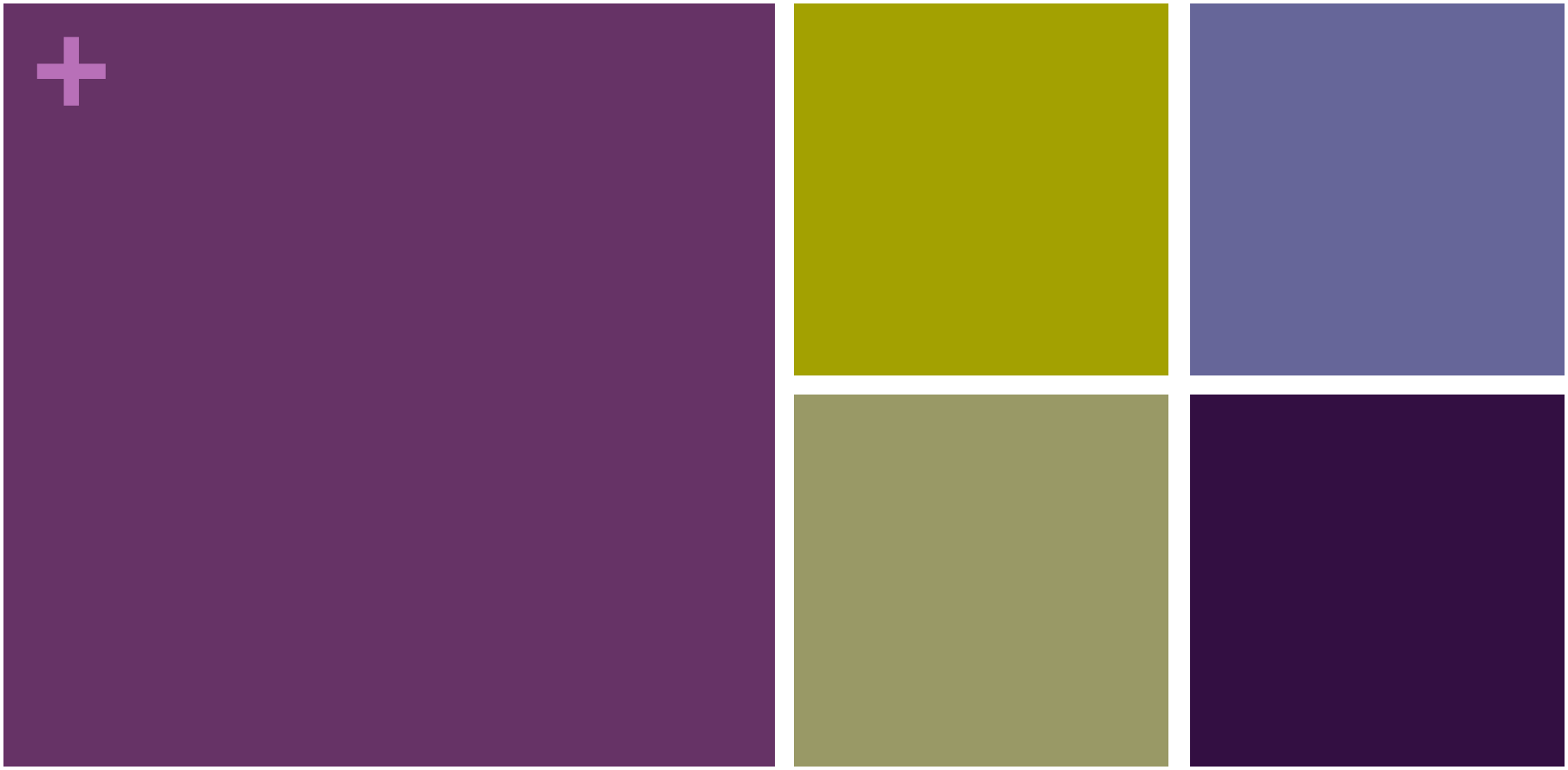
Warm Up

www.pollev.com/python002



Working with Functions in Python

Introduction to Programming - Python

+

Functions

Functions

- A function is a group of statements that exist within a program for the purpose of performing a specific task
- Since the beginning of the semester we have been using a number of Python's built-in functions, including:
  - print()
  - range()
  - len()
  - random.randint()
  - … etc

3 reasons to use functions

Organize your code

Reuse your code

Collaborate with others

Defining Functions

- Functions, like variables must be named and created before you can use them
- The same naming rules apply for both variables and functions
  - You can't use any of Python's keywords
  - No spaces
  - The first character must be A-Z or a-z or the "_" character
  - After the first character you can use A-Z, a-z, "_" or 0-9
  - Uppercase and lowercase characters are distinct

Defining functions

```
def myfunction():
  print ("Printed from inside a function")


# call the function
myfunction()
```

Flow of Execution – With Functions

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

Flow of Execution with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
And now we're done.
```

+

Multiple Functions

Multiple functions

```
def hello():
  print (“Hello there!”)


def goodbye():
  print (“See ya!”)


hello()
goodbye()
```

Calling functions inside functions

```
def main():
  print (“I have a message for you.”)
  message()
  print (“Goodbye!”)
def message():
  print (“The password is ‘foo’”)


main()
```

Programming Challenge: Hollow Square

- Write a program that prints the pattern to the right using functions

Programming Challenge: User Controlled Hollow Rectangle

- Ask the user for the height for a rectangle
- Then draw a rectangle of that height

+

Local Variables

Local Variables

- Functions are like "mini programs"
- You can create variables inside functions just as you would in your main program

Local Variables

```
def bugs():
  numbugs = int(input('How many bugs? '))
  print (numbugs)


bugs()
```

Local Variables

- However, variables that are defined inside of a function are considered "local" to that function.
- This means that they only exist within that function. Objects outside the "scope" of the function will not be able to access that variable

Local Variables

```
def bugs():
  numbugs = int(input('How many bugs? '))
  print (numbugs)


bugs()
print (numbugs) # error!  Variable numbugs
                # doesn't exist in this scope!
```

Local Variables

- Different functions can have their own local variables that use the same variable name
- These local variables will not overwrite one another since they exist in different "scopes"

Local Variables

```
def newjersey():
  numbugs = 1000
  print (“NJ has”, numbugs, “bugs”)

def newyork():
  numbugs = 2000
  print (“NY has”, numbugs, “bugs”)



newjersey()

newyork()
```

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('Craig')  # send 1 string`

Passing Arguments to a Function

```
def square(num):
  print (num**2)   # num assumes the value of the
                   # argument that is passed to
                   # the function (5)
square(5)
```

Passing Arguments to a Function

- When passing arguments, you need to let your function know what kind of data it should expect in your function definition
- You can do this by establishing a variable name in the function definition. This variable will be auto declared every time you call your function, and will assume the value of the argument passed to the function.

Passing Multiple Arguments to a Function

- You can actually pass any number of arguments to a function
- One way to do this is to pass in arguments "by position"

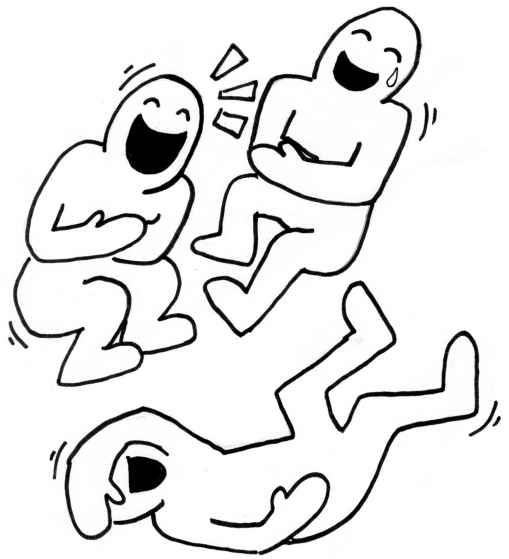
Passing Multiple Arguments to a Function

```
def average(num1, num2, num3):
  sum = num1+num2+num3
  avg = sum / 3
  print (avg)


average(100,90,92)c
```

Programming Challenge

- Write a "joke" generator that prints out a random knock knock joke.
- Extension: Write a "drum roll" function that pauses the program for dramatic effect! Have the drum roll function accept a parameter that controls how long it should pause.



Programming Challenge

- Write a function that accepts a restaurant check and a tip %
- Print out the tip that should be left on the table as well as the total bill
- If the tip is less than 15% you should tell the user that they might want to leave a little more on the table

Arguments vs Parameters

```
def average(num1, num2, num3):
  sum = num1+num2+num3
  avg = sum / 3
  print (avg)


average(100,90,92)
```

Parameters

Arguments

+

Argument Mechanics

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

Argument Mechanics

```
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

- We call this behavior "passing by value"
- We are essentially creating two copies of the data that is being passed – one that stays in the main program and one that is passed as an argument into our function
- This behavior allows us to set up a "one way" communication mechanism – we can send data into a function as an argument, but the function cannot communicate back by updating or changing the argument in any way
- (we will talk about how to communicate back to the caller in just a second!)

+

Global Variables

Global Variables

- When you create a variable inside a function we say that the variable is "local" to that function
- This means that it can only be accessed by statements inside the function that created it
- When a variable is created outside all of your functions it is considered a "global variable"
- Global variables can be accessed by any statement in your program file, including by statements in any function
- All of the variables we have been creating so far in class have been global variables

Global Variables

```
name = 'Craig'


def showname():
    print ("Function:", name)


print ("Main program:", name)
showname()
```

Global Variables

- If you want to be able to change a global variable inside of a function you must first tell Python that you wish to do this using the "global" keyword inside your function

Global Variables

```
name = 'Craig'

def showname():
  global name
  print ("Function 1:", name)
  name = 'John'
  print ("Function 2:", name)



print ("Main program 1:", name)

showname()

print ("Main program 2:", name)
```

Global Variables

- Global variables can make debugging difficult
- Functions that use global variables are generally dependent on those variables, making your code less portable
- With that said, there are many situations where using global variables makes a lot of sense.

+

Value Returning Functions

Value Returning Functions

- Value returning functions are functions that return a value to the part of the program that initiated the function call
- They are almost identical to the type of functions we have been writing so far, but they have the added ability to send back information at the end of the function call
- We have secretly been using these all semester!
  - `somestring = input("Tell me your name")`
  - `somenumber = random.randint(1,5)`

Writing your own value returning functions

- You use almost the same syntax for writing a value returning function as you would for writing a normal function
- The only difference is that you need to include a "return" statement in your function to tell Python that you intend to return a value to the calling program
- The return statement causes a function to end immediately. It's like the break statement for a loop.
- A function will not proceed past its return statement once encountered. Control of the program is returned back to the caller.

Value Returning Functions

```
def myfunction(arg1, arg2):
  statement
  statement
  …
  statement
  return expression



# call the function

returnvalue = myfunction(10, 50)
```

Programming Challenge: Combined Age

- Write a function that takes two age values as integers, adds them up and returns the result as an integer

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
```

Returning Boolean Values

- Boolean values can drastically simplify decision and repetition structures

- Example:
  - Write a program that asks the user for a part number
  - Only accept part #'s that are on the following list:
    - 100
    - 200
    - 300
    - 400
    - 500
  - Continually prompt the user for a part # until they enter a correct value

Returning multiple values

- Functions can also return multiple values using the following syntax:

```
def testfunction():
      x = 5
      y = 10
      return x, y


p, q = testfunction()
```

Programming Challenge: Maximum of two values

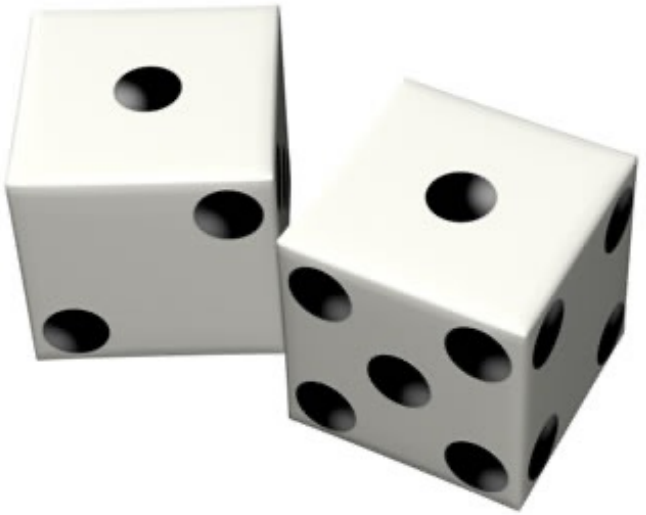
- Write a function named "maximum" that accepts two integer values and returns the one that the greater of the two to the calling program

Programming Challenge: Feet to Inches

- Write a function that converts feet to inches. It should accept a number of feet as an argument and return the equivalent number of inches.

Programming Challenge: Two Dice

- Write a function that simulates the rolling of two dice
- The function should accept a size parameter (i.e. how many sides does each die have)
- The function should return two values which represent the result of each roll
- Extension:
  - Make sure both numbers that you return are different (i.e. you can't roll doubles or snake eyes)
  - Build in an argument that lets you specify whether you want to enforce the no doubles policy



+

Nested Function Calls

Nested Function Calls

- We can nest function calls within one another.
- We have been doing this all along!
  - `num = int(input("Enter a positive number: "))`
- The innermost function call executes first and then the outermost function call

Example

```
def double(a):
        return a*2


answer = len(str(double(4)))
print(answer)
```

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

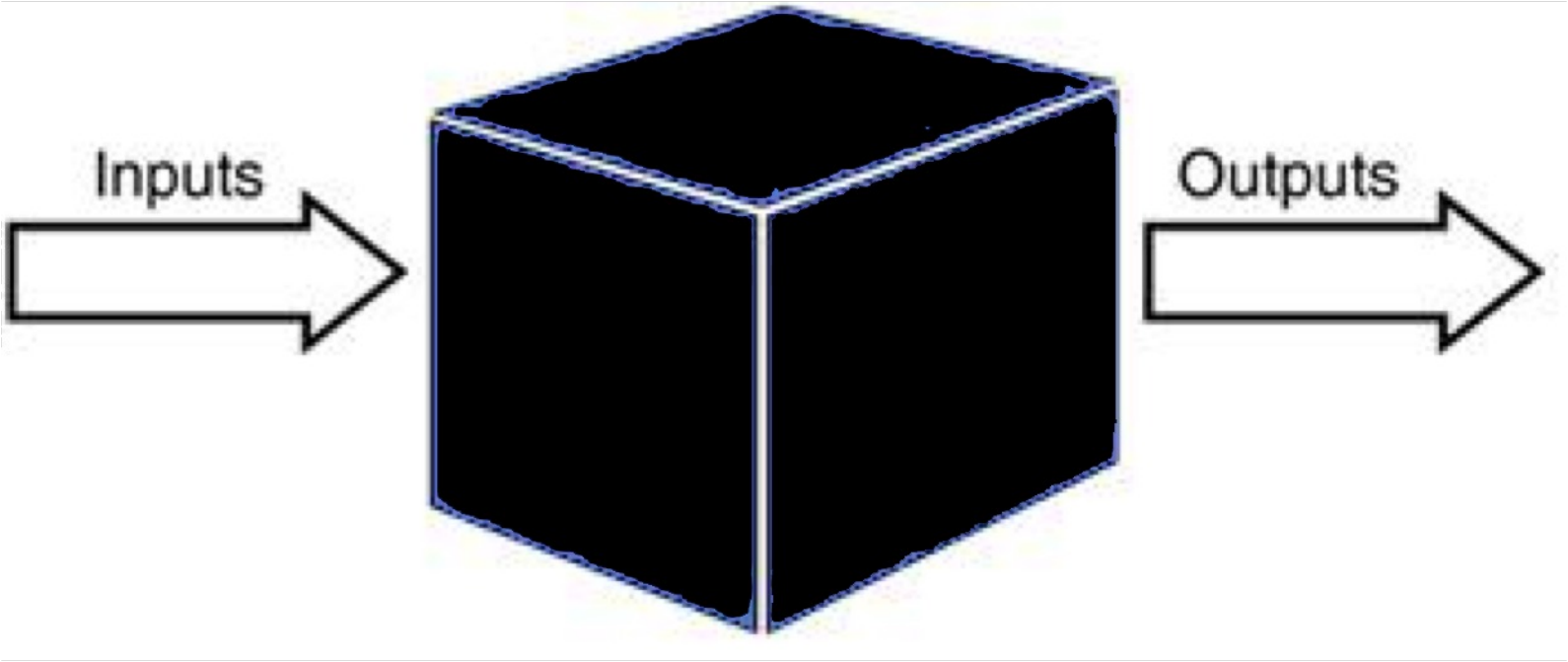
Modules

- 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

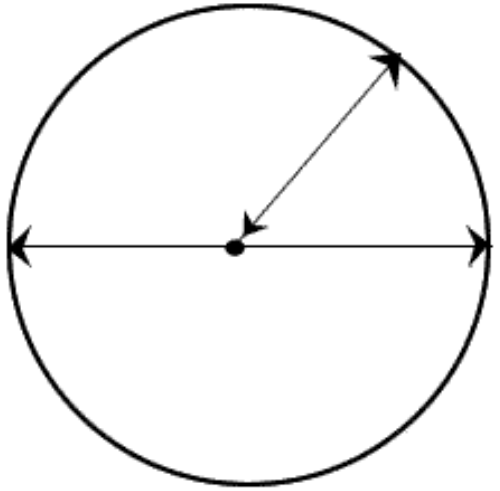
```
myfunctions.function1()
myfunctions.dosomethingelse()
```

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!