Lesson:

Python Keywords, Identifiers, Comments, Indentation and Statements

1) Explain the significance of Python keywords and provide examples of five keywords.

Ans:

In Python, keywords are reserved words that have special meaning within the language. They are fundamental to Python's syntax and structure, and they cannot be used as identifiers, such as variable names, function names, or any other identifier. Keywords define the language's control flow, data structure management, and more.

Significance of Python Keywords:

1. **Control Flow:** Keywords like `if`, `else`, `while`, and `for` help in controlling the flow of the program, allowing for conditional execution and looping.
2. **Function Definition:** Keywords such as `def` and `return` are used to define functions and return values from them.
3. **Data Types and Operations:** Keywords like `True`, `False`, and `None` define built-in constant values that represent certain data types.
4. **Error Handling:** Keywords like `try`, `except`, and `finally` are used to manage exceptions, allowing for error handling and ensuring code robustness.
5. **Class and Object Definition:** Keywords like `class` and `self` are used in

object-oriented programming to define classes and manage instances.

Examples of Five Python Keywords:

1. **`if`**
   - Used to make decisions in code based on conditions.

Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
```

   -
2. **`for`**
   - Used for iterating over a sequence (like a list, tuple, or string).

Example:

```
for i in range(5):
    print(i)
```

   -
3. **`def`**
   - Used to define a function.

Example:

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

   -
4. **`class`**
   - Used to define a new user-defined class.

Example:

```
class Dog:
    def __init__(self, name):
        self.name = name
```

-
5. **`return`**
   - Used to exit a function and return a value.

Example:

```
def add(a, b):
    return a + b
```

2) Describe the rules for defining identifiers in Python and provide an example.

Ans :

In Python, identifiers (names for variables, functions, classes, etc.) must follow these rules:

1. **Start with a Letter or Underscore**: Identifiers must begin with a letter (a-z, A-Z) or an underscore (`_`), but cannot start with a digit.
2. **Subsequent Characters**: After the first character, identifiers can have letters, digits (0-9), or underscores.
3. **Case-Sensitive**: Identifiers are case-sensitive, so `variable`, `Variable`, and `VARIABLE` are different.
4. **No Reserved Words**: Identifiers cannot be a Python keyword (e.g., `for`, `while`, `if`).

Example:

```
my_variable = 10  # Valid identifier

_variable1 = 20   # Valid identifier

1st_variable = 30 # Invalid identifier, cannot start
with a digit
```

3) What are comments in Python, and why are they useful? Provide an example.

Comments in Python are lines of text within your code that are ignored by the Python interpreter. They are useful for documenting your code, explaining complex logic, or leaving notes for yourself or other developers who might work on the code in the future.

Comments make code easier to understand, especially when revisiting it after some time or when collaborating with others.

Example:

```
x = 10  # This comment explains the variable assignment



def add(a, b):

    return a + b  # Adds two numbers and returns the result
```

4) Why is proper indentation important in
Python?

Ans:

Proper indentation in Python is crucial because it is a fundamental part of the language's syntax. Unlike many other programming languages, which use braces `{}` or keywords to define blocks of code, Python uses indentation to define the structure and flow of a program. Here's why it matters:

1. **Code Blocks Definition**: In Python, indentation is used to define blocks of code. For example, all the code inside a loop, function, or conditional statement must be indented consistently. Without proper indentation, Python will not know where the block of code starts and ends, leading to syntax errors.
2. **Readability**: Proper indentation improves the readability of your code, making it easier for others (and yourself) to understand the structure and flow of the program. Consistent indentation helps to clearly indicate which statements are part of loops, functions, or conditional branches.
3. **Avoiding Errors**: Incorrect or inconsistent indentation can lead to unexpected behavior in your program. Since Python relies on indentation to understand the code structure,

even a small mistake, such as using spaces instead of tabs or vice versa, can cause the program to fail.

5)What happens if indentation is incorrect in Python?

Ans:

In Python, correct indentation is crucial because it defines the structure of the code. Python uses indentation to indicate blocks of code, such as the body of loops, functions, conditionals, and classes. If the indentation is incorrect, several things can happen:

1. **IndentationError:** If your code is not indented consistently (e.g., mixing spaces and tabs or incorrect levels of indentation), Python will raise an `IndentationError`. This error occurs when Python expects a certain level of indentation but finds something else.

Logical Errors: Even if the code runs without errors, incorrect indentation can lead to logical errors, where the code does not perform as expected. For instance, if a line of code is incorrectly indented, it might be executed at the wrong time or skipped entirely.

6) Differentiate between expression and statement in Python with examples.

Ans:

In Python, an **expression** and a **statement** are both fundamental concepts, but they serve different purposes in a program.

Expression

An **expression** is a combination of values, variables, operators, and function calls that can be evaluated to produce a value. In other words, an expression is anything that returns a value.

Examples of expressions:

```
3 + 4            # An expression that evaluates to
7

len("hello")     # An expression that evaluates to
5

a * b            # An expression that evaluates to
the product of 'a' and 'b'
```

Statement

A **statement** is an instruction that the Python interpreter can execute. Statements perform an action and do not necessarily return a value (although they can). Statements can include expressions.

Examples of statements:

```
a = 3 + 4        # An assignment statement; it
performs the action of assigning the value 7 to 'a'

print(a)         # A print statement; it performs
the action of printing the value of 'a'

if a > 5:        # An if statement; it performs a
conditional action

    print("a is greater than 5")
```

Data Science With Generative AI
Course