All about functions

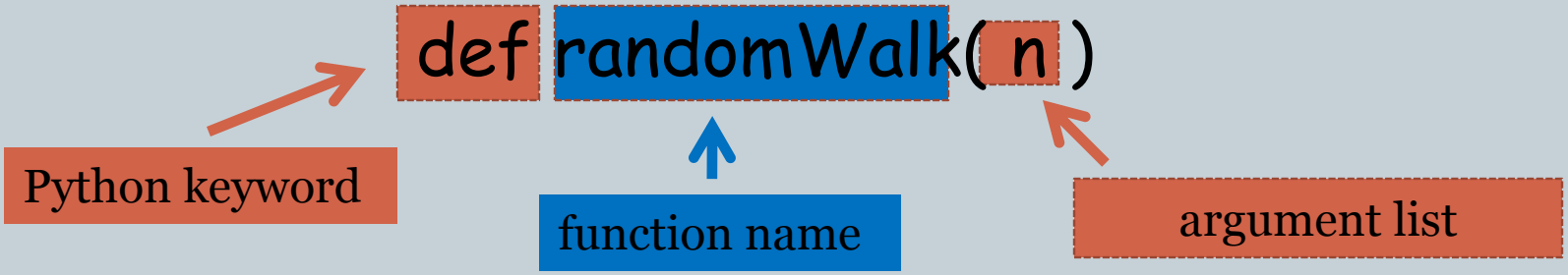
FEB 21ST

The function randomWalk

```
# This function takes the barrier distance n as an argument, simulates
# the random walk until it hits the barrier (n or -n), and returns the
# length of the random walk

def randomWalk( n ):
    location = 0 # tracks the location of the person
    length = 0 # tracks the length of the random walk

    # Loop terminates when the location reaches n or -n
    while abs(location) != n:
        step = random.randint(0, 1) #returns 0 or 1, each with prob. 1/2
        if step == 0:
            step = -1
        location = location + step
        length = length + 1

    return length
```

Notes about this function

- The first line of the function:



- The body of the function is indented.
- It is as though `n` is input to the function.
- A function can have one or more arguments
- The last line of the function is usually a return:

```
return length
```

The rest of the program

```
n = input("Enter a positive integer: ")
print randomWalk(n)
```

- randomWalk(n) is a call to the function randomWalk providing it the number n that the user as input as an argument.
- In order to execute the print statement, the function call randomWalk(n) needs to be executed first.
- This means that "control" is transferred to the function and we start executing the function starting with its first line.
- The value that the function returns essentially replaces the function call.

Averaging over 100 simulations

```
n = input("Enter a positive integer: ")

count = 0 # tracks the number of times the walk is repeated
sum = 0 # sum of the lengths of the walk; needed for average
while count < 100:
    sum = sum + randomWalk(n)
    count = count + 1

print float(sum)/100
```

Making another function

```
# This function repeats a random walk with barrier n as many times
# as specified by the argument numRepititions and returns the length
# of the walk, averaged over all the repititions

def manyRandomWalks(n, numRepititions):
    count = 0 # tracks the number of times the walk is repeated
    sum = 0 # sum of the lengths of the walk; needed for average

    # Repeats the random walk as many times as specified by numRepititions
    while count < numRepitions:
        sum = sum + randomWalk(n)
        count = count + 1

    return float(sum)/100
```

The rest of the program

```
n = input("Enter a positive integer: ")
print manyRandomWalks(n, 100)
```

- The function call needs to supply arguments in the correct order, i.e., in the order specified in the function definition.

- Names in the function call have nothing to do with names in the function definition. We could have written

```
m = input("Enter a positive integer: ")
print manyRandomWalks(m, 100)
```

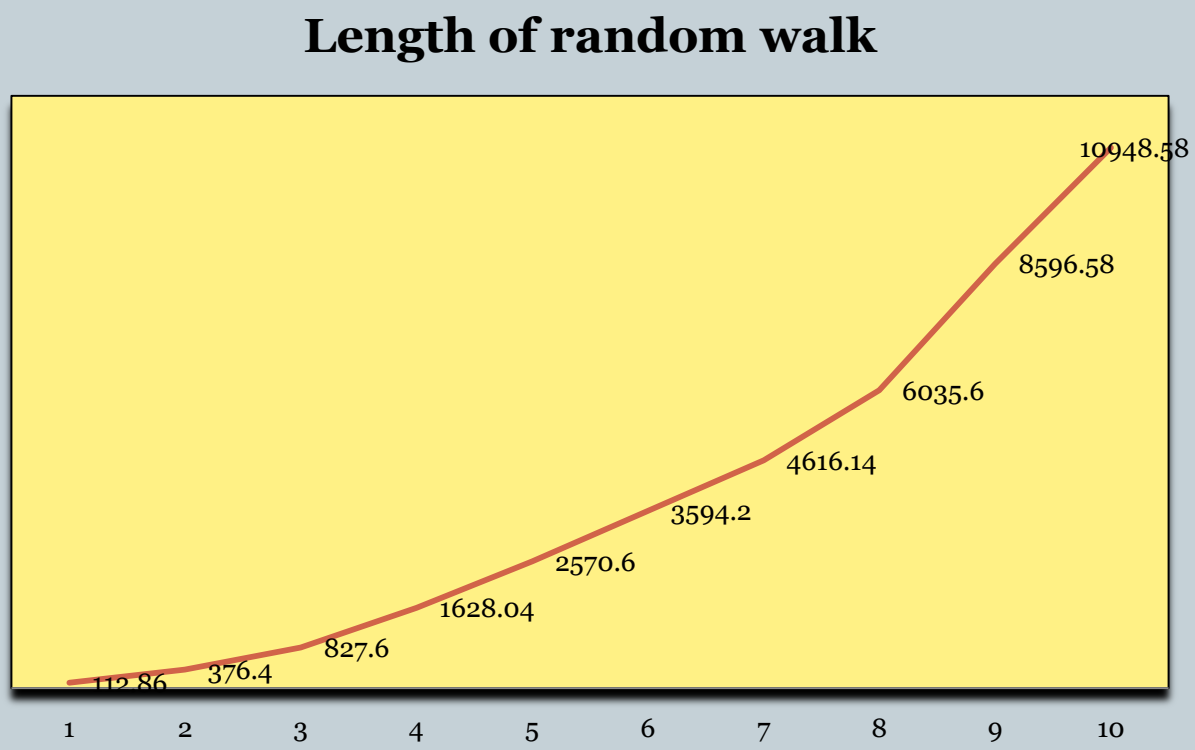
And the value of `m` and the value `100` would be used for `n` and `numRepititions` in the function.

Trying this out for different barrier values

```
m = 10 # tracks the value of the barrier
# m travels through 10, 20, ..., 100 in this loop and we compute and print the
# average walk length for each m
while m <= 100:
    print manyRandomWalks(m, 100)
    m = m + 10
```

Sample output

112.86
376.4
827.6
1628.04
2570.6
3594.2
4616.14
6035.6
8596.58
10948.58



The manyRandomWalks functions

- Definition:

```
def manyRandomWalks(n, numRepititions):
        ...
        ...
        return float(sum)/100
```

- The first line of the function definition is called the *function header*. The rest of the function is called the *function body*.
- The names `n` and `numRepititions` in the function header are called `parameters` of the function.
- Call to this function:

```
print manyRandomWalks(m, 100)
```

- The expressions `m` and `100` are called function *arguments*.

More on the manyRandomWalks function

- Arguments in a function call could be complicated expressions that will be evaluated to a value first before being sent in to the function.

  Example: manyRandomWalks(80/x, y + 1)

- In fact, arguments could be expressions involving calls to other functions.

  Example: manyRandomWalks(int(math.sqrt(x)), y + 1)

More on the randomWalks function

- One way in which Python matches arguments to parameters is by reading them left to right and matching 1st argument to 1st parameter, 2nd argument to 2nd parameter, etc.

- This is called the *positional style* of parameter passing.

- So

  ```
  manyRandomWalks(10, 100)
  ```

  and

  ```
  manyRandomWalks(100, 10)
  ```

  will return very different values.

- In this way of parameter passing the number of arguments and the number of parameters also have to exactly match.

Keyword arguments

- You can avoid matching by position by using *keyword arguments* in the function call.
- **Example:** `manyRandomWalks(numRepititions = 200, n = 20)`
- Here `numRepititions` and `n` are function parameters.
- Since the actual parameters are explicitly being provided values in the function call, the matching of arguments to parameters is no longer positional.
- The above function call is identical to the call `manyRandomWalks(n = 20, numRepititions = 200)`

Keyword parameters

- There is a way to define *default* values of parameters.
- **Example:** `def manyRandomWalks(n, numRepititions = 100)`
- This function can now be called with one or two arguments and in different styles.
- **Examples**: Try these out
  - `manyRandomWalks(10)`
    (The default value of 100 us used for `numRepititions`; 10 is used for `n`)
  - `manyRandomWalks(40, 150)`
    (40 is used for `n`, 150 for `numRepititions`)

Another example

```
def test(x = 3, y = 100, z = 200):
    return x - y + z
```

Examples of function calls:

1. `test(10)` (10 is used for `x`; default values 100 for `y` and 200 for `z`)
2. `test(10, 20)` (10 is used for `x`, 20 for `y`; default value 200 for `z`)
3. `test(z = 35)` (default values 3 for `x`, 100 for `y`; 35 for `z`)
4. `test(10, z = 35)` (10 for `x`, default value 100 for `y`, 35 for `z`)
5. `test(z = 50, 10, 12)` (Error: positional arguments come first, then keyword arguments)

Things that functions return

- Functions don't have to explicitly return values. For example:

```
def printGreeting(name):
        print "Hello", name, "how are you?"
```

- How would you call such a function?

Example:

```
printGreeting("Michelle")
```

- What would happen if you executed?

```
x = printGreeting("Michelle")
```