AIMERZ.ai
Aim.Act.Achieve

NOTES

Basic Python Syntax

Basic Python Syntax

Indentations: Understanding Python's Rules and Their Significance

Python uses **indentation** (spaces or tabs) to define the structure of code. Unlike other programming languages that use braces {} to mark code blocks, Python relies entirely on **whitespace** (indentation) for defining blocks of code like functions, loops, and conditional statements.

1. What is Indentation?

- **Indentation** refers to spaces at the beginning of a line of code.
- In Python, indentation is **mandatory** and used to group code blocks.
- Each level of indentation indicates a new block, and the number of spaces used must be consistent.

Imagine indentation as the layout of paragraphs in a document. Just like paragraphs are separated by indentation to organise thoughts, Python uses indentation to organise code.

2. Why is Indentation Important in Python?

- Python uses indentation to identify which code belongs to which block.
- Every time you write a loop, conditional, or function, the code inside must be indented correctly.
- If the indentation is wrong, Python will throw an error.

Notes:

- Indentation defines the scope (where a code block starts and ends).
- Improper indentation leads to syntax errors and makes your code unreadable.

3. Rules for Indentation

- Python uses a **minimum of one space** for indentation, but commonly, **4 spaces** are used as standard practice.
- You must be consistent with indentation throughout your code. You can't mix spaces and tabs in the same code block.

4. Example of Correct Indentation

Let's see how Python uses indentation to define code blocks in an if statement.

```python
age = 18

if age >= 18:       # Start of the 'if' block
    print("You are an adult.")   # Indented block of code
else:               # Start of the 'else' block
    print("You are a minor.")    # Indented block of code
```

Output:

```
You are an adult.
```

Explanation:

- The code inside the if and else blocks is indented by 4 spaces.
- This tells Python which lines belong to each block.

5. Example of Incorrect Indentation

If you don't follow consistent indentation, Python will raise an error:

```python
age = 18

if age >= 18:
print("You are an adult.")   # Missing indentation!
else:
    print("You are a minor.")
```

Output:

```
Python
“IndentationError: expected an indented block”
```

6. Indentation in Loops

Here’s an example of using indentation in a loop:

```
Python
for i in range(5):   # Start of the 'for' loop
    print(i)         # Indented block: belongs to the 'for' loop
print("Loop finished")  # Not indented: outside the loop
```

Output:

```
Python
0
1
2
3
4
Loop finished
```

Explanation:

- The code block inside the loop (`print(i)`) is indented.
- Once the indentation ends, the code (`print("Loop finished")`) is outside the loop.

7. Indentation in Functions

Indentation also works in functions. Every block of code inside a function must be indented.

```
Python
def greet(name):         # Function definition
    print(f"Hello, {name}")  # Indented: part of the function
    return name          # Indented: part of the function
```

AIMERZ.ai Aim.Act.Achieve

Explanation:

- All lines inside the `greet()` function are indented by 4 spaces.
- This shows Python that the code belongs to the function.

8. Mixing Tabs and Spaces

Python discourages mixing tabs and spaces for indentation. Choose one method (commonly spaces) and stick to it for the entire codebase.

Python Style Guide (PEP 8) recommends using 4 spaces per indentation level.

Common Errors to Avoid:

1. **Inconsistent Indentation:**

```python
if True:
    print("Hello")
  print("World")  # This line is incorrectly indented
```

Output:

```
Python

“IndentationError: unindent does not match any outer indentation level”
```

2. **Mixing Tabs and Spaces**:

```python
if True:
    print("Hello")  # Using spaces
 print("World")    # Using tabs (error!)
```

AIMERZ.ai Aim.Act.Achieve

Output:

```
Python

"TabError: inconsistent use of tabs and spaces in indentation"
```

Statements in Python: Understanding Different Types

In Python, a statement is a unit of code that performs a specific action. Statements are the building blocks of any Python program. There are several types of statements, including assignment statements and expression statements.

1. Assignment Statements

Assignment Statements are used to assign values to variables. The general format is:

```
Python
variable_name = value
```

Explanation:

Purpose: To store data in a variable.
Syntax: Use the = operator.
Variable: A container for storing data values.

Examples:

```
Python
x = 10        # Assigns the value 10 to variable x
name = "Alice" # Assigns the string "Alice" to variable name
```

Explanation:

- x is assigned the value 10.
- name is assigned the value "Alice".

2. Expression Statements

Expression Statements evaluate an expression and return a result. These statements typically involve calculations or evaluations.

Purpose: To perform operations and evaluations.
Syntax: Consists of expressions like arithmetic operations.

Examples:

```python
result = 5 + 3      # Evaluates the expression 5 + 3 and assigns the result to 'result'
print(result)       # Evaluates 'result' and outputs its value
```

Output:

```
8
```

Explanation:

- 5 + 3 is an expression that evaluates to 8.
- print(result) is an expression that outputs the value of result.

3. Conditional Statements

Conditional Statements control the flow of the program based on conditions. They are used to execute different blocks of code depending on certain conditions.

Purpose: To make decisions in code.
Syntax: Includes if, elif, and else.

Examples:

```python
age = 18

if age >= 18:
```

```
    print("Adult")
else:
    print("Minor")
```

Output:

```
Python
Adult
```

Explanation:

- Checks if age is greater than or equal to 18.
- Executes the corresponding block of code based on the condition.

4. Loop Statements

Loop Statements are used to repeat a block of code multiple times. There are mainly two types of loops: for and while.

Purpose: To repeat actions.
Syntax: Uses for or while keywords.

Examples:

```
Python
# 'for' loop
for i in range(3):
    print(i)
```

```
Python
# 'while' loop
count = 0
while count < 3:
    print(count)
    count += 1
```

Output for both will be same:

```
Python
0
1
2
```

Explanation:

- for loop iterates over a sequence of numbers.
- while loop continues until the condition is False.

5. Function Statements

Function Statements define reusable blocks of code. Functions are used to encapsulate code into a single unit that can be called multiple times.

Purpose: To define and reuse code.
Syntax: Uses the def keyword.

Examples:

```
Python
def greet(name):
    return f"Hello, {name}!"

message = greet("Alice")
print(message)
```

Output:

```
Python
Hello, Alice!
```

Explanation:

- Defines a function `greet` that returns a greeting message.
- Calls the function and prints the result.

AIHERZ.ai Aim.Act.Achieve

6. Import Statements

Import Statements are used to include external modules and libraries into your Python script.

Purpose: To use code from other modules.
Syntax: Uses the import keyword.

Examples:

```python
import math

print(math.sqrt(16))  # Uses the sqrt function from the math module
```

Output:

```
4.0
```

Explanation:

- Imports the `math` module and uses its `sqrt` function to compute the square root of `16`.

AIMERZ.ai Aim.Act.Achieve

THANK YOU

AIMERZ.ai
Aim.Act.Achieve