CSSE 120 – Introduction to Software Development

Concept: *Functions with Parameters*

Defining **functions**

```
def convert_and_return(celsius):
    fahrenheit = ((9 / 5) * celsius) + 32
    return fahrenheit
```

A ***function*** is a chunk of code that has a name. Here (to the right) is a portion of an example of the notation for ***defining*** a function.

The ***name*** of the function follows the keyword `def`. The variables in the parentheses after the name of the function are called ***parameters***. This function ***returns*** a value. (Functions that have no ***return*** statement return the special value ***None***.)

Why have functions?

Functions are powerful for 2 reasons:

- They help ***organize a program into logical chunks***. That makes it easier to:
    - Test the program (by testing the chunks, called ***unit testing***).
    - Modify the program (by focusing your interest on the chunks of interest).
    - Write correct code (by understanding the organization of the program).
    - ***Encapsulate*** (enclose and hide) the behavior of a function inside its definition, thus separating:
        - the ***specification*** (*what* the function accomplishes) of the function
        - from its ***implementation*** (*how* it accomplishes its specification).
- You can ***re-use functions***. That is, you can call them over and over again, with different values for the parameters to achieve different results.

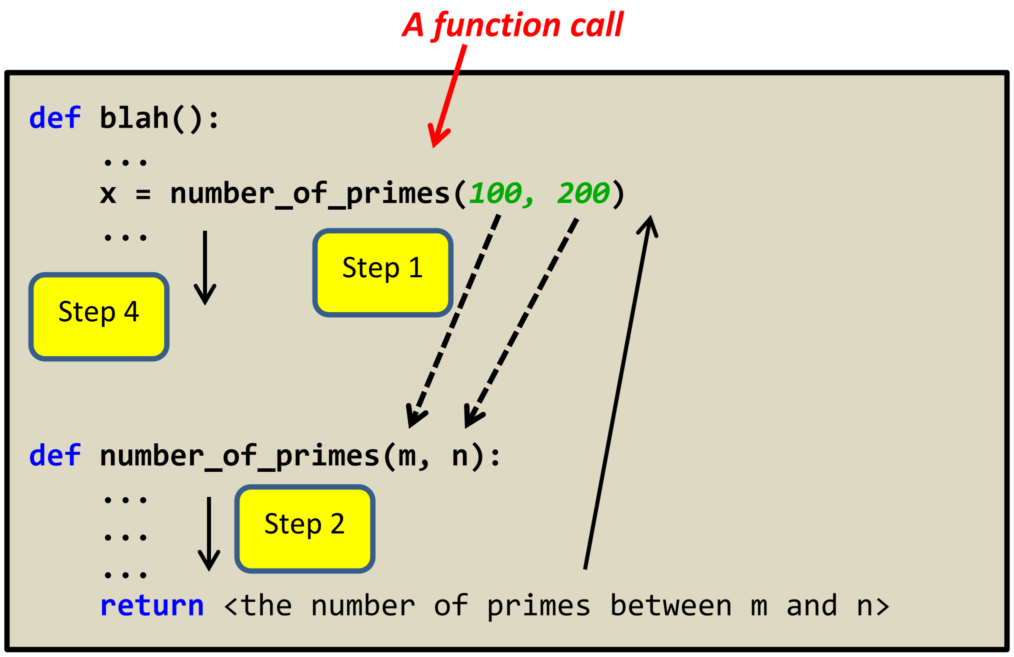
(continues on the next page)

Calling functions

You ***call*** (aka ***invoke***) a function by writing its ***name followed by parentheses***, with the ***actual arguments*** placed inside the parentheses.[1]

When you call a function:

1. The actual ***arguments*** of the function ***call*** (the values in the parentheses) are sent into the formal ***parameters*** of the function ***definition***.
2. ***Execution continues*** at the beginning of the definition of the called function.
3. When the function's ***return*** statement is executed, the returned value is sent back to the calling function. Or, if the end of the function is reached without a ***return*** statement, the special value ***None*** is sent back to the calling function.
4. ***Execution continues*** from the place where the function call appeared, with the returned value replacing the function call.



```
def blah():
    ...
    x = number_of_primes(100, 200)
    ...

def number_of_primes(m, n):
    ...
    ...
    ...
    return <the number of primes between m and n>
```

Note especially the ***two-way transfer of information***:

- When a function is called, the values of the ***arguments*** are sent ***TO*** the function, with the ***parameters RECEIVING*** those values.
    - So this is how information goes ***FROM the caller INTO a called function***.
- When a function executes a ***`return`*** statement (or reaches its end), its returned value is sent ***BACK*** from the function, with the ***caller RECEIVING*** that value.
    - So this is how information goes ***FROM the function BACK TO the caller***.
    - If there is no explicit ***`return`*** statement, the value ***None*** is returned automatically.
    - The caller will typically ***capture*** the returned value in a ***variable***, using that variable in subsequent statements, as shown in the diagram above.

[1] You ***MUST*** have the parentheses even when there are no arguments. It is the parentheses that tell the interpreter to ***call*** the function instead of just ***referring*** to it. Avoid this common mistake:
`y = blah` where you meant `y = blah()`