Functions in Python

What is a function in Python?

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function

```
def function_name(parameters):
	"""docstring"""
	statement(s)
```

1. Keyword `def` that marks the start of the function header.
2. A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of the function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7. An optional `return` statement to return a value from the function.

Example of a function

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

How to call a function in python?

Once we have defined a function, we can call it from another function, program, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

Note: In python, the function definition should always be present before the function call. Otherwise, we will get an error. For example,

```python
# function call
greet('Paul')

# function definition
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")

# Error: name 'greet' is not defined
```

Docstrings

The first string after the function header is called the docstring and is short for documentation string. It is briefly used to explain what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as the `__doc__` attribute of the function.

For example:

Try running the following into the Python shell to see the output.

```
>>> print(greet.__doc__)

    This function greets to
    the person passed in as
    a parameter
```

The return statement

The `return` statement is used to exit a function and go back to the place from where it was called.

Syntax of return

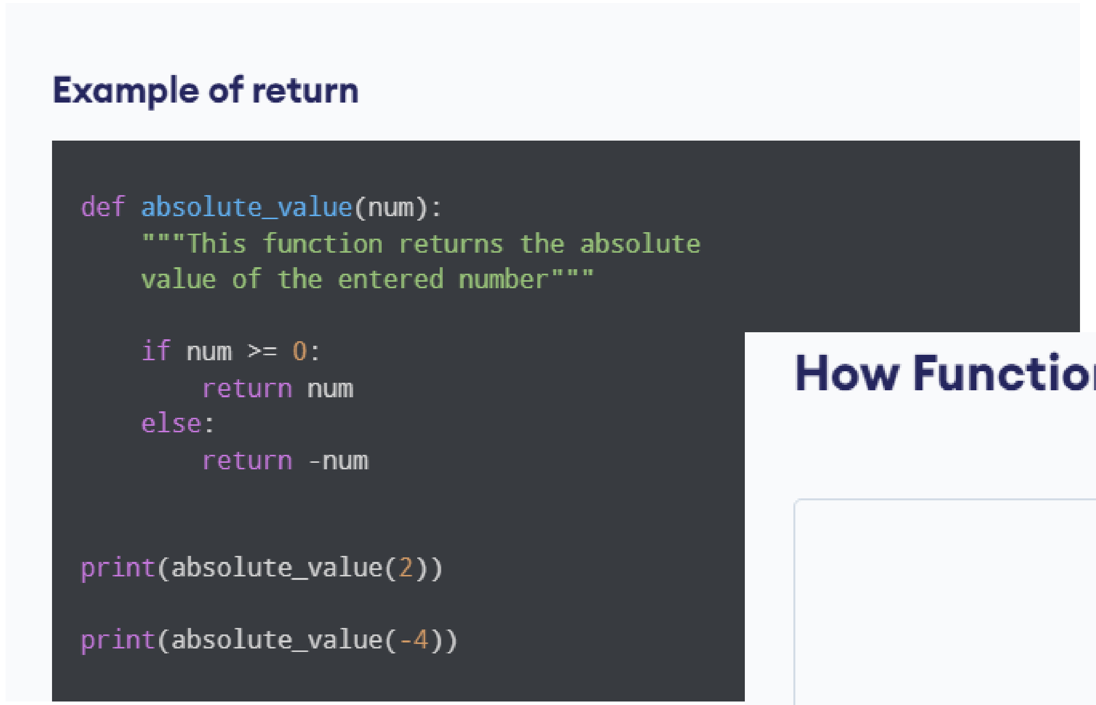
```
return [expression_list]
```

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the `return` statement itself is not present inside a function, then the function will return the `None` object.

For example:

```
>>> print(greet("May"))
Hello, May. Good morning!
None
```

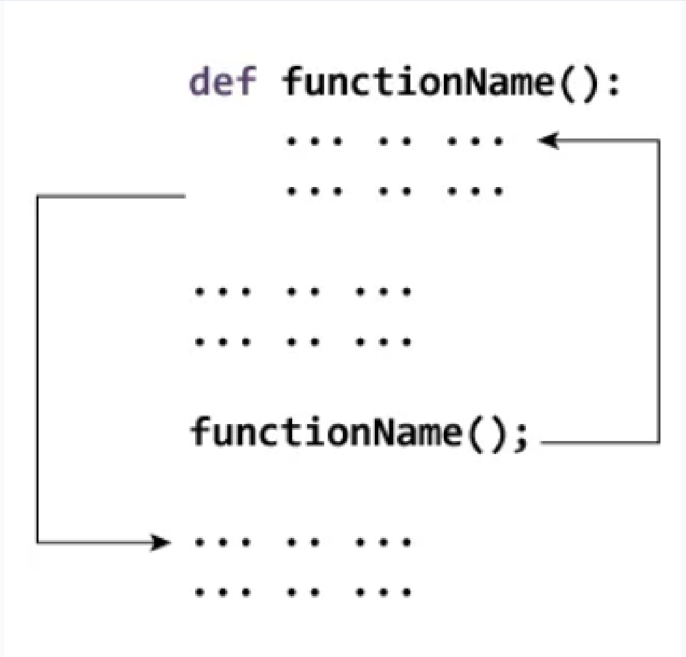
Here, `None` is the returned value since `greet()` directly prints the name and no `return` statement is used.



Example of return

```python
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num


print(absolute_value(2))

print(absolute_value(-4))
```

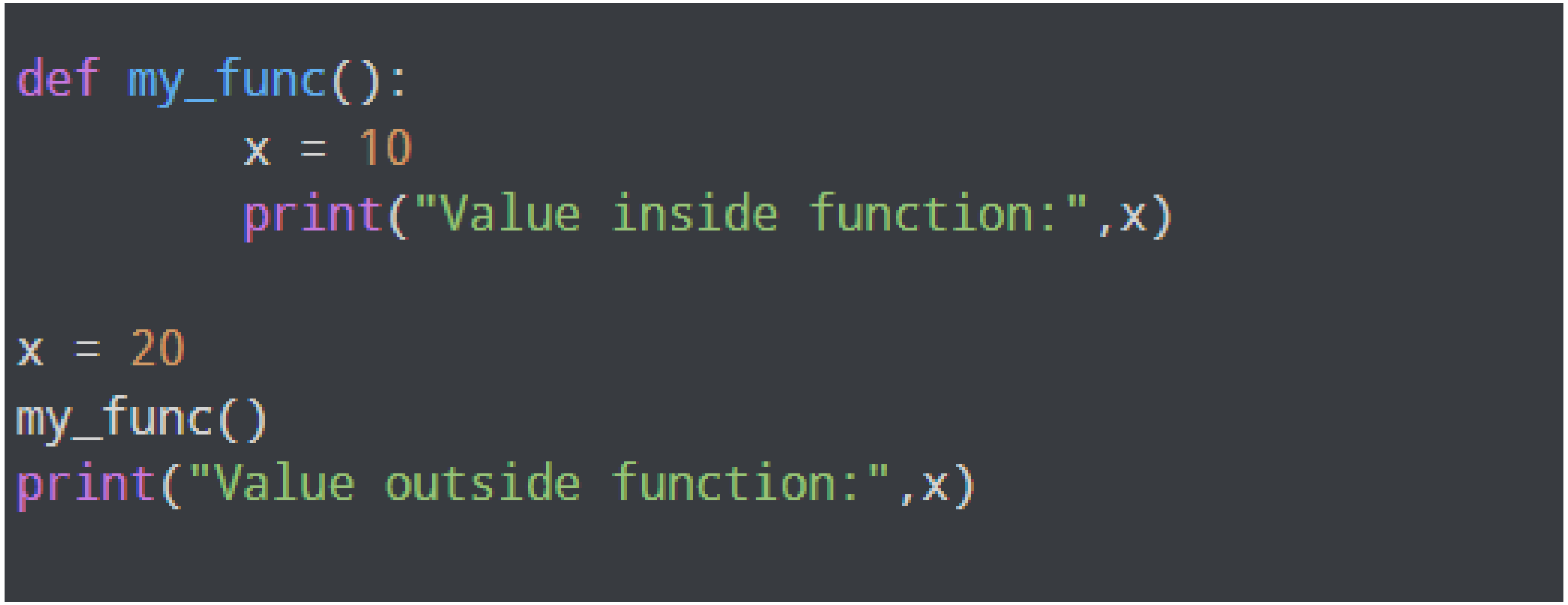
How Function works in Python?



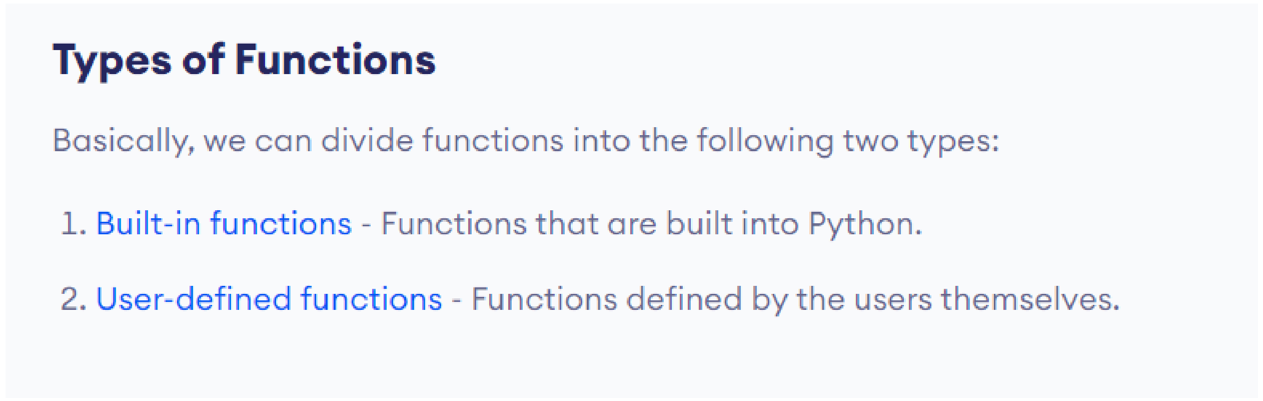
Working of functions in Python

Scope and Lifetime of variables

- Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope.
- The lifetime of a variable is the period throughout which the variable exists in the memory. The lifetime of variables inside a function is as long as the function executes.
- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.



```
def my_func():
        x = 10
        print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```



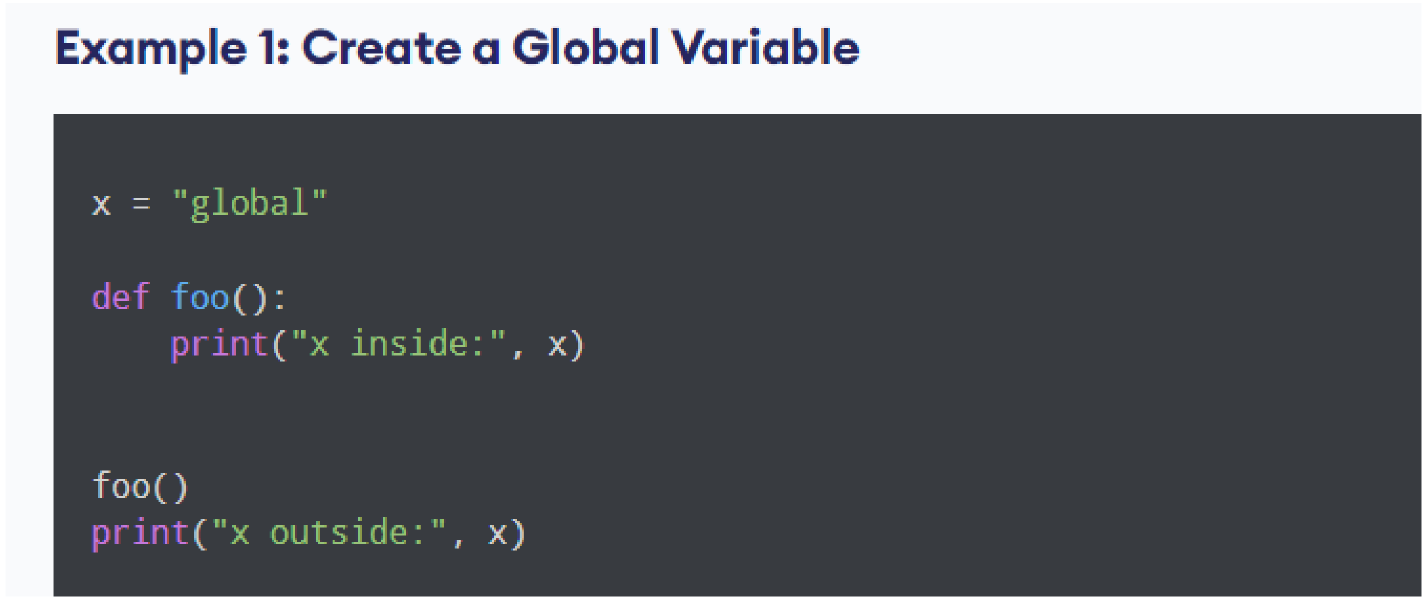
Types of Functions

Basically, we can divide functions into the following two types:

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.



Example 1: Create a Global Variable

```python
x = "global"

def foo():
    print("x inside:", x)


foo()
print("x outside:", x)
```

What if you want to change the value of `x` inside a function?

```python
x = "global"

def foo():
    x = x * 2
    print(x)

foo()
```

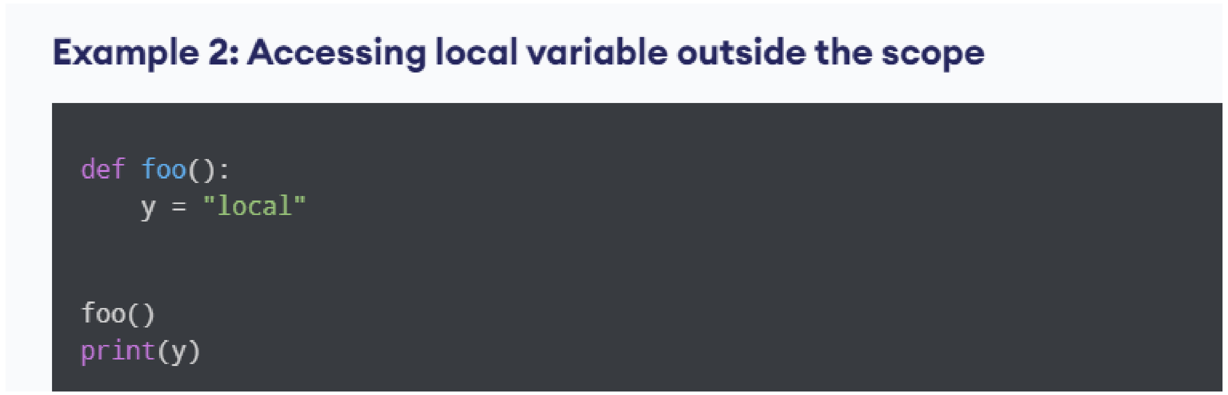
Output

```
UnboundLocalError: local variable 'x' referenced before assignment
```

The output shows an error because Python treats `x` as a local variable and `x` is also not defined inside `foo()`.

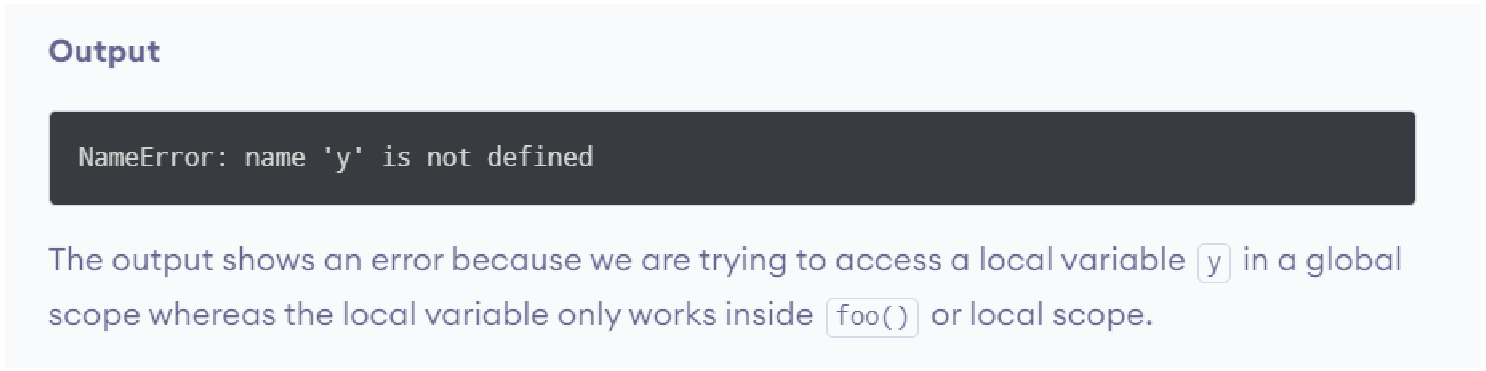
Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.



Example 2: Accessing local variable outside the scope

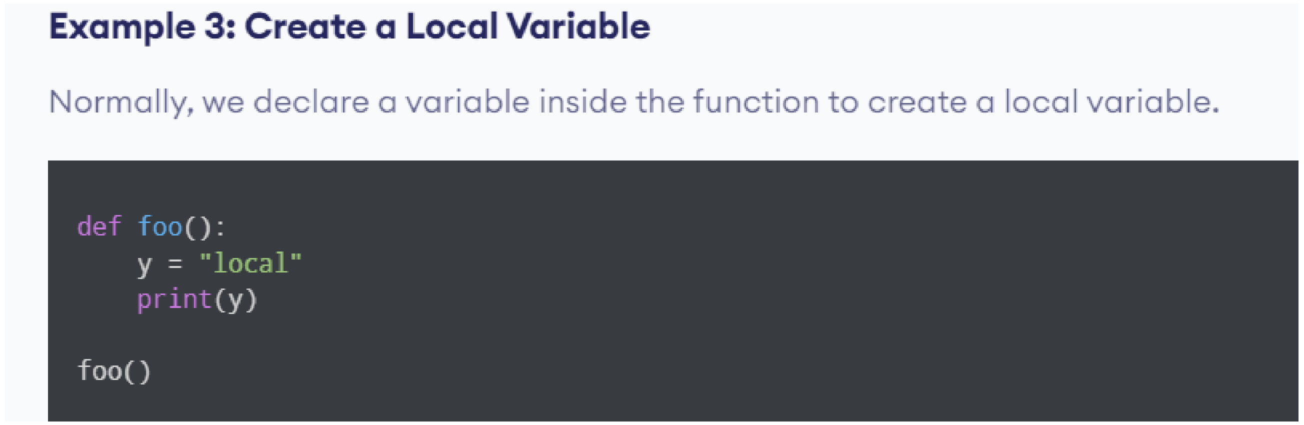
```python
def foo():
    y = "local"


foo()
print(y)
```



Output

```
NameError: name 'y' is not defined
```

The output shows an error because we are trying to access a local variable `y` in a global scope whereas the local variable only works inside `foo()` or local scope.

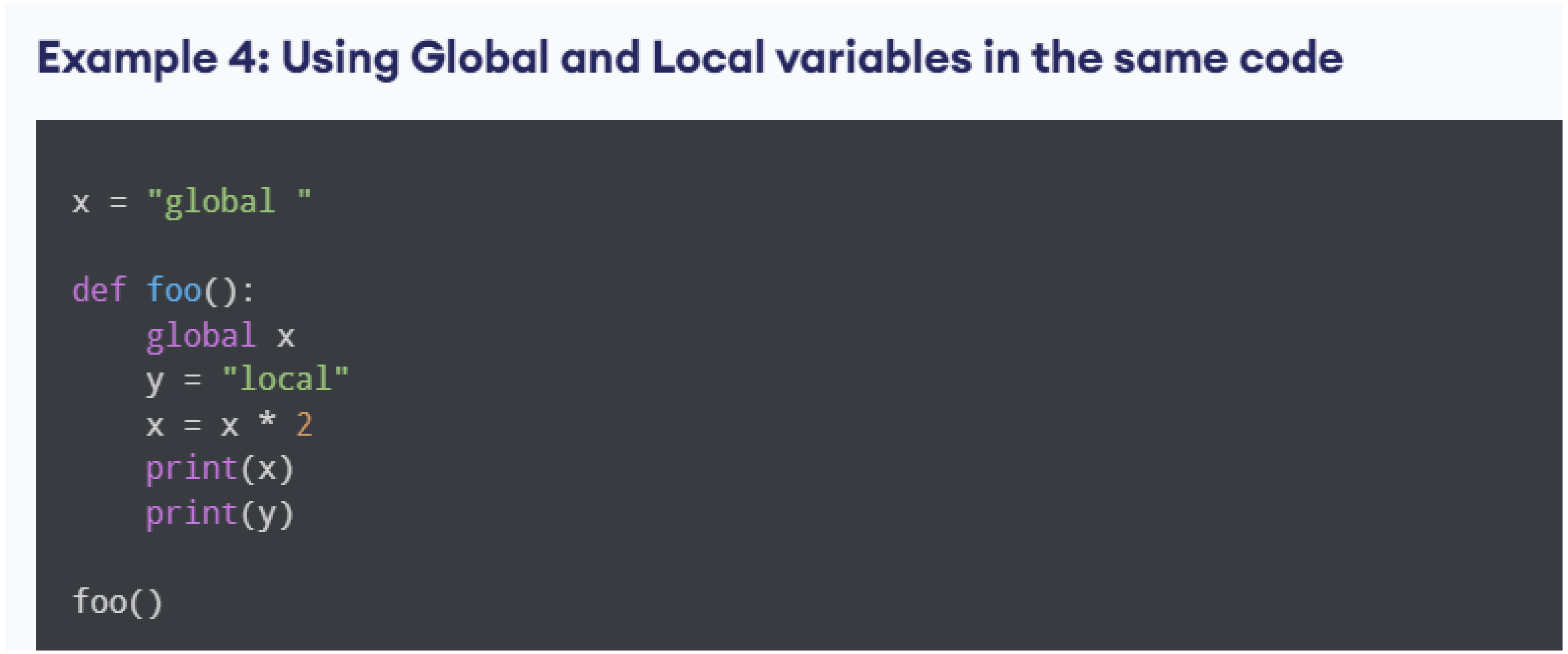


Example 3: Create a Local Variable

Normally, we declare a variable inside the function to create a local variable.

```python
def foo():
    y = "local"
    print(y)

foo()
```

Global and local variables



Example 4: Using Global and Local variables in the same code

```python
x = "global "

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

Example 5: Global variable and Local variable with same name

```python
x = 5

def foo():
    x = 10
    print("local x:", x)


foo()
print("global x:", x)
```

Creating Python Inner Functions

A function defined inside another function is known as an **inner function** or a **nested function**. In Python, this kind of function can access names in the enclosing function. Here's an example of how to create an inner function in Python:

Python >>>

```
>>> def outer_func():
...     def inner_func():
...         print("Hello, World!")
...     inner_func()
...

>>> outer_func()
Hello, World!
```

Another example of a more complex nested function:

```
def factorial(number):
    if not isinstance(number, int):
        raise TypeError("The number must be whole.")
    if number < 0:
        raise ValueError("The number must be non-negative.")
    #Factorial calculation
    def inner_factorial(number):
        if number <= 1:
            return 1
        return number * inner_factorial(number - 1)
    return inner_factorial(number)
 factorial(4)
```

OUTPUT

```
24
```