# **Practical No. 1:** Install Python IDE

- ## Practical related questions

**1. Write steps to install python on windows.**

➔ 1) Download Python Installer.
2) Run the Installer
3) Customize Installation
4) Installation Process
5) Verify Installation

**2. Print the version of Python.**

➔ Python 3.12.2

**3. List key features of python.**

➔ 1) Simple and Easy to Learn
2) Interpreted Language
3) Dynamically Typed
4) Object-Oriented Programming
5) Extensive Standard Library
6) Platform Independent
7) Free and Open Source
8) High-Level Language
9) Portable

**4. What are the applications of python ?**

➔ 1) Web Development
2) Data Science and Analytics
3) Machine Learning and Artificial Intelligence

4) Software Development
5) Networking
6) Cybersecurity and Ethical Hacking
7) Cloud Computing
8) Internet of Things (IoT)

**5. State use of pip & pep.**

➔

1) **pip** – Used to install, upgrade, and manage Python packages from the Python Package Index (PyPI) and other repositories.

2) **PEP** – Python Enhancement Proposal; used to propose, discuss, and document new features, implementation details, and best practices for the Python language.

# **Practical No. 2 :** Write simple Python program to display message on screen

## Practical related questions

1. **List different modes of Programming in Python**

➔ 1) Interactive Mode
   2) Script mode

2. **State the steps involved in executing the program using Script Mode.**

➔ Step : 1 - Open the text editor.
Step : 2 - Write python code.
Step : 3 - Save the File
Step : 4 - Open Command Prompt (CMD)
Step : 5 - Navigate to the File Location
Step : 6 - Execute the Program
Step : 7 - View the Output

3. **Write a Python program to display "MSBTE" using Script Mode.**

➔

Steps to run in Script Mode:

1) Open **Notepad** or any text editor.
2) Type the code.
   - print("MSBTE")
3) Save the file as `msbte.py` .
4) Open **Command Prompt** and navigate to the file location using `cd`.
5) Run the program using:
   - python msbte.py
6) Output:
   - MSBTE

## **Practical No. 3 :** Write simple Python program using operators: Arithmetic Operators, Logical Operators, Bitwise Operators

- ## Practical related questions

### 1. Describe ternary operator in Python.

➔ The ternary operator in Python is a conditional expression that allows you to evaluate something in a single line instead of using a full `if-else` statement.

**Syntax:**
<value_if_true> if <condition> else <value_if_false>

**Example:**

```
a = 10
b = 20
max_value = a if a > b else b
print("Maximum value is:", max_value)
```

Here, If the condition `a > b` is True, `a` is assigned to `max_value`.
If False, `b` is assigned.

**Output:**

```
Maximum value is: 20
```

### 2. Describe about different Bitwise operators in Python with appropriate examples.

➔ Bitwise operators are used to perform bit-level operations on integers. They operate on bits and perform bit-by-bit operations.
Here, are the bitwise operators that are used in python:

1) **& – Bitwise AND**

The bitwise AND operator performs a logical AND operation on each bit of two numbers. It returns 1 only if both bits are 1, otherwise it returns 0.

**Example:**

```
a = 10
b = 4
print("a & b =", a & b)
```

**Output:**

```
a & b = 0
```

2) **| – Bitwise OR**

The bitwise OR operator compares each bit of two numbers and returns 1 if 1 bit is 1 and other bit is 0. If both the bits are same it returns 0.

**Example:**

```
a = 10
b = 4
print("a | b =", a | b)
```

**Output:**

```
a | b = 14
```

3) **^ – Bitwise XOR**

The bitwise XOR (exclusive OR) operator returns 1 if the bits are different in the two numbers; otherwise, it returns 0.

**Example:**

```
a = 10
b = 4
print("a ^ b =", a ^ b)
```

**Output:**

```
a ^ b = 14
```

4) **~ – Bitwise NOT**

The bitwise NOT operator inverts all the bits of the number. It changes every 1 to 0 and every 0 to 1. In Python, it returns the negative of the number plus one (i.e., `~a = -a - 1`).

**Example:**

```
a = 10
b = 4
print("~a =", ~a)
```

**Output:**

```
~a = -11
```

**5) << – Left Shift**

The left shift operator shifts the bits of a number to the left by the specified number of positions. It adds zeros from the right and multiplies the number by `2^n`.

**Example:**

```
a = 10
b = 4
print("a << 1 =", a << 1)
```

**Output:**

```
a << 1 = 20
```

**6) >> – Right Shift**

The right shift operator shifts the bits of a number to the right by the specified number of positions. It divides the number by `2^n`.

**Example:**

```
a = 10
b = 4
print("a >> 1 =", a >> 1)
```

**Output:**

```
a >> 1 = 5
```

**3. Describe about different Logical operators in Python with appropriate examples.**

➔

1) **Operator: and**

**Name:** Logical AND

Returns `True` if both the conditions are `True`, otherwise returns `False.`

**Example:**
```
a = 5
b = 10
if a > 0 and b > 0:
    print("Both numbers are positive")
else:
    print("One or both numbers are not positive")
```

**Output:**
```
Both numbers are positive
```

2) **Operator:** `or`
**Name:** Logical OR
Returns `True` if at least one condition is `True`, otherwise returns `False`.

**Example:**
```
a = 5
b = -2
if a > 0 or b > 0:
    print("At least one number is positive")
else:
    print("Both numbers are not positive")
```

**Output:**
```
At least one number is positive
```

3) **Operator:** `not`
**Name:** Logical NOT
Reverses the result of the condition. If the condition is `True`, it returns `False`; if it is `False`, it returns `True`.

**Example:**
```
a = 5
if not a < 0:
    print("Number is not negative")
else:
    print("Number is negative")
```

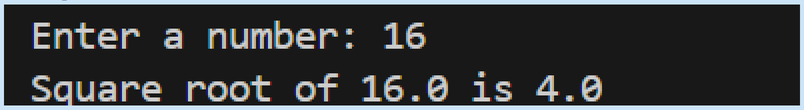
```
Output:
Number is not negative
```

**4. Write a program to find the square root of a number.**

➔

```python
num = float(input("Enter a number: "))
square_root = num ** 0.5
print("Square root of", num, "is", square_root)
```

**Output:**



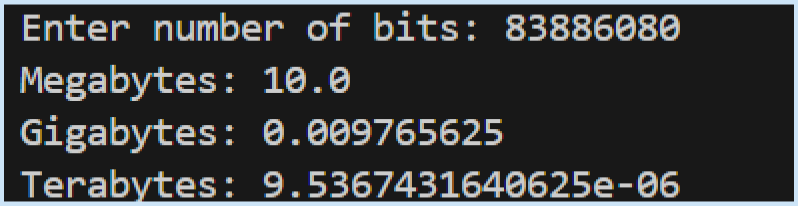
```
Enter a number: 16
Square root of 16.0 is 4.0
```

**5. Write a program to convert bits to Megabytes, Gigabytes and Terabytes.**

➔

```python
bits = int(input("Enter number of bits: "))
bytes_value = bits / 8
megabytes = bytes_value / (1024 * 1024)
gigabytes = bytes_value / (1024 * 1024 * 1024)
terabytes = bytes_value / (1024 * 1024 * 1024 * 1024)
print("Megabytes:", megabytes)
print("Gigabytes:", gigabytes)
print("Terabytes:", terabytes)
```

**Output:**



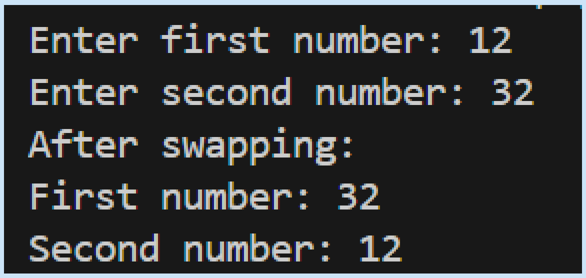
```
Enter number of bits: 83886080
Megabytes: 10.0
Gigabytes: 0.009765625
Terabytes: 9.5367431640625e-06
```

**6. Write a program to swap the value of two variables.**

➔

```python
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
a, b = b, a
print("After swapping:")
print("First number:", a)
print("Second number:", b)
```

**Output:**



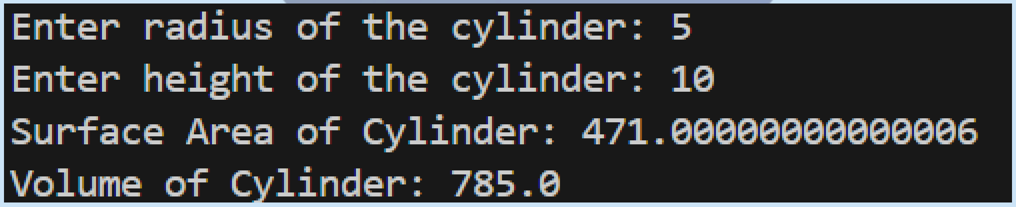
```
Enter first number: 12
Enter second number: 32
After swapping:
First number: 32
Second number: 12
```

**7. Write a program to calculate surface volume and area of a cylinder.**

➔

```
pi = 3.14
radius = float(input("Enter radius of the cylinder: "))
height = float(input("Enter height of the cylinder: "))
surface_area = 2 * pi * radius * (radius + height)
volume = pi * radius * radius * height
print("Surface Area of Cylinder:", surface_area)
print("Volume of Cylinder:", volume)
```

**Output:**



```
Enter radius of the cylinder: 5
Enter height of the cylinder: 10
Surface Area of Cylinder: 471.00000000000006
Volume of Cylinder: 785.0
```

**Practical No. 4 :** Write simple Python program to demonstrate use of conditional statements: if' statement, 'if … else' statement, Nested 'if' statement

## Practical related questions

1. **Differentiate between if-else and nested-if statement about different Logical operators in Python with appropriate examples.**

➔

- **If-else :-**
  Used to choose between two blocks based on a single condition. The if part executes the true statement block in the condition and the else part is executed for the false statement block of the condition.

  **Example-**
```
marks = 80
attendance = 85
if marks >= 75 and attendance >= 75:
    print("Eligible for Award")
else:
    print("Not Eligible")
```

  **Output-**
  Eligible for Award

- **Nested-if :-**
  Used when one condition depends on another condition. The nested-if statements are used to execute multiple conditional statements in one module or problem statement.

  **Example-**
```
marks = 35
attendance = 80
if marks >= 40:
    if attendance >= 75 or marks >= 50:
        print("Allowed to sit for exam")
    else:
        print("Not allowed")
else:
    print("Fail")
```

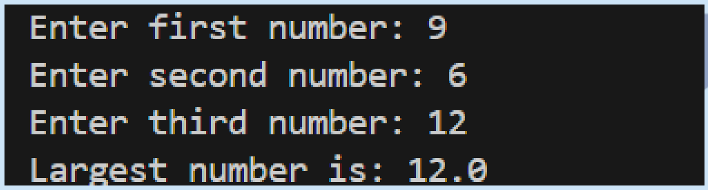
**Output-**
Fail

2. **Write a program to check the largest number among the three numbers.**

➔

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))
if num1 >= num2 and num1 >= num3:
    print("Largest number is:", num1)
elif num2 >= num1 and num2 >= num3:
    print("Largest number is:", num2)
else:
    print("Largest number is:", num3)
```

**Output -**



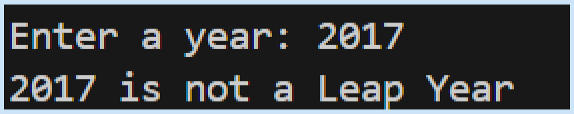
```
Enter first number: 9
Enter second number: 6
Enter third number: 12
Largest number is: 12.0
```

3. **Write a program to check if the input year is a leap year of not.**

➔

```
year = int(input("Enter a year: "))
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(year, "is a Leap Year")
else:
    print(year, "is not a Leap Year")
```

**Output-**



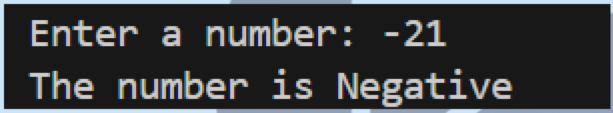
```
Enter a year: 2017
2017 is not a Leap Year
```

### 4. Write a program to check if a Number is Positive, Negative or Zero

➔

```
num = float(input("Enter a number: "))
if num > 0:
    print("The number is Positive")
elif num < 0:
    print("The number is Negative")
else:
    print("The number is Zero")
```

**Output-**



```
Enter a number: -21
The number is Negative
```

### 5. Write a program that takes the marks of 5 subjects and displays the grades.

➔

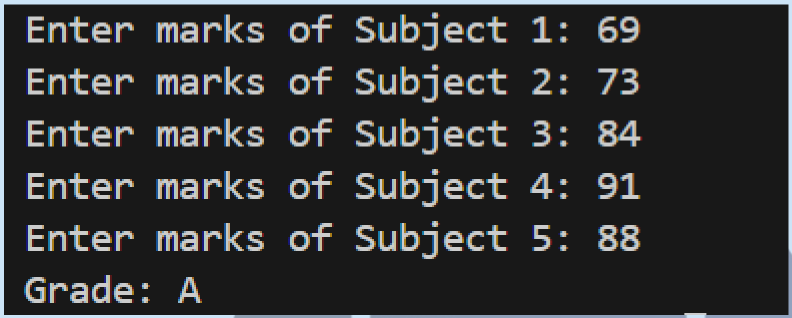
```
sub1 = float(input("Enter marks of Subject 1: "))
sub2 = float(input("Enter marks of Subject 2: "))
sub3 = float(input("Enter marks of Subject 3: "))
sub4 = float(input("Enter marks of Subject 4: "))
sub5 = float(input("Enter marks of Subject 5: "))

total = sub1 + sub2 + sub3 + sub4 + sub5
average = total / 5

if average >= 90:
    print("Grade: A+")
elif average >= 80:
    print("Grade: A")
elif average >= 70:
    print("Grade: B+")
```

```
elif average >= 60:
    print("Grade: B")
elif average >= 50:
    print("Grade: C")
elif average >= 40:
    print("Grade: D")
else:
    print("Grade: Fail")
```

**Output-**



```
Enter marks of Subject 1: 69
Enter marks of Subject 2: 73
Enter marks of Subject 3: 84
Enter marks of Subject 4: 91
Enter marks of Subject 5: 88
Grade: A
```

## 6. List operators used in if conditional statement.

➔

1) == (Equal to)
2) != (Not equal to)
3) > (Greater than)
4) < (Less than)
5) >= (Greater than or equal to)
6) <= (Less than or equal to)

## Practical No. 5 : Write Python program to demonstrate use of looping statements: ‘while’ loop, ‘for’ loop and Nested loop

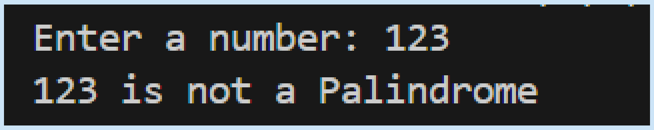
- **Practical related questions**

1. **Write a Python program that takes a number and checks whether it is a palindrome or not.**

➔

```
num = int(input("Enter a number: "))
reverse = 0
temp = num
while temp > 0:
    digit = temp % 10
    reverse = (reverse * 10) + digit
    temp = temp // 10
if num == reverse:
    print(num, "is a Palindrome")
else:
    print(num, "is not a Palindrome")
```

**Output:**



```
Enter a number: 123
123 is not a Palindrome
```

2. **Write a Python program to print all even numbers between 1 to 100 using while loop.**

➔

```
num = 1
while num <= 100:
    if num % 2 == 0:
        print(num)
    num = num + 1
```

**Output:**

```
2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76,
78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100
```

**3. Print the following patterns using loop:**

a)
```
1010101
 10101
  101
   1
```

➔

```python
rows = 4
for i in range(rows):
    for space in range(i):
        print(" ", end="")
    for j in range(2 * (rows - i) - 1):
        if j % 2 == 0:
            print("1", end="")
        else:
            print("0", end="")
    print()
```

b)
```
  *
 ***
*****
 ***
  *
```

➔

```python
rows = 3
for i in range(1, rows + 1):
    print(" " * (rows - i) + "*" * (2 * i - 1))

for i in range(rows - 1, 0, -1):
    print(" " * (rows - i) + "*" * (2 * i - 1))
```

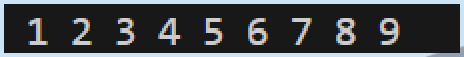
**4. Change the following Python code from using a while loop to for loop:**

x=1
while x<10:
    print x,
    x+=1

➔

```
for x in range(1, 10):
    print(x, end=" ")
```

**Output:**



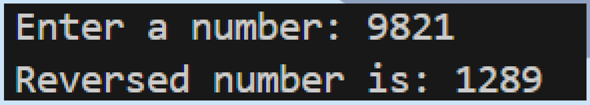
```
1 2 3 4 5 6 7 8 9
```

**5. Write a Python Program to Reverse a given number.**

➔

```
num = int(input("Enter a number: "))
reverse = 0
while num > 0:
    digit = num % 10
    reverse = (reverse * 10) + digit
    num = num // 10
print("Reversed number is:", reverse)
```

**Output:**



```
Enter a number: 9821
Reversed number is: 1289
```

**6. Write a python program to find Factorial of given number.**

➔

```
num = int(input("Enter a number: "))
factorial = 1
for i in range(1, num + 1):
    factorial = factorial * i
print("Factorial of", num, "is:", factorial)
```

**Output:**

```
Enter a number: 5
Factorial of 5 is: 120
```

### 7. Write a python program to find Fibonacci series for given number.

➔

```
n = int(input("Enter the number of terms: "))
a = 0
b = 1
if n <= 0:
    print("Please enter a positive number.")
else:
    print("Fibonacci series:")
    for i in range(n):
        print(a, end=" ")
        c = a + b
        a = b
        b = c
```

**Output:**

```
Enter the number of terms: 6
Fibonacci series:
0 1 1 2 3 5
```

### 8. Find sum of four digit number.

➔

```
num = int(input("Enter a four-digit number: "))
sum_of_digits = 0
while num > 0:
    digit = num % 10
    sum_of_digits = sum_of_digits + digit
    num = num // 10
print("Sum of digits is:", sum_of_digits)
```

**Output:**

```
Enter a four-digit number: 8543
Sum of digits is: 20
```

## **Practical No. 6 :** Write Python program to demonstrate use of loop control statements: continue, pass, break

- **Practical related questions**

1. **Describe Keyword "continue" with example.**

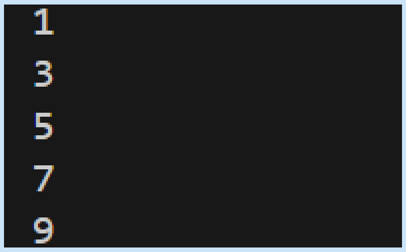
➔

The `continue` statement is used inside loops (`for` or `while`) to skip the rest of the current iteration and immediately move to the next iteration. When Python encounters `continue`, it ignores all statements below it in the loop body for the current pass. Control jumps back to the loop's condition check (for `while`) or to the next item (for `for`).

**Example:**

```python
# Program to print odd numbers between 1 and 10 using continue
num = 0
while num < 10:
    num += 1
    if num % 2 == 0:
        continue
    print(num)
```

**Output:**



```
1
3
5
7
9
```

## 2. Describe pass with example.

➔

The pass statement in Python is used as a placeholder when a statement is syntactically required but you do not want any code to execute. It essentially does nothing and is often used as a stub for future code. When the pass statement is executed, it does nothing and allows the program to continue execution without any error. It is useful in situations where the program requires a block of code syntactically, but you don't have a logic for it yet or want to leave it blank temporarily.

**Example:**

```
# Program demonstrating pass in a for loop
for i in range(1, 6):
    if i % 2 == 0:
        pass
    else:
        print(i)
```

**Output:**

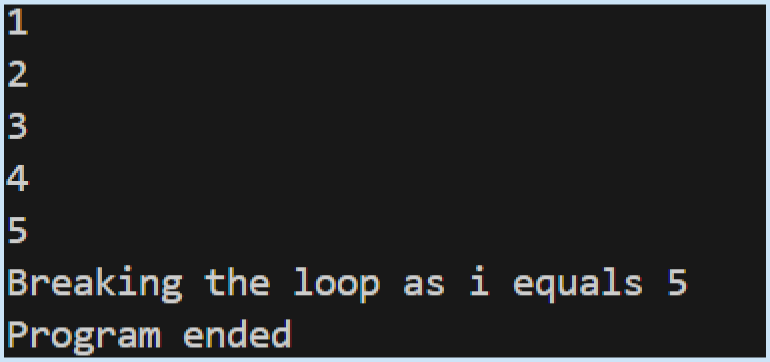
```
1
3
5
```

## 3. Write program to demonstrate use of break.

➔

```
for i in range(1, 11):
    print(i)
    if i == 5:
        print("Breaking the loop as i equals 5")
        break
print("Program ended")
```

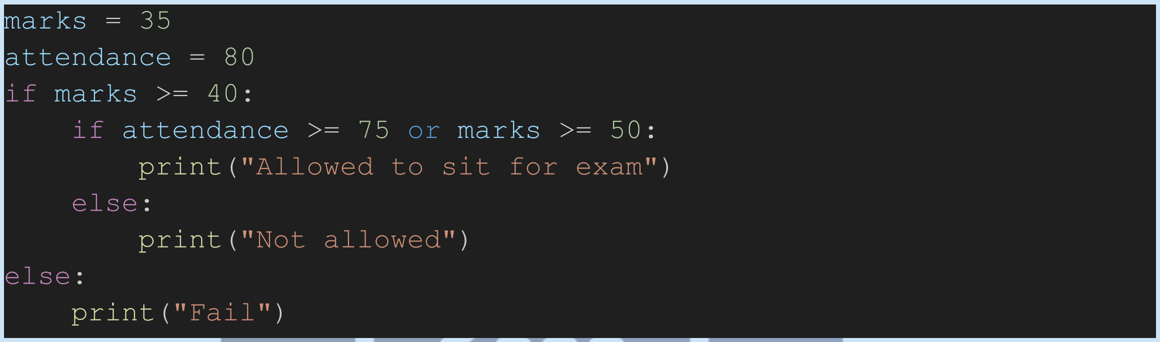
**Output:**



```
1
2
3
4
5
Breaking the loop as i equals 5
Program ended
```

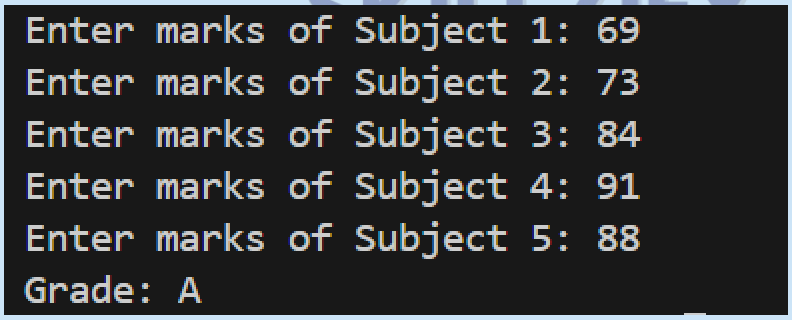
4. **Write program to demonstrate use of nested if-else.**

➔



```python
marks = 35
attendance = 80
if marks >= 40:
    if attendance >= 75 or marks >= 50:
        print("Allowed to sit for exam")
    else:
        print("Not allowed")
else:
    print("Fail")
```

**Output:**



```
 Enter marks of Subject 1: 69
 Enter marks of Subject 2: 73
 Enter marks of Subject 3: 84
 Enter marks of Subject 4: 91
 Enter marks of Subject 5: 88
 Grade: A
```

## **Practical No. 7 :** Write Python program to perform following operations on Lists: Create list, Access list, Update list (Add item, Remove item), and Delete list

- **Practical related questions**

**1. Write syntax for a method to sort a list.**

➔

In Python, to sort a list, you can use the `sort()` method for sorting in-place, or the `sorted()` function for returning a new sorted list.

a) Using `sort()` method
**Syntax:** `list_name.sort()`

b) Using `sorted()` method
**Syntax:** `sorted_list = sorted(list_name)`

**2. Justify the statement "Lists are mutable" not.**

➔

The statement "Lists are mutable" refers to the fact that in Python, lists can be changed after their creation. This means you can modify the contents of a list, such as adding, removing, or changing elements, without creating a new list.

**3. Describe various list functions.**

➔

a) append() -
Adds an element to the end of the list.

Syntax -
`list.append(element)`

b) insert() -

Inserts an element at a specified index in the list. Shifts the subsequent elements to the right.

Syntax -
```
list.insert(index, element)
```

c) remove() -

Removes the first occurrence of the specified element from the list.

Syntax -
```
list.remove(element)
```

d) pop() -

Removes and returns the element at the specified index. If no index is specified, it removes and returns the last element. Raises an `IndexError` if the list is empty .

Syntax -
1) `list.pop(index)`
2) `list.pop()`

e) clear() -

Removes all elements from the list, leaving it empty.

Syntax -
```
list.clear()
```

f) sort() -

Sorts the list in ascending order by default. You can pass the `reverse=True` argument to sort in descending order.

Syntax -
```
1) list.sort()
2) ist.sort(reverse=True)
```

g) reverse() -

Reverses the order of elements in the list.

Syntax -

`list.reverse()`

**4. Write a Python program to find common items from two lists.**

➔

```python
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_items = list(set(list1) & set(list2))
print("Common items:", common_items)
```

**Output:**

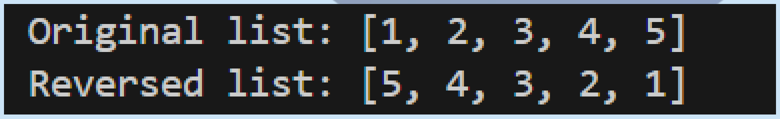
```
Common items: [4, 5]
```

**5. Write a Python program to reverse a list.**

➔

```python
my_list = [1, 2, 3, 4, 5]
print("Original list:", my_list)
my_list.reverse()
print("Reversed list:", my_list)
```

**Output:**



```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

**Practical No. 8 :** Write python program to use built-in functions/methods on list: cmp, len, max, list, append, count, extend, insert, pop, remove, etc.

## Practical related questions

1. **Write syntax for a method to reverse a list.**

➔

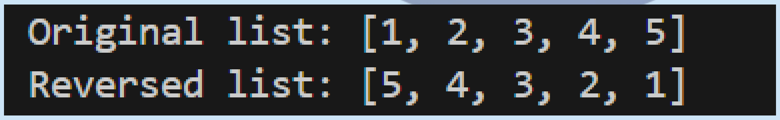
The `reverse()` method is used to reverse a list. It reverses the order of elements in the list in-place, meaning the original list is modified.

**Syntax -**
list_name.reverse()

**Example -**

```
my_list = [1, 2, 3, 4, 5]
print("Original list:", my_list)
my_list.reverse()
print("Reversed list:", my_list)
```

**Output -**



```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

2. **Describe various list functions.**

➔

a) append() -
Adds an element to the end of the list.

Syntax -
`list.append(element)`

b) insert() -
Inserts an element at a specified index in the list. Shifts the subsequent elements to the right.

Syntax -
```
list.insert(index, element)
```

c) remove() -
Removes the first occurrence of the specified element from the list.

Syntax -
```
list.remove(element)
```

d) pop() -
Removes and returns the element at the specified index. If no index is specified, it removes and returns the last element. Raises an `IndexError` if the list is empty .

Syntax -
```
1) list.pop(index)
2) list.pop()
```

e) clear() -
Removes all elements from the list, leaving it empty.

Syntax -
```
list.clear()
```

f) sort() -
Sorts the list in ascending order by default. You can pass the `reverse=True` argument to sort in descending order.

Syntax -
```
1)   list.sort()
2)   ist.sort(reverse=True)
```

g) reverse() -

Reverses the order of elements in the list.

Syntax -

```
list.reverse()
```

## 3. Describe the use of pop operator in list.

➔

The `pop()` method in Python is used to remove and return an element from a list. This method can either remove the element at a specified index or remove the last element if no index is provided.

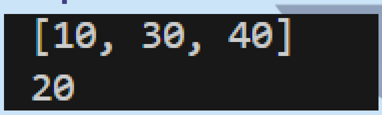
**Syntax -**

list_name.pop(index)

**Example -**

```
my_list = [10, 20, 30, 40]
removed_item = my_list.pop(1)
print(my_list)
print(removed_item)
```

**Output -**



```
[10, 30, 40]
20
```

## 4. Describe the use extend & pop method in list.

➔

a) `extend()` Method:

The `extend()` method is used to add all elements from an iterable (like a list, tuple, or set) to the end of the current list. This method modifies the original list and does not return a new list.

**Syntax:**

list_name.extend(iterable)

**Example:**

```
list1 = [1, 2, 3]
list2 = [4, 5]
list1.extend(list2)
print(list1)
```

**Output:**

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

b) `pop()` Method:

The `pop()` method is used to remove and return an element at a specific index in a list. If no index is specified, it removes and returns the last element. It modifies the original list by removing the element.

**Syntax :**

list_name.pop(index)
list_name.pop()

**Example:**

```
my_list = [10, 20, 30, 40]
removed_item = my_list.pop(1)
print(my_list)
print(removed_item)
removed_item = my_list.pop()
print(my_list)
print(removed_item)
```

**Output:**

```
[10, 30, 40]
20
[10, 30]
40
```

### 5. Write a Python program to find common items from two lists.

➔

```
list1 = [10, 20, 30, 40, 50]
list2 = [30, 40, 50, 60, 70]
common_items = [item for item in list1 if item in list2]
```

```
print("Common items:", common_items)
```

**Output:**
Common items: [30, 40, 50]

### 6. Write a Python program to sort a list.

➔

```
my_list = [30, 10, 50, 20, 40]
my_list.sort()
print("Sorted list:", my_list)
```

**Output:**
Sorted list: [10, 20, 30, 40, 50]

## Practical No. 9 : Write python program to perform following operations on tuple: Create, Access, Print, Delete & Convert tuple into list and vice-versa

- **Practical related questions**

1. **Define empty tuple. Write syntax to create empty tuple.**

➔

A tuple is an ordered, immutable collection of elements. An empty tuple is a tuple that contains no elements.

**Syntax to create an empty tuple:**
empty_tuple = ()

**Example:**

```
empty_tuple = ()
print("Empty tuple:", empty_tuple)
```

**Output:**
Empty tuple: ()

2. **Write syntax to copy specific elements existing tuple into new tuple.**

➔

**Syntax to Copy Specific Elements from an Existing Tuple into a New Tuple:**
To copy specific elements from an existing tuple into a new tuple, you can use slicing.

**Slicing -**
Slicing allows you to select a range of elements from the tuple and store them in a new tuple.

**Example:**

```
existing_tuple = (10, 20, 30, 40, 50)
new_tuple = existing_tuple[1:4]
print("New tuple:", new_tuple)
```

**Output:**

New tuple: (20, 30, 40)

## 3. Compare tuple with list (Any 4 points).

➔

| Tuple | List |
|---|---|
| Immutable (Cannot be changed after creation). | Mutable (Can be changed after creation) |
| Defined with parentheses () | Defined with square brackets [ ] |
| Faster than lists due to immutability | Slower compared to tuples |
| Used for fixed collections of data | Used for collections that may change during execution |

## 4. Create a tuple and find the minimum and maximum number from it.

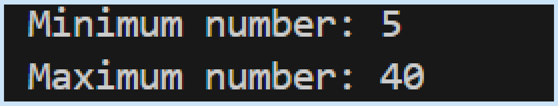
➔

```
my_tuple = (10, 20, 5, 40, 30)

min_num = min(my_tuple)
max_num = max(my_tuple)

print("Minimum number:", min_num)
print("Maximum number:", max_num)
```

**Output:**



```
Minimum number: 5
Maximum number: 40
```

## 5. Write a Python program to find the repeated items of a tuple.

➔

```
my_tuple = (10, 20, 30, 20, 40, 30, 50, 20)
repeated_items = []
for item in my_tuple:
    if my_tuple.count(item) > 1 and item not in repeated_items:
        repeated_items.append(item)
print("Repeated items:", repeated_items)
```

**Output:**

```
Repeated items: [20, 30]
```

## 6. Print the number in words for Example: 1234 => One Two Three Four.

➔

```
num_dict = {0: 'Zero', 1: 'One', 2: 'Two', 3: 'Three', 4: 'Four',
            5: 'Five', 6: 'Six', 7: 'Seven', 8: 'Eight', 9: 'Nine'}
num = 1234
for digit in str(num):
    print(num_dict[int(digit)], end=' ')
```

**Output:**

```
One Two Three Four
```

## **Practical No. 10 :** Write python program to perform following operations on the Set: Create set, Access Set, Update Set, Delete Set

- **Practical related questions**

**1. Describe the various set operations.**

➔

1) **Add** (`add()`):

   The add operation adds a single element to the set.

   **Syntax**:

   `set1.add(element)`

   **Example:**

```
set1 = {1, 2, 3}
set1.add(4)
print(set1)
```

   **Output:**

```
{1, 2, 3, 4}
```

2) **Remove**(`remove()`):

   The remove operation removes an element from the set, and if the element is not found, it raises a KeyError.

   **Syntax**:

   `set1.remove(element)`

   **Example:**

```
set1 = {1, 2, 3}
set1.remove(2)  # Removes 2
```

```
print(set1)
```

**Output:**
{1, 3}

3) **Clear (`clear()`):**
The clear operation removes all elements from the set.

**Syntax:**
`set1.clear()`

**Example:**

```
set1 = {1, 2, 3}
set1.clear()
print(set1)
```

**Output:**
set()

## 2. Describe the various methods of set.

➔

**1) add()**
Adds a single element to the set.

**Syntax:**
`set.add(element)`

**Example:**

```
s = {1, 2, 3}
s.add(4)
print(s)
```

**Output:**
{1, 2, 3, 4}

**2) update()**
Adds multiple elements from another set, list, tuple, etc.

**Syntax:**
`set.update(iterable)`

**Example:**
```
s = {1, 2}
s.update([3, 4])
print(s)
```

**Output:**
{1, 2, 3, 4}

3) **remove()**
Removes a specific element from the set. Raises an error if the element is not present.

**Syntax:**
`set.remove(element)`

**Example:**
```
s = {1, 2, 3}
s.remove(2)
print(s)
```

**Output:**
{1, 3}

4) **pop()**
Removes and returns a random element from the set.

**Syntax:**
`set.pop()`

**Example:**
```
s = {1, 2, 3}
s.pop()
print(s)
```

**Output:**
{2, 3}

5) **clear()**
Removes all elements from the set.

**Syntax:**
```
set.clear()
```

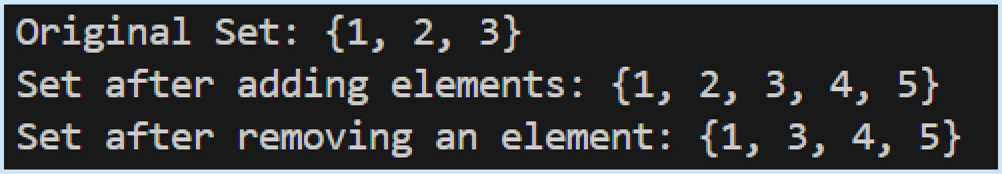
**Example:**
```
s = {1, 2, 3}
s.clear()
print(s)
```

**Output:**
set()

**3. Write a Python program to create a set, add member(s) in a set and remove one item from set.**

➔
```
my_set = {1, 2, 3}
print("Original Set:", my_set)
my_set.add(4)
my_set.add(5)
print("Set after adding elements:", my_set)
my_set.remove(2)
print("Set after removing an element:", my_set)
```

**Output:**


```
Original Set: {1, 2, 3}
Set after adding elements: {1, 2, 3, 4, 5}
Set after removing an element: {1, 3, 4, 5}
```

## 4. Write a Python program to find maximum and the minimum value in a set.

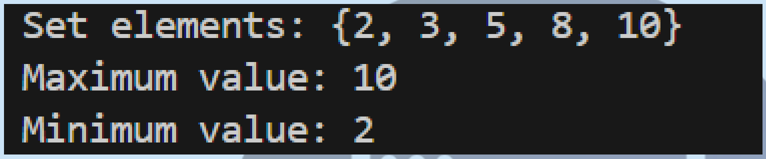
➔

```
my_set = {5, 10, 3, 8, 2}

maximum = max(my_set)
minimum = min(my_set)

print("Set elements:", my_set)
print("Maximum value:", maximum)
print("Minimum value:", minimum)
```

**Output:**



```
Set elements: {2, 3, 5, 8, 10}
Maximum value: 10
Minimum value: 2
```

## 5. Write a Python program to find the length of a set.

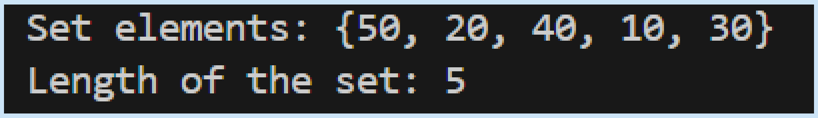
➔

```
my_set = {10, 20, 30, 40, 50}

length = len(my_set)

print("Set elements:", my_set)
print("Length of the set:", length)
```

**Output:**



```
Set elements: {50, 20, 40, 10, 30}
Length of the set: 5
```