Input/output and variables

Lesson Plan

PW SKILLS

Today's checklist:

- Variables In python
- Printing In python
- Input in Python
- Naming of variables in Python
- Comments in Python

1. Variables In python

Static Variables:

- Static variables typically retain their value throughout the execution of a program once assigned.
- They are often associated with a fixed memory location allocated during compile-time.
- Static variables may have limited scope and are not typically affected by the flow of execution or function calls.
- In some languages like C and C++, static variables within functions retain their value between function calls.
- Static variables are commonly used for values that need to be shared across multiple instances of a class or functions.
- They are initialized once and maintain their value until explicitly modified.

Dynamic Variables:

Dynamic variables, on the other hand, can change their value, type, or scope during the execution of a program.
They are often associated with memory allocation and deallocation at runtime.
Dynamic variables can be reassigned to different values or types at any point in the program.
Their scope may vary based on where they are declared or defined, and they may come into existence or be destroyed dynamically based on program flow.
Dynamic variables are commonly used in dynamically typed languages like Python, JavaScript, and Ruby, where variable types are determined at runtime.
Dynamic variables provide flexibility but may require careful handling to avoid unexpected behavior.

Python code example :-

```
x = 5  # x is an integer
print(x)  # Output: 5

x = "hello"  # Now x is a string
print(x)  # Output: hello

x = [1, 2, 3]  # Now x is a list
print(x)  # Output: [1, 2, 3]
```

Explanation:

- Here, `x` is a dynamic variable that changes its value and type during the execution of the program.
- Initially, `x` is assigned an integer value `5`, then reassigned to a string `"hello"`, and finally to a list `[1, 2, 3]`.

Dynamic variables in Python provide flexibility in programming but require careful handling to avoid unexpected behavior, especially when reassigning variables with different types or values.

2. Printing In python

Printing in Python is a fundamental aspect of programming, allowing you to display information, debug code, and communicate with users. Python provides various ways to print output, including basic printing and string formatting techniques. Here's an explanation suitable for students:

Basic Printing:

In Python, the `print()` function is used to display output to the console. It takes one or more arguments and prints them to the standard output.

```
print("Hello, world!")
```

Explanation:

- This line of code prints the string "Hello, world!" to the console.
- The `print()` function is a built-in function in Python that takes any number of arguments and displays them as text on the console.

Formatting String:

String formatting allows you to create dynamic strings by injecting values into placeholders within a string. Python provides several methods for string formatting:

1. **Using f-strings (formatted string literals):**

- Introduced in Python 3.6, f-strings provide a concise and readable way to format strings.
- You can place variables or expressions directly inside curly braces {} within a string, preceded by an 'f' or 'F' prefix.

```
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

Explanation:

- In this example, `{name}` and `{age}` are placeholders inside the string.
- The variables `name` and `age` are directly substituted into these placeholders when the string is formatted.

2. **Using the format() method:**

The format() method allows you to insert values into placeholders within a string by specifying the position of the placeholders

```
name = "Bob"
age = 30
print("My name is {} and I am {} years old.".format(name, age))
```

Explanation:

- In this example, `{}` is a placeholder inside the string.
- The `format()` method replaces each placeholder `{}` with the corresponding value provided as arguments to the method.

3. **Using % operator (old-style formatting):**

This method is less preferred compared to f-strings and format() method but is still supported in Python.

name = "Charlie"
age = 35
print("My name is %s and I am %d years old." % (name, age))

Explanation:

- `%s` and `%d` are placeholders for string and integer values, respectively.
- The `%` operator is used to format the string by replacing `%s` with the string value `name` and `%d` with the integer value `age`.

Examples and Outputs:

```
# Using f-strings
name = "Eve"
age = 40
print(f"My name is {name} and I am {age} years old.")

# Using format() method
name = "Frank"
age = 45
print("My name is {} and I am {} years old.".format(name, age))

# Using % operator (old-style formatting)
name = "Grace"
age = 50
print("My name is %s and I am %d years old." % (name, age))
```

Output:

My name is Eve and I am 40 years old.
My name is Frank and I am 45 years old.
My name is Grace and I am 50 years old.

3. Input in Python

In Python, the input() function is used to accept user input from the console. It allows a program to pause and wait for the user to enter some data, which can then be stored in a variable for further processing. Here's a beginner-friendly explanation of how to use the input() function:

Using the input() Function:
Basic Usage:

The input() function takes an optional prompt message as an argument, which is displayed to the user before waiting for input.
After displaying the prompt, the program waits for the user to type something and press Enter.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

Accepting Different Data Types:

By default, the input() function treats user input as a string.
You can convert the input to other data types using type casting functions like int(), float(), etc.

```
age = int(input("Enter your age: "))
print("You are", age, "years old.")
```

Handling User Input:

- You can use conditional statements or other logic to handle different user inputs.
- It's essential to validate and sanitize user input to ensure the program behaves correctly.

```
num = input("Enter a number: ")
if num.isdigit():
    num = int(num)
    print("You entered", num)
else:
    print("Invalid input. Please enter a valid number.")
```

Example :

```
# Accepting user input for name
name = input("Enter your name: ")
print("Hello, " + name + "!")

# Accepting user input for age
age = int(input("Enter your age: "))
print("You are", age, "years old.")

# Handling different types of user input
num = input("Enter a number: ")
if num.isdigit():
    num = int(num)
    print("You entered", num)
else:
    print("Invalid input. Please enter a valid number.")
```

Output (Example):

Enter your name: John
Hello, John!
Enter your age: 25
You are 25 years old.
Enter a number: 10
You entered 10
The input() function is a handy tool for interacting with users and creating interactive programs. However, remember to handle user input carefully to avoid errors and vulnerabilities in your code.

4. Identifiers and Naming of Variables in Python

Identifiers are like names for objects in Python. An identifier is a name used to identify a variable, function, class, module, or other objects in Python. Identifiers are case-sensitive and can consist of letters (both uppercase and lowercase), digits, and underscores (_). However, an identifier cannot start with a digit.

Naming Conventions:

1. **Keywords** : Keywords in Python are reserved words that serve special purposes in the language and cannot be used as ordinary identifiers (such as variable names, function names, etc.). Attempting to use a keyword as an identifier will result in a SyntaxError. For example, you cannot name a variable "if" or "while" because these are keywords used for control flow in Python.

Example:

```
# Incorrect usage
if = 5
```

2. **Character Restrictions :** Python identifiers can consist of lowercase letters (a-z), uppercase letters (A-Z), digits (0-9), and underscores (_). However, they cannot begin with a digit or consist solely of digits. This means that an identifier like "variable_1" is valid, but "1variable" is not.

Example:

```
# Valid identifier
variable_1 = 10

# Invalid identifier
1variable = 10  # SyntaxError: invalid syntax
```

3. **Starting with an Underscore :** Identifiers can begin with an underscore (_), making it a valid identifier. This convention is often used for private variables or functions within a module. While it's valid, it's also a convention in Python to indicate that the identifier is "private" and should not be accessed from outside the module directly.

Example:

```
_internal_variable = 10

def _internal_function():
    pass
```

4. **Case Sensitivity :** Python identifier names are case-sensitive, meaning that "abc" and "ABC" are considered as different identifiers. It's essential to pay attention to letter casing when using identifiers. Conventionally, lowercase is used for variable names, function names, and module names, CamelCase is used for class names, and uppercase is used for constants.

Example:

```
# Lowercase for variable names, function names, and module names
 my_variable = 10

 def my_function():
     pass

 import my_module

 # CamelCase for class names
 class MyClass:
     def __init__(self):
         pass

 # Uppercase for constants
 MY_CONSTANT = 100
```

By following these naming conventions, your Python code will be more readable, consistent, and easier to understand for both yourself and other programmers.

Examples of Valid Identifiers:

- `variable_name`
- `myFunction`
- `ClassName`
- `module_name`
- `CONSTANT_VALUE`

KEYWORDS

- Keywords in Python are predefined words that hold a special meaning and have a specific purpose within the Python language.
- These keywords are reserved and already defined by the Python compiler or interpreter, and they cannot be used as variable names or function names.
- Keywords are used to define the syntax and structure of the Python language, forming the building blocks for creating programs.
- Python keywords have predefined functionality and cannot be redefined or modified by the programmer.
- It's crucial to avoid using keywords as identifiers to prevent conflicts and syntax errors in your code.
- There are a total of 35 keywords. To know all the keywords run this code: `help(keywords)`.

```
[1] help('keywords')

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Examples of Keywords:

- `if`, `else`, `elif`: Used for conditional statements.
- `for`, `while`: Used for loop control.
- `def`, `class`: Used for defining functions and classes.
- `import`, `from`, `as`: Used for module import.
- `True`, `False`, `None`: Used for boolean values and null values.

5. Comments in Python

In Python, comments are used to provide explanations or annotations within the code. They are ignored by the Python interpreter during execution and serve the purpose of making the code more readable and understandable. Here's how you can use comments in Python along with an example:

Single-Line Comments:

Single-line comments start with the # character and continue until the end of the line. They are typically used for short comments or explanations on a single line.

```
# This is a single-line comment
print("Hello, World!")  # This is also a single-line comment
```

Multi-Line Comments:

Python doesn't have a built-in syntax for multi-line comments like some other programming languages. However, you can use multi-line strings (triple quotes) as a workaround to achieve a similar effect. These strings are not assigned to any variable and are essentially treated as comments because they are not used anywhere in the code.

```
"""
  This is a multi-line comment using a docstring.

  It serves as documentation for the function.
  You can write multiple lines of explanation here.
  """
```