Indentation, Comments & Docstrings Scope and Rules

Table of contents

01

Indentation

Indentation – Before We Begin

Python

```
if True:
----print("This is indented and part of the if block.")
print("This is not indented and outside the if block.")
```

Indentation

- Indentation in Python is one of the most **critical** aspects of the language.
- Indentation refers to the **spaces** at the beginning of a line of code.
- In Python, indentation is used to determine the **grouping of statements**, which is essential for defining the structure of your code.
- Python does not enforce a **specific number** of spaces for indentation, but it must be consistent within a block of code.
- You typically indent after keywords like **if, for, while, def, and class**.

Indentation Levels/ Depth

Python

```
if True:
----if True:
--------for i in range(5):
------------print(i)
--------pass
----pass
```

Common Mistakes with Indentation

1. Inconsistent Indentation

Python

```
if True:
    print("This is fine.")
  print("This is not fine.") # This will cause an IndentationError
```

Correct Way

```
if True:
    print("This is fine.")
print("This is not fine.") # This will be smooth
```

Common Mistakes with Indentation

2. Missing Indentation

Python

```python
if True:
print("This is not fine.") # This will cause an IndentationError
```

Correct Way

```python
if True:
    print("This is fine.") # This will be smooth
```

02

Comments & Docstrings

Comments

Comments are text in your code that is **NOT executed** by Python. They are used to explain the code and make it more readable for humans.

Python supports **two** types of comments:

1. Single-line Comments
2. Multi-line Comments

Comments (Single-Line)

Single-line comments start with the **hash character (#)** and extend to the end of the line.

Standalone Single-Line Comment

```
# This is a single-line comment
x = 10
```

Inline Single-Line Comment

```
x = 10 # This is an inline comment
```

Comments (Multi-Line)

Python **doesn't** have a built-in syntax for multi-line comments.

However, you can use a series of single-line comments or a multi-line string (with triple quotes ‘ ‘ ‘ or “ “ “) that is not assigned to a variable.

NOTE: Multi-line string is technically not a comment but a string literal. However, if it's not assigned to any variable, Python will ignore it, making it functionally equivalent to a comment.

Example with single-line comments

```
# This is a multi-line comment
# that spans multiple lines.
x = 10
```

Example with a multi-line string

```
"""
This is a multi-line comment
that spans multiple lines.
"""
x = 10
```

Docstrings

Documentation Comments, aka, **Docstrings** are a specific kind of comment used to describe what a **function**, **class**, or **module** does.

They are written using triple quotes (**‘‘‘** or **“““**) and should be placed as the first statement in the **function, class, or module**.

Python

```python
def add(a, b):
    """
    This function adds two numbers.

    Parameters:
    a (int, float): The first number.
    b (int, float): The second number.

    Returns:
    int, float: The sum of the two numbers.
    """
    return a + b
```

Accessing Docstrings

Docstrings can be accessed using the **.__doc__** attribute or the **help()** function.

This allows you to easily view the documentation for any module, class, function, or method.

1. print(add.__doc__)

2. print(help(add))

Python

```python
def add(a, b):
    """
    This function adds two numbers.

    Parameters:
    a (int, float): The first number.
    b (int, float): The second number.

    Returns:
    int, float: The sum of the two numbers.
    """
    return a + b

print(add.__doc__) # You can use this
print(help(add))   # Or this
```

03

Scope and Rules

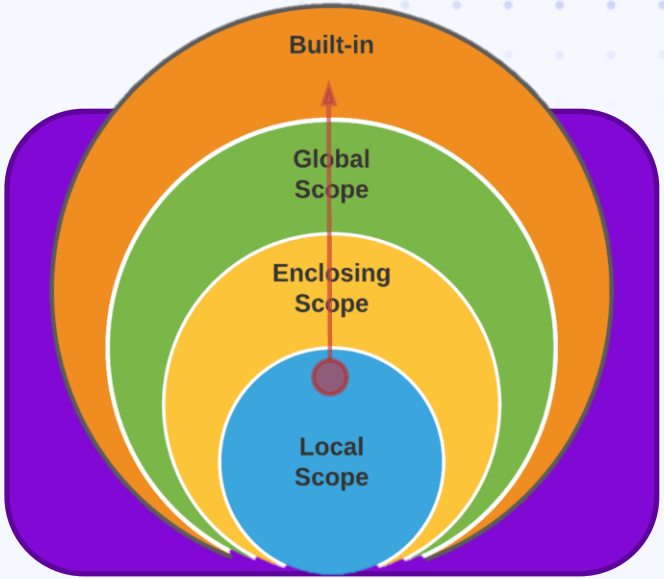
Scope

The region or context within a program where a variable is accessible is called **Scope**.

Variable scope is crucial because it determines where variables can be used throughout your code.

There are **four** types of scopes:

1. Local
2. Enclosed
3. Global
4. Built-in



Credit: Python Scope - Complete Tutorial (hands-on.cloud) - **Andrei Maksimov**

Local Scope

A variable defined **inside** a function is local to that function. It is only accessible within that function.

Python

```python
def my_function():
    x = 10  # x is a local variable
    print(x)  # prints 10

my_function()
print(x)  # This will raise an error because x is not accessible outside the function.
```

Enclosing Scope

This is relevant in **nested** functions. A variable in the enclosing (outer) function is accessible in the nested (inner) function.

Python

```python
def outer_function():
    x = 20  # x is in the enclosing scope

    def inner_function():
        print(x)  # inner_function can access x from outer_function

    inner_function()

outer_function()
```

Global Scope

Variable declared **outside** of the function (at the top-level) have global scope. They can be accessed from anywhere within the code.

Python

```python
x = 30  # x is a global variable

def my_function():
    print(x)  # Accessible here

my_function()
print(x)  # Also accessible here
```

Built-in Scope

These are names **preassigned** in Python. They include functions like **print()**, **len()**, etc., and are always accessible.

Python

```python
s="Python"
print(s) # Output is Python

def func():
    if(True):
        print("Inside the function") # Output is Inside the function

func()
```

Global Keyword

Global keyword can be used in these **two** scenarios:

1. For **creating** global variable within a local scope
2. For **modifying** a global variable within a local scope

Python

```python
def myfunc():

    # create a global variable
    global x
    x = 10

myfunc()

print(x)
```

Python

```python
x = 30

def my_function():
    global x
    x = 40  # modifies the global variable x
    print(x)  # prints 40

my_function()
print(x)  # prints 40
```

Nonlocal Keyword

The **nonlocal** keyword is used to modify a variable from the outer function inside the inner function.

Without Nonlocal	With Nonlocal

`def outer():`
`  x = "outer"`
 	
`  def inner():`
`  x = "inner"`
`  print(x)`
 	
`  inner()`
`  print(x)`
 	
`outer()`
	`def outer():`
	`  x = "outer"`
	
	`  def inner():`
	`# Refer to the variable in the enclosing scope`
	`  nonlocal x`
	`  x = "inner"`
	`  print(x)`
	
	`  inner()`
	`  print(x)`
	`outer()`

LEGB Rule

01 — **Local** → Names assigned within a function, and not declared as global or nonlocal

02 — **Enclosing** → Names in the local scope of any enclosing functions, from inner to outer.

03 — **Global** → Names assigned at the top level of a module or declared as global in a def within the file.

04 — **Built-in** → Names preassigned in the built-in names module (open, range, etc.).

Knowledge Reinforcement

1

Question

What is the output of the following code?

Python

```python
x = 10

def my_function():
    x = 20
    print(x)

my_function()
print(x)
```

Answer

a) 10 20

b) 20 10

c) 20 20

d) Error

1

Question

What is the output of the following code?

```python
x = 10

def my_function():
    x = 20
    print(x)

my_function()
print(x)
```

Answer

a) 10 20

b) 20 10

c) 20 20

d) Error

2

Question

Which of the following is the correct way to write a multi-line **comment** in Python?

Answer

a) # This is a comment

b) '''This is a multi-line comment'''

c) """This is a multi-line comment"""

d) All of them

2

Question

Which of the following is the correct way to write a multi-line **comment** in Python?

Answer

a) # This is a comment

b) '''This is a multi-line comment'''

c) """This is a multi-line comment"""

d) All of them

3

Question

Which of the following best describes a **docstring** in Python?

Answer

a) A comment used to describe complex code logic

b) A special type of comment used to document functions, classes, and modules

c) A multi-line comment used to prevent a section of code from running

d) A single-line comment written after code to explain it

3

Question

Which of the following best describes a **docstring** in Python?

Answer

a) A comment used to describe complex code logic

b) A special type of comment used to document functions, classes, and modules

c) A multi-line comment used to prevent a section of code from running

d) A single-line comment written after code to explain it

4

Question

What will happen if you mix tabs and spaces for **indentation** in a Python script?

Answer

A) The code will run correctly

B) It may produce an IndentationError

C) The code will be ignored

D) It will automatically convert to spaces

4

Question

What will happen if you mix tabs and spaces for **indentation** in a Python script?

Answer

A) The code will run correctly

B) It may produce an IndentationError

C) The code will be ignored

D) It will automatically convert to spaces

5

Question

What should you include in a function's **docstring**?

Answer

a) Function's purpose, parameters, return values, and any exceptions

b) A detailed breakdown of each line of code in the function

c) Only the function's return type

d) The code itself, explained line by line

5

Question

What should you include in a function's **docstring**?

Answer

a) Function's purpose, parameters, return values, and any exceptions

b) A detailed breakdown of each line of code in the function

c) Only the function's return type

d) The code itself, explained line by line

WATCH

Level up your coding with each episode in this focused Python series.

Next Video!

Practice SET