12. Scope of the Variable

October 19, 2022

1 Global Variables

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

[1]:
```
x = 10
```

[2]:
```
def foo():
    print("x inside function:", x)
```

[3]:
```
foo()
```

```
x inside function: 10
```

[4]:
```
print("x outside function:", x)
```

```
x outside function: 10
```

[5]:
```
foo()
print("x outside function:", x)
```

```
x inside function: 10
x outside function: 10
```

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

[10]:
```
x="global"
def foo():
    x=x*2
    print(x)
```

[11]:
```
foo()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In [11], line 1
----> 1 foo()
```

```
Cell In [10], line 3, in foo()
      2 def foo():
----> 3     x=x*2
      4     print(x)

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

- To make this work, we use theglobalkeyword.
- `global` keyword allows to modify the variable outside of the current scope.
- It is used to create a global variable and make changes to the variable in a local context.
- we may have some scenarios where we need to modify the global variable from inside afunction.

[12]:
```
c=0 # global variable
def add():
    global c
    c = c + 2 # increment by 2
    print("Inside add():", c)
```

[13]:
```
add()
print("In main:", c)
```

```
Inside add(): 2
In main: 2
```

- In the above program, we define c as a global keyword inside the `add()` function.
- Then, we increment the variable `c` by 2, i.e `c = c + 2`. After that, we call the `add()` function.Finally, we print the global variable `c`.
- As we can see, change also occurred on the global variable outside the function, c = 2.

1.2 Rules of `global` Keyword

- When we create a variable inside a function, it is local by default.
- When we define a variable outside of a function, it is global by default. You don't have touse global keyword.
- We use global keyword to read and write a global variable inside a function.
- Use of global keyword outside a function has no effect.

2 Local Variables

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

[14]:
```
def example2():
    y="local"
    print(y)
```

[15]:
```
example2()
```

```
local
```

[16]:
```
print(y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [16], line 1
----> 1 print(y)

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

- The output shows an error because we are trying to access a local variable y in a global scopewhereas the local variable only works inside `example2()` or local scope.

2.1 Using Global and Local variables in the same code

[17]:
```
x="global"
def foo():
    global x
    y="local"
    x=x*2
    print(x)
    print(y)
```

[18]:
```
foo()
```

```
globalglobal
local
```

- In the above code, we declare x as a global and y as a local variable in the `foo()`
- Then, we use multiplication operator `*` to modify the global variable x and we print both x and y.
- After calling the `foo()`, the value of x becomes global because we used the `x * 2` to print two times global.
- After that, we print the value of local variable y i.e local.

2.2 Global variable and Local variable with same name

[19]:
```
x=5

def square():
    x=10
    print("local x sq:", x**2)
```

[20]:
```
square()
print("global x:", x)
```

```
local x sq: 100
global x: 5
```

- In the above code, we used the same name `x` for both global variable and local variable.
- We get a different result when we print the same variable because the variable is declared inboth scopes, i.e. the local scope inside `square()` and global scope outside `square()`.
- When we print the variable inside `square()` it outputs local x. This is called the **local scope of the variable**.
- Similarly, when we print the variable outside the `square()`, it outputs global `x = 5`. This is called the **global scope of the variable**.

3 Nonlocal Variables

- Nonlocal variables are used in nested functions whose local scope is not defined.
- This means that the variable can be neither in the local nor the global scope.

[21]:
```
def outer():
    x="local"
    print("outer:", x)

    # nested function
    def inner():
        x="nonlocal"
        print("inner:", x)

    inner()
```

[22]:
```
outer()
```

```
outer: local
inner: nonlocal
```

[23]:
```
# Create a nonlocal variable
def outer():
    x="local"
    # print("outer:", x)

    # nested function
    def inner():
        x="nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)
```

[24]:
```
outer()
```

```
inner: nonlocal
```

```
outer: local
```

[25]:
```
# Create a nonlocal variable
def outer():
    x="local"
    print("outer:", x)

    # nested function
    def inner():
        nonlocal x
        x="nonlocal"
        print("inner:", x)

    inner()
```

[26]:
```
outer()
```

```
outer: local
inner: nonlocal
```

- In the above code, there is a nested `inner()` function.
- We use `nonlocal` keyword to create a nonlocal variable.
- The `inner()` function is defined in the scope of another function `outer()`.
- If we change the value of a nonlocal variable, the changes appear in the local variable.

[27]:
```
# Create a nonlocal variable
def outer():
    x="local"
    # print("outer:", x)

    # nested function
    def inner():
        nonlocal x
        x="nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)
```

[28]:
```
outer()
```

```
inner: nonlocal
outer: nonlocal
```