Functions with arguments and parameters

Arguments and parameters are a mechanism by which a function may receive outside information that can influence how the function works. A *parameter* is a variable that is placed inside the function's parentheses when it is defined. That variable may be used within the body of the function in any way a normal variable would. The parameter represents a piece of information the function must receive from outside its definition in order to function.

Syntax for defining a function with parameters:

```
def name_of_function(param1, param2, ...):
   statement                     # This block of statements
   statement                     # is called the body of
   statement                     # the function definition.
   …
```

Syntax for calling a function with arguments:

```
name_of_function(argument1, argument2, ...)
```

In order to call a function that needs arguments, we must give those parameters values. This is done by putting *arguments* inside the function call. Arguments can be anything Whenever Python sees a function call to a function that takes parameters, before jumping to the start of the new function body, it assigns any parameters in the function definition the corresponding values from where the function is called.

```
def name_of_function(param1, param2, ...):
   statement
   statement

name_of_function(argument1, argument2, ...)
```

Equivalently, imagine that a function that takes arguments has some hidden variable assignment statements at the beginning of the body that are changed automatically every time the function is called:

```
def name_of_function(param1, param2, …):
   param1 = argument1              # Python does this
   param2 = argument2              # behind the scenes.
   # if there were more arguments,
   # there would be more assignments statements here…
   statement
   statement
```

Example:

```
def sing_song(name, age):
   print("Happy birthday to you!  Happy birthday to you!")
   print("Happy birthday dear", name, "Happy birthday to you!")
   print("You are now", age, "years old!")

def main():
   sing_song("Brian", 84)
   sing_song("Meg", 27)

main()
```

Output:
```
Happy birthday to you!  Happy birthday to you!
Happy birthday dear Brian Happy birthday to you!
You are now 84 years old!
Happy birthday to you!  Happy birthday to you!
Happy birthday dear Meg Happy birthday to you!
You are now 27 years old!
```

You can also call functions with arguments by using variables instead of literals:

Example:
```
def sing_song(name, age):
   print(“Happy birthday to you!  Happy birthday to you!”)
   print(“Happy birthday dear”, name, “Happy birthday to you!”)
   print(“You are now”, age, “years old!”)

def main():
   username = input(“What is your name? ”)
   their_age = int(input(“What is your age? “))
   sing_song(username, their_age)

main()
```
Output:
```
What is your name? Sheldon Cooper
What is your age? 31
Happy birthday to you!  Happy birthday to you!
Happy birthday dear Sheldon Cooper Happy birthday to you!
You are now 31 years old!
```

Common mistakes:
- It is illegal to call a function that takes arguments with more or fewer than the appropriate number of arguments.
    - For example, in the version of `sing_song` immediately above, if we called the function using `sing_song(“Alice”)` or `sing_song()` or `sing_song(“Alice”, 39, “Bob”, 42)`, our program would crash because that function, based on the definition above, always takes exactly *two* arguments.
- It is illegal to call a function that takes arguments of the wrong data type.
    - For example, there is a math function in Python called `math.sqrt` that takes one argument that must be a number. If you call it using `math.sqrt(“apple”)`, your program will crash because `“apple”` is a string, not a number.

Notice that the names of the variables do not have to match between the arguments in the function *call* and the function *definition*. It's OK to use the same variable names in both places, but realize that **the transfer of information is one-way only**. Information is passed from where the function is called to where it is defined, but any changes you make to the arguments are not passed back:

```
def some_function(x):
   print("Inside the function, x is", x)
   x = 17
   print("Inside the function, x is changed to", x)

def main():
   x = 2
   print("Before the function call, x is", x)
   some_function(x)
   print("After the function call, x is", x)

main()
```

Output:

```
Before the function call, x is 2
Inside the function, x is 2
Inside the function, x is 17
After the function call, x is 2
```

Why does this happen? This happens because there is *no permanent connection* between the `x` in `main` and the `x` in `some_function`. They are completely separate variables because they were defined in separate functions, even though they happen to share a name (by coincidence). When `main` calls `some_function(x)`, `main's` value for `x` is passed to `some_function`'s version of `x`, but that's where the connection ends. When `some_function` assigns 17 to `x`, nothing is transferred back to `main`.

In fact, any variables you assign to inside a function will never affect anything outside of it (until we learn about returning values, but that's for later). The easiest mistake to make with this is assigning to a new variable inside a function then trying to use it outside of that function.

```
def sing_song(name, age):
   print("Happy birthday to you!  Happy birthday to you!")
   print("Happy birthday dear", name, "Happy birthday to you!")
   print("You are now", age, "years old!")
   age_next_year = age + 1

def main():
   sing_song("Brian", 84)
   print("Next year you will be", age_next_year, "years old!")

main()
```

The line in bold will crash your program because `age_next_year` is assigned to inside of `sing_song`, making it a local variable to `sing_song`. It is invisible to all other functions in your program besides `sing_song`.