Python Functions

Function arguments

A function can take `arguments` and `return values`:

In the following example, the function **say_hello** receives the argument "name" and prints a greeting:

```
>>> def say_hello(name):
...     print(f'Hello {name}')
...
>>> say_hello('Carlos')
# Hello Carlos

>>> say_hello('Wanda')
# Hello Wanda

>>> say_hello('Rose')
# Hello Rose
```

Keyword Arguments

To improve code readability, we should be as explicit as possible. We can achieve this in our functions by using `Keyword Arguments`:

```
>>> def say_hi(name, greeting):
...     print(f"{greeting} {name}")
...
>>> # without keyword arguments
>>> say_hi('John', 'Hello')
# Hello John

>>> # with keyword arguments
>>> say_hi(name='Anna', greeting='Hi')
# Hi Anna
```

Return Values

When creating a function using the `def` statement, you can specify what the return value should be with a `return` statement. A return statement consists of the following:

- The `return` keyword.
- The value or expression that the function should return.

```
>>> def sum_two_numbers(number_1, number_2):
...     return number_1 + number_2
...
>>> result = sum_two_numbers(7, 8)
```

```
>>> print(result)
# 15
```

Local and Global Scope

- Code in the global scope cannot use any local variables.
- However, a local scope can access global variables.
- Code in a function’s local scope cannot use variables in any other local scope.
- You can use the same name for different variables if they are in different scopes. That is, there can be a local variable named spam and a global variable also named spam.

```
global_variable = 'I am available everywhere'

>>> def some_function():
...     print(global_variable)  # because is global
...     local_variable = "only available within this function"
...     print(local_variable)
...
>>> # the following code will trow error because
>>> # 'local_variable' only exists inside 'some_function'
>>> print(local_variable)
Traceback (most recent call last):
  File "<stdin>", line 10, in <module>
NameError: name 'local_variable' is not defined
```

The global Statement

If you need to modify a global variable from within a function, use the global statement:

```
>>> def spam():
...     global eggs
...     eggs = 'spam'
...
>>> eggs = 'global'
>>> spam()
>>> print(eggs)
```

There are four rules to tell whether a variable is in a local scope or global scope:

1. If a variable is being used in the global scope (that is, outside all functions), then it is always a global variable.
2. If there is a global statement for that variable in a function, it is a global variable.
3. Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.

4. But if the variable is not used in an assignment statement, it is a global variable.

Lambda Functions

In Python, a lambda function is a single-line, anonymous function, which can have any number of arguments, but it can only have one expression.

This function:

```
>>> def add(x, y):
...     return x + y
...
>>> add(5, 3)
# 8
```

Is equivalent to the *lambda* function:

```
>>> add = lambda x, y: x + y
>>> add(5, 3)
# 8
```

Like regular nested functions, lambdas also work as lexical closures:

```
>>> def make_adder(n):
...     return lambda x: x + n
...
>>> plus_3 = make_adder(3)
>>> plus_5 = make_adder(5)

>>> plus_3(4)
# 7
>>> plus_5(4)
# 9
```