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

Ans: In Python, keywords are reserved words that have predefined meanings and cannot be used as identifiers (such as variable names or function names) in the program. These keywords are part of the Python language syntax and serve specific purposes.

To print all the Python keywords, you can use the keyword library in Python.

Ex: import keyword

# Get all the Python keywords

keywords = keyword.kwlist

# Print all the keywords

for keyword in keywords:

print(keyword)

When you run this code, it will output 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

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

In Python, there are certain rules and conventions to follow when creating variables. Here are the main rules for creating variables in Python:

Variable Names: Variable names can contain letters (a-z, A-Z), digits (0-9), and underscores (\_). They must start with a letter or an underscore. Python is case-sensitive, so myVariable and myvariable are considered different variables.

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

Naming Conventions: It is recommended to follow certain naming conventions to improve code readability. The most common convention is to use lowercase letters and underscores to separate words in variable names (e.g., my\_variable). This convention is known as "snake\_case."

Meaningful Names: Choose variable names that are descriptive and meaningful. This helps in understanding the purpose of the variable when reading the code.

Avoid Starting with Digits: Variable names should not start with a digit. For example, 2variable is not a valid variable name, but variable2 is.

No Spaces or Special Characters: Variable names cannot contain spaces or special characters like !, @, $, %, etc. They can only contain letters, digits, and underscores.

Length Limit: There is no strict limit on the length of variable names in Python, but it is good practice to keep them reasonably short and meaningful.

EX: name , age , my\_variable.

**Q.3. What are the standards and conventions followed for the nomenclature of variables in python to improve code readability and maintainability?**

In Python, there are standard conventions and naming conventions that are widely followed to improve code readability and maintainability. These conventions help make the code more consistent and easier to understand, especially when working on collaborative projects or when sharing code with others. Here are some commonly followed standards and conventions for variable naming in Python:

Use descriptive names: Choose variable names that are descriptive and convey the purpose or meaning of the variable. This makes the code more self-explanatory and easier to understand.

Follow snake\_case: The most common naming convention for variables in Python is to use lowercase letters and underscores to separate words in variable names. For example: my\_variable, number\_of\_students, user\_name. This convention is known as "snake\_case."

Avoid single-letter names: While single-letter variable names like x, y, or i may be used in certain contexts (e.g., mathematical calculations or loop indices), it is generally recommended to use more meaningful names that reflect the purpose of the variable.

Use nouns or noun phrases: Variables typically represent objects, values, or concepts, so it is advisable to use nouns or noun phrases for variable names. For example: name, age, customer\_list.

Be consistent and readable: Maintain consistent naming conventions throughout your codebase. This helps improve readability and makes it easier for others to understand and navigate the code. For example, if you use is\_active for one variable, avoid using active\_status for a similar purpose elsewhere.

Avoid reserved keywords: Do not use Python's reserved keywords (such as if, for, while, etc.) as variable names. Using reserved keywords can lead to syntax errors or confusion.

Use meaningful abbreviations (when appropriate): If an abbreviation is widely recognized and understood, it can be used in variable names to improve readability and reduce the length of long variable names. However, be cautious not to use overly cryptic or ambiguous abbreviations that may confuse others.

Use uppercase for constants: By convention, constants (variables whose values are not intended to change) are often written in uppercase letters, with words separated by underscores. For example: MAX\_VALUE, PI, CONFIG\_FILE\_PATH.

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

If a keyword is used as a variable name in Python, it will result in a syntax error. Keywords in Python have predefined meanings and are reserved for specific purposes within the language syntax. Attempting to use a keyword as a variable name goes against the rules of the language and will cause the interpreter to raise an error.

Ex: if = 10 # SyntaxError: invalid syntax

In this case, if is a reserved keyword used for conditional statements in Python, so assigning a value to it as a variable name is not allowed.

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

In Python, the def keyword is used to define a function. Functions are blocks of code that perform a specific task and can be reused throughout a program. The def keyword, followed by the function name and a set of parentheses, marks the beginning of a function definition.

EX: def function\_name(parameters):

def: The def keyword indicates the start of a function definition.

function\_name: This is the name given to the function. Choose a descriptive and meaningful name that reflects the purpose of the function.

parameters: Parameters are optional. They are variables declared within the parentheses that allow the function to accept input values or arguments.

function body: This is the indented block of code that defines the functionality of the function. It contains the instructions or operations that the function performs when called.

def add\_numbers(a, b):

In this example, the function is named add\_numbers, and it takes two parameters a and b. The function body adds the values of a and b and stores the result in the variable sum. The return statement is used to specify the value that the function should output.

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

The special character \ in Python is called the backslash. It is used as an escape character to include special characters or sequences in strings and to specify certain types of characters.

Here are some common uses of the backslash in Python:

Escape Sequences: The backslash is used to create escape sequences within strings. An escape sequence is a combination of a backslash followed by one or more characters that represent a special character or sequence. For example:

\': Represents a single quote character.

\": Represents a double quote character.

\\: Represents a backslash itself.

\n: Represents a newline character.

\t: Represents a tab character.

\r: Represents a carriage return character**.**

Ex: print("This is a string with a \"quoted\" word.") # Output: This is a string with a "quoted" word.

print('This string has a \nnew line.')

# Output:

# This string has a

# new line.

**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 that contains elements of the same data type. Here's an example of a homogeneous list that contains integers

**Ex:** **numbers = [1, 2, 3, 4, 5]**

(ii) Heterogeneous Set:

A heterogeneous set is a set that contains elements of different data types. Here's an example of a heterogeneous set:

Ex: my\_set = {1, "apple", 3.14, True}

In this example, the my\_set set contains elements of different data types, including an integer (1), a string ("apple"), a float (3.14), and a boolean value (True), making it a heterogeneous set.

(iii) Homogeneous Tuple:

A homogeneous tuple is a tuple that contains elements of the same data type. Here's an example of a homogeneous tuple that contains strings:

Ex: fruits = ("apple", "banana", "orange", "mango")

It's important to note that in Python, lists ([]), sets ({}), and tuples (()) can contain elements of different data types. The homogeneous or heterogeneous nature of these data structures depends on the data types of the elements they contain.

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

In Python, data types can be classified as either mutable or immutable based on whether their values can be changed after they are created. Understanding the difference between mutable and immutable data types is important as it affects how the data is stored, accessed, and modified.

Immutable Data Types:

Immutable data types are those whose values cannot be changed once they are assigned. If any modification is made to an immutable object, a new object is created with the modified value. Examples of immutable data types in Python include:

Numeric data types: int, float, complex

String: str

Tuple: tuple

FrozenSet: frozenset

string = "Hello"

print(string) # Output: Hello

# Attempting to modify the string

string[0] = 'J' # Raises a TypeError

# Creating a new string

new\_string = string + " World"

print(new\_string) # Output: Hello World

In the example above, the attempt to modify the string by assigning a new character at index 0 results in a TypeError. This is because strings are immutable, so we cannot change their individual characters. Instead, a new string is created by concatenating the original string with another string.

Mutable Data Types:

Mutable data types are those whose values can be modified after they are assigned. Changes made to mutable objects affect the original object directly. Examples of mutable data types in Python include:

List: list

Dictionary: dict

Set: set

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

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

# Modifying the list

my\_list[2] = 5

print(my\_list) # Output: [1, 2, 5, 4]

# Appending an element

my\_list.append(6)

print(my\_list) # Output: [1, 2, 5, 4, 6]

In the example above, the list my\_list is modified by assigning a new value to the element at index 2. The modification is reflected directly in the original list. Additionally, an element is appended to the list using the append() method.

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

\*\*\*

\*\*\*\*\*

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

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

Input: rows = 5

for i in range(rows):

for j in range(i+1):

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

print()

The outer loop for i in range(rows) controls the number of rows in the pattern. In this case, there are 5 rows.

The inner loop for j in range(i+1) controls the number of stars to be printed in each row. The value of i+1 determines the number of stars in each row, starting from 1 in the first row, 2 in the second row, and so on.

The statement print("\*", end="") prints a single star without a newline character.

After each row is printed, the print() statement is used to move to the next line, creating a new row.

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

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

**|||||||**

**|||||**

**|||**

**|**

Input: rows = 5

while rows > 0:

count = rows

while count > 0:

print("|", end="")

count -= 1

print()

rows -= 1

The variable rows is initially set to 5, representing the number of rows in the pattern.

The outer while loop while rows > 0 controls the number of rows in the pattern. It continues as long as the value of rows is greater than 0.

Inside the outer while loop, the variable count is set to the value of rows. This variable keeps track of the number of vertical bars to be printed in each row.

The inner while loop while count > 0 controls the number of vertical bars to be printed in each row. It continues as long as the value of count is greater than 0.

Inside the inner while loop, the statement print("|", end="") is used to print a single vertical bar without a newline character.

After each row is printed, the print() statement is used to move to the next line, creating a new row.

Finally, the rows variable is decremented by 1 at the end of each iteration of the outer while loop, ensuring that the pattern decreases in size with each row.