returns nothing and just prints to Shell

```python
def printCheckerboard(rows, cols, symbol1, symbol2):
    '''
    input: rows, cols, symbol1, symbol2
    processing: prints a custom checkerboard
    output: NOTHING
    '''
    for r in range(rows):
        for c in range(cols):
            # if the sum of the row # and column #
            # is even, then draw @
            # else draw #
            if (r+c) % 2 == 0:
                print(symbol1, end=" ")
            else:
                print(symbol2, end=" ")
        print() # make new line at the end of the row

printCheckerboard(5, 5, "$", "*")
```

Solution 2: the function returns an output string

```python
def makeCheckerboard(rows, cols, symbol1, symbol2):
    '''
    input: rows, cols, symbol1, symbol2
    processing: creates a checkerboard pattern
    output: a string of the checkerboard pattern
    '''
    output = ""
    for r in range(rows):
        for c in range(cols):
            if (r+c) % 2 == 0:
                # instead of printing, we concatenate
                # print(symbol1, end=" ")
                output += symbol1 + " "
            else:
                #print(symbol2, end=" ")
                output += symbol2 + " "
        # instead of printing new line
        # we add \n to string
        # print()
        output += "\n"

    return output

print(makeCheckerboard(5, 5, "$", "*")) # print what's returned
```

I prefer this solution!

I like it when functions return things :)

It also runs faster!

Modules

- All programming languages come pre-packaged with a standard library of functions that are designed to make your job as a programmer easier

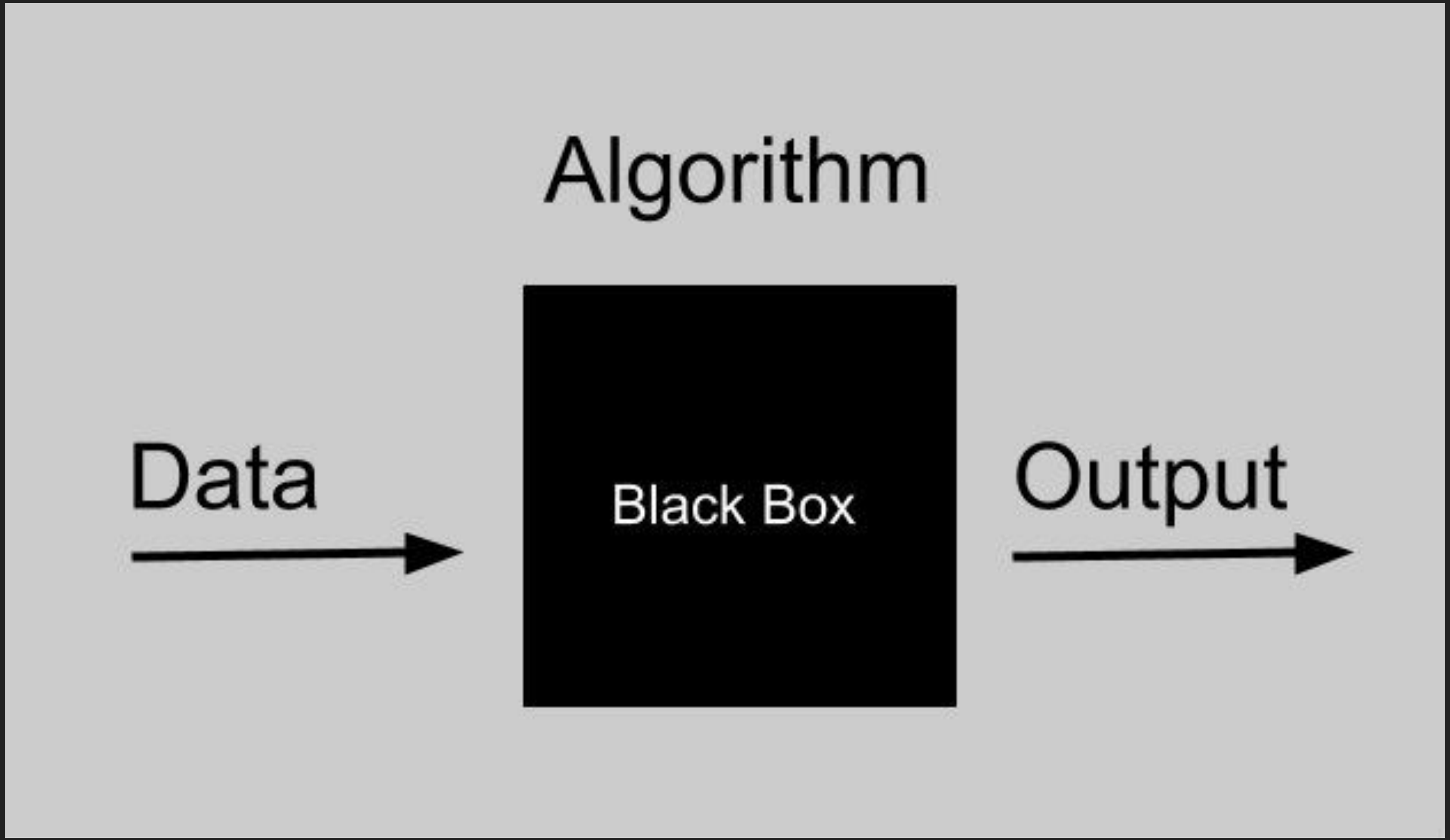
- Some of these functions are built right into the "core" of Python (print, input, range, etc)

- 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`

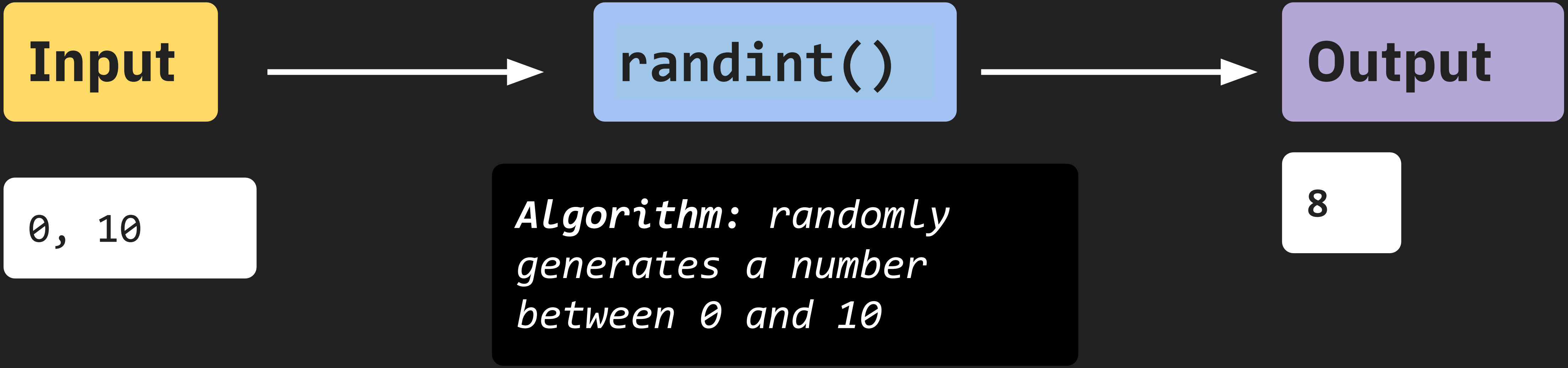
Modules

- 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

“Black Box” model



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



```
0, 10
```

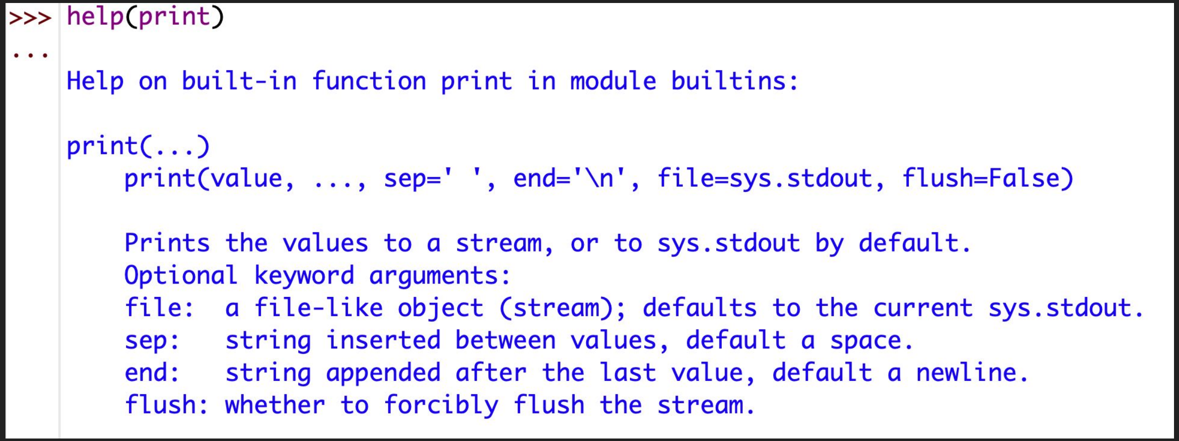
Algorithm: *randomly generates a number between 0 and 10*

```
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

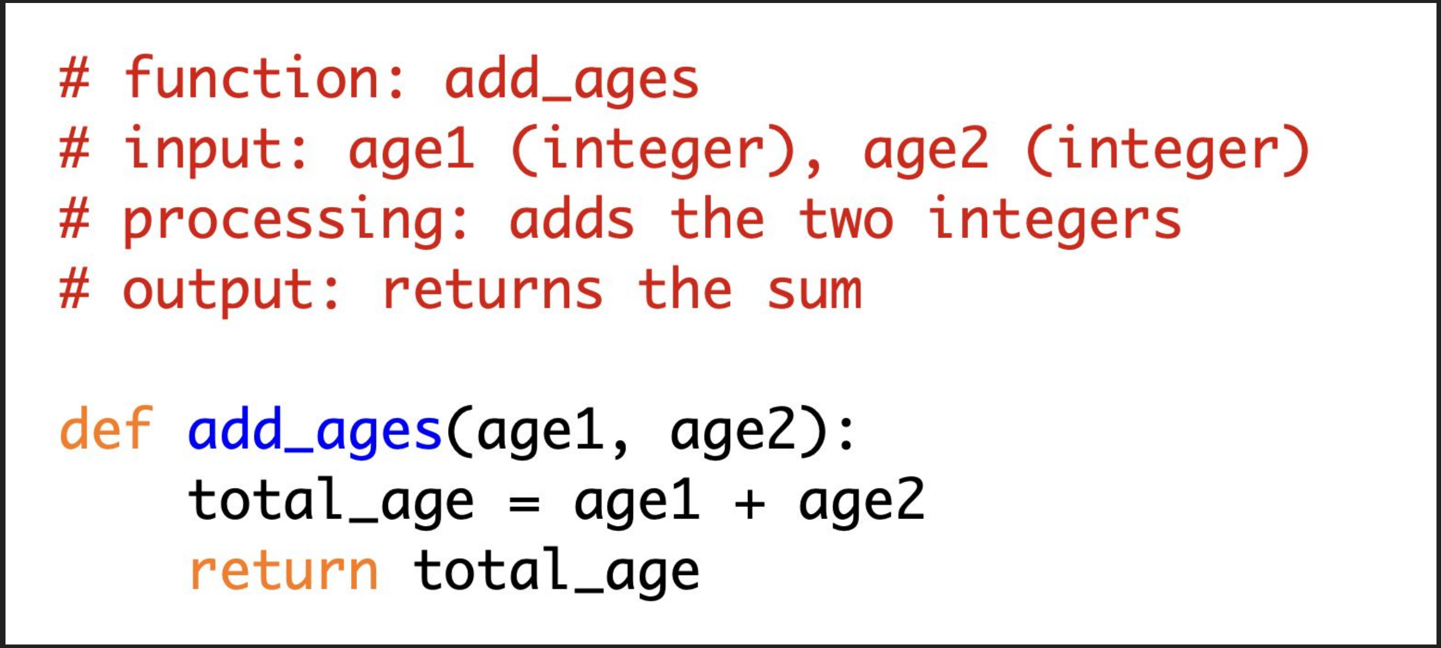


```
>>> help(print)
...
    Help on built-in function print in module builtins:

    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.
```

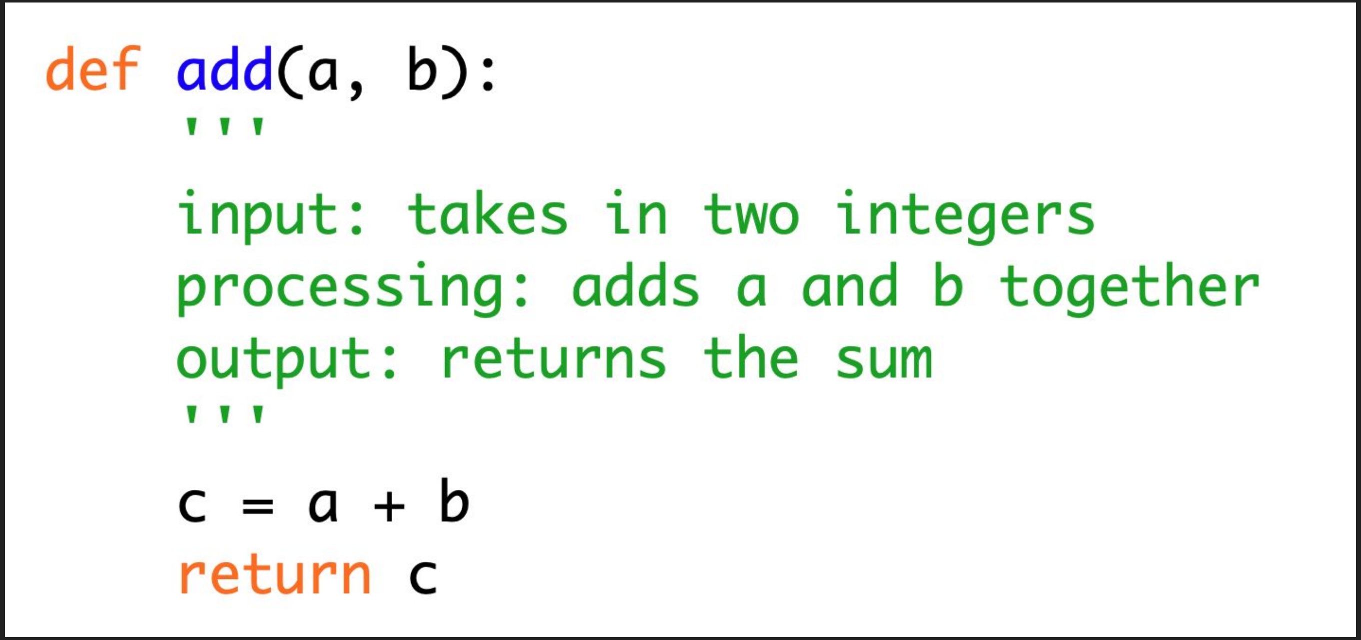
IPO Notation

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

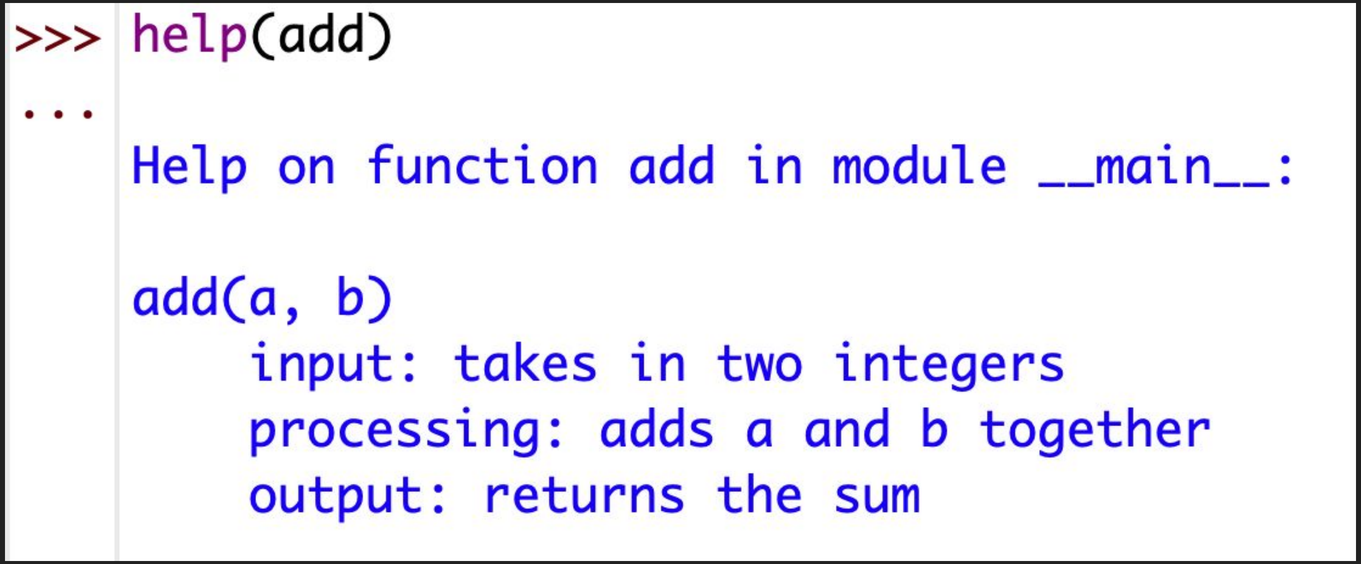
- Example:



```python
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing: adds the two integers
# output: returns the sum

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

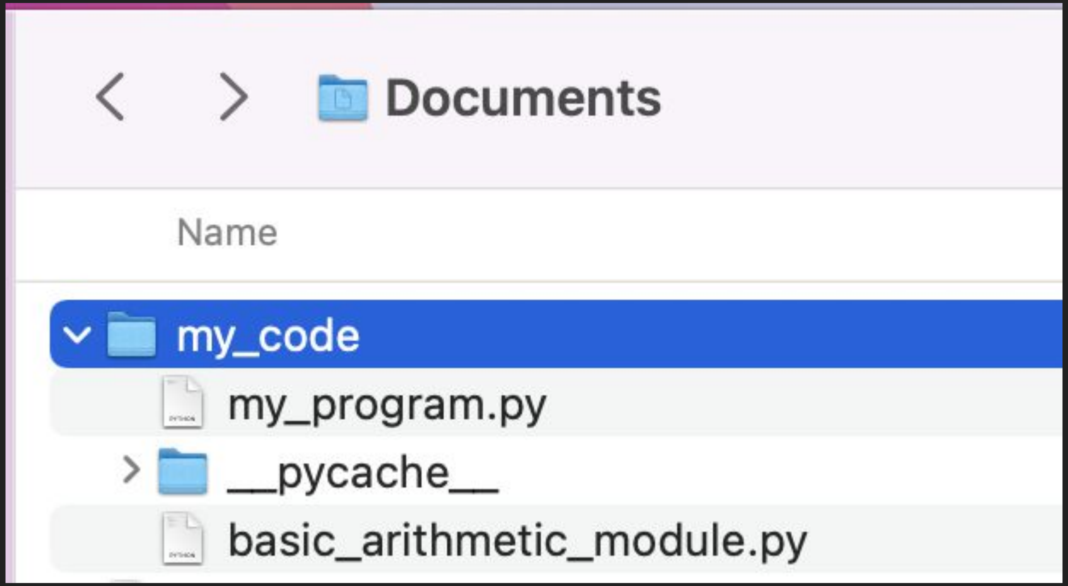


```python
def add(a, b):
    '''
    input: takes in two integers
    processing: adds a and b together
    output: returns the sum
    '''
    c = a + b
    return c
```



```
>>> 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

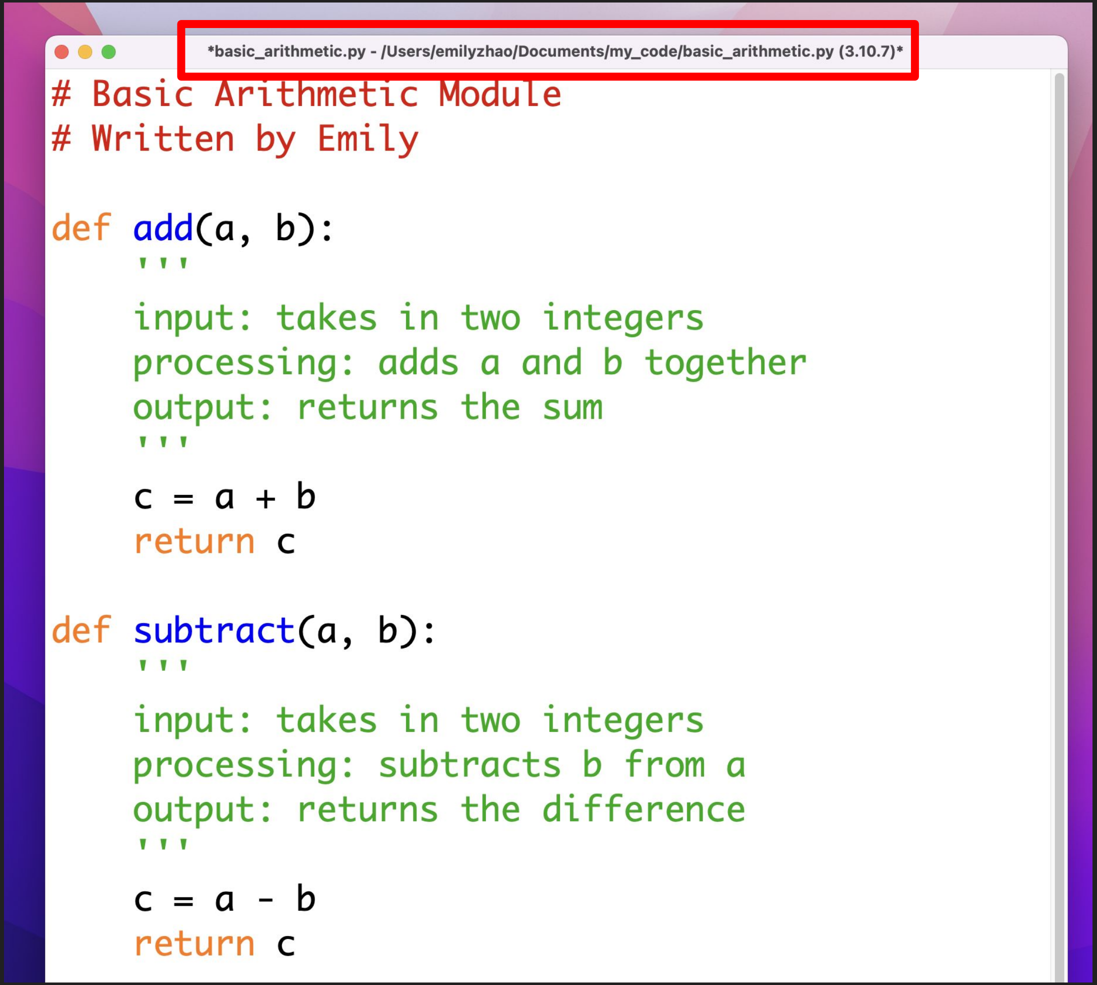


Your `module.py` and your `program.py` **need to be in the same folder!**

— I have placed them both in a folder called my_code

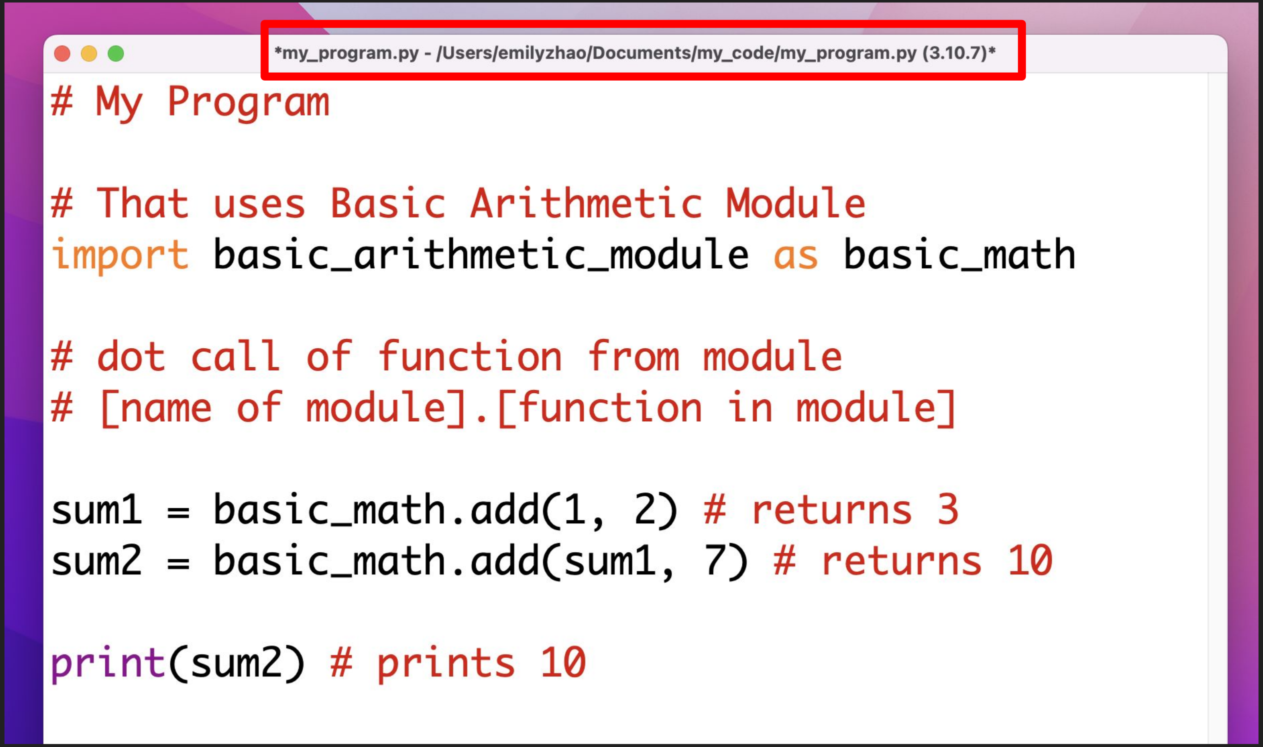
Note: a folder called _pycache_ will appear after you run your program. Do not worry about it, but don't delete it!

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

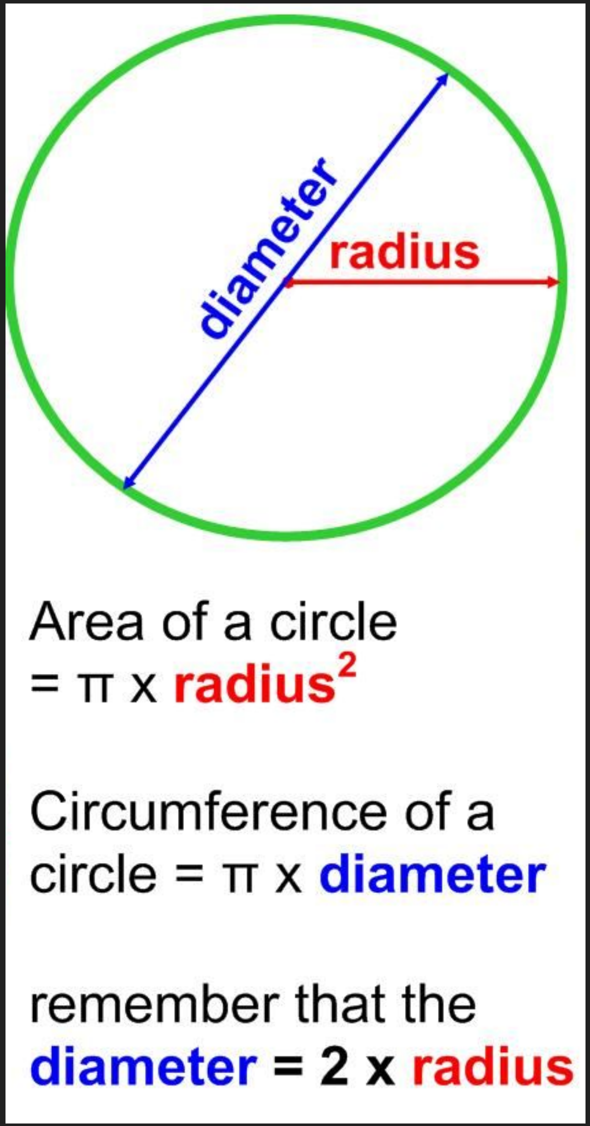
Programming Challenge

Create a module called “geometry_helper”

Write two functions in this module:

- Area of circle
- Perimeter of circle

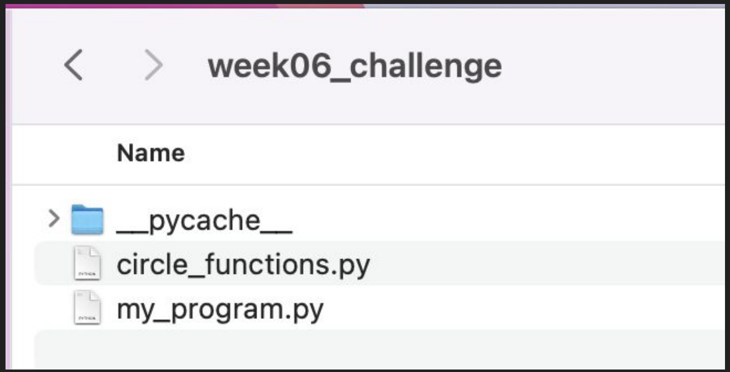
Each of these functions will accept one argument (a radius) and will print out the result to the user.



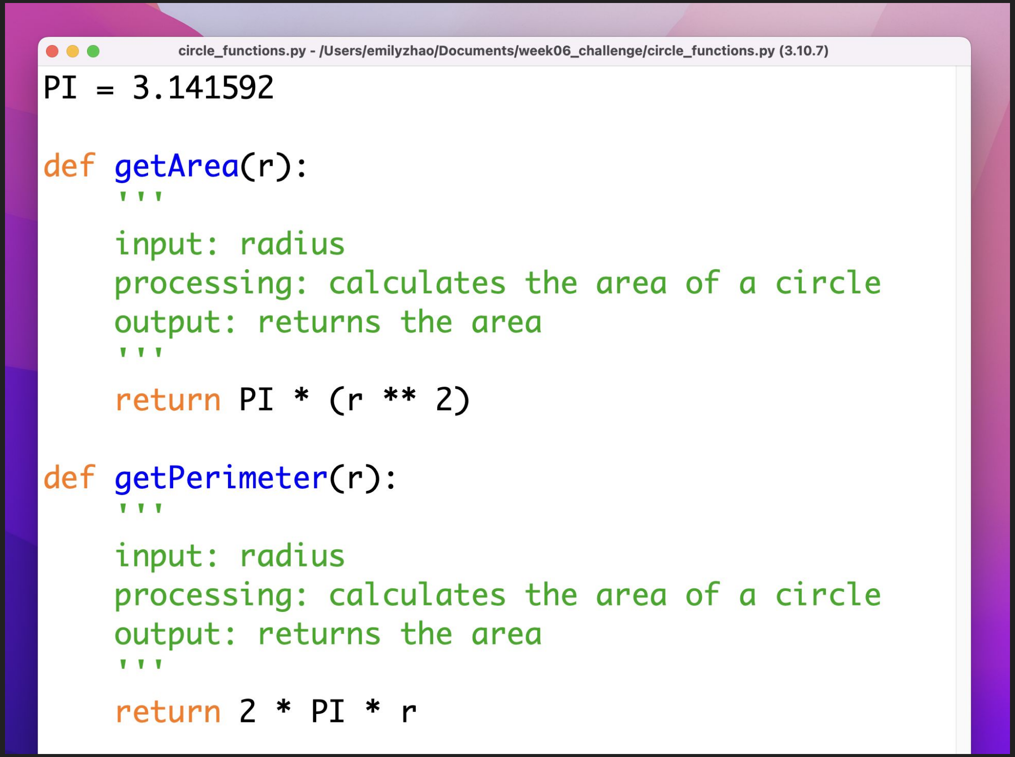
Area of a circle $= \pi \times \text{radius}^2$

Circumference of a circle $= \pi \times \text{diameter}$

remember that the **diameter = 2 x radius**

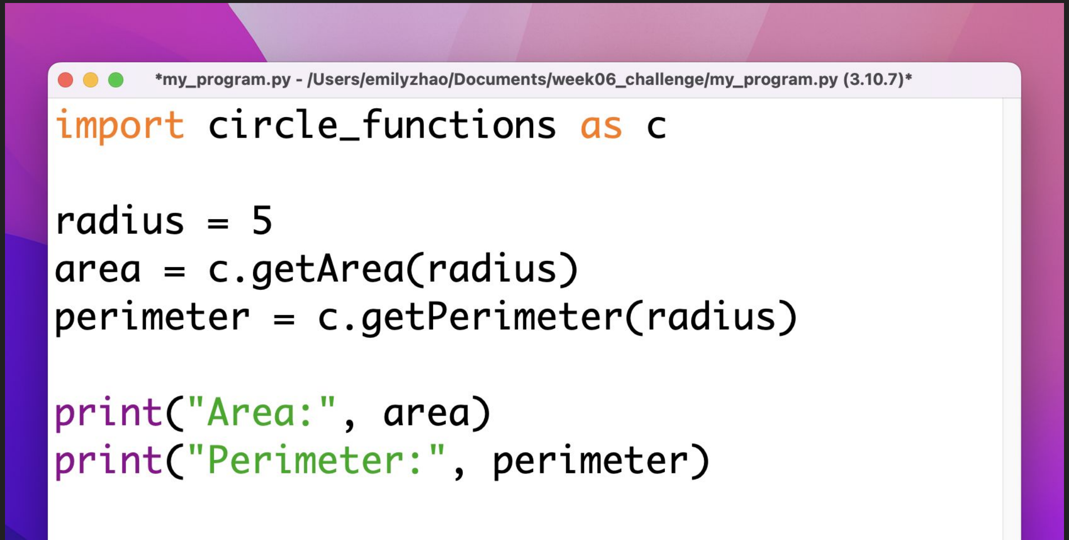


```
__pycache__
circle_functions.py
my_program.py
```



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)
```

Midterms Topics Survey

Midterm Topics

pollev.com/emilyzhao

Midterm Prep Quiz

Homework

- TA Review Session tomorrow (Friday)
- Midterm Prep Quiz due tomorrow (Friday)
- Study for Midterm (next Thursday)
- Assignment #6 (due next Thursday)