PYTHON PROGRAMMING-IAT-1 Solution

Course Code:1BPLC105B/205B

1a.Explain the role of the input() function, print() function with an example.
Answer:
input() Function
The input() function is used to take input from the user (from the keyboard).
Whatever the user types is always read as a string.
Syntax:
variable = input("message for user")
Example:
```
name = input("Enter your name: ")
print("Hello", name)
```
Output:
Hello Manasa
2. print() Function
The print() function is used to display output (text, numbers, or variables) on the screen.
Syntax:
print(value1, value2, ...)
Example:
```
x = 10
y = 20
print("The sum is", x + y)
```
Output:
The sum is 30
Example:
```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("The sum is", num1 + num2)
```

Output :
Enter first number: 5
Enter second number: 10
The sum is 15

1b.Write a Python program to check whether a given number is prime.

Program:

```
num = int(input("Enter a number: "))
# 1 and numbers less than 1 are not prime
if num <= 1:
print(num, "is not a prime number")
else:
# check for factors
for i in range(2, num):
if num % i == 0:
print(num, "is not a prime number")
break
else:
print(num, "is a prime number")
```

Output:
Enter a number: 7

7 is a prime number

2.a Explain order of operations in Python with an example.

Answer:

1. Parentheses
2. Exponentiation
3. Multiplication and both Division
4. Operators with the same precedence are evaluated from left-to-right

```
>>> 2 ** 3 ** 2     # The right-most ** operator gets done first!
512
>>> (2 ** 3) ** 2   # Use parentheses to force the order you want!
64
```

Operations on strings

You cannot perform mathematical operations on strings

For strings, the + operator represents concatenation, not addition

The * operator also works on strings; it performs repetition

```
>>> message - 1          # Error
>>> "Hello" / 123        # Error
>>> message * "Hello"    # Error
>>> "15" + 2             # Error
```

```
fruit = "banana"
baked_good = " nut bread"
print(fruit + baked_good)
```

The output of this program is banana nut bread

2.b Discuss type conversion functions in Python.

Answer:

int, str, float

The int function can take a floating point number or a string, and turn it into an int.For floating point numbers, it discards the decimal portion of the number - a process we call truncation towards zero on the number line.

The type converter float can turn an integer, a float,or a syntactically legal string into a float The type converter str turns its argument into a

String

```
>>> int(3.14)
3
>>> int(3.9999)              # This doesn't round to the closest int!
3
>>> int(3.0)
3
>>> int(-3.999)              # Note that the result is closer to zero
-3
>>> int(minutes / 60)
10
>>> int("2345")              # Parse a string to produce an int
2345
>>> int(17)                  # It even works if arg is already an int
17
>>> int("23 bottles")
```

```
>>> float(17)
17.0
>>> float("123.45")
123.45
```

```
>>> str(17)
'17'
>>> str(123.45)
'123.45'
```

3.a Differentiate between for loop and while loop with examples.

Answer:

1. For Loop

Definition:

A for loop is used when you know how many times you want to repeat a block of code — for example, looping through a list or a range of numbers.

Syntax:

for variable in sequence:

block of code

Example:

for i in range(5):

print("Hello", i)

range(5) → generates numbers from 0 to 4

The loop runs 5 times

Output:

Hello 0

Hello 1

Hello 2

Hello 3

Hello 4

While Loop

Definition:

A while loop is used when you don't know exactly how many times to repeat, but want to continue looping as long as a condition is true.

Syntax:

while condition:

block of code

Example:

count = 0

while count < 5:

print("Hello", count)

count += 1
The loop continues as long as count < 5
count increases by 1 each time
Output:
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4

3.b Write a Python function to compute the sum of first N natural numbers.
Program:

```
def sum_of_natural_numbers(n):
total = 0
for i in range(1, n + 1):
total += i
return total

num = int(input("Enter a number: "))
print("Sum of first", num, "natural numbers is:",
sum_of_natural_numbers(num))
```

OUTPUT:
Enter a number: 5
Sum of first 5 natural numbers is: 15

4.a What is a tuple? How is it different from a list?
Answer:
A tuple in Python is an ordered collection of items, similar to a list, but it is immutable — meaning its elements cannot be changed, added, or removed once the tuple is created. Tuples are written within parentheses ( ) and can contain elements of different data types, such as numbers, strings, or even other tuples. For example, t1 = (10, 20, 30) is a tuple containing three integers. Since tuples are immutable, they are often used to store fixed collections of data, such as coordinates or database records, where the values should not be modified. You can

still access elements in a tuple using indexing and slicing, just like with lists.Difference between Tuple and List The main difference between a tuple and a list is mutability. A list is mutable, meaning its elements can be changed, added, or removed after creation, whereas a tuple is immutable, so its contents cannot be altered. Lists are written with square brackets [ ], while tuples are written with parentheses ( ). Because tuples are immutable, they are generally faster and more memory efficient than lists. Lists are used when you need to modify or update data frequently, while tuples are preferred when the data should remain constant throughout the program.

Example:

```
# Creating a list and a tuple
my_list = [10, 20, 30]
my_tuple = (10, 20, 30)
# Accessing elements (same for both)
print("List element at index 1:", my_list[1])
print("Tuple element at index 1:", my_tuple[1])
# Modifying elements
my_list[0] = 100 # ✓Allowed (lists are mutable)
print("Modified list:", my_list)
my_tuple[0] = 100 # ✗Error (tuples are immutable)
#TypeError: 'tuple' object does not support item assignment
```

4.b Write a program to split a string into a list of words.

Answer:

Converting a String to a List:

Use the split() method to break a string into a list of words.

Default behavior: splits at any whitespace.

```
>>song = "The rain in Spain..."
>>words = song.split()
>>print(words)
```

Output:

```
['The', 'rain', 'in', 'Spain...']
```

You can specify a delimiter string to split at specific characters.

Example:

song.split("ai")
Output:
['The r', 'n in Sp', 'n...']

5.a Discuss list methods append(), insert(), and remove() with examples.
Answer:
1. append() method
The append() method is used to add an element to the end of a list.It increases the length of the list by one each time it is called.
Syntax:
list.append(element)

Example:
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
Output:
['apple', 'banana', 'cherry', 'orange']

The element "orange" is added to the end of the list.
2. insert() method
The insert() method is used to add an element at a specific position (index) in the list.It takes two arguments — the index position and the element to be inserted.

Syntax:
list.insert(index, element)
Example:
numbers = [10, 20, 40, 50]
numbers.insert(2, 30)
print(numbers)
Output:
[10, 20, 30, 40, 50]
The value 30 is inserted at index 2 (between 20 and 40).
3. remove() method

The remove() method is used to delete the first occurrence of a specific element from a list.
Syntax:
list.remove(element)

Example:
colors = ["red", "blue", "green", "blue"]
colors.remove("blue")
print(colors)
Output:
['red', 'green', 'blue']

Only the first "blue" in the list is removed — not all occurrences.

5.b Write a Python program to remove vowels from a given string.
Program:

```
text = input("Enter a string: ")
# Define vowels
vowels = "aeiouAEIOU"
# Create an empty string to store result
result = ""
# Loop through each character
for char in text:
if char not in vowels:
result += char
# Display result
print("String after removing vowels:", result)
```

Output:
Enter a string: Beautiful Day
String after removing vowels: Btfl Dy

6.a Explain the in and not in operators with examples.

Answer:
'in' and 'not in' are Boolean operators.
They test if an element exists in a list or sequence.
Return True if found, False if not.

Example:
horsemen = ['war', 'famine', 'pestilence', 'death']
'pestilence' in horsemen → True
'debauchery' in horsemen → False
'debauchery' not in horsemen → True

6.b Compare strings and lists with examples.
Answer:
Definition A string is a sequence of characters enclosed in quotes.A list is a collection of items (numbers, strings,etc.) enclosed in square brackets.

Syntax: "Hello" or 'Hello' [10, 20, 30]
Type Immutable (cannot be changed after creation) Mutable (can be changed after creation) Elements Only characters Can contain any data type (numbers, strings, lists,etc.)Indexing Supported Supported Slicing Supported Supported Methods Have string-specific methods (e.g.,.upper(), .lower()) Have list-specific methods (e.g., .append(),.remove())

Examples
String Example
s = "hello"
print(s[1]) # Access character at index 1 → 'e'
print(s.upper()) # Convert to uppercase → 'HELLO'
List Example
lst = [10, 20, 30]
print(lst[1]) # Access element at index 1 → 20
lst.append(40) # Add new element
print(lst) # Output: [10, 20, 30, 40]
Mutability Difference

Strings are Immutable
s = "hello"
# s[0] = 'H' ❌(Not allowed)
s = "Hello" # You must create a new string
Lists are Mutable
lst = [1, 2, 3]
lst[0] = 10 # ✅Allowed
print(lst) # Output: [10, 2, 3]
Similarities
Both support:
Indexing (access by position)
Slicing (extract a part)
Looping (for loop)
Example:
for ch in "abc":
print(ch)
for num in [1, 2, 3]:
print(num)

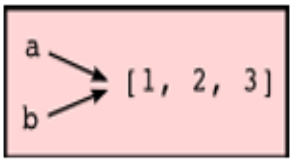
7.a Write a Python program to remove odd numbers from a list.
Program:
numbers = [10, 15, 20, 25, 30, 35, 40]
# Create a new list that contains only even numbers
even_numbers = []
for num in numbers:
if num % 2 == 0: # Check if number is even
even_numbers.append(num)
# Display the result
print("Original list:", numbers)
print("List after removing odd numbers:", even_numbers)
Output:
Original list: [10, 15, 20, 25, 30, 35, 40]
List after removing odd numbers: [10, 20, 30, 40]

7.b Explain aliasing in lists with an example.
Answer:

Since variables refer to objects, if we assign one variable to another, both variables refer to the same object:

```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b
True
```

In this case, the state snapshot looks like this:



Because the same list has two different names, `a` and `b`, we say that it is **aliased**. Changes made with one alias affect the other:

```
>>> b[0] = 5
>>> a
[5, 2, 3]
```