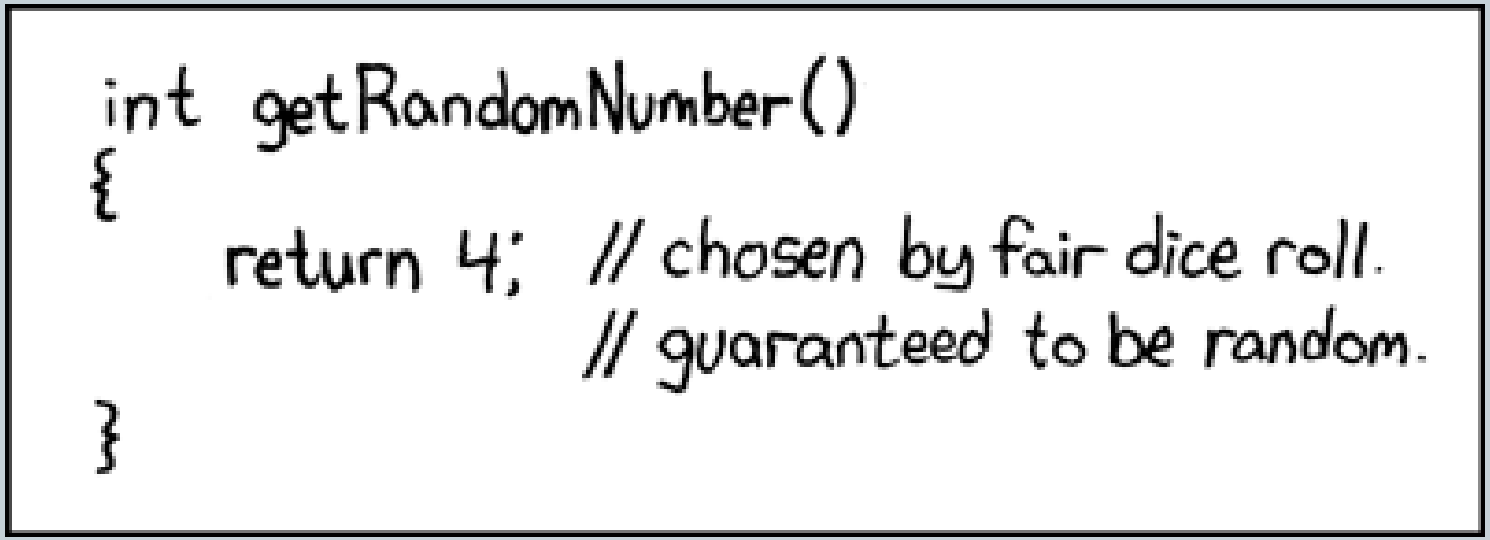
Functions

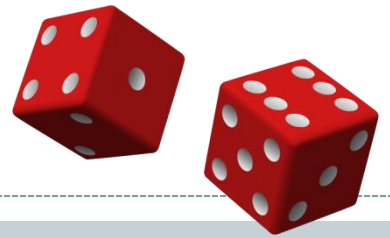


http://xkcd.com/221/

Outline

- Functions
  - Library Functions
    - Helper functions
      - Perform calculations
      - Output data
      - Consolidate similar code to one location
  - Functions
    - Flow of control
    - Anatomy/Terminology
    - Parameters
    - Return Values
    - Calling (Using) a Function

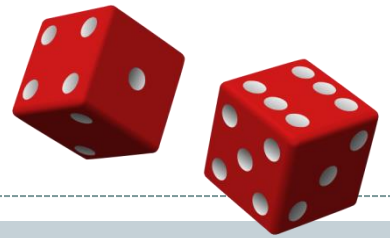
Programs Thus Far



- One big `list of code:`

```
import random

rolls  = 0
sum    = 0
target = random.randint(2, 12)

print("Rolling dice until I get " + str(target) + ".")
while sum != target:
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    sum = dice1 + dice2
    print(str(dice1) + " + " + str(dice2) + " = " + str(sum))
    rolls += 1

print("It took " + str(rolls) + " rolls.")
```

Programs Thus Far

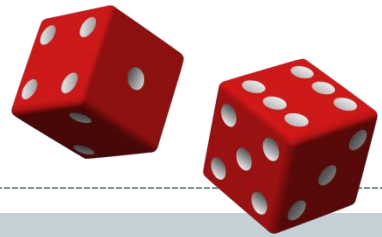


- One big `list of code:`

```
import random

rolls  = 0
sum    = 0
target = random.randint(2, 12)

print("Rolling dice until I get " + str(target) + "
while sum != target:
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    sum = dice1 + dice2
    print(str(dice1) + " + " + str(dice2) + " = " +
    rolls += 1

print("It took " + str(rolls) + " rolls.")
```

```
% python DiceRolling.py
Rolling dice until I get 4.
6 + 1 = 7
3 + 3 = 6
5 + 5 = 10
5 + 1 = 6
3 + 3 = 6
6 + 2 = 8
1 + 4 = 5
4 + 3 = 7
5 + 5 = 10
5 + 4 = 9
4 + 1 = 5
1 + 6 = 7
6 + 4 = 10
2 + 2 = 4
It took 14 rolls.
```

Programs Thus Far



- Problems with one list of code:
  - Doesn't scale to complex programs
  - Often find ourselves repeating similar code


```
import random

rolls  = 0
sum    = 0
target = random.randint(2, 12)

print("Rolling dice until I get " + str(target) + ".")
while sum != target:
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    sum = dice1 + dice2
    print(str(dice1) + " + " + str(dice2) + " = " + str(sum))
    rolls += 1

print("It took " + str(rolls) + " rolls.")
```

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num     = int(sys.argv[1])
r       = float(sys.argv[2])
x       = int(input(“Input an integer: “))
rand    = random.randint(1,6)
v       = math.sqrt(144)
listStr = oldStr.split()
```

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num     = int(sys.argv[1])
r       = float(sys.argv[2])
x       = int(input("Input an integer: "))
rand    = random.randint(1,6)
v       = math.sqrt(144)
listStr = oldStr.split()
```

Some functions return a value.

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num      = int(sys.argv[1])
r        = float(sys.argv[2])
x        = int(input("Input an integer: "))
rand     = random.randint(1,6)
v        = math.sqrt(144)
listStr = oldStr.split()
```

Some functions return nothing

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num     = int(sys.argv[1])
r       = float(sys.argv[2])
x       = int(input("Input an integer: "))
rand    = random.randint(1,6)
v       = math.sqrt(144)
listStr = oldStr.split()
```

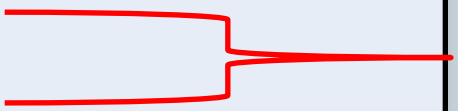
Some functions take a single parameter.

Some functions take a single parameter.

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num     = int(sys.argv[1])
r       = float(sys.argv[2])
x       = int(input("Input an integer: "))
rand    = random.randint(1,6)
v       = math.sqrt(144)
listStr = oldStr.split()
```



Some functions take no parameters

Using Library Functions

- Library Functions
  - Already seen loads of "helper" functions:

```
print("Hello world")
num     = int(sys.argv[1])
r       = float(sys.argv[2])
x       = int(input("Input an integer: "))
rand    = random.randint(1,6)
v       = math.sqrt(144)
listStr = oldStr.split()
```

Some functions take two parameters.

Functions

- Functions:
  - Like a mathematical function
    - Given some inputs, produce an output value
  - Functions allows building modular programs
    - Reuse code, only invent the wheel once
  - When a function is called:
    - Control jumps to the function code
    - Argument passed to function copied to parameter variables used in method
    - Function executes and (optionally) returns a value
    - Execution returns to calling code

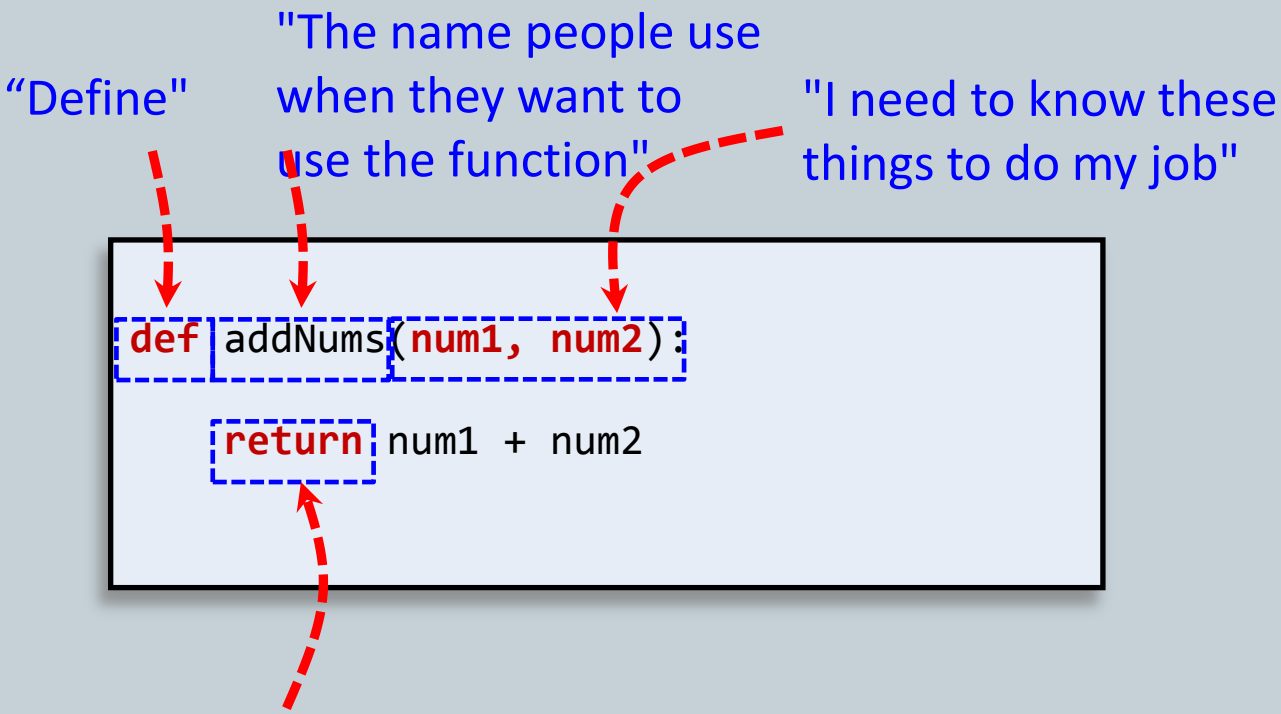
Flow of Control

```
def printWorld():
    print("world", end = "")

def addNums(num1, num2):
    result = num1
    result = num1 + num2
    return result

print("Hello", end = " ")
printWorld()
print(", 1 + 2 = ", end = "")
a = addNums(1, 2)
print(a)
```

```
% python MethodJumping.py
Hello world, 1 + 2 = 3
```

Anatomy of a Function

- Goal: helper function that can add two numbers



Pass by Value

- Python passes parameters by value (by copy)
  - Changes to primitive type parameters do not persist after function returns
    - Primitive types: `int, double, char, long, boolean`

```
def addNums(num1, num2):
    result = num1
    result = num1 + num2
    return result
```

```
c = 2
d = 3
print("sum = " + str(sum(c, d)))
print("c = " + str(c))
print("d = " + str(d))
```

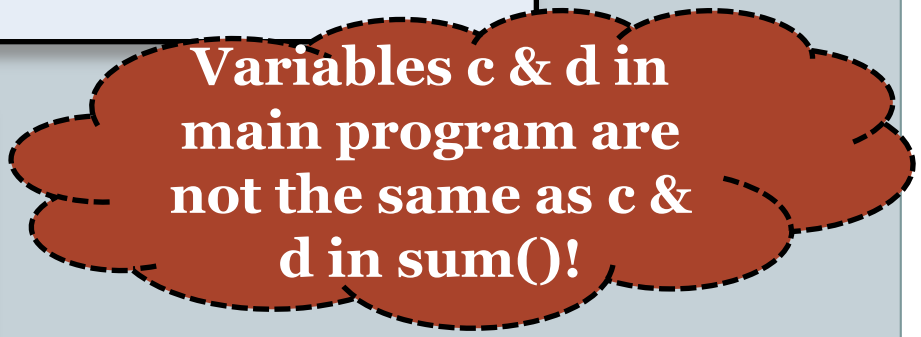
```
% python PassByVal.py
sum = 5
c = 2
d = 3
```

Pass by Value, Puzzler

```
def sum(c, d):
    result = c + d
    c = 0
    d = 0
    return result
```

```
c = 2
d = 3
print("sum = " + str(sum(c, d)))
print("c = " + str(c))
print("d = " + str(d))
```

```
% python PassByVal.py
sum = 5
c = 2
d = 3
```



List Parameters

- Lists can be passed as arguments

```
import random

def average(nums):
    total = 0
    for i in range(0,len(nums)):
        total += nums[i];
    return total / len(nums)

vals = []
for i in range(0, 1000):
    vals.append(random.randint(0,10))
print("avg " + str(average(vals)))
```

```
% python AverageList.py
avg 5.508
```

Quiz: Variable Scope

What lines are the following variables in scope?

nums 3-7

total 4-7

vals 9-12

i 5-6, 10-11

```
01: import random
02:
03: def average(nums):
04:     total = 0
05:     for i in range(0,len(nums)):
06:         total += nums[i];
07:     return total / len(nums)
08:
09: vals = []
10: for i in range(0, 1000):
11:     vals.append(random.randint(0,10))
12: print("avg " + str(average(vals)))
```

Quiz: Variable Scope

What is the value of `total` printed at the end of the program?

~ 5

What if we remove line 4?

Unbound local error: local variable 'total' referenced before assignment

```
01: import random
02:
03: def average(nums):
04:     total = 0
05:     for i in range(0,len(nums)):
06:         total += nums[i];
07:     return total / len(nums)
08:
09: vals = []
10: for i in range(0, 1000):
11:     vals.append(random.randint(0,10))
12: print("avg " + str(average(vals)))
```

Quiz: Variable Scope

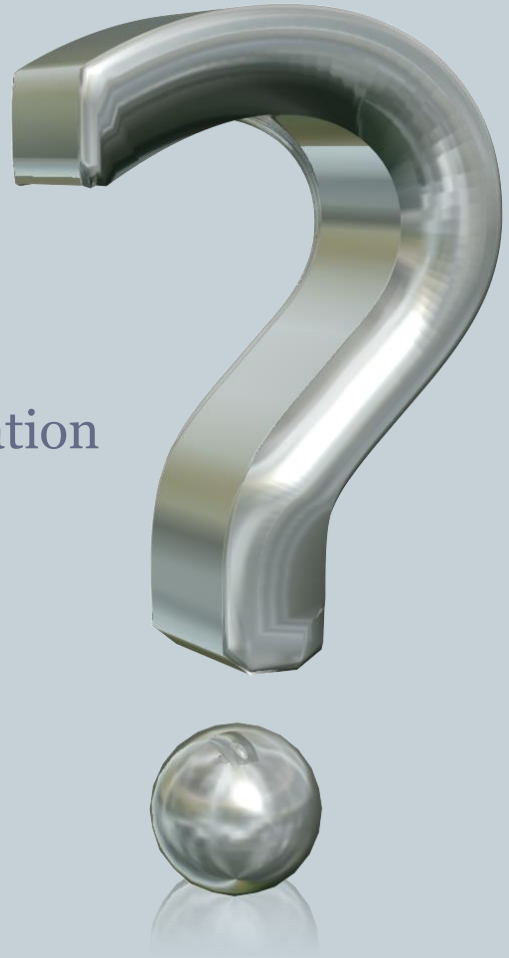
What if we remove line 9?

Name Error: name 'vals' not defined

```
01: import random
02:
03: def average(nums):
04:     total = 0
05:     for i in range(0,len(nums)):
06:         total += nums[i];
07:     return total / len(nums)
08:
09: vals = []
10: for i in range(0, 1000):
11:     vals.append(random.randint(0,10))
12: print("avg " + str(average(vals)))
```

Summary

- Functions
  - Library Functions
    - Helper functions
      - Perform calculations
      - Output data
      - Consolidate similar code to one location
  - Functions
    - Flow of control
    - Anatomy/Terminology
    - Parameters
    - Return Values
    - Calling (Using) a Function



Your Turn

- Write a function that returns the distance between two points. You should have 4 parameters, x1, y1, x2, y2, and the distance formula is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Submit your function to the Moodle dropbox for Activity05. 1 point for turning something in, 2 points for turning in something correct.