Worksheet 5
Basic Functions

Function

A **function** is a nice **shorthand for a formula**. It has an **input, does something, ad produces and output**.

- *for example, we can create a formula for the number of standard drinks in a drink*

```
std_drinks = volume * percentage * 0.789

# Calculate the number of standard drinks contained in `volume`
# of alcoholic drink `percentage` % of alcohol
volume = 0.375
percentage = 13.5
drinks = volume * percentage * 0.789
print(drinks)
3.9943125000000004
```

- *this can be converted into a function that takes the inputs, volume and percentage and then returns the standard drink for any variable*

```
def std_drinks(volume, percentage):
    return volume * percentage * 0.789

print(std_drinks(0.375, 13.5))
print(std_drinks(0.586, 3.8))
print(std_drinks(1.5, 0))

3.9943125000000004
1.7569452
0.0
```

All functions have **paramenters** which are **very similar to variables**. The **values inputed** are arguments. The **expression after** `return` is what the function will **output.** For example:

- *parameters = volume, percentage*
- *argument* = `0.375, 13.5`
- *output = volume * percentage * 0.789*

Basic Anatomy of a Function Definition

The basic components of functions are:

- the **function name**;
- the **parameters**;
- the **body**, where the actual computation associated with the function occurs;
- some **mechanism for returning a value and exiting** the function.
  - when using a `return`statement, the code stops and exits

For example, a function name would be as follows:

```
def my_first_function():
```

Parameters:

```
def my_first_function(num0, num1, num2):
```

Body, can be any code that makes use of the variables:

```
def my_first_function(num0, num1, num2):
    maxnum = num0
    if num1 > maxnum:
        maxnum = num1
    if num2 > maxnum:
        maxnum = num2
```

Return statement:

```
def my_first_function(num0, num1, num2):
    maxnum = num0
    if num1 > maxnum:
        maxnum = num1
    if num2 > maxnum:
        maxnum = num2
    return maxnum
```

Multi-Parameter Functions

Functions can be defined with **multiple parameters** using a **comma to separate arguments**.

- Must have **different names** *unlike the following example:*

```
def function(a, a, a):
  return a
File "program.py", line 1
    def function(a, a, a):
    ^
SyntaxError: duplicate argument 'a' in function definition
```

- The **ordering** of the arguments will determine what function argument is set to what value

```
def swapped_sum(a, b):
    return b+a
print(swapped_sum("a", "b"))
print(swapped_sum("b", "c"))
ba
cb
```

- Functions can have **no parameters** and **empty parentheses** where any function call will have similarly empty parentheses

```
def error_code():
    return -1
print(error_code())
-1
```

Worksheet 6
Basic Methods

Methods

Methods are **unique functions that belong to specific data types**. They are written and called differently to functions.

- *for example, the method upper() is only for str type and converts it to uppercase*

```
my_str = "take the pie out of the oven"
print(my_str)
print(my_str.upper())
take the pie out of the oven
TAKE THE PIE OUT OF THE OVEN
```

or

```
print("goddamn it, it's burnt".upper())
GODDAMN IT, IT'S BURNT
```

The Format Method

The **`format` method** allows the **formatting of `str` type**.

```
num1 = 5
num2 = 6.7
print("The quotient of {} and {} is {:.2f}".format(num1,num2,num1/num2))
The quotient of 5 and 6.7 is 0.75
```

*The following example uses the format method to **generate a new string by substituting a value**.*

- *the **empty curly braces {}** are used to tell the format method **where to put the string***

```
print("I own {} horses".format(10))
I own 10 horses
```

There is also the **format string or f-string**. The string is **prefixed with f and variable names/statements are directly inserted into the braces**

- everything about the **`format` method can apply to f-strings**

Formating Floats

The **`float` type** can be **formatted using curly brackets**.

- *for example, `{:6.2f}` tells `format` to use at least 6 characters in total and use exactly two digits after the decimal point*

```
pi = 3.1415926
# what do you think < does?
print('{:<6.2f}'.format(pi))  # pad on the right
print('{:6.2f}'.format(pi))   # pad on the left
3.14
  3.14
```

Formating Different Types

`Format` also has the ability to embed different types (not just numbers) in strings like so

```
print("strings: {:s}".format("9743"))
print("binary number: {:b}".format(9743))
print("integer base 10: {:d}".format(9743))
# Characters
print("Call me on my {:c}".format(9743))  # yes, it's a telephone
```

Formatting Keywords

`Format` insists on the colon before the format specification (e.g. `.2f`). If there is **nothing before the colon**, Python will **substitute in the arguments to `format` in sequence**

- *i.e. the first argument to `format` will be substituted for the first occurrence of curly braces, the second for the second, etc.*
- you can also, index the arguments to `format` in the same way that you index the elements of a list, *for example:*

```
print("{0} {0} It's off to work we go".format("Hi Ho!"))
Hi Ho! Hi Ho! It's off to work we go
```

You can also name the arguments to **format** using **keywords** *for example:*

```
print("""{t1} and {t2}
    Agreed to have a battle;
For {t1} said {t2}
  Had spoiled his nice new rattle.""".format(t1="Tweedledum",
t2="Tweedledee"))
Tweedledum and Tweedledee
    Agreed to have a battle;
For Tweedledum said Tweedledee
    Had spoiled his nice new rattle.
```