Functions

Functions are blocks of organized, reusable code that is used to perform a single, related action. Functions accept parameters and may or may not return a value.

```
def func(fargs, *args, **kwargs):
    """
    Function docstring
    """
    function_suite
    return [expression]
```

`fargs` - Normal function argument

`*args` and `**kwargs` allow you to pass an **unspecified number of arguments** to a function.

- `*args` is used to send a **non-keyworded** variable length argument list to the function
- `**kwargs` if you want to handle **named arguments** in a function

Example for `*args`:

```
def sample(farg, *args):
    print("Normal arg: " + farg)
    for arg in args:
        print("another arg through *args: ", arg)

sample('normal arg', 'first', 'second')

# Output
>>>
Normal arg: normal arg
another arg through *args: first
another arg through *args: second
```

Example for `**kwargs**`:

```
def sample(farg, **kwargs):
    print("Normal arg:" + farg)
    for key,value in kwargs.items():
        print(f"{key} = {value}")

sample('normal arg', first="1st", second="second")

# Output
>>>
Normal arg:normal arg
first = 1st
second = second
```

- Things to note:
    - By calling `return [expression]`, you exit from the function, returning a value to the caller. A `return` statement with no arguments is the same as `return None`

Scope of variables

- Global Variables
  - defined **outside a function body**, and has a **global scope**
  - accessed **throughout the program** by all functions
- Local Variables
  - defined **within a function body**, and has a **local scope**
  - accessible **only within the functions** in which they are declared

```
total = 0 # this is in the global scope

def sum(x, y):
    total = x + y # total is in the local scope
    return total

print(str(sum(10, 20)), total)

# Output:
>> 30 0

# This is because total = 10 + 20 is in the local scope and the value of total
in the global scope remains unchanged
```