1)

In [ ]:
```
'''
In Python, keywords are special reserved words that have specific
meanings and functionalities within the language.
These keywords cannot be used as identifiers (variable names, function names, etc.)
because they are already predefined and have a particular purpose in Python syntax.
Using a keyword as an identifier will result in a syntax error.

Example:

False     class    finally    is    return
None      continue for        lambda    try
True      def      from       nonlocal while
and       del      global     not       with
as        elif     if         or        yield
assert    else     import     pass
break     except   in         raise

'''
```

In [1]:
```
import keyword

all_keywords = keyword.kwlist
print(all_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']
```

2)

In [ ]:
```
'''
Rules to create variables in Python:

1) Valid Name Characters: Variable names can contain letters (both uppercase and lowercase), digits, and underscores ().
2) Case-Sensitivity: Python is case-sensitive, which means that variables with different cases are considered different.
3) Reserved Keywords: You cannot use reserved keywords (also known as Python keywords) as variable names.
4) No Spaces: Variable names cannot contain spaces. Use underscores (_) to separate words if needed.
5) Avoid Special Characters: Special characters such as !, @, #, $, %, etc., are not allowed in variable names.
6) Descriptive and Readable: It's recommended to use descriptive and meaningful names for variables.

'''
```

3)

In [ ]:
```
'''
In Python, adhering to certain standards and conventions for variable names can greatly
improve code readability and maintainability. The most widely followed standards are outlined
in Python Enhancement Proposal 8 (PEP 8), which is the official style guide for Python code.

1) Variable Names: Use descriptive names that convey the meaning of the variable's purpose.
Variable names should be in lowercase letters and can contain underscores (_) to separate words for readability.

2) Constants: Constants are variables that are not meant to be modified.
PEP 8 suggests using ALL_CAPS with underscores to indicate constants.

3) Avoid Single-Letter Names: Except for simple loop variables, try to avoid using single-letter
variable names as they are not informative.

4) Avoid Ambiguous Names: Choose meaningful and unambiguous names for variables.

5) Use Function Names for Functions: Functions should follow the same naming conventions as variables,
using lowercase letters with underscores.

6) Use Class Names for Classes: Class names should follow the CapWords convention
(also known as CamelCase) where each word starts with a capital letter.

7) Module Names: Module names should be in lowercase with underscores.
If a module name is composed of multiple words, separate them with underscores.

8) Avoid Shadowing Built-in Names: Do not use variable names that are already used as
built-in functions or keywords, as it can lead to confusion and unexpected behavior.

'''
```

4)

In [ ]:
```
'''
If a keyword is used as a variable name in Python, it will result in a syntax error.
Keywords are reserved words with predefined meanings in the Python language,
and they cannot be used as identifiers (variable names, function names, etc.).
'''
```

In [4]:
```
if = 10
```

```
  Cell In[4], line 1
    if = 10
       ^
SyntaxError: invalid syntax
```

5)

In [ ]:
```
'''
The def keyword in Python is used to define user-defined functions.
By using the def keyword, one can create functions and give them a name, so we can call
and execute that block of code whenever needed.
'''
```

In [5]:
```
def add_numbers(a, b):
    return a + b
```

6)

In [ ]:
```
'''
In Python, the special character \ (backslash) is used as an escape character.
It is used to represent special sequences within strings and other literals.
When a backslash is followed by a character or a combination of characters,
it creates an escape sequence, which has a special meaning and behavior.
'''
```

In [6]:
```
# Newline
print("Hello,\nWorld!")

# Tab
print("Name:\tJohn")

# Carriage return
print("Hello,\rWorld!")

# Backslash
print("This is a backslash: \\")

# Single quote
print('It\'s raining outside.')

# Double quote
print("She said, \"Hello!\"")
```

```
Hello,
World!
Name:   John
World!
This is a backslash: \
It's raining outside.
She said, "Hello!"
```

7)

In [ ]:
```
'''
(i) Homogeneous list: A homogeneous list is a list that contains elements of the same data type.
(ii) A heterogeneous set is a set that contains elements of different data types.
(iii) A homogeneous tuple is a tuple that contains elements of the same data type.
'''
```

In [8]:
```
# Homogeneous list (integers)
homogeneous_list = [1, 2, 3, 4, 5]
print(homogeneous_list)
```

```
[1, 2, 3, 4, 5]
```

In [9]:
```
# Heterogeneous set (mixed data types)
heterogeneous_set = {1, "hello", 3.14, 5}
print(heterogeneous_set)
```

```
{1, 3.14, 5, 'hello'}
```

In [10]:
```
# Homogeneous tuple (strings)
homogeneous_tuple = ("apple", "banana", "orange")
print(homogeneous_tuple)
```

```
('apple', 'banana', 'orange')
```

8)

In [ ]:
```
'''
Immutable Data Types:

An immutable data type is a type whose value cannot be changed after it is created.
When we perform any operation that seems to modify an immutable object, it actually creates a
new object with the modified value, leaving the original object unchanged.

Examples of immutable data types in Python include:

int: Integer values, such as 10, 100, -5, etc.
float: Floating-point numbers, like 3.14, 2.71, etc.
str: Strings of characters, e.g., "hello", "Python", etc.
tuple: Ordered collections of items, enclosed in parentheses.

Mutable Data Types:

A mutable data type is a type whose value can be changed or modified after it is created.
When we modify a mutable object, the changes directly affect the original object, and no new object is created.

Examples of mutable data types in Python include:

list: Ordered collections of items, enclosed in square brackets.
dict: Dictionaries, key-value pairs, enclosed in curly braces.
set: Unordered collections of unique items, enclosed in curly braces.

'''
```

9)

In [13]:
```
rows = 5

for i in range(1, rows + 1):
    spaces = rows - i
    stars = 2 * i - 1
    print(" " * spaces + "*" * stars)
```

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

10)

In [14]:
```
total_rows = 5
row_num = total_rows

while row_num > 0:
    num_pipes = 2 * row_num - 1
    num_spaces = total_rows - row_num
    print(" " * num_spaces + "|" * num_pipes)
    row_num -= 1
```

```
|||||||||
 |||||||
  |||||
   |||
    |
```