

Functions that take arguments

We know that variables only exist inside the functions in which they are first assigned a value:

```
# THIS PROGRAM DOESN'T WORK!
def sing_song():
    print("Happy birthday to you! Happy birthday to you!")
    print("Happy birthday dear", name, "Happy birthday to you!")

def main():
    name = input("What is your name? ")
    sing_song()

main()
```

The program above won't work because `name` is a variable that is first assigned a value inside of the function `main`. It only exists inside of `main`, and is invisible to all other functions, including `sing_song`. Therefore, we need some way for `main` to pass information to `sing_song`, so that we know who to sing the birthday song to. In general, we'd like a way for one function to pass *any* piece of information to another function.

Syntax for defining a function that takes arguments (or parameters):


```
def name_of_function(variable1, variable2, ...):
    statement                # This group of statements
    statement                # is called the body of
    statement                # the function definition.
    ...
```

Syntax for calling a function that takes arguments (or parameters):

```
name_of_function(value1, value2, ...)
```

Whenever Python sees a function call to a function that takes arguments, before jumping to the start of the new function body, it assigns any variables in the function definition the corresponding values from where the function is called.

```
def name_of_function(variable1, variable2, ...):
    statement
    statement
```



```
name_of_function(value1, value2, ...)
```

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(variable1, variable2, ...):
    variable1 = value1        # Python does this
    variable2 = value2        # 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 less 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`.