```
# Explain the significance of Python keywords and provide examples of five keywords.
Ans:- Python keywords are reserved words that have special meanings and cannot be used
        as identifiers (names for variables, functions, etc.) in the code. These keywords are
        an essential part of the language syntax, defining the structure and logic of the code.
        They play a crucial role in controlling the flow of a program, defining functions, and managing data types.

# Here are examples of five Python keywords along with their significance:

1. if:

 Used for conditional statements
It allows the execution of a block of code only if a specified condition is true.

x = 10
if x > 5:
    print("x is greater than 5")

2. for:
Used for loop constructs.
It iterates over a sequence (such as a list, tuple, or string)
and executes a block of code for each item in the sequence.

fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

3. def:
 Used for defining functions.
It is followed by the function name and a block of code that represents the function's body.

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

4. class:
    Used for defining classes,
    which are a way to structure and organize code by creating objects with attributes and methods.

class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        print(self.name + " says Woof!")

my_dog = Dog("Buddy")
my_dog.bark()


5. return:
 Used inside functions to indicate the value that the function should return to the caller.
It terminates the function's execution and sends the specified value back.

def add_numbers(a, b):
    result = a + b
    return result

sum_result = add_numbers(5, 3)
print("Sum:", sum_result)


# Q. Descrive the rules for defining identifiers in Python and provide an example
Ans:-
In Python, identifiers are names given to variables, functions, classes, modules
or any other entities in the program. Identifiers follow certain rules for their definition.
Here are the rules for defining identifiers in Python:

 a.Valid Characters:

Identifiers can include letters (both uppercase and lowercase), digits, and underscore (_).
They must start with a letter (a-z, A-Z) or an underscore (_).
Case Sensitivity:

 b.Python is case-sensitive:
    so uppercase and lowercase letters are considered different. For example, myVar and myvar are different identifiers.

c. Reserved Words:

Identifiers cannot be the same as Python keywords or reserved words, as they have special meanings in the language.
```

No Spaces or Special Characters:

Spaces and special characters (except underscore) are not allowed in identifiers.
Length Limitation:

There is no fixed limit on the length of an identifier, but it's advisable to keep them reasonably short and descriptive.

```
# Q.What are comments in Python, and why are they useful6 Provide an example

Ans:- Comments in Python are annotations or explanatory notes within the code
that are ignored by the Python interpreter during program execution.
They are used to provide information, explanations, or documentation about
the code for developers or anyone reading the code.
Comments are a crucial aspect of writing maintainable and understandable code.

there are two type of comment in pyhton

1. single line comment
These comments are written on a single line and are preceded by
    the # symbol. Everything after the # on that line is treated as a comment.
    for example:
     # print("hello world")

2. Multi line comment:
    # While Python does not have a specific syntax for multi-line comments,
    # triple-quotes (''' or """) are often used to create multi-line strings.
    # These strings are not assigned to any variable and are treated as comment.


# Q. Why is proper indentation important in Python?

Ans:-Proper indentation is crucial in Python because
it is used to define the block structure of the code.
In Python, indentation is not just a matter of style,
it is a syntactical requirement.
The interpreter uses indentation to determine
the grouping of statements within loops, conditional statements, functions, and classes.
The key reasons why proper indentation is important in Python are:

1. Readibility
2.Block structure
3.Maintainbility
4.Scope and nesting


# Q. What happens if indentation is incorrect in Python?

Ans:-
In Python, indentation is not just a matter of style but is a syntactical requirement.
Incorrect indentation can lead to various issues, and
the Python interpreter will raise an IndentationError when it encounters inconsistent or improperly indented code.
Here are some common scenarios and what happens when indentation is incorrect:

1. IndentationError: Unexpected Indent:

If there is an unexpected increase in indentation
where it is not expected, Python raises an IndentationError.
This often occurs when a block of code (inside a function, loop, or conditional statement) is not indented properly.

exaple :
    def example_function():
print("Indented incorrectly")  # IndentationError here

2. IndentationError:
    Unindent does not match any outer indentation level:

  If there is a decrease in indentation where it is not expected,
        Python raises an IndentationError.
This can happen if there is an extra unindentation or if the unindentation does not match the outer block.

exaple:
    if True:
    print("Properly indented")
  print("Unexpected unindent")  # IndentationError here

3. Mixing Tabs and Spaces:

Mixing tabs and spaces for indentation is not allowed in Python.
```

```
If there is inconsistency in using tabs and spaces within the same block, Python raises an IndentationError.

example :
    def mixed_indentation():
    print("Indented with spaces")
 print("Indented with tabs")  # IndentationError here


4. Incorrect Nesting:
       Incorrect nesting, where blocks of code are not indented properly to reflect
      their intended structure, can result in IndentationError.
    exaple:
        if True:
print("Incorrect nesting")  # IndentationError here


# Q. Differentiate 0etween expression and statement in Python with examples

Ans:-
    In Python, expressions and statements are two fundamental components of the language,
    but they serve different purposes and have distinct characteristics.

1. Expression:
     An expression is a piece of code that produces a value when executed.
        It can be as simple as a literal value, a variable, or a more complex combination of operators and functions.
        Expressions are often used within statements to compute a value.

    example :
        # Literal expression
result = 5 + 3

# Variable expression
x = 10
y = x * 2

# Complex expression
z = (x + y) / 2


2.  Statement:
       A statement is a complete line of code that performs an action. Unlike expressions
        statements may not necessarily produce a value. Instead
        they often involve control flow, assignments, or declarations.
        Statements are executed one after another in a script or program.
example:
    # Assignment statement
a = 10

# Conditional statement
if a > 5:
    print("a is greater than 5")
else:
    print("a is not greater than 5")

# Loop statement
for i in range(3):
    print(i)
```