Functions and Parameters

Functions!

A Function is a group of commands, given a name.

There are lots of Functions built into Python. We can tell what is and isn't a Function by looking for parentheses after a name - for example, int() or range().

We can use Functions as many times as we want once they're created!

But why though?

Functions help us by:

- Shortening our code
- Letting us reuse code multiple times without copy+paste
- Making our code more readable

- The def keyword
- A name
- Parentheses
- A colon
- A body

We need 5 things to create a new function:

- The def keyword
- A name

def

- Parentheses
- A colon
- A body

We need 5 things to create a new function:

- The def keyword
- A name
- Parentheses
- A colon
- A body

def does something

- The def keyword
- A name
- Parentheses
- A colon
- A body

```
def does something()
```

- The def keyword
- A name
- Parentheses
- A colon
- A body

```
def does something():
```

- The def keyword
- A name
- Parentheses
- A colon
- A body

```
def does_something():
    print("woah")
```

How to make a new function 2 electric boogaloo

Like variables, Functions need to exist **before** they can be used.

Because of this, we should always define our Functions at the top of our programs.

The code inside of the Function definition will **NOT** be run when the interpreter reads over those lines of code - they will only be run when the Function is <u>called</u>.

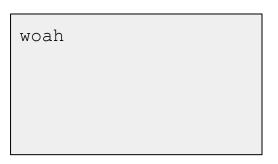
Calling a Function

When I tell a Function to do its thing, the word for that is <u>call</u> - we are <u>calling</u> the Function!

I can <u>call</u> a Function by simply saying its name!

```
def does_something():
    print("woah")

does_something()
```



Adding Parameters

Parameters

Parameters are pieces of information that we can give to our functions when we call them.

When we are defining our own functions, we can make them accept parameters by placing their names inside their parentheses, like so:

```
def does_something(x):
    print(x)

does_something(10)
```

More info on parameters

When we say a Function is going to accept a parameter, we are enabling immediate use of that variable inside our Function, because it will already have data inside!

The act of sending information through a parameter is called <u>passing</u>. We <u>pass</u> parameters to our <u>Functions</u> when we <u>call</u> them!

More Parameters!

We can tell our Functions to accept more than 1 parameter by placing multiple variable names, separated by commas, inside the parentheses on the definition line, like so:

```
def print_sum(num1, num2):
    print(num1 + num2)

print_sum(10, 13)
num1 = 4
num2 = 2
print sum(num1, num2)
```

Multiple Parameters

When a Function does accept multiple parameters, the values in the Function call will be mapped in order to the variables in the Function definition.

```
def print_sum(num1, num2):
    print(num1 + num2)

num1 = 4
num2 = 2
print sum(num2, num1)
```

Making Parameters Optional

```
circle(40)
circle(40, 180)
circle(40, 180, 3)
```

Sometimes, we can <u>call</u> the same <u>Function</u> with different numbers of parameters.

We can make the Functions we define do the same thing!

How to do that

In order to make a parameter optional, we need to give it a **default value**. This is the value the variable will have in the Function if that slot isn't filled during the Function's <u>call</u>. This is how it's done!

```
def print_sum(num1, num2 = 0):
    print(num1 + num2)

print_sum(10, 4)
print_sum(8)
```

Ordering Optional Parameters

If your Function is going to include optional parameters, all *non*-optional parameters **must** come first in the definition.

Example:

```
def sum_vals(num1, num2 = 0):
    print(num1 + num2)

vs.
def sum_vals(num1 = 0, num2):
    print(num1 + num2)
```

Using Optional Parameters

Just like when we use multiple parameters normally, optional parameters will be filled in from left to right, according to the Function's definition.

Example:

```
def sum_vals(num1 = 0, num2 = 0, num3 = 0):
    print(num1 + num2 + num3)

sum_vals()
sum_vals(9)
sum_vals(9, 2)
sum_vals(9, 2, 8)
```

Using Optional Parameters Out of Order

But what if I want to use only an optional parameter that comes later in the order? When we're <u>calling</u> the <u>Function</u>, we can put the parameters' names in the call to specify which parameter you're giving a value to. The parameters don't even need to be in order!

```
def sum_vals(num1 = 0, num2 = 0, num3 = 0):
    print(num1 + num2 + num3)

sum_vals(num3 = 5)
sum_vals(num2 = 3, num3 = 10, num1 = 2)
```