More about functions:
Keyword arguments and parameters

FEB 18TH, 2015

Parameters versus Arguments

- Parameters are variables used in a function header.
- Parameters get assigned values when a function is called.

```
def foo(x, y, z):
    x = y + z
    return x + y + z
```

- Here `x`, `y`, and `z` are parameters of the function `foo`.
- Inside the function `foo`, they can be treated as variables that acquire values provided by a function call (e.g., `foo(2, 7, 3)`).

Parameters versus Arguments

- 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)`

Matching arguments to parameters

- 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”)
```

The object None

- None is a built-in constant in Python that is used to indicate the absence of a value.

- In the example,

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

  x is assigned the value None. You can see this by trying

```
print(x)
```

- To understand None better try:
  - type(x)
  - bool(x)

- Unlike True and False which can be assigned to even though they are listed as built-in Python constants, None cannot be assigned to.