Day 21

Function Arguments and return statement

There are four types of arguments that we can provide in a function:

- Default Arguments
- Keyword Arguments
- Variable length Arguments
- Required Arguments

Default arguments:

We can provide a default value while creating a function. This way the function assumes a default value even if a value is not provided in the function call for that argument.

Example:

```
def name(fname, mname = "Jhon", lname = "Whatson"):
    print("Hello,", fname, mname, lname)
name("Amy")
```

Output:

Hello, Amy Jhon Whatson

Keyword arguments:

We can provide arguments with key = value, this way the interpreter recognizes the arguments by the parameter name. Hence, the the order in which the arguments are passed does not matter.

Example:

```
def name(fname, mname, lname):
    print("Hello,", fname, mname, lname)

name(mname = "Peter", lname = "Wesker", fname = "Jade")
```

Output:

Hello, Jade Peter Wesker

Required arguments:

In case we don't pass the arguments with a key = value syntax, then it is necessary to pass the arguments in the correct positional order and the number of arguments passed should match with actual function definition.

Example 1: when number of arguments passed does not match to the actual function definition.

```
def name(fname, mname, lname):
    print("Hello,", fname, mname, lname)

name("Peter", "Quill")
```

Output:

name("Peter", "Quill")
TypeError: name() missing 1 required positional argument: 'lname'

Example 2: when number of arguments passed matches to the actual function definition.

```
def name(fname, mname, lname):
  print("Hello,", fname, mname, lname)

name("Peter", "Ego", "Quill")
```

Output:

Hello, Peter Ego Quill

Variable-length arguments:

Sometimes we may need to pass more arguments than those defined in the actual function. This can be done using variable-length arguments.

There are two ways to achieve this:

Arbitrary Arguments:

While creating a function, pass a * before the parameter name while defining the function. The function accesses the arguments by processing them in the form of tuple.

Example:

```
def name(*name):
  print("Hello,", name[0], name[1], name[2])

name("James", "Buchanan", "Barnes")
```

Output:

Hello, James Buchanan Barnes

Keyword Arbitrary Arguments:

While creating a function, pass a * before the parameter name while defining the function. The function accesses the arguments by processing them in the form of dictionary.

Example:

```
def name(**name):
  print("Hello,", name["fname"], name["mname"], name["lname"])

name(mname = "Buchanan", lname = "Barnes", fname = "James")
```

Output:

Hello, James Buchanan Barnes

return Statement

The return statement is used to return the value of the expression back to the calling function.

Example:

```
def name(fname, mname, lname):
   return "Hello, " + fname + " " + mname + " " + lname

print(name("James", "Buchanan", "Barnes"))
```

Output:

Hello, James Buchanan Barnes