**Python Basic – 1**

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

In Python, a keyword is defined as the reserved words that have special meaning and predefined functionalities in the language. These keywords cannot be used as identifiers and these are case-sensitive. **They are used to define the syntax and structure of the Python programming language.** As of python 3.11 version, there are a total of 3 keywords.

The following code prints all the python keywords using keyword library.

import keyword

all\_keywords = keyword.kwlist

print(all\_keywords)

The above code displays the following output:

**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 in memory locations and are essential for any programming language.

Rules to create variables in python:

* Variable names can consist of letters (both lowercase and uppercase), digits, and underscores.
* It must begin with a letter (a-z, A-Z) or an underscore (\_) but not with a digit.
* Variable names should not contain special characters like !, @, #, $, %, etc. The only allowed special character is the underscore (\_).
* Variable names cannot be the same as Python keywords.
* Variable names cannot contain spaces.

. By following these rules, you can create valid variable names in Python and write clean and readable code.

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

In Python, naming standards and conventions for variables is essential for writing code that is readable, maintainable, and consistent.

* Using descriptive and meaningful names for variables that reflect their purpose. Single-character names or abbreviations should be avoided unless they have a clear context.
* Using lowercase letters and separate words with underscores.
* If we have constants using all capitals with underscore is preferable for their names. This convention makes it clear that the variable is intended to be a constant.
* To indicate that a variable is intended for internal use within a class or module and not part of the public interface, prefix its name with a single underscore (\_). This is a convention, as Python does not have true private variables.
* For class names, use CamelCase (also known as CapWords) to differentiate them from functions and variables.
* Avoid using names that could be easily confused with built-in functions or modules, as this can lead to bugs and confusion.

By following these standards and conventions, the Python code will be more consistent and easier for others to read, understand, and maintain.

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

If you use a keyword as a variable name in Python, you will encounter a syntax error. Python's keywords are reserved for specific purposes and cannot be used as identifiers (variable names, function names, class names, etc.). Attempting to use a keyword as a variable name goes against the language's syntax rules and will result in a clear error message.

if = 10

print(if)

When you try to run this code, you will get a syntax error like this: **SyntaxError: invalid syntax**

Python will raise a SyntaxError because "if" is a keyword and cannot be used as a variable name. To resolve this error, you need to choose a different variable name that is not a reserved keyword.

1. **For what purpose def keyword is used?**

The def keyword in Python is used to define user-defined functions. Functions are blocks of organized, reusable code that perform a specific task or a set of tasks. Defining functions allows you to break down complex problems into smaller, manageable pieces, promoting code reusability and maintainability.

Here's the basic syntax for defining a function using the def keyword:

def function\_name(parameters):

# Function body (block of code)

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

In Python, the special character \ is called the "backslash" or "escape" character. It is used to escape special characters or create special sequences within strings and other literals. When the backslash is followed by certain characters, it creates escape sequences that represent specific characters or actions.

Here are some common escape sequences and their meanings:

**\n:** Newline - Inserts a new line in the string.

**\t:** Tab - Inserts a tab in the string.

1. **Give an example of the following conditions:**

**(i) Homogeneous list**

**(ii) Heterogeneous set**

**(iii) Homogeneous tuple**

**(i)** **Homogeneous List**: A homogeneous list is a list that contains elements of the same data type.

list = [1, 2, 3, 4, 5]

print(list) # Output: [1, 2, 3, 4, 5]

**(ii) Heterogeneous Set:** A heterogeneous set is a set that contains elements of different data types.

set = {1, "hello", 3.14, True}

print(set) # Output: {1, 'hello', 3.14, True}

**(iii) Homogeneous Tuple:** A homogeneous tuple is a tuple that contains elements of the same data type.

tuple = ("apple", "banana", "orange")

print(tuple) # Output: ('apple', 'banana', 'orange')

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

In Python, data types can be classified into two categories: mutable and immutable. These categories describe how the data stored in these objects can be modified after creation.

* **Mutable Datatypes** are objects whose contents can be changed or modified after they are created. This means you can add, remove, or modify elements without creating a new object.

**Example:** Lists and dictionaries

* **Immutable Datatypes** are objects whose contents cannot be changed after they are created. If you want to modify the data in an immutable object, you have to create a new object with the desired changes.

**Example:** Strings and tuples

tuple = (1, 2, 3)

# tuple[0] = 100

# This line will raise an error, as tuples are immutable

new\_tuple = tuple + (4,)

# Create a new tuple by concatenation

print(new\_tuple)

# Output: (1, 2, 3, 4)

* **List:** A list is a mutable, ordered collection of items in Python. It allows you to store multiple values of different data types within square brackets [], separated by commas. Lists are versatile and widely used for various purposes, such as holding data, iterating through elements, and modifying their content.

list = [1, 2, 3, 4]

list.append(5) # Add an element to the list

list[0] = 100 # Modify the first element

print(list) # Output: [100, 2, 3, 4, 5]

* **Dictionary:** A dictionary is a mutable, unordered collection of key-value pairs in Python. It uses curly braces {} to enclose its elements, and each key-value pair is separated by a colon :. The keys must be unique and immutable (strings, numbers, or tuples), while the values can be of any data type.

dict = {'a': 1, 'b': 2, 'c': 3}

print("Original Dictionary:", dict)

* **String:** A string is an immutable sequence of characters in Python. Strings are used to represent textual data and are enclosed within single quotes ' ', double quotes " ", or triple quotes ''' ''' or """ """. Being immutable means you cannot change the characters in a string after it is created.

string = "Hello"

print("Original String:", string)

* **Tuple:** A tuple is an immutable, ordered collection of items in Python. It uses parentheses () to enclose its elements, separated by commas. Tuples are similar to lists, but once created, their elements cannot be changed. They are often used for fixed collections of data.

tuple = (1, 2, 3)

print("Original Tuple:", tuple)

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

**\***

**\*\*\***

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

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

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

To create the given structure using only for loops, we can use nested loops.

n=5

for i in range(1, n+ 1):

for j in range(1, 2\*i):

print('\*', end='')

print()

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

**|||||||||**

**|||||||**

**|||||**

**|||**

**|**

To create the given structure using a while loop, we can use a decrementing loop counter to control the number of pipes '|' to be printed on each line.

n=5

while n>0:

count=n

while count>0:

print('|', end='')

count -= 1

print()

n -= 1