In [ ]:
```
Q.1. What are keywords in python? Using the keyword library,
print all the python keywords.

A) Keywords in Python are reserved words that have specific meanings
and purposes within the language. These keywords cannot be used
as identifiers (variable names, function names, etc.)
because they are already used by Python for specific purposes.

import keyword

# Get all the Python keywords
all_keywords = keyword.kwlist

# Print the keywords
for keyword in all_keywords:
    print(keyword)

    These are the keywords that are reserved in Python
and have specific meanings and purposes within the language.
```

In [ ]:
```
Q.2. What are the rules to create variables in python?
A)
In Python, variables are used to store data values and
provide a way to refer to them by a name. Here are the
rules for creating variables in Python:

Variable names must start with a letter (a-z, A-Z) or
an underscore (_). They cannot start with a number.

Variable names can contain letters (a-z, A-Z),
digits (0-9), and underscores (_).
They must be a combination of these characters.

Variable names are case-sensitive, which means
myVariable and myvariable are different variables.

Here are some valid examples of variable names:

name
age
my_variable
total_count
is_valid
    And here are some examples of invalid variable names:
        123variable  # Starts with a number
if  # Uses a Python keyword
my-variable  # Contains a hyphen (invalid)
```

In [ ]:
```
Q.3. What are the standards and conventions followed
for the nomenclature of variables in python to improve
code readability and maintainability?

A) In Python, there are widely adopted standards and conventions for
variable naming to improve code readability and maintainability.
These conventions are outlined in the official Python style guide,
known as PEP 8. Here are the key guidelines:

* Snake Case: Variable names should be in all lowercase letters
with words separated by underscores. For example:
```

```
my_variable
total_count
user_name

* Descriptive Names: Use meaningful and descriptive names that convey
the purpose or meaning of the variable. This helps in understanding
the code without needing additional comments. For example:
num_students
average_score
is_valid_input

* Avoid Abbreviations: Generally, it's recommended to avoid excessive
abbreviations in variable names. Instead, use clear and understandable
names that indicate the purpose of the variable. For example:

i (acceptable for loop counter)
index (preferred for loop counter if more context is needed)

* Use English Words: If we Stick to using English words for variable names,
as Python itself uses English for its keywords and standard library.

* Consistency: By Maintaining  consistency in variable naming throughout the
codebase. If we use the same naming conventions and styles consistently
to ensure uniformity and make the code more readable.

* Avoid Shadowing Built-in Names:  By Avoiding  variable names that
could potentially shadow built-in functions or module names.
This helps prevent conflicts and confusion. For example,
don't use names like list, str, or file for  variables.

By following these standards and conventions,  We can enhance code
readability and maintainability, making it easier for others
(including yourself) to understand and maintain the code.
```

In [ ]:

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

A) If a keyword is used as a variable name in Python,
it will result in a syntax error. Python reserves keywords for
specific purposes within the language, and they cannot be used
as identifiers (such as variable names, function names, etc.).

When we try to use a keyword as a variable name, Python's
parser will raise a SyntaxError indicating that the keyword
is not allowed as an identifier. For example, let's consider
the keyword "if" being used as a variable name:


    if = 10
If we run this code, we'll get the following error:

SyntaxError: invalid syntax
    Python recognizes "if" as a keyword and expects it to be used for
    conditional statements, not as a variable name. To resolve this
    issue, we should choose a different variable name that is not a
    reserved keyword.

It is important to be mindful of Python's keywords and  by avoid using
them as variable names to prevent syntax errors and it is important to
ensure the proper functioning of the code.
```

In [ ]:

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

The def keyword in Python is used to define a function.
```

```
Functions are reusable blocks of code that perform specific
tasks. They allow us to encapsulate a set of instructions and
execute them whenever needed by calling the function's name.

Here's the basic syntax for defining a function using the def keyword:
def function_name(parameters):
    # Function body
    # Code goes here
    # More code
    # ...

# Example function call
function_name(argument_values)
Let's break down the components of the def statement:

def: It is the keyword used to indicate the start of a function definition.
function_name: This is the name of the function, which  can choose to
    be meaningful and descriptive.
parameters: These are optional placeholders that represent the inputs
    or arguments passed to the function. They can be used within the
    function's body to perform computations or produce results.
Function body: The indented block of code following the def statement
    constitutes the function's body. It contains the instructions or
    statements that are executed when the function is called.
argument_values: These are the actual values passed to the function
    when calling it. They correspond to the function's parameters and
    provide the necessary data for the function to work with.

    Here's an example to illustrate the usage of def to define
    and call a simple function:

        def greet(name):
    print("Hello, " + name + "!")

greet("Alice")  # Output: Hello, Alice!
greet("Bob")    # Output: Hello, Bob!

In this example, the greet function is defined using the def keyword.
It takes one parameter, name, and prints a greeting message with the
provided name. The function is then called with different arguments
to produce the desired output.

In summary, the def keyword is used to define functions in Python,
allowing us to create reusable and modular code blocks.
```

In [ ]:

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

In Python, the special character '\' is known as the backslash.
It serves as an escape character, which means it is used to
indicate that the character following it has a special meaning.
The backslash allows you to include special characters or
create special sequences within strings or other literals.

Here are some common use cases for the backslash in Python:

Escape Sequences: The backslash is used to escape special characters
    within strings. For example:
        print("This is a \"quote\"")  # Output: This is a "quote"
print("This is a \nnew line")  # Output: This is a
                               #           new line

    In the first example, the backslash escapes the double quotes,
    allowing them to be included within the string.
```

```
    In the second example, '\n' represents a newline character,
    which causes a line break in the output.

Raw Strings: Adding an 'r' before a string literal creates
    a raw string, where backslashes are treated as literal
    characters rather than escape characters. This is useful,
    for example, when working with file paths or regular expressions.
    For example:

path = r"C:\Users\John\Documents"  # Raw string
print(path)  # Output: C:\Users\John\Documents

In this example, the 'r' prefix before the string creates a
raw string, allowing the backslashes to be treated as literal characters.

Unicode Escape: The backslash can be used to represent Unicode
    characters using their hexadecimal or octal representation.
    For example:

        print('\u03B1')  # Output: α (Greek letter alpha)
print('\U0001F600')  # Output: 😀 (smiling face emoji)


    In this case, '\u' is used for a 16-bit Unicode character,
    while '\U' is used for a 32-bit Unicode character.


        Continuation Lines: The backslash can be used to continue
            a logical line of code onto the next physical line.
            This is helpful when you have long statements or expressions.
            For example:

                total = 10 + 20 + \
                        30 + 40
print(total)  # Output: 100



The backslash at the end of the first line indicates that the
statement continues on the next line.

These are some of the common operations and uses of the
backslash character ('') in Python. It provides flexibility
for including special characters, creating escape sequences,
representing Unicode characters, working with raw strings,
and continuing lines of code.
```

In [ ]:

```
Q.7. Give an example of the following conditions:
(i) Homogeneous list
(ii) Heterogeneous set
(iii) Homogeneous tuple

(i) Homogeneous list: A homogeneous list in Python is a list
    where all the elements have the same data type.
    Here's an example of a homogeneous list containing integers:+

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

    In this example, all the elements of the list numbers are integers.

(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:
```

```
        data_set = {1, "apple", 3.14, True}

        In this example, the set data_set contains elements of
        different data types, such as an integer, a string, a float,
        and a boolean.

(iii) Homogeneous tuple: A homogeneous tuple in Python is a tuple where
    all the elements have the same data type. Here's an example of a
    homogeneous tuple containing strings:

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

    In this example, all the elements of the tuple fruits are strings.

Note: The examples provided above are for illustrative purposes
    and do not represent an exhaustive list of elements. We can
    have more elements in a list, set, or tuple, depending on the
    requirements.
```

In [ ]:

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

A)In Python, data types can be classified as either mutable or
immutable based on whether their values can be changed after
they are created. Here's an explanation of mutable and immutable data types:

Mutable Data Types:
Mutable data types are those whose values can be modified after
they are created. This means you can change their state, add, remove,
or modify elements without creating a new object.
The key characteristic of mutable data types is that they have methods
or operations that can modify their internal state.

Examples of mutable data types in Python include:

Lists: Lists are ordered, mutable sequences of elements.
Dictionaries: Dictionaries are mutable collections of key-value pairs.
Sets: Sets are mutable collections of unique elements.
Here's an example to demonstrate the mutability of lists:


numbers = [1, 2, 3, 4, 5]
numbers.append(6)  # Modifying the list by adding an element
numbers[0] = 10  # Modifying an element by index assignment
print(numbers)  # Output: [10, 2, 3, 4, 5, 6]
In this example, we can modify the list numbers by adding an
element using the append() method and by assigning a new value to an
element using index assignment.

Immutable Data Types:
Immutable data types are those whose values cannot be changed
after they are created. This means we cannot modify their state once
they are assigned. If we need to change the value, We have to create a
new object. Immutable objects are considered safer and more predictable,
as they guarantee that their values remain constant throughout their
lifetime.

Examples of immutable data types in Python include:

Integers: Integer values are immutable.
Floats: Floating-point numbers are immutable.
Strings: Strings are immutable sequences of characters.
```

```
Tuples: Tuples are immutable sequences of elements.
Here's an example to demonstrate the immutability of strings:

message = "Hello"
message += " World"  # Creating a new string by concatenation
print(message)  # Output: "Hello World"
In this example, we cannot modify the original string "Hello".
Instead, we create a new string by concatenating " World" and
assign it back to the message variable.

Immutable objects provide advantages such as ease of reasoning
about their behavior, thread safety, and efficient caching.
However, when you need to modify their values frequently,
mutable data types like lists are more suitable.

It's important to understand the mutability and immutability of
data types in Python as it affects the work with and manipulate
data in the programs.
```

In [ ]:
```
Q.9. Write a code to create the given structure using only for loop.
    *
   ***
  *****
 *******
*********
```

In [1]:
```
rows = 5  # number of rows in the structure

for i in range(rows):
    # Print spaces before the stars
    for j in range(rows - i - 1):
        print(" ", end="")

    # Print stars
    for k in range(2 * i + 1):
        print("*", end="")

    # Move to the next line after each row
    print()
```

```
    *
   ***
  *****
 *******
*********
```

In [ ]:
```
Q.10. Write a code to create the given structure using while loop.
|||||||||
 |||||||
  |||||
   |||
    |
```

In [4]:
```
n = int(input("Enter the number of lines: "))
i=1
symbols = ((n*2)-1) #number of symbols to be printed initially

while(n>0):
    b=1
    while (b<i):
        print (" ",end = '')
        b=b+1
```

```
    j=1
    while (symbols>=j):
        print ("|",end = '')
        j+=1
    print ()
    n= n-1
    i=i+1
    symbols = symbols-2
```

```
Enter the number of lines: 5
|||||||||
 |||||||
  |||||
   |||
    |
```

In [ ]: