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

**Ans:** Keywords in Python are reserved words that have special meanings and purposes within the language. These words cannot be used as identifiers (variable names, function names, class names, etc.) because they are already used by Python for specific tasks and operations. Keywords play a fundamental role in defining the syntax and structure of the Python language. We can use the **keyword** library in Python to access and print all the keywords.

Python code

import keyword

print(keyword.kwlist)

['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?**

**Ans:**In Python, variables are used to store data values.

**Naming Rules**:

* + Variable names must start with a letter (a-z, A-Z) or an underscore **\_**.
  + The subsequent characters can be letters, digits (0-9), or underscores.
  + Variable names are case-sensitive, meaning **myVar** and **myvar** are different variables.

1. **Naming Conventions**:
   * Use descriptive names that reflect the purpose of the variable.
   * Use lowercase letters for variable names (e.g., **my\_variable**).
   * Use underscores to separate words in variable names for improved readability (e.g., **word\_count**).
2. **Reserved Keywords**:
   * Avoid using reserved keywords as variable names. Keywords are words that have special meanings in Python and cannot be used for other purposes (e.g., **if**, **else**, **while**, **for**, **def**, etc.).
3. **Avoid Starting with Digits**:
   * Variable names cannot start with a digit (0-9). They must start with a letter or underscore.
4. **No Special Characters**:
   * Special characters (such as **!**, **@**, **#**, **$**, **%**, etc.) cannot be used in variable names.
5. **No Spaces**:
   * Variable names cannot contain spaces. If we need to represent multiple words, use underscores or capitalize the first letter of each word (CamelCase).
6. **Use Meaningful Names**:
   * Choose variable names that make the purpose of the variable clear and understandable.

**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, following standardized naming conventions for variables helps improve code readability and maintainability. The conventions are not strict rules enforced by the language, but they are widely adopted practices that make our code more consistent and understandable, especially when collaborating with others. Here are some common naming conventions:

1. **Snake Case for Variable Names**:
   * Use lowercase letters for variable names.
   * Separate words using underscores (**\_**) to improve readability.
   * Example: **word\_count**, **user\_input**, **total\_sum**
2. **Camel Case for Function and Method Names**:
   * Start with a lowercase letter.
   * Capitalize the first letter of each subsequent concatenated word.
   * Example: **calculateTotal**, **getUserData**, **processData**
3. **Pascal Case for Class Names**:
   * Start each word with an uppercase letter.
   * Do not use underscores.
   * Example: **MyClass**, **EmployeeRecord**, **BookStore**
4. **Single Leading Underscore**:
   * Use a single leading underscore to indicate that a variable or method is intended for internal use within a module.
   * Example: **\_internal\_var**, **\_helper\_function**
5. **Double Leading Underscore (Name Mangling)**:
   * Use a double leading underscore to indicate name mangling in class definitions. This makes the attribute name more unique to the class.
   * Example: **\_\_private\_var**
6. **Uppercase for Constants**:
   * Use uppercase letters for constant values.
   * Separate words using underscores.
   * Example: **MAX\_VALUE**, **PI**, **DEFAULT\_SETTINGS**
7. **Meaningful Names**:
   * Use descriptive names that convey the purpose of the variable or function.
   * Avoid single-letter variable names except in simple loop constructs (e.g., **for i in range(...)**).
8. **Avoid Ambiguous Names**:
   * Choose names that are not easily confused with built-in functions or modules.
   * Avoid using names of existing classes or modules.
9. **Consistency**:
   * Maintain consistent naming conventions throughout your codebase.
   * Follow the conventions used in the libraries you're working with to ensure consistency.

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

**Ans:** If a keyword is used as a variable name in Python, it will encounter a **SyntaxError**. Keywords are reserved words in the Python language that have special meanings and purposes. You cannot use them as variable names, function names, class names, or any other identifiers.

Example

if = 50

SyntaxError: invalid syntax

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

**Ans:** The **def** keyword in Python is used to define a user-defined function. Functions are blocks of reusable code that can be defined once and then called (invoked) multiple times throughout the program. The **def** keyword marks the beginning of a function definition and specifies the name of the function, its parameters (if any), and the code that the function executes.

Eample

def f(name):

print("my name is", name)

f("maitri")

my name is maitri

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

**Ans:**

The special character **'\'** in Python is called the **backslash**. It is used as an escape character to indicate that the character following it has a special meaning. It's commonly used to represent characters that are difficult to input or that have a special purpose within strings and other text data.

Here are some common uses of the backslash:

1. **Escape Sequences**: Backslash is used to create escape sequences that represent special characters:
   * **\n**: Newline (line break)
   * **\t**: Tab
   * **\'**: Single quote
   * **\"**: Double quote
   * **\\**: Backslash
2. **Unicode Escapes**: Backslash is used to represent Unicode characters using their hexadecimal value:
   * **\uXXXX**: Unicode character with 4 hexadecimal digits
   * **\UXXXXXXXX**: Unicode character with 8 hexadecimal digits
3. **Raw Strings**: Prefixing a string with **r** (e.g., **r"This is a raw string"**) treats backslashes as literal characters, useful for regular expressions and file paths.
4. **Line Continuation**: A backslash at the end of a line indicates that the line continues on the next line.

Here are some examples of using the backslash in Python:

Example

print("This is a line\nwith a newline")

This is a line

with a newline

print("She said, \"Hello!\"")

She said, "Hello!"

print('C:\\Users\\Username\\Documents')

C:\Users\Username\Documents

print("Unicode character: \u03B1")

Unicode character: α

print("Unicode character: \U0001F602")

Unicode character: 😂

In each of these cases, the backslash indicates that the character following it has a special meaning or purpose.

Top of Form

**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 contains elements of the same data type. In this example, all elements are integers:

Python code

homogeneous\_list = [5, 10, 15, 20, 25]

(ii) **Heterogeneous Set**: A heterogeneous set contains elements of different data types. Here, the set contains a mix of strings and numbers:

Python code

heterogeneous\_set = {"apple", 42, 3.14, "banana"}

(iii) **Homogeneous Tuple**: A homogeneous tuple contains elements of the same data type. In this case, the elements are all strings:

Python code

homogeneous\_tuple = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

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

**Ans:**

Mutable and immutable are terms used to describe whether the value of a data type can be changed after it's created. In Python, some data types are mutable, meaning their values can be modified in-place, while others are immutable, meaning their values cannot be changed once they are created.

**Mutable Data Types**: Mutable data types are those that allow changes to their values without creating a new object. This means you can modify the content of a mutable object after it's created, and any references to the object will reflect these changes.

Examples of mutable data types in Python:

* Lists: You can modify, add, or remove elements from a list after creation.
* Dictionaries: You can add, modify, or remove key-value pairs.
* Sets: You can add or remove elements.

Example using a mutable data type (list):

List=[1,2,3]

List[2]=4

print(List)

[1, 2, 4]

**Immutable Data Types**: Immutable data types are those that do not allow changes to their values after they are created. If you want to modify an immutable object's value, you have to create a new object with the desired value.

Examples of immutable data types in Python:

* Numbers (int, float, complex)
* Strings
* Tuples

Example using an immutable data type (string):

Tuple=(1,2,3)

Tuple[2]=4

Traceback (most recent call last):

File "<pyshell#59>", line 1, in <module>

Tuple[2]=4

TypeError: 'tuple' object does not support item assignment

NewTuple=[1,2,4]

print(NewTuple)

[1, 2, 4]

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

**\***

**\*\*\***

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

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

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

**Ans:**

Python Code

for i in range(1,6):

print("\*"\*(2\*i-1))

\*

\*\*\*

\*\*\*\*\*

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

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

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

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

**|||||||**

**|||||**

**|||**

**|**

**Ans:**

Python Code

while i <= 5:

print("|"\*(2\*(6-i)-1))

i+=1

|||||||||

|||||||

|||||

|||

|