codecademy

Function Arguments

Default argument is fallback value

In Python, a default parameter is defined with a fallback value as a default argument. Such parameters are optional during a function call. If no argument is provided, the default value is used, and if an argument is provided, it will overwrite the default value.

```
def greet(name, msg="How do you do?"):
  print("Hello ", name + ', ' + msg)

greet("Ankit")
greet("Ankit", "How do you do?")

"""
this code will print the following for both
the calls -
`Hello  Ankit, How do you do?`
"""
```

Mutable Default Arguments

Python's default arguments are evaluated only once when the function is defined, not each time the function is called. This means that if a mutable default argument is used and is mutated, it is mutated for all future calls to the function as well. This leads to buggy behaviour as the programmer expects the default value of that argument in each function call.

```
# Here, an empty list is used as a default
argument of the function.
def append(number, number_list=[]):
  number_list.append(number)
  print(number_list)
  return number_list

# Below are 3 calls to the `append` function
and their expected and actual outputs:
append(5) # expecting: [5], actual: [5]
append(7) # expecting: [7], actual: [5, 7]
append(2) # expecting: [2], actual: [5, 7, 2]
```

Python Default Arguments

A Python function cannot define a default argument in its signature before any required parameters that do not have a default argument. Default arguments are ones set using the form `parameter=value`. If no input value is provided for such arguments, it will take on the default value.

```
# Correct order of declaring default argments in a function
def greet(name, msg = "Good morning!"):
  print("Hello ", name + ', ' + msg)

# The function can be called in the following ways
greet("Ankit")
greet("Kyla","How are you?")

# The following function definition would be incorrect
def greet(msg = "Good morning!", name):
  print("Hello ", name + ', ' + msg)
# It would cause a "SyntaxError: non-default argument follows default argument"
```

Python function default return value

If we do not not specify a return value for a Python function, it returns `None`. This is the default behaviour.

```
# Function returning None
def my_function(): pass

print(my_function())

#Output
None
```

Python variable None check

To check if a Python variable is None we can make use of the statement `variable is None`.
If the above statement evaluates to True, the `variable` value is None.

```
# Variable check for None
if variable_name is None:
    print "variable is None"
else:
    print "variable is NOT None"
```

Python function arguments

A function can be called using the argument name as a keyword instead of relying on its positional value. Functions define the argument names in its composition then those names can be used when calling the function.

```
# The function will take arg1 and arg2
def func_with_args(arg1, arg2):
  print(arg1 + ' ' + arg2)

func_with_args('First', 'Second')
# Prints:
# First Second

func_with_args(arg2='Second', arg1='First')
# Prints
# First Second
```