***ASSIGNMENT\_6***

**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 variable names or identifiers. These keywords are part of the Python language syntax and serve specific purposes.

To print all the Python keywords, you can make use of the keyword library. Here's an example code snippet that accomplishes this:

**import keyword**

**all\_keywords = keyword.kwlist**

**print(all\_keywords)**

By importing the keyword library, we can access the kwlist attribute, which contains a list of all the Python keywords. Printing all\_keywords will display the complete list of Python keywords, including words like if, for, while, def, and many more.

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

***ANS:***

When creating variables in Python, you need to follow a few rules. Here are the key rules to keep in mind:

Variable names must start with a letter or an underscore (\_). They cannot begin with a number.

Variable names can contain letters (both uppercase and lowercase), numbers, and underscores. However, they cannot contain spaces or special characters such as @, $, %, etc.

Python is case-sensitive, so variables with different letter cases are considered different. For example, myVariable and myvariable are distinct variables.

Variable names should be descriptive and meaningful, helping to convey the purpose or content of the variable.

Avoid using Python keywords as variable names, as they are reserved for specific purposes in the language.

It is recommended to use lowercase letters for variable names, following the convention known as "snake\_case." For example, my\_variable is a commonly used format.

Here are a few examples of valid variable names:

**name**

**age**

**my\_variable**

**\_total**

**x**

Remembering and adhering to these rules will help you create valid variable names in Python.

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

***ANS:***

Python follows a set of standards and conventions for variable naming to enhance code readability and maintainability. These conventions are outlined in PEP 8, the official style guide for Python code. Here are the key standards and conventions to follow:

Use descriptive and meaningful names: Variable names should clearly convey the purpose or content of the variable. This makes the code more readable and easier to understand.

Use lowercase letters: For most variables, lowercase letters are preferred. This convention is known as "snake\_case," where words are separated by underscores. For example: my\_variable, student\_name, total\_count.

Avoid single-character names: Unless used as a loop counter or an iterator, it's generally recommended to avoid single-character variable names like x or i. Instead, opt for more descriptive names that indicate the variable's purpose.

Constants should be in uppercase: If a variable represents a constant value that should not be changed, use uppercase letters and underscores to separate words. For example: PI, MAX\_VALUE, DEFAULT\_SIZE.

Be consistent with naming conventions: Maintain consistency throughout your codebase. If you use a specific naming convention, stick to it consistently. This ensures readability and avoids confusion.

Avoid reserved keywords: Do not use Python keywords as variable names. Keywords are reserved for specific purposes in the language and should not be used as identifiers.

By following these standards and conventions, your Python code will become more readable, maintainable, and consistent, making it easier for others (and your future self) to understand and work with your code.

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

***ANS:***

If you use a keyword as a variable name in Python, you will encounter a syntax error. Keywords in Python are reserved words that have predefined meanings and serve specific purposes within the language. They cannot be used as variable names or identifiers.

When you attempt to use a keyword as a variable name, Python will raise a syntax error to indicate the issue. The error message will typically state that you have an invalid syntax or an unexpected token. Here's an example of what the error message might look like:

**>>> def = 10**

**File "<stdin>", line 1**

**def = 10**

**^**

SyntaxError: **invalid syntax**

In the above example, the keyword def is used as a variable name, which results in a syntax error. Python recognizes def as a keyword reserved for defining functions, so using it as a variable name is not allowed.

To avoid such errors, it is important to choose variable names that do not conflict with Python keywords and follow the naming conventions mentioned earlier.

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

***ANS:***

The def keyword in Python is used to define a function. Functions are blocks of reusable code that perform a specific task. They help in organizing code, promoting reusability, and enhancing code modularity.

When you use the def keyword, you are declaring and defining a function. Here's the basic syntax for defining a function in Python:

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

**# Function body**

**# Code statements**

**# Return statement (optional)**

Let's break down the components of a function declaration:

def: This keyword is used to indicate that you are defining a function.

function\_name: This is the name you give to your function, which should follow the variable naming conventions. It should be descriptive and reflect the purpose of the function.

parameters: These are optional inputs that you can pass to the function. Parameters allow you to provide data to the function for processing.

Function body: This is the block of code that is executed when the function is called. It consists of one or more statements that perform a specific task or series of tasks.

Return statement: The return statement is optional. It is used to specify the value(s) that the function will send back as a result when it is called.

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

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

**return a + b**

In the above code, the def keyword is used to define the add\_numbers function that takes two parameters a and b. The function body adds these two parameters and returns the result using the return statement.

Overall, the def keyword is crucial in defining functions, enabling you to encapsulate a block of code and execute it whenever needed by calling the function.

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

***ANS:***

The special character \ in Python is known as the backslash. It serves as an escape character, indicating that the character immediately following it has a special meaning.

The backslash \ is used to escape characters that would otherwise have a different interpretation or function within a string or code. It allows you to include special characters or represent certain characters that are not easily typable or have a special purpose. Here are some common uses of the backslash:

Escaping special characters: If you want to include characters such as quotes or backslashes themselves within a string, you can use the backslash to escape them. For example:

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

**Output: She said, "Hello!"**

Newline character: The backslash followed by the letter n (\n) represents the newline character. It is used to create a line break or move to a new line within a string. For example:

**print("Hello\nWorld")**

**Output:**

**Hello**

**World**

Tab character: The backslash followed by the letter t (\t) represents the tab character. It is used to insert horizontal tabs within a string. For example:

**print("Name:\tJohn")**

**Output: Name: John**

Unicode characters: The backslash can also be used to represent Unicode characters using their hexadecimal or Unicode code points. For example:

**print("\u03B1") # Prints the Greek letter alpha (α)**

**Output: α**

These are just a few examples of how the backslash \ is used as an escape character in Python. It helps to include special characters, control the formatting of strings, and represent characters that are not easily typable.

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

**(i) Homogeneous list**

**(ii) Heterogeneous set**

**(iii) Homogeneous tuple**

***ANS:***

**(i) Homogeneous list:**

A homogeneous list in Python is a list that contains elements of the same data type. Here's an example of a homogeneous list of integers:

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

In this example, the numbers list contains only integer values.

**(ii) Heterogeneous set:**

A heterogeneous set in Python is a set that can contain elements of different data types. Here's an example of a heterogeneous set:

my\_set = {1, "hello", 3.14, True}

In this example, the my\_set set includes elements of different data types, such as an integer, a string, a floating-point number, and a boolean.

**(iii) Homogeneous tuple:**

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

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

In this example, the fruits tuple contains only string values.

Note that Python lists (i), sets (ii), and tuples (iii) can be either homogeneous or heterogeneous. The examples provided demonstrate the respective conditions you mentioned.

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

***ANS:***

In Python, data types can be categorized as either mutable or immutable. This categorization refers to whether the data's value can be changed after it is created. Let's explore mutable and immutable data types with proper explanations and examples:

**Immutable Data Types:**

Immutable data types are those whose values cannot be modified once they are assigned. Any attempt to modify an immutable object will result in the creation of a new object with the modified value. Examples of immutable data types in Python include:

Numeric Types: int, float, complex

Boolean Type: bool

String Type: str

Tuple Type: tuple

Example:

**x = 10 # Immutable object of type int**

**y = "Hello" # Immutable object of type str**

**x = x + 5 # Creating a new object with modified value (15)**

**y = y + " World" # Creating a new object with modified value ("Hello World")**

**print(x) # Output: 15**

**print(y) # Output: Hello World**

In the above example, when modifying the value of x or concatenating the string y, new objects are created because the original objects are immutable. The original values remain unchanged, and new objects are assigned to the variables.

**Mutable Data Types:**

Mutable data types, on the other hand, are objects whose values can be modified after they are assigned. Changes to mutable objects are reflected in the same object, without the need to create a new one. Examples of mutable data types in Python include:

List Type: list

Dictionary Type: dict

Set Type: set

Example:

**numbers = [1, 2, 3, 4] # Mutable object of type list**

**student = {"name": "John", "age": 20} # Mutable object of type dict**

**numbers.append(5) # Modifying the existing list by adding a new element**

**student["age"] = 21 # Modifying the existing dictionary by changing the value of a key**

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

**print(student) # Output: {"name": "John", "age": 21}**

In the above example, the list numbers and the dictionary student are mutable objects. Modifying them directly changes their values in place without creating new objects. The original objects are modified directly.

Understanding the distinction between mutable and immutable data types is crucial because it affects how objects behave when modified and how memory is utilized in your program.

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

**\***

**\*\*\***

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

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

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

***ANS:***

Here's a code snippet that uses a for loop to create the given structure:

**rows = 5**

**for i in range(rows):**

**for j in range(i + 1):**

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

**print()**

**Output:**

**\***

**\*\***

**\*\*\***

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

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

Explanation:

The variable rows represents the number of rows in the structure.

The outer loop for i in range(rows): iterates from 0 to rows-1.

The inner loop for j in range(i + 1): iterates from 0 to i (inclusive) for each row i.

In each iteration of the inner loop, an asterisk (\*) is printed using print("\*", end="").

After printing all the asterisks for a row, a newline character is printed using print() to move to the next line.

The nested loops ensure that the correct number of asterisks is printed for each row, following the given structure.

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

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

**|||||||**

**|||||**

**|||**

***ANS:***

Here's a code snippet that uses a while loop to create the given structure:

**rows = 5**

**i = rows**

**while i >= 1:**

**j = 1**

**while j <= i:**

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

**j += 1**

**print()**

**i -= 2**

Output:

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

**|||||**

**|||**

Explanation:

The variable rows represents the number of rows in the structure.

The outer while loop while i >= 1: starts with i initialized to rows and continues until i becomes less than 1.

The inner while loop while j <= i: starts with j initialized to 1 and continues until j becomes greater than i.

In each iteration of the inner loop, a vertical bar (|) is printed using print("|", end="").

After printing all the vertical bars for a row, a newline character is printed using print() to move to the next line.

Inside the outer loop, the variable i is decremented by 2 in each iteration to achieve the desired pattern of decreasing vertical bars.

The while loop structure allows us to create the given structure by manipulating the loop conditions and variables to control the number of vertical bars printed in each row.