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

AnsKeywords in Python are reserved words that have predefined meanings and cannot be used as identifiers (such as variable names or function names). These keywords are part of the Python language syntax and are used to define control structures, data types, and other language elements.

In Python, you can access the list of keywords using the keyword library. Here's an example of how to print all the Python keywords using the keyword library:

import keyword

all\_keywords = keyword.kwlist

print(all\_keywords)

When you run this code, it will output a list of all the Python keywords:

['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']

These keywords are reserved and have special meanings in Python, so it's important not to use them as variable names or identifiers in your code.

**Q.2. What are the rules to create variables in python?**

Ans In Python, variables are used to store and manipulate data. Here are the rules to create variables in Python:

Variable Naming: Variable names in Python can contain letters (a-z, A-Z), digits (0-9), and underscores (\_). However, they must start with a letter or an underscore. Python is case-sensitive, so myVar and myvar are considered different variables.

Reserved Keywords: You cannot use reserved keywords as variable names because they have predefined meanings in Python. For example, you cannot use if, for, while, def, etc., as variable names.

Avoid Built-in Functions: It's good practice to avoid using the names of built-in functions, such as print, input, sum, etc., as variable names.

No Spaces or Special Characters: Variable names cannot contain spaces or special characters like !, @, #, $, etc. Use underscores (\_) to separate words in variable names for readability (e.g., my\_variable).

Descriptive and Readable: Choose meaningful and descriptive variable names that represent the purpose or content of the variable. This helps improve code readability and maintainability.

Case Styles: Python conventionally uses snake\_case for variable names. It means using all lowercase letters with words separated by underscores (e.g., my\_variable, user\_name).

**Q.3. What are the standards and conventions followed for the nomenclature of variables in**

**python to improve code readability and maintainability?**

Ans In Python, there are several standards and conventions followed for naming variables to improve code readability and maintainability. Here are some commonly accepted practices:

Snake Case: Python conventionally uses snake\_case for variable names. It means using all lowercase letters with words separated by underscores. For example: my\_variable, user\_name, total\_score.

Descriptive and Meaningful Names: Choose variable names that are descriptive and convey the purpose or content of the variable. This helps in understanding the code's functionality. For example, instead of using generic names like x or temp, use names like age, username, total\_score.

Avoid Abbreviations: Aim for clarity by avoiding unnecessary abbreviations. Use descriptive names even if they are slightly longer. For example, use customer\_name instead of cust\_nm.

Singular and Plural Distinction: Use singular names for variables that represent a single entity and plural names for variables that represent a collection or multiple entities. For example, car and cars, person and people.

Constants: If you have variables that are intended to be constants (values that do not change), use uppercase letters with underscores to separate words. For example: PI, MAX\_SIZE, DEFAULT\_NAME.

Avoid Reserved Keywords: Do not use reserved keywords as variable names. These are words that have predefined meanings in Python. For example, avoid using names like if, for, while, def, etc.

Consistency: Maintain consistent naming conventions throughout your codebase. This helps in making the code more readable and reduces confusion. For example, if you use snake\_case for variable names, stick to it consistently.

**Q.4. What will happen if a keyword is used as a variable name?**

Ans it will result in a syntax error. keywords are reserved words that have predefined meanings and cannot be used as identifiers (such as variable names, function names, or class names).

When you attempt to use a keyword as a variable name, will raise a syntax error, indicating that the keyword cannot be used as an identifier. This error prevents the misuse of keywords and helps maintain the integrity of the language syntax.

For example, let's say you try to use the keyword if as a variable name:

if = 10

In this case, raise a SyntaxError with the message "invalid syntax" because if is a keyword and cannot be used as a variable name.

To avoid such errors, it's important to choose variable names that are not reserved keywords and follow the naming conventions mentioned earlier.

**Q.5. For what purpose def keyword is used?**

**Ans The def keyword marks the beginning of a function definition and specifies the name of the function, followed by a set of parentheses and a colon.**

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

**def function\_name(parameters):**

**Let's break down the components:**

**def: This keyword indicates the start of a function definition.**

**function\_name: This is the name of the function, which you choose and should follow the variable naming rules.**

**parameters: These are optional inputs that the function can accept. They are enclosed in parentheses and can be used within the function body to perform specific operations.**

**:: The colon indicates the start of the function's code block.**

**Function body: This is the indented block of code that defines what the function does. It can contain any valid Python statements and can include control structures, variable declarations, calculations, and more.**

**Here's an example of a simple function that adds two numbers together:**

**def add\_numbers(a, b):**

**sum = a + b**

**return sum**

**In this example, the add\_numbers function takes two parameters a and b. It calculates their sum and returns the result using the return keyword.**

**You can then call the function by its name and pass the necessary arguments:**

**result = add\_numbers(3, 5)**

**print(result)**

**Q.6. What is the operation of this special character ‘\’?**

Ans . It is used to escape or indicate the presence of special characters or sequences within strings and characters. The \ character is followed by another character to represent a specific meaning.

Here are some common uses of the escape character \

Escape Sequences: The escape character is used to represent special characters or sequences within strings. For example:

\': Represents a single quote within a string.

\": Represents a double quote within a string.

\\: Represents a backslash within a string.

\n: Represents a newline character.

\t: Represents a tab character.

\r: Represents a carriage return character.

Example:

print('He said, "It\'s raining outside."')

print("Hello\nWorld")

Raw Strings: If you want to treat a string literally and disable the interpretation of escape sequences, you can use the r prefix before the string. This creates a raw string. Example:

path = r'C:\Users\Username\Documents'

print(path)

Line Continuation: The backslash \ can be used to continue a single logical line of code onto the next physical line. This is useful for improving code readability. Example:

total = 10 + \

20 + \

30

print(total)

**Q.7. Give an example of the following conditions:**

**(i) Homogeneous list**

**(ii) Heterogeneous set**

**(iii) Homogeneous tuple**

**Ans i) Homogeneous List:**

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

**my\_list = [1, 2, 3, 4, 5]**

**In this example, all the elements in the list (1, 2, 3, 4, 5) are integers, making it a homogeneous list.**

**(ii) Heterogeneous Set:**

**A heterogeneous set in a set where the elements can be of different data types. Here's an example of a heterogeneous set containing different types of elements:**

**my\_set = {1, 'two', 3.0, (4, 5)}**

**In this example, the set my\_set contains elements of different types, including an integer (1), a string ('two'), a floating-point number (3.0), and a tuple ((4, 5)). This makes it a heterogeneous set.**

**(iii) Homogeneous Tuple:**

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

**my\_tuple = ('apple', 'banana', 'cherry', 'date')**

**Q.8. Explain the mutable and immutable data types with proper explanation & examples.**

Ans Mutable Data Types:

Mutable data types are those whose values can be modified after they are created.

When you modify a mutable object, it is modified in-place, meaning the original object is changed.

Examples of mutable data types in Python include lists, sets, and dictionaries.

Example 1: Lists

my\_list = [1, 2, 3]

my\_list[0] = 10

print(my\_list)

In the above example, the first element of the list is modified from 1 to 10.

Example 2: Dictionaries

my\_dict = {'name': 'John', 'age': 25}

my\_dict['age'] = 30

print(my\_dict) }

In this example, the value corresponding to the key 'age' in the dictionary is changed from 25 to 30.

Immutable Data Types:

Immutable data types are those whose values cannot be modified after they are created.

When you modify an immutable object, a new object is created with the modified value.

Examples of immutable data types in Python include integers, floats, strings, and tuples.

Example 1: Strings

my\_string = "Hello"

new\_string = my\_string + " World"

print(new\_string)

In this example, the concatenation of the original string and the new string creates a new string object.

Example 2: Tuples

my\_tuple = (1, 2, 3)

new\_tuple = my\_tuple + (4,)

print(new\_tuple)

**Q.9. Write a code to create the given structure using only for loop.**

**\***

**\*\*\***

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

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

ANS for i in range(1, 6): # Number of rows

for j in range(i):

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

print()

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

In this code, we have an outer for loop that iterates over the rows from 1 to 5. For each row, there is an inner for loop that prints the asterisk (\*) a certain number of times based on the current row number.

The inner loop iterates j from 0 to i-1, where i represents the current row number. This ensures that the number of asterisks printed on each row corresponds to the row number itself. The end='' argument in the print() function is used to avoid printing a newline after each asterisk.

**Q.10. Write a code to create the given structure using while loop.**

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

**|||||||**

**|||||**

**|||**

**|**

ANS the code to create the given structure using a while loop:

row = 5

while row >= 1:

print('|' \* row)

row -= 1

|||||||||

|||||||

|||||

|||

|

In this code, we initialize the variable row with the value 5, representing the number of rows in the structure. We enter a while loop that continues as long as row is greater than or equal to 1.

Within the loop, we print the string '|' repeated row times using the print() function. Then, we decrement the value of row by 1 in each iteration.