

# More about functions



**FEB 21ST**

# The manyRandomWalks functions



- **Definition:**

```
def manyRandomWalks(n, numRepetitions):  
    ...  
    ...  
    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 `numRepetitions` 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 1<sup>st</sup> argument to 1<sup>st</sup> parameter, 2<sup>nd</sup> argument to 2<sup>nd</sup> 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(numRepetitions = 200, n = 20)`
- Here `numRepetitions` 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, numRepetitions = 200)`

# Keyword parameters



- There is a way to define *default* values of parameters.
- **Example:** `def manyRandomWalks(n, numRepetitions = 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 is used for `numRepetitions`; 10 is used for `n`)
  - `manyRandomWalks(40, 150)`  
(40 is used for `n`, 150 for `numRepetitions`)

# 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")
```