CONDITIONAL STATEMENTS QUESTIONS & ANSWERS

Section-A

1. What is a conditional statement in Python?
   Ans.

In Python, a conditional statement allows you to make decisions in your code based on whether a certain condition is true or false. The most common conditional statements are `if`, `elif` (else if), and `else`.

Example: Using `if`, `elif`, and `else`.

```
# Example
```

```
# Example
number = -15

if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

```
The number is negative.
```

In this example, the program checks the value of the `temperature` variable and prints a message based on the conditions specified. Since `temperature` is 25, the output is "It's a nice day."

2. Explain the difference between the `if`, `elif`, and `else` statements.
   In Python, the `if` statement is used to execute a block of code only if a certain condition is true. The `elif` (else if) statement is used to check additional conditions if the previous conditions are false. The `else` statement is used to execute a block of code if none of the previous conditions are true.
   In this example, we'll determine the type of a number (positive, negative, or zero) using `if`, `elif`, and `else`.

```
# Example
number = -15

if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

```
The number is negative.
```

Here, the program checks if the number is greater than 0 using the `if` statement. If true, it prints "The number is positive." If false, it moves to the `elif` statement, which checks if the number is less than 0. If true, it prints "The number is negative." If both conditions are false, it goes to the `else` statement and prints "The number is zero."

3. How do you write a basic `if` statement in Python?

To demonstrate a basic `if` statement in Python, let's create a program that checks whether a number is positive or not.

```
# Example
```

```
# Example
number = 10

if number > 0:
    print("The number is positive.")
```

```
The number is positive.
```

In this example, the program evaluates the condition `number > 0`. If it's true, the indented block of code under `if` will be executed and "The number is positive." will be printed. If the condition is false, the block will be skipped.

4. What is the purpose of indentation in Python conditional statements?

   Indentation is a crucial aspect of Python's syntax, especially in conditional statements. It is used to define blocks of code and indicate the scope of statements within conditions. Let's illustrate this with an example:

```
# Example
number = 10

if number > 0:
    print("The number is positive.")
    print("This statement is inside the if block.")

print("This statement is outside the if block.")
```

```
The number is positive.
This statement is inside the if block.
This statement is outside the if block.
```

   In the example, the indentation (spaces or tabs at the beginning of lines) defines the scope of the `if` block. The two `print` statements inside the `if` block are executed only if the condition is true. The last `print` statement, not indented, is executed regardless of the condition.

5. How do you use the `elif` statement in Python?

   The `elif` statement (short for "else if") is used in Python to check multiple conditions in a series. It comes after an `if` statement and before an optional `else` statement. Let's look at an example:

```
# Example
score = 75

if score >= 90:
    print("Excellent!")
elif 80 <= score < 90:
    print("Very Good.")
elif 70 <= score < 80:
    print("Good.")
elif 60 <= score < 70:
    print("Fair.")
else:
    print("Needs Improvement.")
```

```
Good.
```

In this example, the `elif` statements allow checking multiple conditions sequentially. If the first condition is not true, it moves to the next one. The last `else` statement is optional and serves as a catch-all for cases that don't match any previous condition.

6. Explain the concept of nested `if` statements in Python.

In Python, nested `if` statements are used to handle more complex conditional logic. A nested `if` statement is an `if` statement inside another `if` statement. Each `if` statement is executed based on the condition of the outer `if` statement being true. Let's look at an example:

```python
# Example
score = 75

if score >= 60:
    print("Passing grade.")
    if score >= 90:
        print("Excellent!")
    elif score >= 80:
        print("Very Good.")
    elif score >= 70:
        print("Good.")
    else:
        print("Fair.")
else:
    print("Failing grade.")
```

```
Passing grade.
Good.
```

In this example, the outer `if` statement checks if the score is greater than or equal to 60. If true, it enters the block and prints "Passing grade." Then, there is another `if` statement nested inside, checking for specific grade ranges. This allows for more fine-grained evaluation based on different conditions.

7. What is the ternary conditional expression, and how is it used in Python?

In Python, the ternary conditional expression is a concise way to write a simple `if-else` statement. It is also known as the conditional expression or ternary operator. The syntax is:

```
x if condition else y
```

Here, x is the value to be returned if the condition is true, and y is the value to be returned if the condition is false.

Let's look at an example:

```
# Example
age = 20
message = "Teenager" if age >= 13 and age <= 19 else "Not a te
print(message)
```

```
Teenager
```

In this example, if the age is between 13 and 19 (inclusive), the value "Teenager" is assigned to the `message` variable; otherwise, "Not a teenager" is assigned.

8. How can you use the `pass` statement in an if block in Python?

In Python, the `pass` statement is a no-operation statement. It serves as a placeholder where syntactically some code is required but where no action is desired or necessary.

When using the `pass` statement within an `if` block, it allows you to create a valid code structure without any executable statements. This can be useful when you're working on building the structure of your program and want to leave some parts incomplete.

Let's see an example:

```
# Example
x = 10

if x > 5:
    print("x is greater than 5")
else:
    pass
```

```
x is greater than 5
```

In this example, the `if` block checks if x is greater than 5, and if true, it prints a message. The `else` block contains the `pass` statement, which does nothing. The program runs successfully without any errors.

9. Discuss the use of the `assert` statement in Python.

In Python, the `assert` statement is used for debugging purposes. It helps you identify bugs more quickly by allowing you to test conditions that you believe should hold true. If the condition specified in the `assert` statement evaluates to `False`, it raises an `AssertionError` exception, indicating that an assumption in your code has been violated.

Here is an example of how the `assert` statement is used:

```
# Example
x = 5

# Assert that x is greater than 0
assert x > 0, "Value of x should be greater than 0"

print("The value of x is:", x)
```

```
The value of x is: 5
```

In this example, the `assert` statement checks if `x` is greater than 0. Since the condition is true, the program continues to execute without any issues. If the condition were `False`, an `AssertionError` would be raised with the specified error message.

It's important to note that using `assert` is not a substitute for proper error handling in production code. It is primarily intended for debugging and should be used judiciously.

10. What is the purpose of the `if __name__ == "__main__":` block in Python scripts?

    The `if __name__ == "__main__":` block in a Python script serves the purpose of determining whether the script is being run as the main program or if it is being imported as a module into another script. This allows you to write code that can be reused in other scripts without executing it immediately upon import.

    When a Python script is executed, the interpreter sets a special built-in variable called `__name__`. If the script is the main program being executed, `__name__` is set to `"__main__"`; otherwise, if the script is imported as a module, `__name__` is set to the module's name.

    Here's an example to illustrate its use:

```
# Example script: my_script.py

def say_hello():
    print("Hello from my_script!")

# The following block will only be executed if this script is r
if __name__ == "__main__":
    say_hello()
```

```
Hello from my_script!
```

In this example, if `my_script.py` is run as the main program, it will execute the `say_hello()` function. However, if `my_script.py` is imported as a module into another script, the `say_hello()` function won't be executed immediately.
This construct is commonly used to separate reusable code from the code that should only run when the script is executed directly.

11. Explain the difference between the `==` and `is` operators in conditional statements.

    The `==` and `is` operators in conditional statements are used for different purposes in Python. The key difference lies in what they compare and how they perform the comparison.
    The `==` operator compares the values of two objects to check if they are equal. It checks whether the objects have the same content, regardless of their identity.
    The `is` operator, on the other hand, checks for object identity. It verifies if two objects refer to the exact same memory location, indicating that they are the same object in memory.
    Here's an example to illustrate the difference:

```
# Example program

# Create two lists with the same content
list1 = [1, 2, 3]
list2 = [1, 2, 3]

# Using == to check if the values are equal
print(list1 == list2)  # Outputs: True

# Using is to check if they are the same object
print(list1 is list2)  # Outputs: False
```

```
True
False
```

In this example, `list1 == list2` returns `True` because the lists have the same content. However, `list1 is list2` returns `False` because they are different objects in memory.

It's important to use `==` when comparing values and `is` when checking for object identity. While `==` is used in most equality comparisons, `is` is specifically used for identity comparisons.

12. How do you combine multiple conditions using logical operators (`and`, `or`) in Python?

    In Python, you can combine multiple conditions using logical operators such as `and` and `or`. These operators allow you to create complex conditions by combining simpler ones.

    Here's an example program illustrating the use of logical operators:

```python
# Example program

# Variables
age = 25
is_student = True

# Combining conditions using 'and'
if age > 18 and is_student:
    print("You are an adult student.")
else:
    print("You are not an adult student.")

# Combining conditions using 'or'
if age < 18 or is_student:
    print("You are either under 18 or a student.")
else:
    print("You are neither under 18 nor a student.")
```

```
You are an adult student.
You are either under 18 or a student.
```

In this example:
The first `if` statement checks whether the age is greater than 18 `and` the person is a student. If both conditions are true, it prints "You are an adult student."
The second `if` statement checks whether the age is less than 18 `or` the person is a student. If at least one condition is true, it prints "You are either under 18 or a student."
Logical operators are powerful tools for creating flexible and expressive conditions in Python.

13. Discuss the concept of truthy and falsy values in Python conditional statements.

In Python, values are considered either truthy or falsy when used in conditional statements. Truthy values are treated as `True`, while falsy values are treated as `False`.
Here's an example program illustrating truthy and falsy values:

```
# Example program

# Truthy values
if 1:
    print("1 is truthy")

if "Hello":
    print("Hello is truthy")

if [1, 2, 3]:
    print("List is truthy")

# Falsy values
if 0:
    print("0 is falsy")
else:
    print("0 is falsy")

if "":
    print("Empty string is falsy")
else:
    print("Empty string is falsy")

if []:
    print("Empty list is falsy")
else:
    print("Empty list is falsy")
```

```
1 is truthy
Hello is truthy
List is truthy
0 is falsy
Empty string is falsy
Empty list is falsy
```

In this example:

Values like `1`, `'Hello'`, and `[1, 2, 3]` are truthy, and the corresponding `if` statements print messages indicating truthiness.

Values like `0`, `''` (empty string), and `[]` (empty list) are falsy, and the corresponding `else` statements print messages indicating falsiness.

Understanding truthy and falsy values is essential for writing effective conditional statements in Python.

14. How do you handle exceptions using the`try`, `except`, `else`, and `finally` blocks in Python?

    In Python, exception handling is done using the `try`, `except`, `else`, and `finally` blocks. The `try` block contains the code that may raise an exception, and the `except` block contains the code to handle the exception.

    Here's an example program illustrating the usage of these blocks:

```python
# Example program

def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed")
    else:
        print(f"Result: {result}")
    finally:
        print("This block always executes")

# Example usage
divide_numbers(10, 2)
divide_numbers(5, 0)
```

```
Result: 5.0
Error: Division by zero is not allowed
This block always executes
```

In this example:
The `divide_numbers` function attempts to divide two numbers. If the division is successful, the `else` block is executed and prints the result. If a `ZeroDivisionError` occurs, the `except` block is executed, handling the exception.
The `finally` block always executes, regardless of whether an exception occurred or not.
This structure allows you to gracefully handle exceptions and ensure that certain code (in the `finally` block) runs regardless of whether an exception occurred or not.

15. Explain the use of the `in` and `not in` operators in Python conditional statements.

The `in` and `not in` operators in Python are used to check whether a value is present in a sequence (such as a string, list, or tuple) or not.
Here's an example program illustrating the usage of these operators:

```
# Example program

# Using 'in' operator
fruits = ['apple', 'orange', 'banana']
if 'apple' in fruits:
    print("Apple is present in the list")
else:
    print("Apple is not present in the list")

# Using 'not in' operator
colors = ('red', 'green', 'blue')
if 'yellow' not in colors:
    print("Yellow is not present in the tuple")
else:
    print("Yellow is present in the tuple")
```

```
Apple is present in the list
Yellow is not present in the tuple
```

In this example:
The `in` operator is used to check if 'apple' is present in the `fruits` list.
The `not in` operator is used to check if 'yellow' is not present in the `colors` tuple.
These operators are useful for conditional statements where you want to test the membership of a value in a sequence or its absence.

16. What is the purpose of the break and continue statements in loop structures within conditional statements?

In Python, the `break` and `continue` statements are used within loop structures to alter the flow of the loop based on certain conditions.
Here's an example program illustrating the use of `break` and `continue` statements within a `while` loop:

```
# Example program

# Using 'break' and 'continue' in a while loop
i = 0
while i < 5:
    i += 1
    if i == 3:
        print("Skipping iteration for i =", i)
        continue  # Skip the rest of the loop body and move to the next iteration
    print("Inside the loop for i =", i)
    if i == 4:
        print("Breaking out of the loop for i =", i)
        break  # Exit the loop when i is 4
```

```
Inside the loop for i = 1
Inside the loop for i = 2
Skipping iteration for i = 3
Inside the loop for i = 4
Breaking out of the loop for i = 4
```

In this example:

The `continue` statement is used to skip the rest of the loop body when `i` is equal to 3.

The `break` statement is used to exit the loop when `i` is equal to 4.

These statements provide flexibility in controlling the execution of loops based on certain conditions, making the code more efficient and readable.

17. How can you use the `assert` statement for debugging in Python?

The `assert` statement in Python is used for debugging purposes. It checks if a given expression is `True`, and if not, it raises an `AssertionError` exception with an optional error message.

Here's an example program illustrating the use of the `assert` statement for debugging:

```python
# Example program

# Using 'assert' for debugging
def divide(a, b):
    assert b != 0, "Cannot divide by zero"  # Check if 'b' is not zero
    return a / b

# Test cases
result1 = divide(10, 2)
print("Result 1:", result1)

result2 = divide(8, 0)  # This will raise an AssertionError
print("Result 2:", result2)
```

```
Result 1: 5.0
Traceback (most recent call last):
  File "example.py", line 11, in
    result2 = divide(8, 0)
  File "example.py", line 5, in divide
    assert b != 0, "Cannot divide by zero"
AssertionError: Cannot divide by zero
```

In this example:
The `assert` statement is used to check if `b` is not zero before performing division.
The first division (`divide(10, 2)`) is successful, and the result is printed.
The second division (`divide(8, 0)`) raises an `AssertionError` because dividing by zero is not allowed. The error message is also displayed.
Using `assert` statements can help identify and fix issues during development by quickly highlighting unexpected conditions.

18. Explain the concept of short-circuit evaluation in Python conditional statements.
In Python conditional statements, short-circuit evaluation is a behavior where the second operand of a logical expression is evaluated only if the first operand does not determine the outcome.
Here's an example program illustrating short-circuit evaluation:

```python
# Example program

# Using short-circuit evaluation
def is_positive(x):
    return x > 0

def is_even(x):
    return x % 2 == 0

# Short-circuit 'and' evaluation
result_and = is_positive(5) and is_even(4)
print("Result (and):", result_and)

# Short-circuit 'or' evaluation
result_or = is_positive(-2) or is_even(6)
print("Result (or):", result_or)
```

```
Result (and): False
Result (or): True
```

In this example:
The function `is_positive` returns `True` if the given number is positive.
The function `is_even` returns `True` if the given number is even.
Short-circuit 'and' evaluation: `is_positive(5)` is `True`, but `is_even(4)` is not evaluated because the first operand already determines the result as `False`.
Short-circuit 'or' evaluation: `is_positive(-2)` is `True`, and `is_even(6)` is not evaluated because the first operand already determines the result as `True`.
Short-circuit evaluation can be beneficial for efficiency, especially when the second operand involves a costly operation that can be skipped if the first operand is sufficient to determine the overall result.

19. How do you use the switch case statement in Python?

Python doesn't have a native `switch` statement like some other programming languages. However, you can achieve similar functionality using a dictionary and functions or lambda expressions. Here's an example:

```python
# Example program with a "switch" using a dictionary

def case1():
    return "This is case 1."

def case2():
    return "This is case 2."

def case3():
    return "This is case 3."

def default_case():
    return "This is the default case."

def switch_case(case_number):
    switch_dict = {
        1: case1,
        2: case2,
        3: case3
    }

    # Use get() to handle default case
    selected_case = switch_dict.get(case_number, default_case)
    return selected_case()

# Test cases
result1 = switch_case(1)
result2 = switch_case(2)
result3 = switch_case(3)
result_default = switch_case(5)

print(result1)
print(result2)
print(result3)
print(result_default)
```

```
This is case 1.
This is case 2.
This is case 3.
This is the default case.
```

In this example:
We define functions `case1`, `case2`, `case3`, and `default_case` to represent different cases and the default case.
The `switch_case` function takes a `case_number` as an argument and uses a dictionary (`switch_dict`) to map case numbers to corresponding functions.
The `get()` method is used to retrieve the function for a specific case. If the case is not found in the dictionary, it defaults to the `default_case` function.
We then call the selected function to get the result for the corresponding case.

While this approach simulates a `switch`-like behavior, keep in mind that Python's approach to handling such situations typically involves using `if`/`elif`/`else` statements or dictionaries.

20. Discuss the differences between the`if` statement and the `if` expression (PEP 308) in Python.

In Python, the `if` statement and the `if` expression (ternary conditional expression) serve different purposes, although both are used for conditional execution. Let's discuss each and provide examples.

1. `if` Statement:

```
# Example of an if statement
x = 10

if x > 5:
    result = "x is greater than 5"
else:
    result = "x is not greater than 5"

print(result)
```

```
x is greater than 5
```

In this example, the `if` statement checks whether `x` is greater than 5. If the condition is true, it executes the code block under the `if` branch; otherwise, it executes the code block under the `else` branch.

2. `if` Expression (PEP 308):

```
# Example of an if expression
x = 10

result = "x is greater than 5" if x > 5 else "x is not greater than 5"

print(result)
```

```
x is greater than 5
```

In this example, the `if` expression provides a more concise way to achieve the same result as the `if` statement. The syntax is `a if condition else b`, and it evaluates to `a` if the condition is true, otherwise `b`.

The key differences are:

`if` statement allows for more complex logic and multiple statements in each branch.
`if` expression is more concise and often used when the result is a simple expression.
`if` expression evaluates both `a` and `b`, while the `if` statement only executes the block corresponding to the true condition.

Section-B

1. What is Python?

The intent of the interviewer is to understand the candidate's knowledge of the programming language and how they define it.

Sample response: Python is a high-level, interactive programming language known for its simplicity and readability. It offers a versatile environment for developing applications in web development, data analysis, and machine learning. Its robust ecosystem helps programmers and developers build efficient solutions. It is a popular choice due to its community support and ability to handle both large-scale and small-scale projects.

2. What are the key features of Python?

The intent of the interviewer is to assess the key features of Python and to explain it to others in a summarized way.

Sample response: Its hey features are readable syntax, a standard library, and an ecosystem of third-party libraries. It supports object-oriented programming and offers cross-platform compatibility. In addition, strong community support is an additional feature.

3. What is PEP 8?

The intent of the interviewer is to assess the candidate's knowledge of coding standards and the official style guide of Python.

Sample response: PEP 8 is the official style guide for Python code, providing guidelines on code layout, naming conventions, and best practices. It ensures consistency, readability, and maintainability in projects which makes collaboration easy in projects.

4. How is memory managed in Python?

The intent of the interviewer is to understand the extent of the candidate's knowledge of Python's memory management.

Sample response: Python's memory is managed through garbage collection, where objects that are no longer referenced are freed. Its memory handles memory allocation and deallocation, that lets developers write code explicitly.

5. Explain the difference between a list and a tuple.

The intent of the interviewer is to assess the candidate's understanding of the difference between and the use cases of list and a tuple.

Sample response: Lists and tuple are sequence data types. The main difference between the two is while tuples represent immutable lists represent mutable. Lists are used when data needs to be modified and tuples are used for fixed data.

6. What is the difference between shallow copy and deep copy in Python?

The intent is to gauge the candidate's understanding of shallow copy and deep copy and their ability to explain them in a brief way.

Sample response: A shallow copy creates a new object in reference to existing data, so any changes in either affects both. In contrast, deep copy creates an independent copy with its own set of data, and they are unaffected by changes in either.

7. How do you handle exceptions in Python?

The intent is to assess the candidate's understanding of exception handling and detecting errors efficiently.

Sample response: Exceptions are handled using try-except block. The potential code is placed in 'try' block and exceptions if caught are handled in 'except' block. This helps in managing errors, logging them, and taking action to prevent program crashes.

8. Explain the concept of generators in Python.

The intent is to test the extent of the candidate's understanding of generators and how they explain it in a summarized manner.

Sample response: Generators in Python are functions that can be used for efficient iteration over a large dataset. They use "yield" keyword to generate value one-at-a time instead of storing them, making them useful for infinite or exceptionally large sets of data.

9. What is the purpose of "self" keyword in Python?

The intent behind this is to gain the candidate's perspective on the keyword and its usage in their daily life.

Sample response: The "self" keyword in Python refers to instance of object being accessed within a class method, allowing class to access and modify its

own attributes. Additionally, it serves as a reference point to enable encapsulation, and differentiate among instance and local variables.

10. What is the difference between a module and a package in Python?

The intent is to test the candidate's knowledge of modules and packages in Python and whether they understand the difference between both the terms.

Sample response: A module is a single file containing a code, on the other hand, package is a form of directory containing various modules. Modules are used to organize codes while package organizes related modules for better management and reusability.

11. How do you comment on a single line and multiple lines in Python?

The intent is to understand the candidate's knowledge of Python syntax and their documentation procedure.

Sample response: In Python, you can comment using # for single line and "''' for multiple line comments. These comments improve readability and provide context to reviewers.

12. Explain the concept of indentation in Python.

The intent behind the question is to test the candidate's knowledge of Python's indentation and its impact on language syntax and readability.

Sample response: Indentation defines the blocks of code. It replaces the need for braces or keywords and is crucial to improve code readability.

13. How do you handle exceptions in Python? Give an example.

The intent behind the question is to assess the ability of the candidate in handling exceptions and give a practical example to highlight their experience in handling one.

Sample response: In Python, exceptions are handles using try-except blocks. Within they try block the code is placed, and in the except block the underlying logic is written. [Give an example to this.]

14. What is the purpose of the if…else statement in Python? Provide an example.

The intent is to analyze the candidate's understanding of conditional statements in Python and explain if…else statements.

Sample response: The is…else statement allows to execute code base on the given conditions and control the flow of the program. So, with this the execution of true conditions is different from the false conditions.

15. How do you define a function in Python? Give an example.

The question's intent is to test the candidate's understanding of various functions in Python along with providing a clear and concise explanation of them.

Sample response: To define a function in Python, use "def" keyword followed by the name of the function and parentheses that includes the parameters.

16. What is a module in Python? How do you import a module?

The intent behind this is to identify the candidate's ability to import a module and the extent of their understanding of a module.

Sample response: In Python, a module is typically a type of file containing Python definitions, statements, and functions, also allowing to organize and reuse a code. If you want to import a module, use 'import' keyword followed by module name.

17. Explain the concept of class and an object in Python.

The intent is to understand the candidate's knowledge of object-oriented programming (OOP) concepts and concepts of class and an object.

Sample response: A class is typically a blueprint or template that defines the properties and behaviors of objects. Objects are instances of a class that represents individual entities with their unique characteristics and behaviors. They encapsulate data and functions within a single entity and enhance readability and modularity.

18. How do you create a file and write data into it using Python?

The intent is to assess the knowledge of candidates on file handling and their ability to write data into file using the programming language.

Sample response: To create a file and write data into it, use the 'open ()' function with file name and mode parameter and for writing use the 'w' mode. And then close the file to prevent data loss.

19. How do you read data from a file in Python?

The interviewer's intent is to assess the candidate's understanding of file handling and reading data from a file.

Sample response: To read data from a file use 'open ()' function to open it in desired mode. Then use methods like 'read ()'or 'readlines ()' to retrieve file's content.

20. Explain the concept of inheritance in Python.

The intent behind the question is to assess the candidate's understanding of the concept of inheritance.

Sample response: Inheritance helps a class to inherit attributes and methods from another class. It promotes code reuse and supports hierarchical organization of class. Create a subclass that inherits properties of the superclass and helps in reuse and extension of existing code.

21. What is the use of range () function in Python?

The intent is to gain the candidate's perception of range function and its purpose in programming.

Sample response: The range () function generates a sequence of numbers within a specified range and is used in loops to iterate over a sequence of values.

22. How do you write a conditional statement (if-else) in Python?

The intent is to assess the candidate's ability to understand conditional statements in Python and writing code using if-else conditions.

Sample response: In Python the following conditional statement is written using:

```
if condition:
    # Code to execute if condition is True
else:
    # Code to execute if condition is False
```

23. How do you handle exceptions in Python?

The intent of the interviewer is to assess the candidate's expertise in handling exceptions in Python.

Sample response: In Python exceptions are handled using try-except blocks. The code with a possibility of exception is placed in the try block and the exceptions are handled in the except block.

24. How do you read from and write to a file in Python?

The intent behind this question is to understand the ability to handle files and then read and write to files.

Sample response: To read a file in Python, use open () functions with the 'r' mode, then use methods like read () or readline () to extract the contents of the file. And use 'w' mode and write () method to add content. Finally, close the file using the close () method.

25. What is object-oriented programming (OOP) in Python?

The intent of the interviewer is to assess the candidate's understanding of OOP programming.

Sample response: Object-oriented programming (OOP) is a programming paradigm where code is organized around objects representing real-world entities and allows encapsulation, inheritance, and polymorphism. OOP makes it easier to develop and manage complex applications.

26. What is a class and how do you create an object in Python?

The intent of the interviewer is to assess the candidate's knowledge of classes and OOP in Python.

Sample response: A class is a blueprint for object creation in Python and defines the properties and behaviors of the objects. To create an object in Python, instantiate the class by calling the constructor using the class name and then follow it with a parenthesis.

27. What is inheritance in Python? How does it work?

The intent behind the question is to test the candidate's understanding of inheritance concept in Python and its usage in the programming language.

Sample response: Inheritance in Python is when a class inherits the attributes and methods of another class. It allows for code reusage and creation of hierarchical relationships among classes. This also allows for code organization and modularity.

28. What are modules in Python? How do you import and use them?

The intent of the interviewer is to assess the candidate's ability to understand the utilization of the code and modules in Python.

Sample response: Modules are files containing Python code that can be imported and used in other Python programs and help in organizing and reusing code. To import a mode use 'import' keyword followed by module name.

29. How do you work with dates and time in Python?

The intent behind this question is to assess the candidate's capability in handling dates, time and libraries and functions in Python.

Sample response: In Python we use the datetime module, which provides classes date and time to handle various operations. It also helps in parsing, formatting, arithmetic calculations, and comparisons.

30. What are some built-in libraries and functions in Python?

The intent behind the question is to assess the candidate's understanding of built-in Python functions and libraries and their familiarity with the capabilities and tools of the programming language.

Sample response: Python provides a range of built-in functions and offers standard libraries like math, datetime and random that enhances Python's functionality and flexibility.

Section - C

1. Describe Python's if-elif-else statement.

Answer: The `if-elif-else` statement is used for conditional branching in Python. It allows you to specify multiple conditions and execute different code blocks based on the first condition that evaluates to `True`.

```
if condition1:
    # Code block to execute if condition1 is True
elif condition2:
    # Code block to execute if condition2 is True
else:
    # Code block to execute if none of the conditions are True
```

2. What is a for loop, and how does it work in Python?

Answer: A `for` loop in Python is used for iterating over a sequence (such as a list, tuple, or string) or any iterable object. It executes a block of code for each item in the sequence.

```
for item in iterable:
# Code block to execute for each item in the iterable
```

3. Explain the while loop and give an example of its usage.

Answer: A `while` loop in Python is used for repeated execution of a block of code as long as a specified condition is `True`. Here's an example:

```
count = 0
while count < 5:
    print(count)
    count += 1
```

4. How do you write an if statement with multiple conditions in Python?

Answer: You can use `if`, `elif`, and `else` to handle multiple conditions in an if-elif-else statement.

```
if condition1:
    # Code block to execute if condition1 is True
elif condition2:
    # Code block to execute if condition2 is True
else:
    # Code block to execute if none of the conditions are True
```

5. What is the purpose of the `else` clause in an if-elif-else statement?

Answer: The `else` clause is executed when none of the preceding conditions in an if-elif-else

```
if condition1:
    # Code block to execute if condition1 is True
elif condition2:
    # Code block to execute if condition2 is True
else:
    # Code block to execute if none of the conditions are True
```

6. Explain the concept of nested loops in Python.

Answer: Nested loops are loops within loops. They are used for more complex iterations, such as iterating over elements in a 2D list.

```
for i in range(3):
for j in range(3):
    # Code block to execute for each combination of i and j
```

7. How can you exit a loop prematurely in Python?

Answer: You can use the `break` statement to exit a loop prematurely when a specific condition is met.

8. What is the `continue` statement used for in Python loops?

Answer: The `continue` statement is used to skip the current iteration of a loop and continue with the next iteration.

```
Copy
for item in iterable:
if condition:
    continue  # Skip the current iteration and continue with t
```

9. Describe the `range()` function in Python and its usage in for loops.

Answer: `range()` generates a sequence of numbers. It's often used in `for` loops to specify the number of iterations.

```
for i in range(start, stop, step):
# Code block to execute for each value of i in the range
```

10. How do you iterate over the elements and their indices in a list using a for loop?

Answer: You can use the `enumerate()` function to iterate over elements and their indices in a list using a `for` loop.

```
for index, item in enumerate(iterable):
# Code block to execute for each item and its index
```

11. What is an infinite loop in Python, and how can you break out of it?

- An infinite loop is a loop that continues to execute without stopping. You can break out of it using the `break` statement.

12. Explain the use of the `pass` statement in Python loops.

- The `pass` statement is a no-op and is often used as a placeholder when syntactically some code is required but no action is needed in a loop.

13. How do you iterate over the keys and values in a dictionary using a for loop? – You can use the `.items()` method to iterate over key-value pairs in a dictionary using a `for` loop.

14. What is the purpose of the `else` block in a while loop in Python? – The `else` block in a `while` loop is executed when the loop's condition becomes `False`.

15. How can you simulate a do-while loop in Python? – You can simulate a do-while loop using a `while` loop with a conditional check at the end.

16. Explain the concept of a loop control variable. – A loop control variable is a variable used to control the flow and behavior of a loop.

17. What is the role of the `in` keyword in Python loops? – The `in` keyword is used to iterate over elements in an iterable, such as a list or string.

18. How do you find the maximum and minimum values in a list using loops in Python? – You can use a `for` loop to iterate over the list and keep track of the maximum and minimum values.

19. What are nested loops, and when are they commonly used? – Nested loops are loops within loops. They are used when you need to iterate over multiple dimensions, like rows and columns of a 2D array.

20. Explain the use of the `range()` function with `for` loops in Python. – `range()` generates a sequence of numbers, often used as an iterable in `for` loops to specify the number of iterations.

21. How can you avoid an infinite loop while waiting for user input in a Python program? – You can include a conditional check for user input and use the `break` statement to exit the loop when a specific input is received.

22. Describe the difference between `for` and `while` loops, and when should each be used? – `for` loops are used when you know the number of iterations in advance, while `while` loops are used when you don't know the exact number of iterations and want to loop until a condition is met.

23. How do you use the `enumerate()` function in a `for` loop in Python? – `enumerate()` returns an iterable of tuples containing the index and value of each element, which can be used in a `for` loop to iterate over elements and their indices.

24. Explain the purpose of the `break` statement in Python loops. – The `break` statement is used to exit a loop prematurely when a specific condition is met.

25. How do you create a loop that iterates over a list in reverse order in Python? – You can use slicing with a negative step size in a `for` loop to iterate over a list in reverse order.

26. What is the purpose of the `continue` statement in Python loops? – The `continue` statement is used to skip the current iteration of a loop and continue with the next iteration.

27. How can you iterate over characters in a string using a `for` loop in Python? – You can use a `for` loop to iterate over characters in a string as if it were a list of characters.

28. Explain the difference between a loop and a recursive function in Python. – A loop is an iterative construct, while a recursive function calls itself to solve a problem.

29. What is an infinite loop, and how can you avoid it in Python? – An infinite loop is a loop that runs indefinitely. You can avoid it by ensuring that there's a condition or a way to exit the loop.

30. How do you iterate over the elements of a list in reverse order using a `for` loop in Python? – You can use the `reversed()` function in a `for` loop to iterate over the elements of a list in reverse order.

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

for item in reversed(my_list):
    # Code block to iterate over elements in reverse order
    print(item)
```

31. Explain the concept of an iterator and give an example in Python.

Answer: An iterator is an object that can be iterated (looped) over. An example is the use of the `iter()` and `next()` functions to create and iterate over an iterator.

```
my_list = [1, 2, 3, 4, 5]
my_iterator = iter(my_list)  # Create an iterator from a list

while True:
    try:
        item = next(my_iterator)
        # Code block to process each item in the iterator
        print(item)
    except StopIteration:
        break  # Exit the loop when the iterator is exhausted
```

32. How can you skip a specific iteration in a for loop without terminating the loop in Python?

Answer: You can use the `continue` statement to skip a specific iteration and continue with the next one in a `for` loop.

```python
my_list = [1, 2, 3, 4, 5]

for item in my_list:
    if item == 3:
        continue  # Skip iteration when item is 3
    # Code block to process items (except when item is 3)
    print(item)
```

33. Describe the `range()` function with three arguments in Python.

Answer: The `range()` function with three arguments allows you to specify the start, stop, and step values for generating a sequence of numbers.

```python
for i in range(start, stop, step):
# Code block to execute for each value of i in the range
```

34. What is the purpose of the `pass` statement in Python loops?

Answer: The `pass` statement is used as a placeholder when syntactically some code is required, but no action is needed in a loop.

```python
for item in iterable:
if condition:
    pass  # Placeholder for future code
# Code block to process items
```

35. How can you use a for loop to iterate over a dictionary's keys and values in Python?

Answer: You can use the `.items()` method to iterate over both keys and values in a dictionary within a `for` loop.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

for key, value in my_dict.items():
    # Code block to process each key and value in the dictiona
    print(f"Key: {key}, Value: {value}")
```

36. What is the role of an iterable in a `for` loop, and how can you create a custom iterable in Python? – An iterable is an object that can be looped over in a `for` loop. You can create a custom iterable by implementing the `__iter__()` and `__next__()` methods in a class.

37. Explain the purpose of the `else` block in a `for` loop in Python. – The `else` block in a `for` loop is executed after the loop completes its iterations and is not interrupted by a `break` statement.

38. How can you use the `zip()` function in Python to iterate over multiple iterables simultaneously? – You can use the `zip()` function to combine multiple iterables and iterate over them simultaneously in a `for` loop.

39. Describe the `range()` function with one argument in Python. – The `range()` function with one argument generates a sequence of numbers starting from 0 up to, but not including, the specified value.

40. How do you create an infinite loop in Python, and when might you use one? – You can create an infinite loop using `while True:`. Infinite loops are used in situations where you need continuous execution, such as server programs.

41. What is the purpose of the `for-else` construct in Python? – The `for-else` construct allows you to specify code to execute if a `for` loop completes without encountering a `break` statement.

42. How can you control the flow of a loop using the `continue` statement in Python? – The `continue` statement allows you to skip the current iteration and continue with the next iteration in a loop.

43. What is the difference between a loop and a conditional statement in Python? – A loop is used for repetitive tasks, while a conditional statement is used for making decisions based on conditions.

44. Explain the purpose of the `break` statement in Python loops. – The `break` statement is used to exit a loop prematurely when a specific condition is met.

45. How can you use the `while` loop to iterate over elements in a list in Python? – You can use a `while` loop with an index variable to iterate over elements in a list.

46. What is the `enumerate()` function in Python, and how is it used in loops? – The `enumerate()` function adds an index to an iterable and is commonly used in `for` loops to iterate over elements and their indices.

47. Explain the use of the `in` operator in Python loops. – The `in` operator is used to check if an element is present in an iterable and is commonly used in loops for membership testing.

48. How do you find the sum of elements in a list using a `for` loop in Python? – You can use a `for` loop to iterate over the elements of a list and accumulate their values to find the sum.

49. Describe the `range()` function with two arguments in Python. – The `range()` function with two arguments generates a sequence of numbers starting from the first argument and up to, but not including, the second argument.

50. How do you create an infinite loop that terminates on user input in Python? – You can create an infinite loop that terminates when specific user input is received by using the `break` statement to exit the loop.

Section - D

1. A
2. A
3. A
4. A
5. C

6. D
7. C
8. A
9. A
10. A
11. B
12. D
13. A
14. D
15. A
16. A
17. C
18. A
19. B
20. A
21. A
22. A
23. B
24. A
25. A
26. A
27. C
28. D
29. B
30. C
31. B
32. A
33. A

Section - E

Python Code:

```
# Create an empty list to store numbers that meet the given conditions
nl = []

# Iterate through numbers from 1500 to 2700 (inclusive)
for x in range(1500, 2701):
    # Check if the number is divisible by 7 and 5 without any remainder
    if (x % 7 == 0) and (x % 5 == 0):
        # If the conditions are met, convert the number to a string and append it to the list
        nl.append(str(x))

# Join the numbers in the list with a comma and print the result
print(','.join(nl))
```

Sample Output:

1.

```
1505,1540,1575,1610,1645,1680,1715,1750,1785,1820,1855,1890,1925,1960,1995,2030,2065,2100,2135,21
2275,2310,2345,2380,2415,2450,2485,2520,2555,2590,2625,2660,2695
```

Sample Solution:

Python Code:

```
# Prompt the user to input a temperature in the format (e.g., 45F, 102C, etc.)
temp = input("Input the temperature you like to convert? (e.g., 45F, 102C etc.) : ")

# Extract the numerical part of the temperature and convert it to an integer
degree = int(temp[:-1])

# Extract the convention part of the temperature input (either 'C' or 'F')
i_convention = temp[-1]

# Check if the input convention is in uppercase 'C' (Celsius)
if i_convention.upper() == "C":
    # Convert the Celsius temperature to Fahrenheit
    result = int(round((9 * degree) / 5 + 32))
    o_convention = "Fahrenheit"  # Set the output convention as Fahrenheit
# Check if the input convention is in uppercase 'F' (Fahrenheit)
elif i_convention.upper() == "F":
    # Convert the Fahrenheit temperature to Celsius
    result = int(round((degree - 32) * 5 / 9))
    o_convention = "Celsius"  # Set the output convention as Celsius
else:
    # If the input convention is neither 'C' nor 'F', print an error message and exit the program
    print("Input proper convention.")
    quit()

# Display the converted temperature in the specified output convention
print("The temperature in", o_convention, "is", result, "degrees.")
```

Sample Output:

2.

```
Input the  temperature you like to convert? (e.g., 45F, 102C etc.) : 104f
The temperature in Celsius is 40 degrees.
```

```
# Import the 'random' module to generate random numbers
import random

# Generate a random number between 1 and 10 (inclusive) as the target number
target_num, guess_num = random.randint(1, 10), 0

# Start a loop that continues until the guessed number matches the target number
while target_num != guess_num:
    # Prompt the user to input a number between 1 and 10 and convert it to an integer
    guess_num = int(input('Guess a number between 1 and 10 until you get it right : '))

# Print a message indicating successful guessing once the correct number is guessed
print('Well guessed!')
```

Sample Output:

3.

```
Guess a number between 1 and 10 until you get it right : 5
Well guessed!
```

Sample Solution:

Python Code:

```
# Set the value of 'n' to 5 (this will determine the number of lines in the pattern)
n = 5

# Iterate through the range of numbers from 0 to 'n' (exclusive)
for i in range(n):
    # Iterate through the range of numbers from 0 to 'i' (exclusive) for each 'i' in the outer loop
    for j in range(i):
        # Print '*' followed by a space without a new line (end="" ensures printing in the same line)
        print('* ', end="")
    # Move to the next line after printing '*' characters for the current 'i'
    print('')

# Iterate through the range of numbers from 'n' down to 1 (inclusive), decreasing by 1 in each iteration
for i in range(n, 0, -1):
    # Iterate through the range of numbers from 0 to 'i' (exclusive) for each 'i' in the outer loop
    for j in range(i):
        # Print '*' followed by a space without a new line (end="" ensures printing in the same line)
        print('* ', end="")
    # Move to the next line after printing '*' characters for the current 'i'
    print('')

```

4.

Python Code:

```
# Prompt the user to input a word
word = input("Input a word to reverse: ")

# Iterate through the characters of the word in reverse order
for char in range(len(word) - 1, -1, -1):
    # Print each character from the word in reverse order without a new line (end="")
    print(word[char], end="")

# Print a new line to separate the reversed word from the next output
print("\n")
```

Sample Output:

```
Input a word to reverse: ecruoser3w
w3resource
```

5.

Sample Solution:

Python Code:

```
# Create a tuple named 'numbers' containing integer values from 1 to 9
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

# Initialize counters for counting odd and even numbers
count_odd = 0
count_even = 0

# Iterate through each element 'x' in the tuple 'numbers'
for x in numbers:
    # Check if the current number 'x' is even by evaluating 'not x % 2'
    if not x % 2:  # If 'x' modulo 2 equals 0, it's even
        # Increment the count of even numbers
        count_even += 1
    else:
        # If 'x' modulo 2 doesn't equal 0, it's odd; increment the count of odd numbers
        count_odd += 1

# Print the total count of even and odd numbers
print("Number of even numbers:", count_even)
print("Number of odd numbers:", count_odd)
```

Sample Output:

```
Number of even numbers : 4
Number of odd numbers : 5
```

6.

7.

Python Code:

```
# Create a list named 'datalist' containing various data types (int, float, complex, bool, string, tuple, list, dictionary)
datalist = [1452, 11.23, 1+2j, True, 'w3resource', (0, -1), [5, 12], {"class": 'V', "section": 'A'}]

# Iterate through each item in the 'datalist'
for item in datalist:
    # Print the type of each item in the list along with the item itself
    print("Type of", item, "is", type(item))

```

Sample Output:

```
Type of  1452  is  <class 'int'>
Type of  11.23  is  <class 'float'>
Type of  (1+2j)  is  <class 'complex'>
Type of  True  is  <class 'bool'>
Type of  w3resource  is  <class 'str'>
Type of  (0, -1)  is  <class 'tuple'>
Type of  [5, 12]  is  <class 'list'>
Type of  {'class': 'V', 'section': 'A'}  is  <class 'dict'>
```

8.

Python Code:

```
# Iterate through numbers from 0 to 5 using the range function
for x in range(6):
    # Check if the current value of 'x' is equal to 3 or 6
    if (x == 3 or x == 6):
        # If 'x' is 3 or 6, skip to the next iteration without executing the code below
        continue
    # Print the value of 'x' with a space and without a newline (end=' ')
    print(x, end=' ')

# Print a new line after printing all numbers satisfying the condition
print("\n")

```

Sample Output:

```
0 1 2 4 5
```

9.

Python Code:

```
# Initialize variables 'x' and 'y' with values 0 and 1, respectively
x, y = 0, 1

# Execute the while loop until the value of 'y' becomes greater than or equal to 50
while y < 50:
    # Print the current value of 'y'
    print(y)

    # Update the values of 'x' and 'y' using simultaneous assignment,
    # where 'x' becomes the previous value of 'y' and 'y' becomes the sum of 'x' and the previous value of 'y'
    x, y = y, x + y
```

Sample Output:

```
1
1
2
3
5
8
13
21
34
```

10.

```python
def fizz_buzz(start, end):
    """
    Print 'Fizz', 'Buzz', or 'FizzBuzz' for numbers in the specified range based on divisibility rules.

    Parameters:
    start (int): The starting number of the range.
    end (int): The ending number of the range.
    """
    if start > end:
        print("Start should be less than or equal to end.")
        return

    for fizzbuzz in range(start, end + 1):
        if fizzbuzz % 3 == 0 and fizzbuzz % 5 == 0:
            print("FizzBuzz")
        elif fizzbuzz % 3 == 0:
            print("Fizz")
        elif fizzbuzz % 5 == 0:
            print("Buzz")
        else:
            print(fizzbuzz)

# Example usage
print("From 1 to 10")
fizz_buzz(1, 10)
print("\nFrom 50 to 75")
fizz_buzz(50, 75)
```

Output:

```
From 1 to 10
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz

From 50 to 75
Buzz
Fizz
52
53
Fizz
Buzz
56
Fizz
.......

68
Fizz
Buzz
71
Fizz
73
74
FizzBuzz
```

11.

```python
# Prompt the user to input the number of rows
row_num = int(input("Input number of rows: "))

# Prompt the user to input the number of columns
col_num = int(input("Input number of columns: "))

# Create a 2D list (a list of lists) filled with zeros using list comprehension
# The outer list will have 'row_num' elements and the inner lists will have 'col_num' elements
multi_list = [[0 for col in range(col_num)] for row in range(row_num)]

# Nested loop to populate the 2D list with multiplication results
for row in range(row_num):
    for col in range(col_num):
        # Set the value at position [row][col] in the 2D list to the product of 'row' and 'col'
        multi_list[row][col] = row * col

# Print the resulting 2D list containing the multiplication table
print(multi_list)
```

Sample Output:

```
Input number of rows: 3
Input number of columns: 4
[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]
```

```python
# Create an empty list named 'lines'
lines = []

# Continue to prompt the user for input indefinitely until a blank line is entered
while True:
    # Prompt the user to input a line of text
    l = input()

    # Check if the entered line is not empty (non-empty string)
    if l:
        # Convert the entered line to uppercase and append it to the 'lines' list
        lines.append(l.upper())
    else:
        # If the entered line is empty, break out of the loop
        break;

# Iterate through each line in the 'lines' list
for l in lines:
    # Print each line (converted to uppercase) from the 'lines' list
    print(l)
```

Sample Output:

```
w3resource
Python

W3RESOURCE
PYTHON
```

12.

Python Code:

```python
# Create an empty list named 'items'
items = []

# Take user input and split it into a list of strings using ',' as the delimiter
num = [x for x in input().split(',')]

# Iterate through each element 'p' in the 'num' list
for p in num:
    # Convert the binary string 'p' to its decimal equivalent 'x'
    x = int(p, 2)

    # Check if 'x' is divisible by 5 (i.e., when divided by 5 there's no remainder)
    if not x % 5:
        # If 'x' is divisible by 5, add the binary string 'p' to the 'items' list
        items.append(p)

# Join the elements in the 'items' list separated by ',' and print the result
print(','.join(items))
```

Sample Output:

```
0001,0010,0011,0100,0101,0110,0111
0101
```

13.

```python
d = l = 0

# Iterate through each character 'c' in the input string 's'
for c in s:
    # Check if the current character 'c' is a digit
    if c.isdigit():
        # If 'c' is a digit, increment the count of digits ('d')
        d = d + 1
    # Check if the current character 'c' is an alphabet letter
    elif c.isalpha():
        # If 'c' is an alphabet letter, increment the count of letters ('l')
        l = l + 1
    else:
        # If 'c' is neither a digit nor an alphabet letter, do nothing ('pass')
        pass

# Print the total count of letters ('l') and digits ('d') in the input string 's'
print("Letters", l)
print("Digits", d)
```

Sample Output:

```
Input a string W3resource
Letters 9
Digits 1
```

14.

15.

Python Code:

```
# Import the 're' module for regular expressions
import re

# Prompt the user to input a password and store it in the variable 'p'
p = input("Input your password")

# Set 'x' to True to enter the while loop
x = True

# Start a while loop that continues until 'x' is True
while x:
    # Check conditions for a valid password:
    # Password length should be between 6 and 12 characters
    if (len(p) < 6 or len(p) > 12):
        break
    # Password should contain at least one lowercase letter
    elif not re.search("[a-z]", p):
        break
    # Password should contain at least one digit
    elif not re.search("[0-9]", p):
        break
    # Password should contain at least one uppercase letter
    elif not re.search("[A-Z]", p):
        break
    # Password should contain at least one special character among '$', '#', '@'
    elif not re.search("[$#@]", p):
        break
    # Password should not contain any whitespace character
    elif re.search("\s", p):
        break
    else:
        # If all conditions are met, print "Valid Password" and set 'x' to False to exit the loop
        print("Valid Password")
        x = False
        break

# If 'x' remains True, print "Not a Valid Password"
if x:
    print("Not a Valid Password")
```

```
Input your passwordW3r@100a
Valid Password
```

16.

Python Code:

```
# Create an empty list named 'items'
items = []

# Iterate through numbers from 100 to 400 (inclusive) using the range function
for i in range(100, 401):
    # Convert the current number 'i' to a string and store it in the variable 's'
    s = str(i)

    # Check if each digit in the current number 'i' is even (divisible by 2)
    if (int(s[0]) % 2 == 0) and (int(s[1]) % 2 == 0) and (int(s[2]) % 2 == 0):
        # If all digits are even, append the string representation of 'i' to the 'items' list
        items.append(s)

# Join the elements in the 'items' list separated by ',' and print the result
print(",".join(items))
```

Sample Output:

```
200,202,204,206,208,220,222,224,226,228,240,242,244,246,248,260,262,264,266,268,280,282,284,286,288,400
```

17.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""
# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (((column == 1 or column == 5) and row != 0) or ((row == 0 or row == 3) and (column > 1 and column < 5))):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or ((row == 0 or row == 6) and (column > 1 and column < 5)) or (column == 5 and row != 0 and row != 6)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

18.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or ((row == 0 or row == 6) and (column > 1 and column < 6)) or (row == 3 and column > 1 and column < 5)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

19.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if ((column == 1 and row != 0 and row != 6) or ((row == 0 or row == 6) and column > 1 and column < 5) or (row == 3 and column > 2 and column < 6) or (column == 5 and row != 0 and row != 2 and
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

20.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or (row == 6 and column != 0 and column < 6)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

21.

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or column == 5 or (row == 2 and (column == 2 or column == 4)) or (row == 3 and column == 3)):
            result_str = result_str + "* "  # Append '*' followed by a space (' ') to the 'result_str'
        else:
            result_str = result_str + "  "  # Append two spaces ('  ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

22.

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (((column == 1 or column == 5) and row != 0 and row != 6) or ((row == 0 or row == 6) and column > 1 and column < 5)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

23. Flowchart :

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Iterate through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or ((row == 0 or row == 3) and column > 0 and column < 5) or ((column == 5 or column == 1) and (row == 1 or row == 2))):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

24.

```
for row in range(0, 7):
    # Iterate through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (column == 1 or ((row == 0 or row == 3) and column > 1 and column < 5) or (column == 5 and row != 0 and row < 3) or (column == row - 1 and row > 2)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

25.

26.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Loop through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Loop through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if ((row == 0 or row == 3 or row == 6) and column > 1 and column < 5) or
            (column == 1 and (row == 1 or row == 2 or row == 6)) or
            (column == 5 and (row == 0 or row == 4 or row == 5))):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern for the first shape
print(result_str)

# Define variables 'row' and 'col' with values 15 and 18 respectively
row = 15
col = 18
result_str = ""  # Reinitialize the 'result_str' to an empty string

# Loop through values from 1 to 'row' using the range function
for i in range(1, row+1):
    # Check conditions to determine the pattern for each row

    # If conditions are met, append 'o' to the 'result_str' in specific positions
    if ((i <= 3) or (i >= 7 and i <= 9) or (i >= 13 and i <= 15)):
        for j in range(1, col):
            result_str = result_str + "o"  # Append 'o' to the 'result_str'
        result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'
    elif (i >= 4 and i <= 6):
        for j in range(1, 5):
            result_str = result_str + "o"  # Append 'o' to the 'result_str'
        result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'
    else:
        for j in range(1, 14):
            result_str = result_str + " "  # Append space (' ') to the 'result_str'
        for j in range(1, 5):
            result_str = result_str + "o"  # Append 'o' to the 'result_str'
        result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern for the second shape
print(result_str)
```

Sample Output:

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


oooooooooooooo
oooooooooooooo
oooooooooooooo
oooo
oooo
oooo
oooooooooooooo
oooooooooooooo
oooooooooooooo
          oooo
          oooo
          oooo
oooooooooooooo
oooooooooooooo
oooooooooooooo
```

Python Code:

```
result_str="";
for row in range(0,7):
    for column in range(0,7):
        if (column == 3 or (row == 0 and column > 0 and column <6)):
            result_str=result_str+"*"
        else:
            result_str=result_str+" "
    result_str=result_str+"\n"
print(result_str);
```

Sample Output:

27.

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

```
# Initialize an empty string named 'result_str'
result_str = ""

# Loop through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Loop through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (((column == 1 or column == 5) and row != 6) or (row == 6 and column > 1 and column < 5)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

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

28.

Python Code:

```
# Initialize an empty string named 'result_str'
result_str = ""

# Loop through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Loop through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (((column == 1 or column == 5) and (row > 4 or row < 2)) or
            row == column and column > 0 and column < 6 or
            (column == 2 and row == 4) or
            (column == 4 and row == 2)):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

29.

Sample Output:

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

```
# Initialize an empty string named 'result_str'
result_str = ""

# Loop through rows from 0 to 6 using the range function
for row in range(0, 7):
    # Loop through columns from 0 to 6 using the range function
    for column in range(0, 7):
        # Check conditions to determine whether to place '*' or ' ' in the result string

        # If conditions are met, place '*' in specific positions based on row and column values
        if (((row == 0 or row == 6) and column >= 0 and column <= 6) or row + column == 6):
            result_str = result_str + "*"  # Append '*' to the 'result_str'
        else:
            result_str = result_str + " "  # Append space (' ') to the 'result_str'

    result_str = result_str + "\n"  # Add a newline character after each row in 'result_str'

# Print the final 'result_str' containing the pattern
print(result_str)
```

Sample Output:

30.
```
*******
     *
    *
   *
  *
 *
*******
```

Python Code:

```
# Request input from the user to provide a dog's age in human years and convert it to an integer
h_age = int(input("Input a dog's age in human years: "))

# Check if the entered age is less than zero; if true, display an error message and exit the program
if h_age < 0:
    print("Age must be a positive number.")
    exit()
# If the entered age is 2 years or less, calculate the dog's age in dog's years using the formula 10.5 times the human years
elif h_age <= 2:
    d_age = h_age * 10.5
# For ages greater than 2, calculate the dog's age in dog's years using the formula 21 plus 4 times the remaining human years after 2
else:
    d_age = 21 + (h_age - 2) * 4

# Display the calculated dog's age in dog's years
print("The dog's age in dog's years is", d_age)
```

Sample Output:

31.
```
Input a dog's age in human years: 12
The dog's age in dog's years is 61
```

Python Code:

```
# Request input from the user to input a letter of the alphabet and assign it to the variable 'l'
l = input("Input a letter of the alphabet: ")

# Check if the input letter 'l' is present in the tuple containing vowels ('a', 'e', 'i', 'o', 'u')
if l in ('a', 'e', 'i', 'o', 'u'):
    print("%s is a vowel." % l)  # Display a message stating that the input letter is a vowel
# If the input letter is 'y', display a message stating that it sometimes represents a vowel and sometimes a consonant
elif l == 'y':
    print("Sometimes the letter y stands for a vowel, sometimes for a consonant.")
# If the input letter doesn't fall into any of the above conditions, it is considered a consonant; display a message stating so
else:
    print("%s is a consonant." % l)

```

Sample Output:

32.
```
Input a letter of the alphabet: u
u is a vowel.
```

```
# Display a list of months to the user
print("List of months: January, February, March, April, May, June, July, August, September, October, November, December")

# Request input from the user to enter the name of a month and assign it to the variable 'month_name'
month_name = input("Input the name of Month: ")

# Check the input 'month_name' and provide the number of days based on the entered month
if month_name == "February":
    print("No. of days: 28/29 days")  # Display the number of days in February (28 or 29 days for leap years)
elif month_name in ("April", "June", "September", "November"):
    print("No. of days: 30 days")  # Display the number of days for months having 30 days
elif month_name in ("January", "March", "May", "July", "August", "October", "December"):
    print("No. of days: 31 days")  # Display the number of days for months having 31 days
else:
    print("Wrong month name")  # If the entered month name doesn't match any of the above conditions, display an error message
```

Sample Output:

33.
```
List of months: January, February, March, April, May, June, July, August, September, October, November, Decemb
er
Input the name of Month: April
No. of days: 30 days
```

```python
# Define a function named 'sum' that takes two parameters 'x' and 'y'
def sum(x, y):
    # Calculate the sum of 'x' and 'y' and assign it to the variable 'sum'
    sum = x + y

    # Check if the calculated sum falls within the range of 15 to 19 (inclusive)
    if sum in range(15, 20):
        return 20  # If the sum falls within the specified range, return 20
    else:
        return sum  # If the sum doesn't fall within the specified range, return the calculated sum

# Call the 'sum' function with different arguments and print the results
print(sum(10, 6))   # Call the function 'sum' with arguments 10 and 6, then print the result
print(sum(10, 2))   # Call the function 'sum' with arguments 10 and 2, then print the result
print(sum(10, 12))  # Call the function 'sum' with arguments 10 and 12, then print the result
```

Sample Output:

```
20
12
22
```

34.

```python
# Request input from the user to input a string and assign it to the variable 'text'
text = input("Input a string: ")

# Remove leading and trailing whitespaces from the input string using the 'strip()' function
text = text.strip()

# Check if the length of the cleaned text is less than 1
if len(text) < 1:
    print("Input a valid text")  # If the text length is less than 1, display a message asking for valid input
else:
    # Check if all characters in the cleaned text are digits (0-9), indicating an integer
    if all(text[i] in "0123456789" for i in range(len(text))):
        print("The string is an integer.")  # If all characters are digits, display a message indicating an integer
    # Check if the text starts with a '+' or '-' and all subsequent characters are digits
    elif (text[0] in "+-") and all(text[i] in "0123456789" for i in range(1, len(text))):
        print("The string is an integer.")  # If the conditions for a signed integer are met, display an integer message
    else:
        print("The string is not an integer.")  # If none of the conditions are met, display a non-integer message
```

Sample Output:

```
Input a string: python
The string is not an integer.
```

35.

Python Code:

```python
# Display a message prompting the user to input lengths of the sides of a triangle
print("Input lengths of the triangle sides: ")

# Request input from the user for the length of side 'x' and convert it to an integer
x = int(input("x: "))

# Request input from the user for the length of side 'y' and convert it to an integer
y = int(input("y: "))

# Request input from the user for the length of side 'z' and convert it to an integer
z = int(input("z: "))

# Check conditions to determine the type of triangle based on the lengths of its sides

# If all sides are equal, display that it's an equilateral triangle
if x == y == z:
    print("Equilateral triangle")
# If at least two sides are equal, display that it's an isosceles triangle
elif x == y or y == z or z == x:
    print("Isosceles triangle")
# If all sides have different lengths, display that it's a scalene triangle
else:
    print("Scalene triangle")
```

36.

Sample Output:

```
x: 6
y: 8
z: 12
Scalene triangle
```

37.

Python Code:

```
# Request input from the user for the name of the month and assign it to the variable 'month'
month = input("Input the month (e.g. January, February etc.): ")

# Request input from the user for the day and convert it to an integer, assigning it to the variable 'day'
day = int(input("Input the day: "))

# Check the input 'month' to determine the season based on the month and day provided

# Check if the input month falls within the winter season
if month in ('January', 'February', 'March'):
    season = 'winter'
# Check if the input month falls within the spring season
elif month in ('April', 'May', 'June'):
    season = 'spring'
# Check if the input month falls within the summer season
elif month in ('July', 'August', 'September'):
    season = 'summer'
# For other months, assign the season as autumn by default
else:
    season = 'autumn'

# Adjust the season based on the specific day within certain months

# Check if the input month is March and the day is after March 19, updating the season to spring
if (month == 'March') and (day > 19):
    season = 'spring'
# Check if the input month is June and the day is after June 20, updating the season to summer
elif (month == 'June') and (day > 20):
    season = 'summer'
# Check if the input month is September and the day is after September 21, updating the season to autumn
elif (month == 'September') and (day > 21):
    season = 'autumn'
# Check if the input month is December and the day is after December 20, updating the season to winter
elif (month == 'December') and (day > 20):
    season = 'winter'

# Display the determined season based on the provided month and day
print("Season is", season)
```

Sample Output:

```
Input the month (e.g. January, February etc.): April
Input the day: 15
Season is spring
```

38.

Python Code:

```
# Request input from the user for the birth year and convert it to an integer, assigning it to the variable 'year'
year = int(input("Input your birth year: "))

# Determine the Chinese zodiac sign based on the provided birth year

# Check if the year corresponds to the Chinese zodiac sign 'Dragon'
if (year - 2000) % 12 == 0:
    sign = 'Dragon'
# Check if the year corresponds to the Chinese zodiac sign 'Snake'
elif (year - 2000) % 12 == 1:
    sign = 'Snake'
# Check if the year corresponds to the Chinese zodiac sign 'Horse'
elif (year - 2000) % 12 == 2:
    sign = 'Horse'
# Check if the year corresponds to the Chinese zodiac sign 'Sheep'
elif (year - 2000) % 12 == 3:
    sign = 'Sheep'
# Check if the year corresponds to the Chinese zodiac sign 'Monkey'
elif (year - 2000) % 12 == 4:
    sign = 'Monkey'
# Check if the year corresponds to the Chinese zodiac sign 'Rooster'
elif (year - 2000) % 12 == 5:
    sign = 'Rooster'
# Check if the year corresponds to the Chinese zodiac sign 'Dog'
elif (year - 2000) % 12 == 6:
    sign = 'Dog'
# Check if the year corresponds to the Chinese zodiac sign 'Pig'
elif (year - 2000) % 12 == 7:
    sign = 'Pig'
# Check if the year corresponds to the Chinese zodiac sign 'Rat'
elif (year - 2000) % 12 == 8:
    sign = 'Rat'
# Check if the year corresponds to the Chinese zodiac sign 'Ox'
elif (year - 2000) % 12 == 9:
    sign = 'Ox'
# Check if the year corresponds to the Chinese zodiac sign 'Tiger'
elif (year - 2000) % 12 == 10:
    sign = 'Tiger'
# If the year doesn't correspond to any specific sign, assign it to 'Hare'
else:
    sign = 'Hare'

# Display the determined Chinese zodiac sign based on the provided birth year
print("Your Zodiac sign :", sign)
```

Sample Output:

```
Input your birth year: 1784
Your Zodiac sign : Dragon
```

39.

```python
# Request input from the user for the birth year and convert it to an integer, assigning it to the variable 'year'
year = int(input("Input your birth year: "))

# Determine the Chinese zodiac sign based on the provided birth year

# Check if the year corresponds to the Chinese zodiac sign 'Dragon'
if (year - 2000) % 12 == 0:
    sign = 'Dragon'
# Check if the year corresponds to the Chinese zodiac sign 'Snake'
elif (year - 2000) % 12 == 1:
    sign = 'Snake'
# Check if the year corresponds to the Chinese zodiac sign 'Horse'
elif (year - 2000) % 12 == 2:
    sign = 'Horse'
# Check if the year corresponds to the Chinese zodiac sign 'Sheep'
elif (year - 2000) % 12 == 3:
    sign = 'Sheep'
# Check if the year corresponds to the Chinese zodiac sign 'Monkey'
elif (year - 2000) % 12 == 4:
    sign = 'Monkey'
# Check if the year corresponds to the Chinese zodiac sign 'Rooster'
elif (year - 2000) % 12 == 5:
    sign = 'Rooster'
# Check if the year corresponds to the Chinese zodiac sign 'Dog'
elif (year - 2000) % 12 == 6:
    sign = 'Dog'
# Check if the year corresponds to the Chinese zodiac sign 'Pig'
elif (year - 2000) % 12 == 7:
    sign = 'Pig'
# Check if the year corresponds to the Chinese zodiac sign 'Rat'
elif (year - 2000) % 12 == 8:
    sign = 'Rat'
# Check if the year corresponds to the Chinese zodiac sign 'Ox'
elif (year - 2000) % 12 == 9:
    sign = 'Ox'
# Check if the year corresponds to the Chinese zodiac sign 'Tiger'
elif (year - 2000) % 12 == 10:
    sign = 'Tiger'
# If the year doesn't correspond to any specific sign, assign it to 'Hare'
else:
    sign = 'Hare'

# Display the determined Chinese zodiac sign based on the provided birth year
print("Your Zodiac sign :", sign)
```

Sample Output:

```
Input your birth year: 1784
Your Zodiac sign : Dragon
```

Python Code:

```python
# Prompt the user to input the first number and convert it to a floating-point number, assigning it to the variable 'a'
a = float(input("Input first number: "))

# Prompt the user to input the second number and convert it to a floating-point number, assigning it to the variable 'b'
b = float(input("Input second number: "))

# Prompt the user to input the third number and convert it to a floating-point number, assigning it to the variable 'c'
c = float(input("Input third number: "))

# Determine the median among the three numbers

# Check if 'a' is greater than 'b'
if a > b:
    # Check if 'a' is less than 'c'
    if a < c:
        median = a
    # Check if 'b' is greater than 'c'
    elif b > c:
        median = b
    else:
        median = c
# If 'a' is not greater than 'b'
else:
    # Check if 'a' is greater than 'c'
    if a > c:
        median = a
    # Check if 'b' is less than 'c'
    elif b < c:
        median = b
    else:
        median = c

# Display the calculated median among the three input numbers
print("The median is", median)
```

Sample Output:

```
Input first number: 25
Input second number: 55
Input third number: 65
The median is 55.0
```

40.

41.

Python Code:

```
# Prompt the user to input a year and convert it to an integer, assigning it to the variable 'year'
year = int(input("Input a year: "))

# Determine if the input year is a leap year

# Check if the year is divisible by 400
if (year % 400 == 0):
    leap_year = True
# Check if the year is divisible by 100
elif (year % 100 == 0):
    leap_year = False
# Check if the year is divisible by 4
elif (year % 4 == 0):
    leap_year = True
else:
    leap_year = False

# Prompt the user to input a month in the range of 1-12 and convert it to an integer, assigning it to the variable 'month'
month = int(input("Input a month [1-12]: "))

# Determine the number of days in the specified month

# Check for months with 31 days
if month in (1, 3, 5, 7, 8, 10, 12):
    month_length = 31
# Check for February
elif month == 2:
    # Check for leap year and assign the appropriate number of days
    if leap_year:
        month_length = 29
    else:
        month_length = 28
# For months with 30 days
else:
    month_length = 30

# Prompt the user to input a day in the range of 1-31 and convert it to an integer, assigning it to the variable 'day'
day = int(input("Input a day [1-31]: "))

# Calculate the next date based on the provided day, month, and year

# Check if the day is less than the total number of days in the month
if day < month_length:
    day += 1
else:
    day = 1
    # Check if the current month is December
    if month == 12:
        month = 1
        year += 1
    else:
        month += 1

# Display the next date in the format [yyyy-mm-dd] based on the updated day, month, and year
print("The next date is [yyyy-mm-dd] %d-%d-%d." % (year, month, day))
```

Output:

```
Input a year:  2012
Input a month [1-12]:  3
Input a day [1-31]:  5
The next date is [yyyy-mm-dd] 2012-3-6.
```

42.

```
# Display a prompt asking the user to input integers to calculate their sum and average,
# specifying that entering 0 will exit the program
print("Input some integers to calculate their sum and average. Input 0 to exit.")

# Initialize variables for count, sum, and the first number to 1
count = 0
sum = 0.0
number = 1

# Iterate until the user inputs 0
while number != 0:
    # Prompt the user to input a number and convert it to an integer
    number = int(input(""))

    # Add the entered number to the sum
    sum = sum + number

    # Increment the count of entered numbers
    count += 1

# Check if no numbers were entered
if count == 0:
    print("Input some numbers")
else:
    # Calculate and display the average and sum of the entered numbers
    print("Average and Sum of the above numbers are: ", sum / (count-1), sum)
```

Sample Output:

```
Input some integers to calculate their sum and average. Input 0 to exit
.
15
16
12
0
Average and Sum of the above numbers are:  14.333333333333334 43.0
```

Python Code:

```
# Prompt the user to input a number and convert it to an integer, assigning it to the variable 'n'
n = int(input("Input a number: "))

# Use a for loop to iterate 10 times, starting from 1 up to 10 (excluding 11)
for i in range(1, 11):
    # Print the multiplication table for the entered number 'n' by multiplying it with each iteration 'i'
    print(n, 'x', i, '=', n * i)
```

Sample Output:

43.

```
Input a number: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Python Code:

```
# Iterate through a range from 0 to 9 (excluding 10)
for i in range(10):
    # Print the string representation of 'i' multiplied by 'i'
    print(str(i) * i)
```

Sample Output:

44.

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

45.

Copy code

```
# list of numbers
list_of_numbers = [2,4,6,9,5]

# for loop to iterate through the list and check each elements of the list
for i in list_of_numbers:
    # check if no. is odd
    if i % 2 != 0:
        # if condition is True print "odd"
        print (i, "is an odd number")
    # check if no. is even
    if i % 2 == 0:
        # if condition is false print "even"
        print (i, "is an even number")
```

46.

```
# list of numbers
list_of_numbers = [2,4,6,9,5]

# for loop to iterate through the list and check each element of the list
for i in list_of_numbers:

    # check if no. is odd
    if i%2 != 0:

        # if condition is True print "odd"
        print(i, "is an odd number")

    # if the no. is not odd then the no. is even therfore print "even"
    else:

        # if condition is True print "even"
        print(i, "is an even number")
```

Output:

```
2 is an even number
4 is an even number
6 is an even number
9 is an odd number
5 is an odd number
```

47.

```
# take a sample string
string ="ShikshaOnline"

# check if value in the string variable
# matches "Shik"
if string == "Shik":
   print ("The first condition is true")

# check if value in the string variable
# matches "Shikzha"
elif string == "Shiksha":
   print("The second condition is true")

# check if value in the string variable
# matches "Online"
elif string == "Online":
   print("The third condition is true")

# check if value in the string variable
# matches "ShikzhaOnline"
elif string=="ShikshaOnline":
   print("The fourth condition is true")

# if none of the above
# conditions evaluate to true
# execute the code inside the else block
else:
   print ("All the above conditions are false")
```

Output:

The fourth condition is true

```
# list of numbers
list_of_numbers = [4,5,9,17,21]

# for loop to iterate through  the list and check each element of the list
for i in list_of_numbers:

  # condition1: check if no. is odd
  # if yes execute the code block inside the first if statement
  if i%2!=0:

    # condition2: check if no. is  also divisible by 3.
    if i%3==0:

      # if condition2 is true
      # execute the below code block
      print (i,"is an odd number & divisible by 3")

    # if codition2 if False
    # execute the below code block

    else:
      print (i, "is an odd number but not divisible by 3")

  # if condition1 is False
  # execute the below code block
  else:
    print(i,"is an even number")
```

Output:

```
4 is an even number
5 is an odd number but not divisible by 3
9 is an odd number & divisible by 3
17 is an odd number but not divisible by :
21 is an odd number & divisible by 3
```

48.

Section - F

1. Write a Python Program to Check weather a Given Number is odd or Even.

```
number = int(input("Enter a number: "))
if num % 2 == 0:
print(num, “is even”)
else:
print(num, “is odd”)
```

2. Write a Python Program to Check weather a candidate Eligible for Vote or not.

```
age = int(input("Enter your age: "))
if age >= 18:
```

```
print(“You are eligible to vote.”)
else:
print(“You are not eligible to vote.”)
```

3. Write a Python program to check weather Given Number is Divisible by 7 or Not.

```
num = int(input("Enter a number: "))
if num % 7 == 0:
print(num, “is divisible by 7.”)
else:
print(num, “is not divisible by 7.”)
```

4. Write a program to display “Hello” if a number entered by user is a multiple of five , otherwise print “Bye”.

```
num = int(input("Enter a number: "))
if num % 5 == 0:
print(“Hello”)
else:
print(“Bye”)
```

5. Write a program to accept percentage from the user and display the grade according to the following **
**** criteria:**

```
** Marks Grade
marks > 80: A+ grade
marks > 70 : A grade
marks > 60: B grade
** marks > 50: C grade**
else Failed.

percentage = float(input("Enter your percentage: "))
if percentage > 80:
print(“your grade is A+”)
elif percentage > 70:
print(“your grade is A”)
elif percentage > 60:
print(“your grade is B”)
elif percentage > 50:
print(“your grade is C”)
else:
print(“your Failed”)
```

6. Write a program that interprets the Body Mass Index (BMI) based on a user’s weight and height.
It should tell them the interpretation of their BMI based on the BMI value.

• Under 18.5 they are underweight
• Over 18.5 but below 25 they have a normal weight
• Over 25 but below 30 they are slightly overweight
• Over 30 but below 35 they are obese
• Above 35 they are clinically obese.
The BMI is calculated by dividing a person's weight (in kg) by the square of their height (in m) .
Take height and weight from the user.

```
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))
bmi = weight/ (height** 2)
if bmi < 18.5:
print("You are underweight.")
elif 18.5 <= bmi < 25:
print("You have a normal weight.")
elif 25 <= bmi < 30:
print("You are slightly overweight.")
elif 30 <= bmi < 35:
print("You are obese.")
else:
print("You are clinically obese.")
```