20 May

Python Basic - 1

* 1. What are keywords in python? Using the keyword library, print all the python keywords.

Keywords are reserved words that have predefined meanings and cannot be used as variable names or identifiers. These keywords are used to define the syntax and structure of the language. Python has a specific library called `keyword` that provides functions and attributes related to keywords.

To print all the Python keywords using the `keyword` library, you can use the following code:

import keyword

all\_keywords = keyword.kwlist

print(all\_keywords)

O/P:

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

* 1. What are the rules to create variables in python?

In Python, variables are used to store data values. To create variables in Python, you need to follow certain rules:

1. Variable names must start with a letter (a-z, A-Z) or an underscore (\_). They cannot start with a number.

2. Variable names can only contain letters, numbers, and underscores. They cannot contain special characters such as @, $, %, etc.

3. Variable names are case-sensitive, meaning `myVariable` and `myvariable` are considered different variables.

4. Variable names should be descriptive and meaningful to indicate the purpose of the variable.

5. Python keywords cannot be used as variable names since they have special meanings.

6. It's best to follow the naming convention called "snake\_case" for variable names, where words are separated by underscores (e.g., `my\_variable`, `student\_name`).

* 1. What are the standards and conventions followed for the nomenclature of variables in python to improve code readability and maintainability?

In Python, there are some commonly followed standards and conventions for naming variables to improve code readability and maintainability. These conventions help make your code more consistent and easier to understand by other developers. Here are some widely accepted naming conventions:

1. Use descriptive and meaningful names: Choose variable names that accurately describe the purpose or content of the variable. Avoid using generic names like `a`, `b`, or `temp`, which can be confusing.

2. Follow snake\_case: Variable names should be written in all lowercase letters, and words should be separated by underscores. This convention is known as snake\_case. For example: `student\_name`, `total\_count`, `is\_valid`.

3. Avoid single-letter names: Unless the variable represents a well-known convention (e.g., `x`, `y` for coordinates), it is generally better to use descriptive names. This helps convey the purpose of the variable more clearly.

4. Use lowercase for constants: If you have a constant variable (a variable whose value should not change), use all uppercase letters with underscores between words. For example: `MAX\_SIZE`, `PI`, `DEFAULT\_COLOR`.

5. Be consistent with naming styles: It's important to maintain consistency in your naming conventions throughout your codebase. If you have multiple-word variable names, be consistent with capitalization and underscores. For example, don't mix `myVariable`, `MyVariable`, and `my\_variable` in the same code.

6. Avoid reserved keywords: Do not use Python's reserved keywords as variable names, as they have special meanings in the language. For example, avoid naming a variable `print`, `for`, or `class`.

7. Use meaningful abbreviations: If using long variable names becomes impractical, it's acceptable to use abbreviations as long as they are commonly understood. For example, `num\_items` can be abbreviated as `num\_items`, `numItems`, or `ni`, depending on the context and readability.

8. Consider the context: Variables should be named in a way that makes sense within the context of their usage. Use names that convey their purpose within the specific code block or function.

* 1. What will happen if a keyword is used as a variable name?

If you attempt to use a keyword as a variable name in Python, you will encounter a syntax error. Python's keywords are reserved and have predefined meanings within the language, so they cannot be used as identifiers or variable names.

* 1. For what purpose def keyword is used?

The def keyword is used to define a function. Functions are blocks of reusable code that perform a specific task. By using the def keyword, you can create your own custom functions with defined parameters, a block of code to execute, and an optional return value.

* 1. What is the operation of this special character ‘\’?

The backslash \ is used to escape characters that would otherwise have a special meaning within a string. When a backslash is followed by a specific character, it creates an escape sequence, which represents a special character or behavior.

Here are some common escape sequences and their meanings:

\\: Represents a literal backslash character.

\': Represents a single quote (apostrophe) character.

\": Represents a double quote character.

\n: Represents a newline character, causing a line break.

\t: Represents a tab character.

\r: Represents a carriage return character.

\b: Represents a backspace character.

\f: Represents a form feed character.

* 1. Give an example of the following conditions:

1. Homogeneous list
2. Heterogeneous set
3. Homogeneous tuple

Here are examples of the following conditions you mentioned:

(i) Homogeneous list:

A homogeneous list is a list where all elements have the same data type. Here's an example of a homogeneous list containing integers:

integer\_list = [1, 2, 3, 4, 5]

In the above example, all elements of the `integer\_list` are integers, making it a homogeneous list.

(ii) Heterogeneous set:

A heterogeneous set is a set where elements can have different data types. Here's an example of a heterogeneous set:

heterogeneous\_set = {1, 'hello', 3.14, True}

In the above example, the set `heterogeneous\_set` contains elements of different data types, including an integer, a string, a float, and a boolean.

(iii) Homogeneous tuple:

A homogeneous tuple is a tuple where all elements have the same data type. Here's an example of a homogeneous tuple containing strings:

string\_tuple = ('apple', 'banana', 'cherry', 'durian')

In the above example, all elements of the `string\_tuple` are strings, making it a homogeneous tuple.

It's important to note that in Python, lists, sets, and tuples can contain elements of different data types (heterogeneous) or the same data type (homogeneous). The examples provided demonstrate homogeneous lists and tuples as well as a heterogeneous set.

* 1. Explain the mutable and immutable data types with proper explanation & examples.

Data types can be classified as either mutable or immutable based on whether their values can be changed after they are created.

1. Immutable Data Types:

Immutable data types are those whose values cannot be changed once they are created. Any operation that appears to modify an immutable object actually creates a new object with the modified value. Some examples of immutable data types in Python include:

- \*\*Integers (`int`)\*\*: Numeric values representing whole numbers (e.g., 5, -3).

- \*\*Floats (`float`)\*\*: Numeric values representing decimal numbers (e.g., 3.14, -0.5).

- \*\*Strings (`str`)\*\*: Sequences of characters (e.g., "hello", 'world').

- \*\*Tuples (`tuple`)\*\*: Ordered collections of items (e.g., (1, 2, 3)).

- \*\*Frozen sets (`frozenset`)\*\*: Immutable sets (e.g., frozenset({1, 2, 3})).

Here's an example illustrating immutability:

# Immutable data types

num = 10

print(num) # Output: 10

num = 20 # Creating a new object with modified value

print(num) # Output: 20

name = "Alice"

print(name) # Output: Alice

name = "Bob" # Creating a new object with modified value

print(name) # Output: Bob

point = (3, 4)

print(point) # Output: (3, 4)

point = (5, 6) # Creating a new object with modified value

print(point) # Output: (5, 6)

In the above example, when we modify the value of the variables `num`, `name`, and `point`, new objects are created instead of modifying the existing objects. This behavior is because these data types are immutable.

2. Mutable Data Types:

Mutable data types are those whose values can be changed after they are created. This means you can modify the object in-place without creating a new object. Some examples of mutable data types in Python include:

- \*\*Lists (`list`)\*\*: Ordered collections of items (e.g., [1, 2, 3]).

- \*\*Dictionaries (`dict`)\*\*: Key-value mappings (e.g., {'a': 1, 'b': 2}).

- \*\*Sets (`set`)\*\*: Unordered collections of unique items (e.g., {1, 2, 3}).

Here's an example illustrating mutability:

# Mutable data types

my\_list = [1, 2, 3]

print(my\_list) # Output: [1, 2, 3]

my\_list.append(4) # Modifying the existing object

print(my\_list) # Output: [1, 2, 3, 4]

my\_dict = {'a': 1, 'b': 2}

print(my\_dict) # Output: {'a': 1, 'b': 2}

my\_dict['c'] = 3 # Modifying the existing object

print(my\_dict) # Output: {'a': 1, 'b': 2, 'c': 3}

my\_set = {1, 2, 3}

print(my\_set) # Output: {1, 2, 3}

my\_set.add(4) # Modifying the existing object

print(my\_set) # Output: {1, 2, 3, 4}

In the above example, we modify the values of the variables `my\_list`, `my\_dict

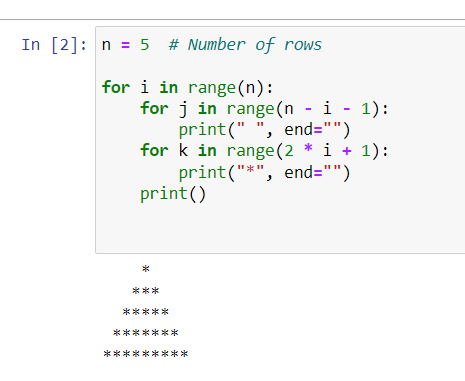
* 1. Write a code to create the given structure using only for loop.

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*



* 1. Write a code to create the given structure using while loop.

|||||||||

|||||||

|||||

|||

|

Program below

