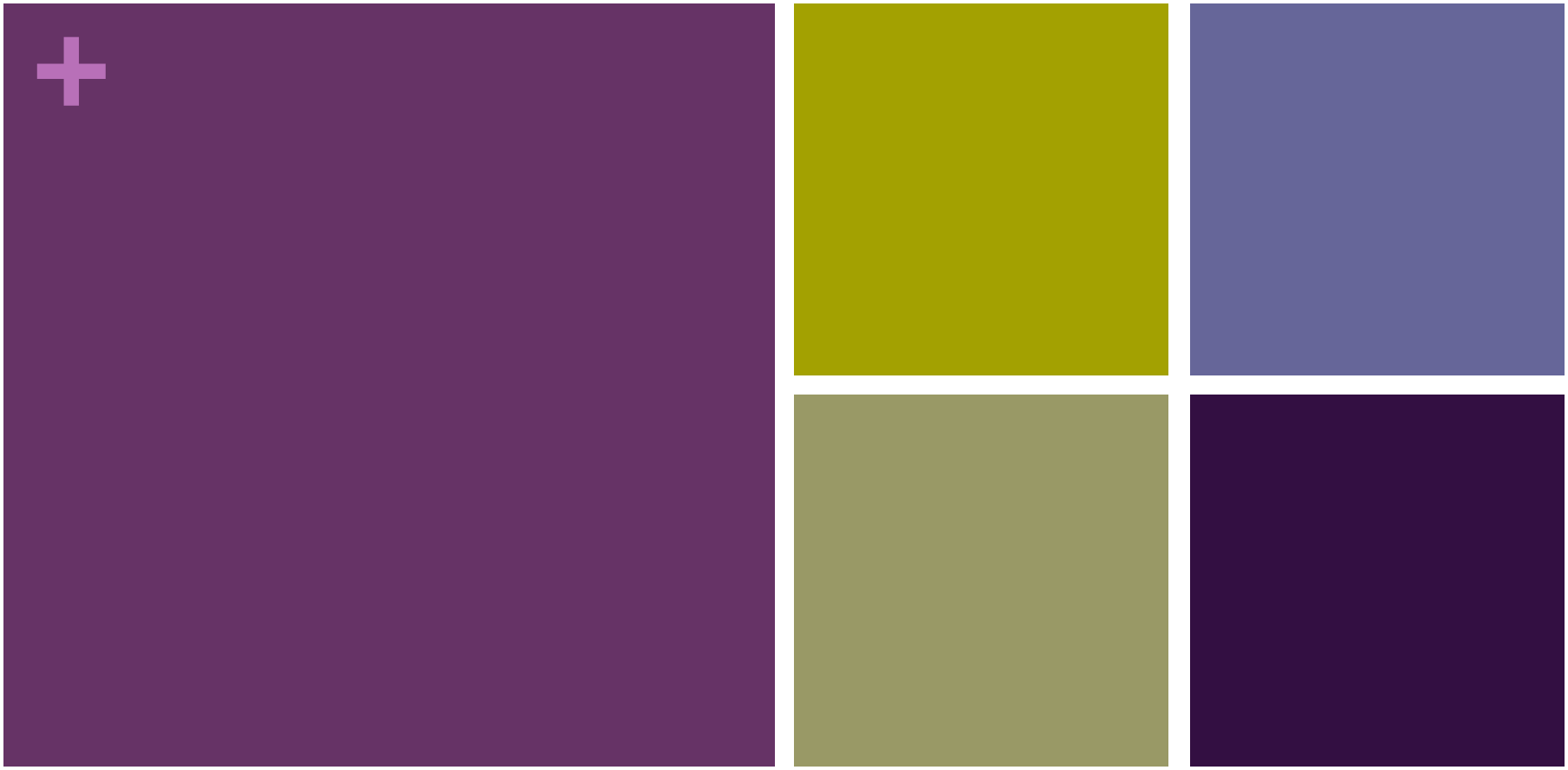


Working with Functions Part 2 (continued)

Introduction to Programming - Python

print() vs return

print() vs return

print()

- displays a value(s) on the screen

```
def lowercase(word):
    print( word.lower() )

lowercase("HAPPY")
```

return

- sends a value(s) back to the part of the program that called the function

```
def lowercase(word):
    return word.lower()

result = lowercase("HAPPY")
print(result)
```

Modules

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

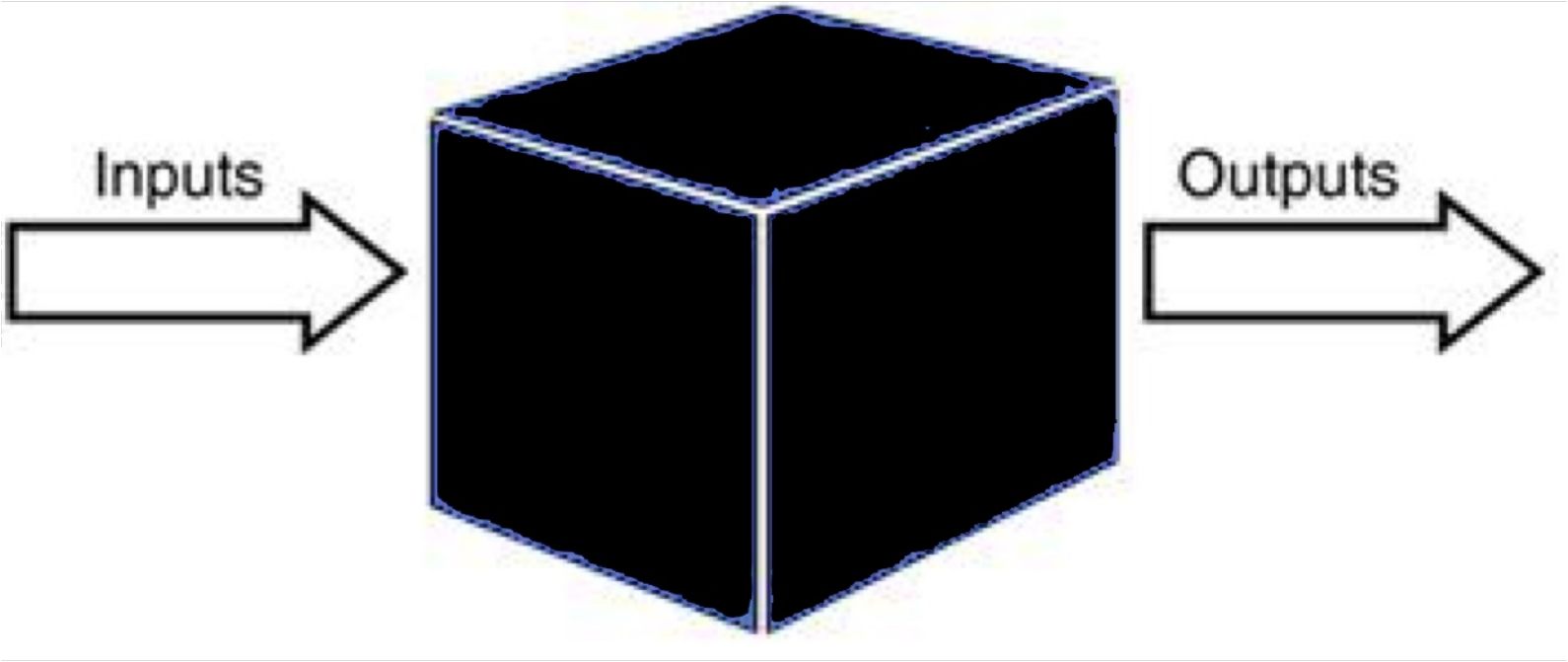
Modules

- On a Mac you can actually see these files here:
  - /Library/Frameworks/Python.framework/Versions/3.2/lib/python3.2/
- To see information about a module, you can do the following in IDLE:
  - import modulename
  - help(modulename)

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



Functions in modules

- We call functions that exist within a module by using "dot notation" to tell Python to run a function that exists in that module
- Example:
  - num = random.randint(1,5)

Listing functions in a module

- You can list the functions that exist in a particular module by using the help() function
- The help() function takes one argument (a string that represents the name of the module) and returns the user manual for that module

Creating your own modules

- You can easily create your own modules that you can populate with your own functions. Here's how:
  - Create a new python script (i.e. "myfunctions.py")
  - Place your function definitions in this script
  - Create a second python script (i..e "myprogram.py")
  - Import your function module using the import statement:

```
import myfunctions
```

  - Call your functions using dot notation

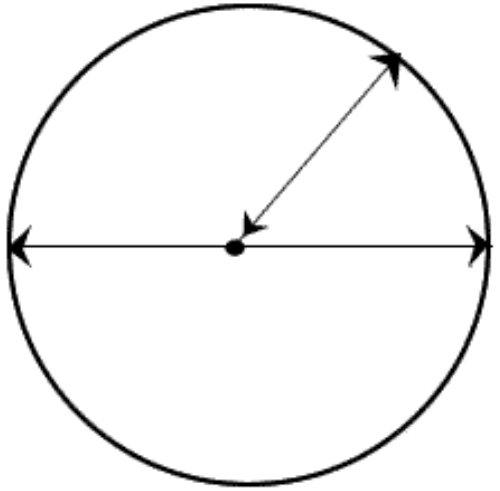
```
myfunctions.function1()
myfunctions.dosomethingelse()
```

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.

Circle

Perimeter $C = \pi d$ or $C = 2\pi r$



Area $A = \pi r^2$

Note: The value of π is 3.1415926 (to 7 decimal places)

Some additional functions inside the random module

- Floating point random #'s
  - `num = random.random()  # generates a float`
    `# between 0 and 1`
  - `num = random.uniform(1,10)  # generates a float`
    `# between 1 and 10`

Seeding the random number generator

- As we mentioned in an earlier class, the computer does not have the ability to generate a truly random #
- It uses a series of complicated mathematical formulas that are based on a known value (usually the system clock)
- The seed function in the random module allows you to specify the "seed" value for the random #'s that the various random number functions generate
- You can seed the random # generator with whatever value you want – and you can even reproduce a random range this way!

+

Worksheet