PYTHON BASICS QUESTIONS 1.What is Python, and why is it popular? Ans-Python is a high-level,interpreted programming language known for its simplicity and readability.It was created by Guido van Rossum and first released in 1991.Python's design philosophy emphasizes code readibility,allowing developers to express concepts in fewer lines of code compared to other languages. Python is popular for several reasons: 1.**EASE OF LEARNING**: its straightforward syntax makes it an excellent choice for beginners. 2.**VERSATILITY**: Python can be used for various applications,including web development, data analysis,artificial intelligence,scientific computing, and automation. 3.**LARGE COMMUNITY AND LIBRARIES**: Python has a vast ecosystem of libraries and framework(like Numpy,Pandas,and Django) that extends its capabilities and simplify complex tasks. 4.**CROSS-PLATFORM COMPATIBILITY**: Python runs on various operating systems, making it highly portable. 5.**STRONG SUPPORT FOR INTEGRATION**: Python can easily integrate with other languages and technologies, enhancing its usability in diverse environments.2.What is an interpreter in Python? Ans-An interpreter in Python is a program that executes Python code line by line.Unlike a compiler, which translates the entire source code into machine code before execution, an interpreter processes the code in real-time,translating and executing it simultaneously.This allows for immediate feedback and easier debugging,making it particularly useful for development and testing.The Python interpreter reads the code,converts it into an intermediate form(bytecode),and then executes it on the Python Virtual Machine(PVM).This approach contributes to Python's ease of use and flexibility.3.What are pre-defined keywords in Python? Ans-Predefined keywords in Python are reserved words that have special meanings and cannot be used as identifiers(such as variable names,function names,or class names). These keywords define the syntax and structure of the Python language. Some python keywords examples are: -'False' -'none' -'break' -'continue' -'class' -'True' etc. These keywords are fundamental to writing Python code and help define control flow, data types, and other programming constructs.4.Can keywords be used as variables names? Ans-No, keywords cannot be used as variable names in Python.Keywords are reserved words that have special meanings in the language, and using them as variable names would lead to syntax errors.5.What is mutability in Python? Ans-Mutability in Python refers to the ability of an object to be changed or modified after it has been created.Objects that can be altered are called Mutable,while those that cannot be changed are called immutable.Mutable objects include lists,dictionaries and sets.6.Why are lists mutable,but tuples are immutable? Ans-Lists are mutable because they are designed to allow modification,such as adding,removing,or changing elements,which makes them flexible for dynamic data handling.This mutability is useful in scenarios where data needs to change frequently. Tuples on the other hand are immutable to provide benifits like improved performance, memory efficiency, and data integrity.Their fixed nature ensures that the data remains constant, making tuples suitable for representing fixed collections of items.This immutability also allows tuples to be used as keys in dictionaries and elements in set which is not possible with mutable types like lists.7.What are the differences between"=="and "is" operators in python? Ans-In Python the '==' and 'is' operators are used for comparison purpose but they serve different purposes: 1.**('=='):EQUALITY OPERATOR)**: - Compares the values of two objects to determine if they are equal. - It checks for value equality, meaning it evaluates to 'True' if the objects have the same content regardless of whether they are same object in memory. -Example: a = [1,2,3] b = [1,2,3] #output: True (same values). 2.**('is'):IDENTITY OPERATOR)**: - Compares the identities of two objects to determine if they are the same object in memory. - It checks for reference equality, meaning it evaluates to 'True' only if both operands refer to the exact same object. - Example: a = [1,2,3] b = a print (a is b) #output: True (same object). 8.What are logical operators in Python? Ans-Logical operators in Python are used to combine conditional statements and evaluate their truth values.The three main logical operators are: 1.**and**: -Returns 'True' if both operands are true;otherwise,it returns 'False'. -example: True and False #output: False. 2.**or**: -Returns 'True' if at least one of the operands is true;returns 'False' only both are false. -Example: True and False #output: True. 3.**not**: -Returns 'True' if the operand is false; returns'False'if the operand is true.It negates the truth value of the operand. -Example: not true #output: False. These operators are commonly used in control flow statements, such as 'if' statements, to create complex logical conditions.9.What is type casting in Python? Ans-Type casting in Python is the process of converting a variable from one data type to another.This allows for operations between different types and ensures that data is in the desired format for processing. Python provides several built-in functions for type casting,including: -'int()' -'float()' -'str()' -'list' -'tuple'. Type casting is useful for ensuring compatibility between different data types in operations and functions.10.What is the difference between implicit and explicit type casting? Ans- The difference between implicit and explicit type casting in Python is as follows: 1.**IMPLICIT TYPE CASTING**(Automatic Type Conversion): -This occurs automatically when Python converts one data type to another without any user intervention. -It usually happens when a smaller data type is converted to a large data type to prevent data loss. -Example: a = 5 #Integer b = 2.0 #Float c = a + b # a is implicitly converted to float. 2.**EXPLICIT TYPE CASTING**(Type Conversion): -This requires the programmer to manually convert one data type to another using built-in functions like 'int()','float()',or'str()'. -It is used when the programmer wants to ensure a specific type conversion. -Example: a = "10" #String b = int(a) #a is explicitly converted to integer. In summary, implicit casting is automatic and done by Python, while explicit casting is manual and done by the programmer.11.What is the purpose of conditional statements in Python? Ans-The purpose of conditional statements in Python is to enable the execution of specific blocks of code based on certain conditions.They allow the program to make decisions and control the flow of execution depending pn whether a condition evaluates to 'True' or 'False'. The primary conditional statements in Python include: 1. 'if' statement 2.'elif' statement 3. 'else' statement.12.How does elif statement work? Ans-The 'elif' statement in Python, short for "else if", is used in conditional statements to check multiple conditions sequentially. Here's how it works: 1.The program first evaluates the condition in the 'if' statement. 2.If the 'if' condition is 'True', the corresponding block of code is executed,and the rest of the 'elif' and 'else' statements are skipped. 3.if the 'if' condition is 'False',the program checks the condition in the first 'elif' statement. 4.if the 'elif' condition is 'True', its block of code is executed,and the remaining 'elif' and 'else' statements are skipped. 5.This process continues for any additional 'elif' statements. 6.If none of the conditions are 'True', the code block under the'else' statement(if present) is executed.13. What is the difference between for and while loops? Ans-The main differences between 'for' and 'while' loops in Python are: 1.PURPOSE: -**'for'LOOP**:- used for iterating over a sequence(like a list,tuple,string,or range) or any iterable object.It is typically used when the number of iterations is known or deterministic. -**'while'LOOP**:-used for repeated execution as long as a specified condition is'True'.It is typically used when the number of iterations is not known in advance and depends on a condition. 2.SYNTAX: -'for' loop: for item in iterable: #code block to execute -'while' loop: while condition: #code block to execute 3.CONTROL: -'for' loop:Automatically iterates through each item in the iterable,making it simpler for fixed iterations. -'while' loop:Requires manual control of the loop variable and condition , which can lead to infinite loops if not handled correctly. 4.ITERATION: -'for' loop: Iterates over elements directly, making it concise and easy to read. -'while' loop: Requires explicit management of the loop variable,which can make it more complex.14.Describe a scenario where a while loop is more suitable than a for loop? Ans- Example Scenario: User input Validation Suppose you want to prompt a user to enter a valid password until they provide one that meets specific criteria(eg, at least 8 characters long, containes a number,etc).In this case,you cannot determine beforehand how many attempts the user will need to make,making a 'while' loop the ideal choice. PRACTICAL QUES

In [2]:
```
#1.write a python program to print "Hello,World!"?
#ANS-
print("Hello,World!")
```

```
Hello,World!
```

In [11]:
```
#2.Write a Python program that displays your name and age?
#ANS-
name = "Faizan"
age = 20
print("Faizan")
print("20")



```

```
Faizan
20
```

In [7]:
```
#3.Write a code to print pre-defined keywords in python using the keyword library?
#ANS-
import keyword
keywords = keyword.kwlist
print("predefined keywords in python:")
for kw in keywords:
    print(kw)
```

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

In [ ]:
```
#4.Write a program that checks if a given word is a python keyword?
#ANS-
import keyword
def is_keyword(word):
    return keyword.iskeyword(word)
word_to_check =input("Enter a word to check if it is a python keywords:")
if is_keyword(word_to_check):
    print(f'"{word_to_check}" is a python keyword.')
else:
    print(f'"{word_to_check}" is not a python keyword.')
```

In [1]:
```
#5.Create a list and tuple in python,and demonstrate how attempting to change an element works differently for each?
#ANS- Here's a demonstration of how attempting to change an element works differently for a list and a tuple in python:
#creating a list
my_list = [1,2,3,4,5]
print("my_list")
# Attempting to change an element in the list
my_list[2] = 10
print("my_list")
#Creating a tuple
my_tuple = (1,2,3,4,5)
print("my_tuple")
#Attempting to change an element in the tuple
try:
    my_tuple[2] = 10 #this will raise an error
except TypeError as e:
    print("Error:")
```

```
my_list
my_list
my_tuple
Error:
```

In [3]:
```
#6.Write a function to demonstrate the behaviour of mutable and immutable arguments?
#ANS-
def mutable_immutable_demo():
    """demonstrates the behavior of mutable and immutable objects in python."""
    #Immutable object:integer
x = 5
y = x
print(f"Before modification: x = {x}, y = {y}")

x = 10
print(f"After modifying x: x = {x}, y = {y}")
# y remains unchanged as integers are immutable

    # Mutable objects: List
a = [1,2,3]
b = a
print(f"\nBefore modification: a = {a}, b = {b}")

a.append(4)
print(f"After modifying a: a= {a},b = {b}")
# b is also modified as lists are mutable and b refers to the same list object as a.
if __name__== "__main__":
    mutable_immutable_demo()
```

```
Before modification: x = 5, y = 5
After modifying x: x = 10, y = 5

Before modification: a = [1, 2, 3], b = [1, 2, 3]
After modifying a: a= [1, 2, 3, 4],b = [1, 2, 3, 4]
```

In [ ]:
```
#7. Same as 6
#ANS- The answer is same as of 6 bcs both the ques are same.
```

In [5]:
```
#8.Write a program to demonstrate the use of logical operators?
#ANS-
def logical_operators_demo():
    """Demonstrates the use of logical operators in python."""
    #Define variables
    x = True
    y = False
# AND operator (both condition must be true)
print(f"x AND y: {x and y}") #output:False
#OR operator (at least one condition must be true)
print(f"x OR y: {x or y}") #output:True
#NOT operator (reverse the boolean value)
print(f"NOT x: {not x}") #output:False
```

```
x AND y: 5
x OR y: 10
NOT x: False
```

In [37]:
```
#9.Write a python program to convert user input from string to integer,float,and boolean types.
#ANS-
user_input = input("20")
#convert to integer
integer_value = int(user_input)
print(f"converted to integer:{integer_value}")
print("could not convert to integer.")

#convert to float
float_value =float(user_input)
print(f"converted to float:{float_value}")
print("could not convert to float.")

#convert to boolean
# in python,empty strings,0,and None are considered False,everything else is True.
boolean_value = bool(user_input)
print(f"converted to boolean:{boolean_value}")
```

```
converted to integer:2
could not convert to integer.
converted to float:2.0
could not convert to float.
converted to booleanTrue
```

In [43]:
```
#10.write code to demonstrate type casting with list elements.
#Ans-
original_list = ["1","2.5","True","4"]

integer_list = [int(x) if x.isdigit() else 0 for x in original_list]
float_list = [float(x) if x.replace('.','',1).isdigit() else 0.0 for x in original_list]
boolean_list = [x.lower() == "true" for x in original_list]
print("Original List: ", original_list)
print("Integer List: ", integer_list)
print("Float List: ", float_list)
print("Boolean List: ", boolean_list)
```

```
Original List:  ['1', '2.5', 'True', '4']
Integer List:  [1, 0, 0, 4]
Float List:  [1.0, 2.5, 0.0, 4.0]
Boolean List:  [False, False, True, False]
```

In [47]:
```
#11.Write a program that checks if a number is positive,negative,or zero.
#ANS-
number = float(input("Enter a number"))
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 positive.
```

In [51]:
```
#12.write a for loop to print numbers from 1 to 10.
#ANS-
for i in range(0,11):
    print(i,end = " ")
```

```
0 1 2 3 4 5 6 7 8 9 10
```

In [53]:
```
#13. write a python program to find the sum of all even numbers between 1 and 50.
#Ans-
sum_even_numbers = 0
for number in range(1,51):
    if number % 2 == 0:
        sum_even_numbers += number
print("The sum of all even numbers between 1 and 50 is:", sum_even_numbers)
```

```
The sum of all even numbers between 1 and 50 is: 650
```

In [55]:
```
#14.Write a program to reverse a string using a while loop.
#ANS-
string = input("Enter a string: ")
reversed_string = ""
index = len(string) - 1
while index >=0:
    reversed_string += string[index]
    index -= 1
print("Reversed string:", reversed_string)
```

```
Reversed string: maR
```

In [57]:
```
#15.Write a python program to calculate the factorial of a number provided by the user using a while loop.
#ANS-
number = int(input("Enter a number: "))
factorial = 1
count = 1
while count <= number:
    factorial *= count
    count += 1
print("The factorial of", number, "is:", factorial)
```

```
The factorial of 21 is: 51090942171709440000
```

In [ ]:
```
# COMPLETED
```

In [ ]:

In [ ]: