Practical No. 1: Install Python IDE

Download Python :

https://www.python.org/downloads/

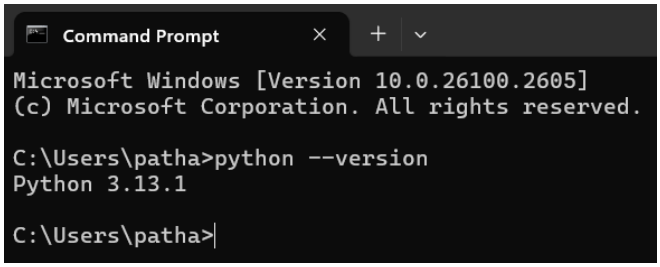
Download PyCharm:

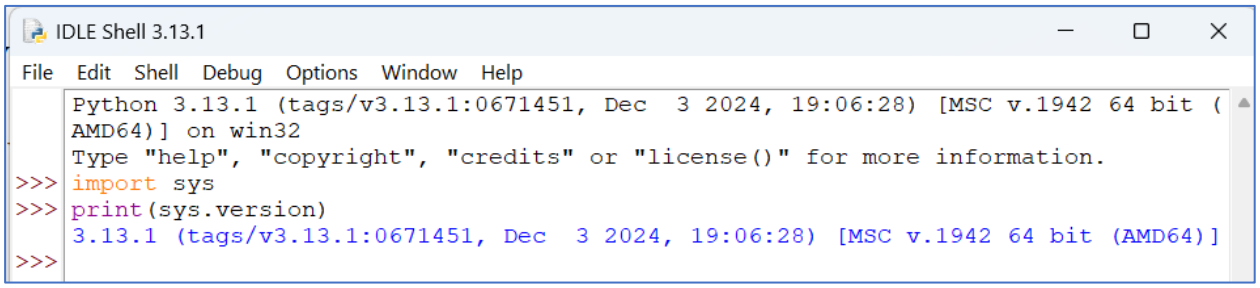
https://www.jetbrains.com/pycharm/download/#section=windows .

1) Write steps to install python on windows.

https://www.geeksforgeeks.org/how-to-install-python-on-windows/

2) Print the version of Python.





3) List key features of python.

Features in Python

- Free and Open Source. ...
- Easy to code. ...
- Easy to Read. ...
- Object-Oriented Language. ...
- GUI Programming Support. ...
- High-Level Language. ...
- Large Community Support. ...
- Easy to Debug.

4) What are the key features of Python?

https://www.geeksforgeeks.org/python-features/

5) What are the applications of Python?

Python is a versatile programming language with many applications, including:

- **Data science**: Python is used to analyse and visualize data, conduct statistical calculations, and build machine learning algorithms

- **Software development**: Python is used to develop websites, software, and other applications
- **Task automation**: Python is used to automate tasks
- **Artificial intelligence**: Python is used to develop artificial intelligence applications
- **Scientific computing**: Python is used for scientific computing, including complex calculations
- **Education**: Python is used to teach programming at the introductory and advanced levels
- **Everyday tasks**: Python is used by non-programmers, such as accountants and scientists, for everyday tasks like organizing finances

Some examples of Python applications include:

- **Data visualization**
  Python can be used to create plots, add labels and legends, and customize three-dimensional plots
- **Console applications**
  Python can be used to create command-line applications that use simple text to complete tasks
- **Machine learning**
  Python can be used to build machine learning algorithms
- **Scientific computing**
  Python can be used for complex calculations, such as those used in machine learning and artificial intelligence

https://www.python.org/about/apps/

6) State use of pip & pep.

In the Python programming language, "pip" is the package installer used to download and manage external libraries (packages) from the Python Package Index (PyPI), while "PEP" stands for "Python Enhancement Proposal" which is a document outlining proposed changes or improvements to the Python language itself

Key points about pip and PEP:

- **Pip:**
  - Function: Installs and manages third-party Python packages.
  - Command usage: pip install <package_name> to install a package.
  - Accesses packages from: Python Package Index (PyPI).
- **PEP:**
  - Function: A formal document proposing changes or additions to the Python language.
  - Process: PEPs are discussed and reviewed by the Python community before being implemented.
  - Example: PEP 8 - Defines the recommended style guide for Python code.

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

1. List different modes of Programming in Python

In Python, the primary "modes" of programming refer to how you interact with the Interpreter : Interactive Mode where you type code line-by-line directly, and Script Mode where you write code in a file (with a .py extension) and execute the entire script at once; essentially, these are the two ways to run Python code.

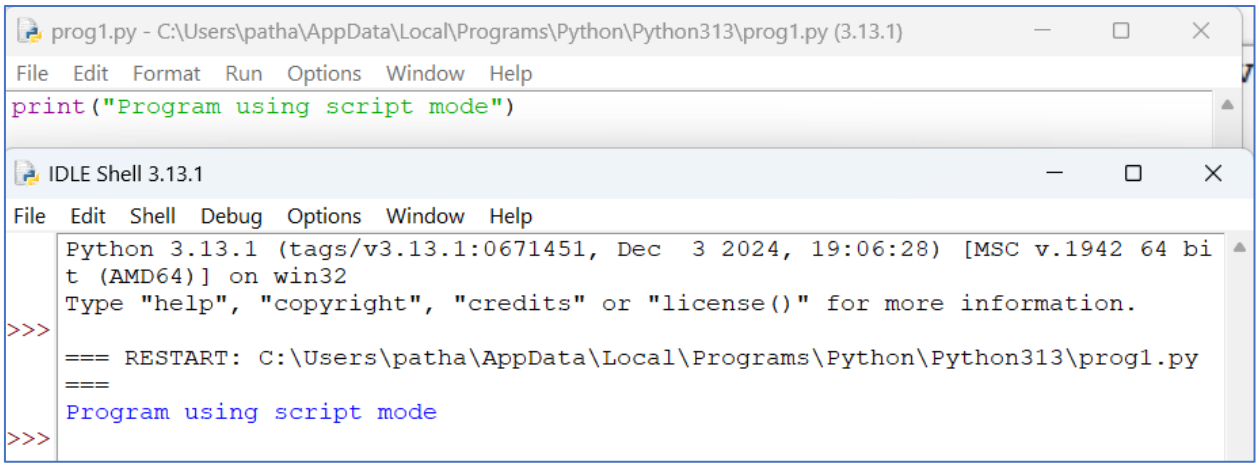
- **Interactive Mode:**
  - Useful for testing small snippets of code or quickly experimenting with syntax.
  - Access the interpreter directly by typing "python" in the command line.
  - Immediate feedback on each line of code executed.
- **Script Mode:**
  - For writing larger, complex programs with multiple lines of code.
  - Save your code in a file with a ".py" extension.
  - Execute the entire script by running the file name in the command line.

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

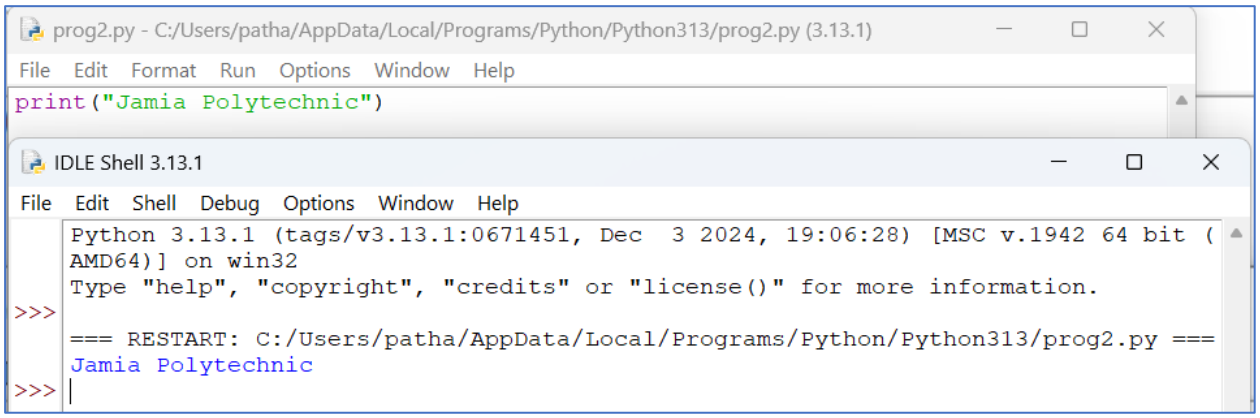
Executing a Python program in script mode involves the following steps:

- **Write the Code:**
  Create a text file using a text editor or an IDE and write your Python code. Save the file with a .py extension, which signifies a Python script.
- **Open a Terminal or Command Prompt:**
  Navigate to the directory where you saved your Python script using the cd command.
- **Execute the Script:**
  Use the python command followed by the name of your script file.
  For example: `python myscript.py`
- **View Output:** The Python interpreter will execute your script, and any output produced by the script will be displayed in the terminal or command prompt window.

https://www.youtube.com/watch?v=swmxFmfHNDY



3. Write a Python program to display “MSBTE” using Script Mode.



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

1. Describe ternary operator in Python.

The ternary operator is a concise way to write an if-else statement in a single line. It's also known as a conditional expression. It returns a true or false value by evaluating a boolean condition.

Syntax : value_if_true if condition else value_if_false

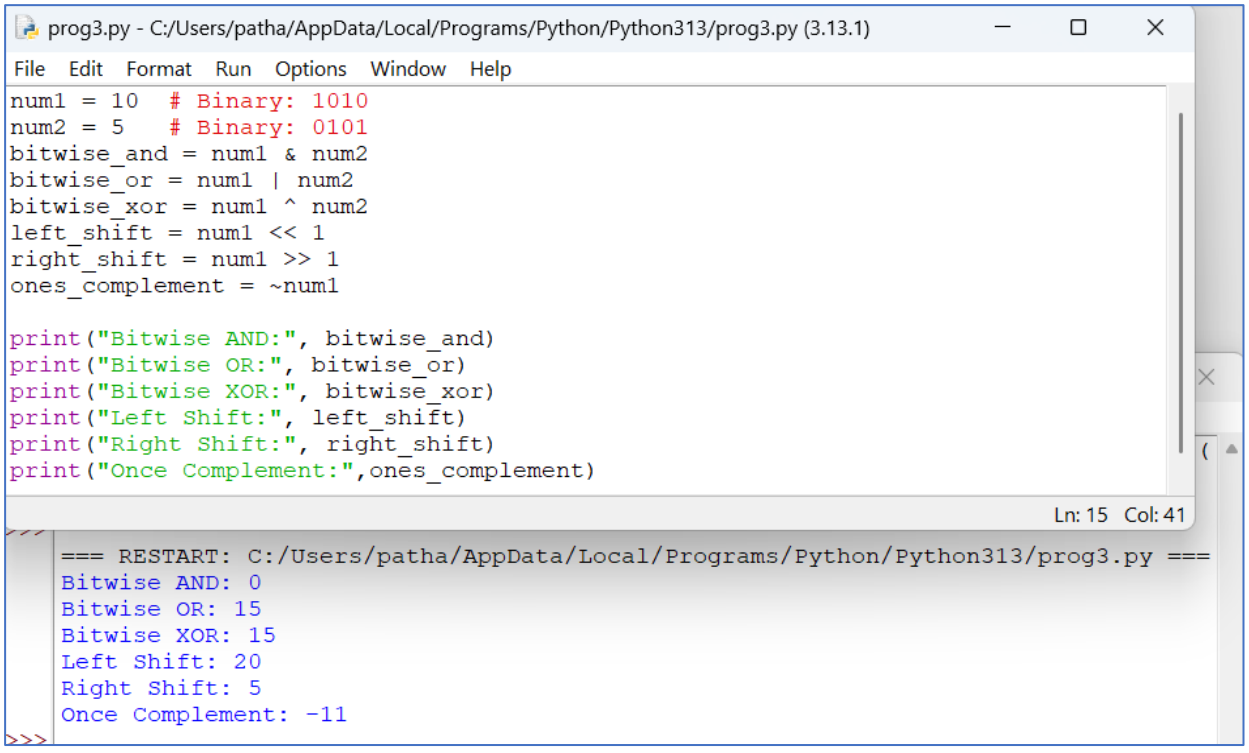
Example :

```
>>> number=10
>>> print("Even" if number % 2 == 0 else "Odd")
Even
```

```
>>> x=10
>>> y=20
>>> print(x if x>y else y)
20
```

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

Bitwise Operators: Python bitwise operators are used to perform bitwise calculations on integers. The integers are first converted into binary and then operations are performed on each bit or corresponding pair of bits, hence the name bitwise operators. The result is then returned in decimal format.



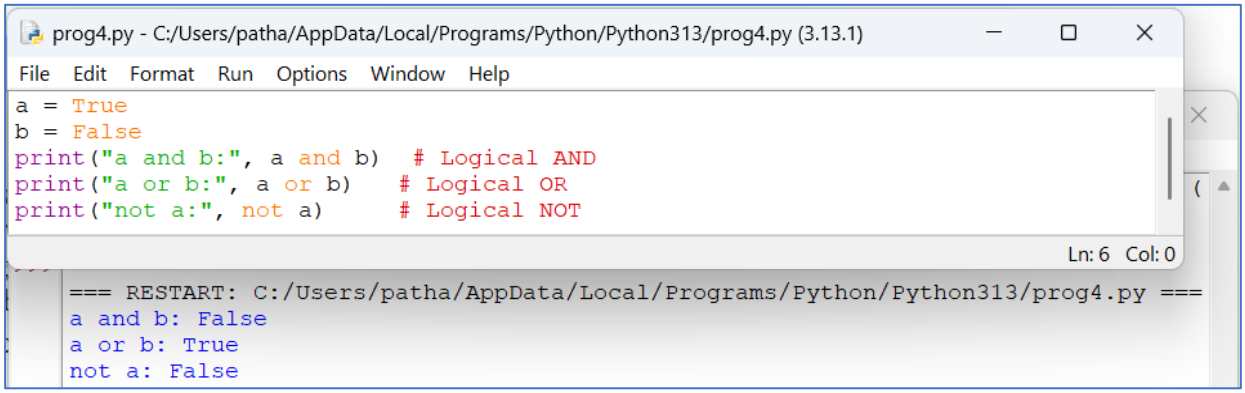
Bitwise Operators in Python | Right-shift, Left-shift, AND, OR, NOT, XOR | Python
https://www.youtube.com/watch?v=Yk3Jwm5YuFs

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

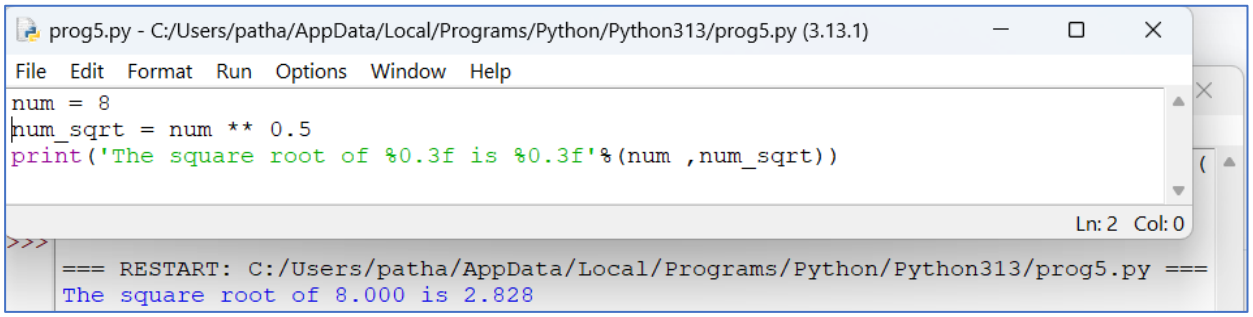
Logical Operators: Python logical operators are used to combine conditional statements, allowing you to perform operations based on multiple conditions. These Python operators, alongside arithmetic operators, are special symbols used to carry out computations on values and variables.

- Logical AND : Returns True if both the operands are true
- Logical OR : If any of the two operands are non-zero then condition becomes true
- Logical NOT : Used to reverse the logical state of its operand

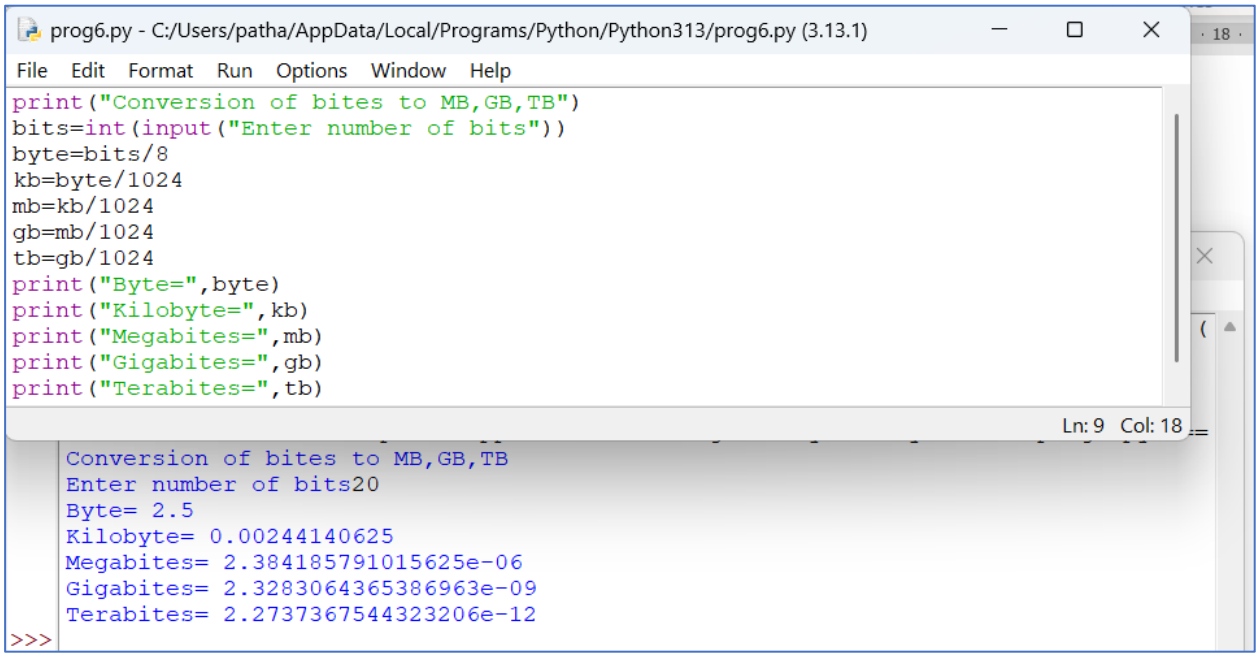
Example :



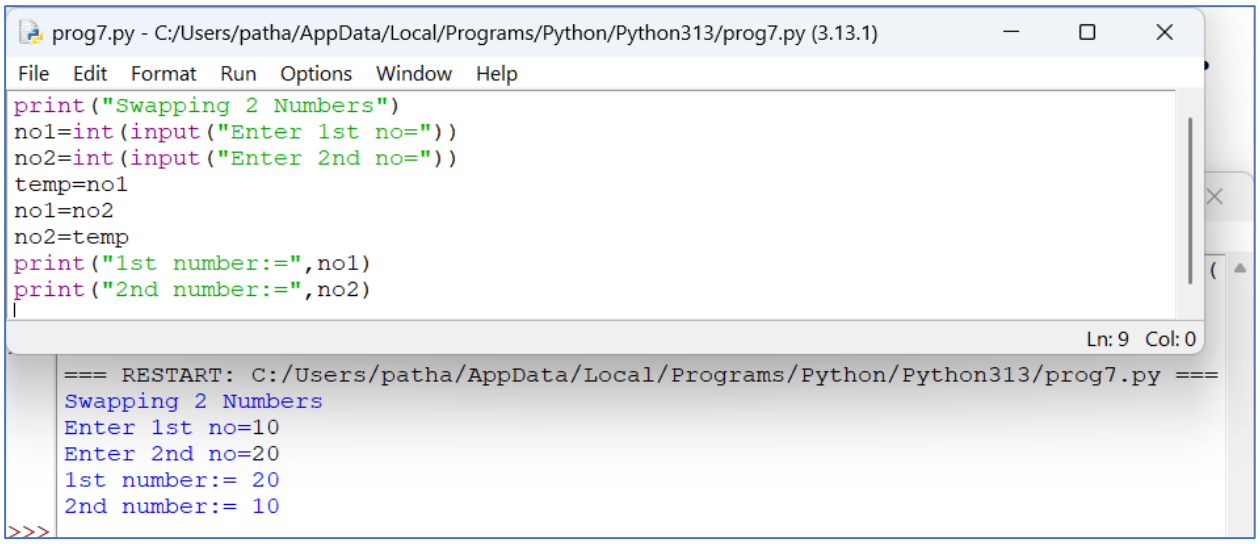
4. Write a program to find the square root of a number.



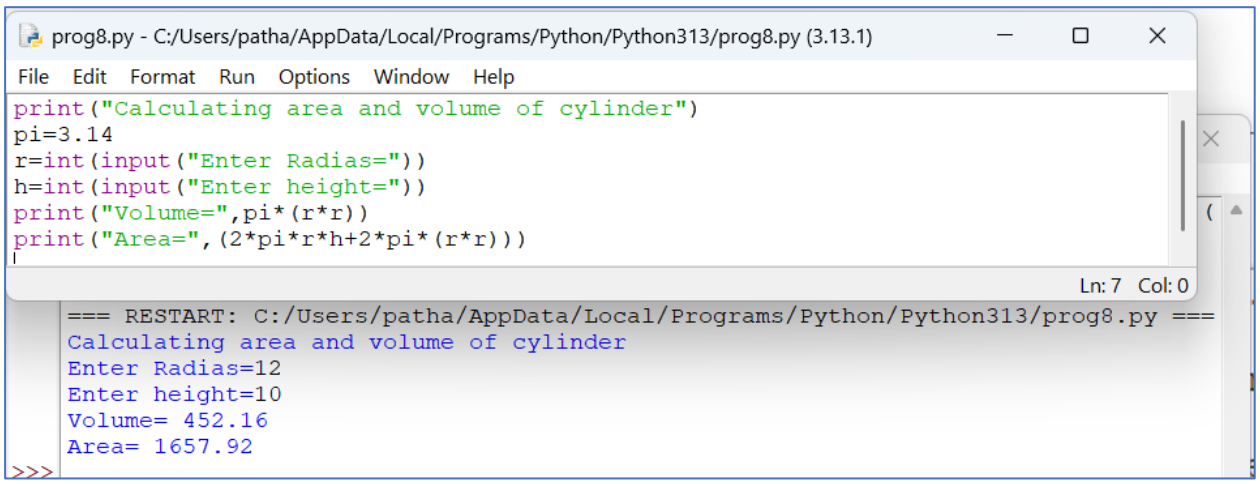
5. Write a program to convert bits to Megabytes, Gigabytes and Terabytes.



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



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



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

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

In Python, if-else and nested-if statements are used for conditional execution of code blocks.

if-else: A simple conditional structure where one block of code is executed if a condition is true, and another block is executed if the condition is false.

Syntax :

```
if condition:
    # Code to execute if the condition is true
else:
    # Code to execute if the condition is false
```

Example :

```
x = 10
y = 5
if x > 5 and y < 10:
    print("Both conditions are true")
else:
    print("At least one condition is false")
```

Nested-if: A more complex conditional structure where an if-else statement is placed inside another if or else block. This allows for testing multiple conditions in a hierarchical manner.

Syntax :

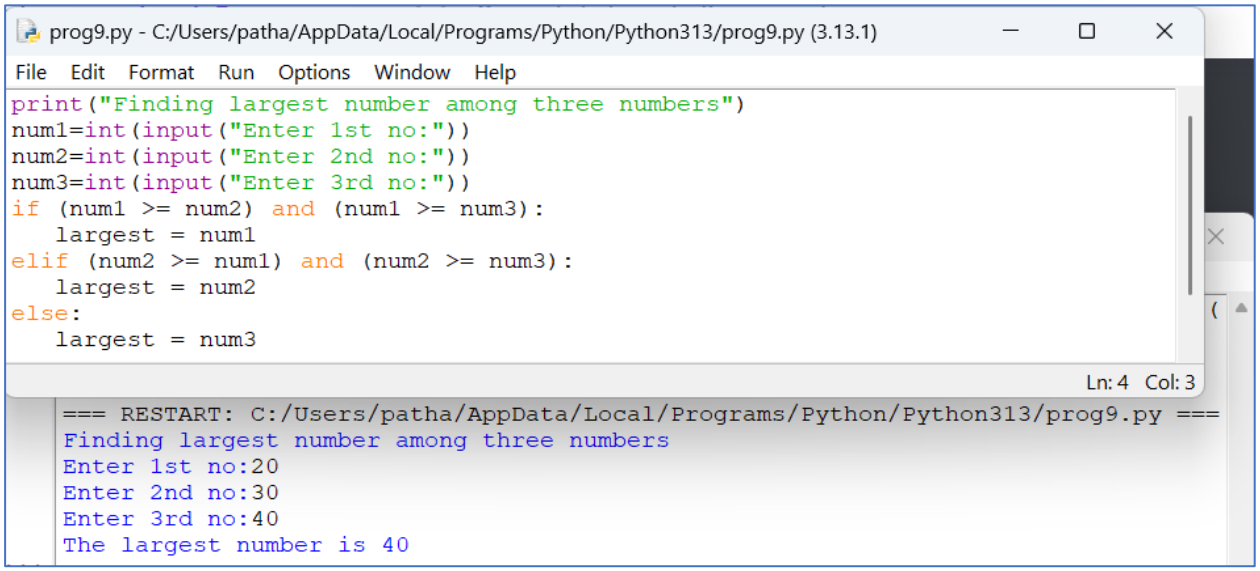
```
if condition1:
    # Code to execute if condition1 is true
    if condition2:
        # Code to execute if both condition1 and condition2 are true
    else:
        # Code to execute if condition1 is true but condition2 is false
else:
    # Code to execute if condition1 is false
```

```
x = 10
y = 5
if x > 5:
    if y < 10:
        print("x is greater than 5 and y is less than 10")
    else:
        print("x is greater than 5 but y is not less than 10")
else:
    print("x is not greater than 5")
```

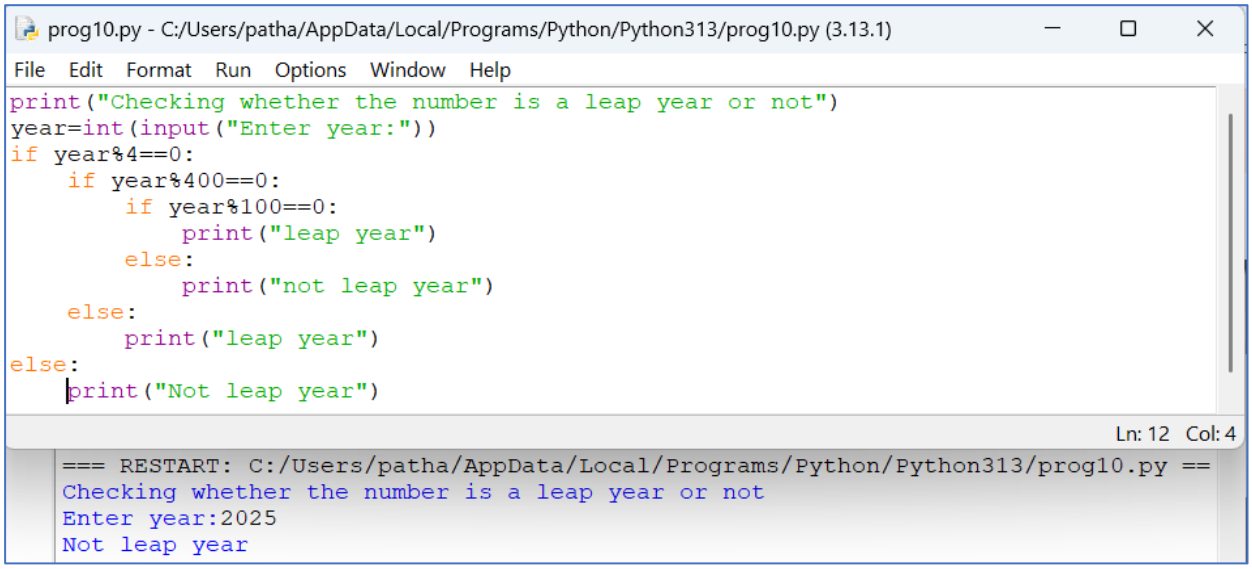
If else, elif in Python 🤔 | Nested If | Python for Beginners

https://www.youtube.com/watch?v=9xiFcK3MRYA&list=PLxCzCOWd7aiEb4apyN1Y8mD-QuUTr3SPQ&index=28

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



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



4. List operators used in if conditional statement.

In Python, you can use various operators within an if conditional statement:

Comparison Operators:

- ==: Equal to
- !=: Not equal to
- <: Less than
- >: Greater than
- <=: Less than or equal to
- >=: Greater than or equal to

Logical Operators:

- and: Returns True if both conditions are true
- or: Returns True if at least one condition is true
- not: Inverts the result of the condition

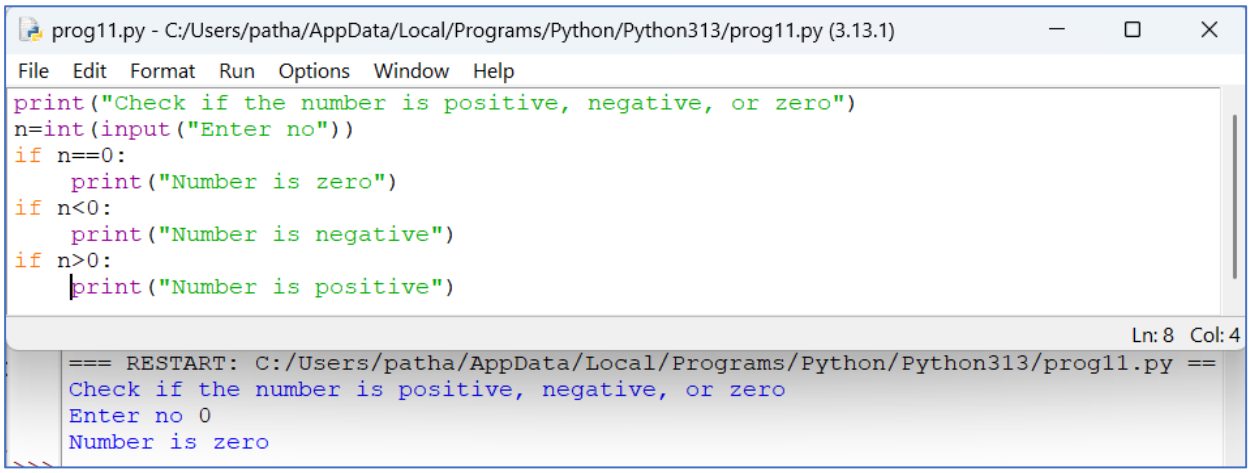
Membership Operators:

- in: Returns True if a value is found in a sequence (e.g., list, tuple, string)
- not in: Returns True if a value is not found in a sequence

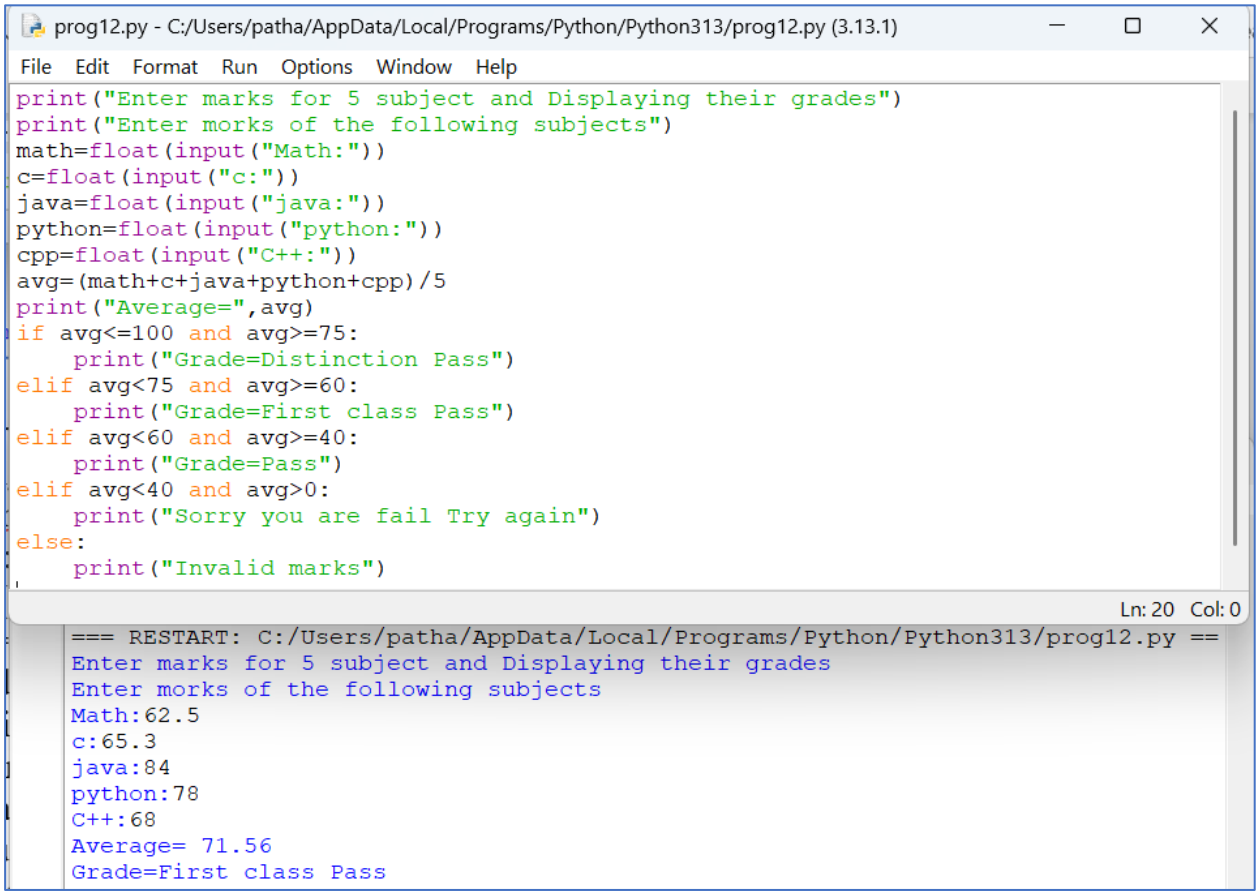
Identity Operators:

- is: Returns True if two variables refer to the same object
- is not: Returns True if two variables do not refer to the same object

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



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



For more examples visit : https://www.programiz.com/python-programming/examples

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

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

prog13.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog13.py (3.13.1)

File Edit Format Run Options Window Help

```
num=int(input("Enter a number to check palindrome or not :"))
rev=0
num1=num
print("Original Number ",num)
while num!=0:
    digit=int(num%10)
    num=int(num/10)
    rev=int(rev*10+digit)
print("Reversed Number",rev)
if num1==rev:
    print("Palindrome Number")
else:
    print("Not Palindrome Number")
```

Ln: 9 Col: 0

```
>>> 
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog13.py ==
Enter a number to check palindrome or not :123
Original Number  123
Reversed Number 321
Not Palindrome Number
>>> 
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog13.py ==
Enter a number to check palindrome or not :3003
Original Number  3003
Reversed Number 3003
Palindrome Number
>>> 
```

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

prog14.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog14.py (3.13.1)

File Edit Format Run Options Window Help

```
num=1
print("Even Numbers from 1-100")
while num<=20:
    if num%2==0:
        print(num)
    num=num+1
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog14.py ==
Even Numbers from 1-100
2
4
6
8
10
12
14
16
18
20
>>> 
```

3. Print the following patterns using loop:

a.

```
1010101
 10101
  101
   1
```

b.

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

prog16.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog16.py (3.13.1)

File Edit Format Run Options Window Help

```
rows=5
k = 1
n = 1
num=0
for i in range(1, rows):
    if(rows%2==0):
        num=0
    else:
        num=1
    for j in range (1, k + 1):
        print(end = " ")
    k = k + 1
    while n <= (2 * (rows - i) - 1):
        if rows%2==0:
            if num==1:
                print("1", end = "")
                n = n + 1
                num=0
            else:
                print("0", end = "")
                n = n + 1
                num=1
        else:
            if num==1:
                print("1", end = "")
                n = n + 1
                num=0
            else:
                print("0", end = "")
                n = n + 1
                num=1
    n = 1
    print()
```

Ln: 34 Col: 0

IDLE Shell 3.13.1

File Edit Shell Debug Options Window Help

```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog16.py ==
 1010101
  10101
   101
    1
>>>
```

prog15.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog15.py (3.13.1)

File Edit Format Run Options Window Help

```
n = 0
rows=3
for i in range(1, rows + 1):
    for j in range (1, (rows - i) + 1):
        print(end = " ")
    while n != (2 * i - 1):
        print("*", end = "")
        n = n + 1
    n = 0
    print()
k = 1
n = 1
for i in range(1, rows):
    for j in range (1, k + 1):
        print(end = " ")
    k = k + 1
    while n <= (2 * (rows - i) - 1):
        print("*", end = "")
        n = n + 1
    n = 1
    print()
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog15.py ==
  *
 ***
*****
 ***
  *
>>>
```

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

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

prog17.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog17.py (3.13.1)

File Edit Format Run Options Window Help

```
##x=1
##while x<10:
##    print (x)
##    x+=1
##for x in range(1,10): print(x);x+=1;
for x in range(1,10):
    print(x)
    x+=1
```

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog17.py ==
1
2
3
4
5
6
7
8
9
>>>
```

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

prog18.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog18.py (3.13.1)

File Edit Format Run Options Window Help

```
num=int(input("Enter a number to reverse"))
rev=0
print("Original Number ",num)
while num!=0:
    digit=int(num%10)
    num=int(num/10)
    rev=int(rev*10+digit)
print("Reversed Number ",rev)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog18.py ==
Enter a number to reverse12345
Original Number  12345
Reversed Number  54321
```

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

prog19.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog19.py (3.13.1)

File Edit Format Run Options Window Help

```
num=int(input("Enter Number to find factorial"))
fact=1
print("Factorial of ",num)
while num>=1:
    fact=fact*num
    num=num-1
print(fact)
```

Ln: 6 Col: 4

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog19.py ==
Enter Number to find factorial 12
Factorial of  12
479001600
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog19.py ==
Enter Number to find factorial5
Factorial of  5
120
```

7. Find sum of four-digit number.

prog21.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog21.py (3.13.1)

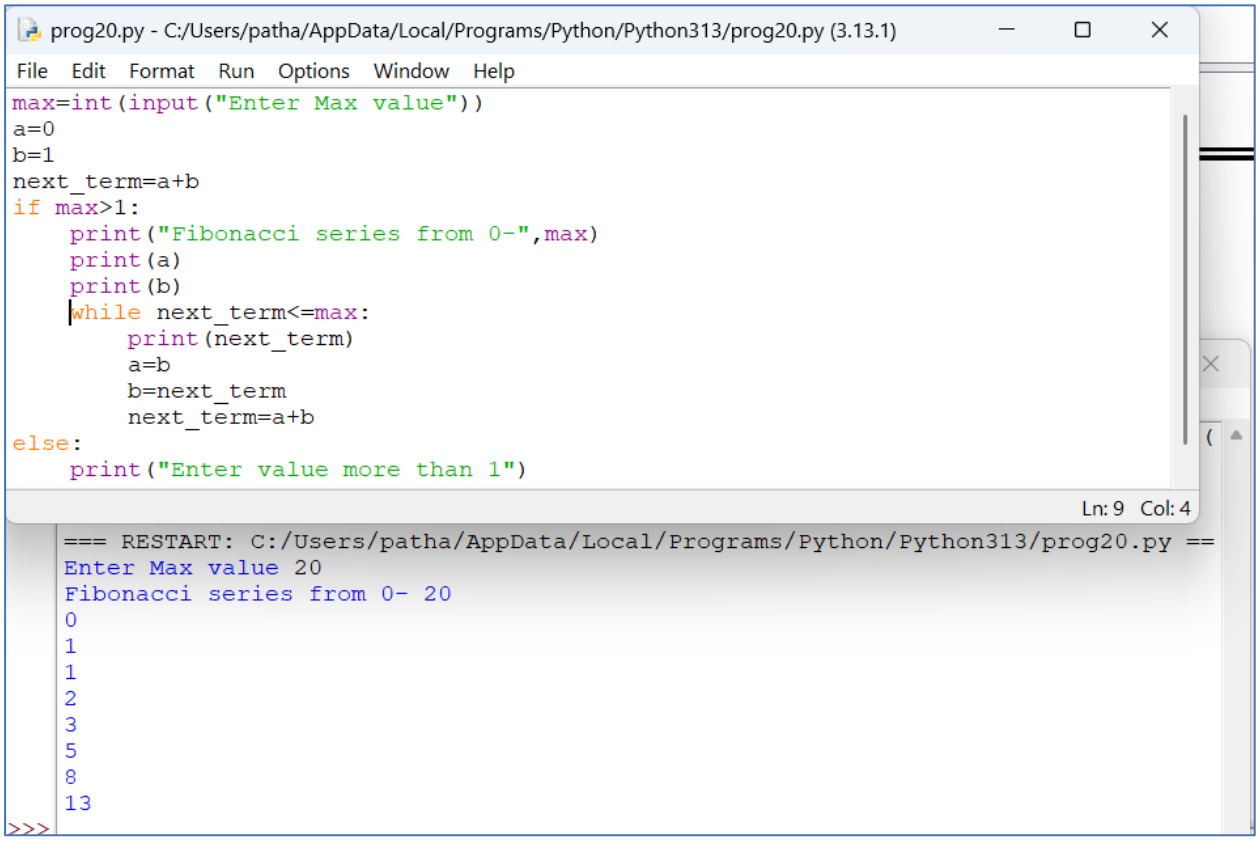
File Edit Format Run Options Window Help

```
number = int(input("Enter a four-digit number: "))
if 1000 <= number <= 9999:
    digit_sum = sum(int(digit) for digit in str(number))
    print("The sum of the digits is:", digit_sum)
else:
    print("Please enter a valid four-digit number.")
```

Ln: 2 Col: 0

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog21.py ==
Enter a four-digit number: 123
Please enter a valid four-digit number.
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog21.py ==
Enter a four-digit number: 1234
The sum of the digits is: 10
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog21.py ==
Enter a four-digit number: 12345
Please enter a valid four-digit number.
>>>
```

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



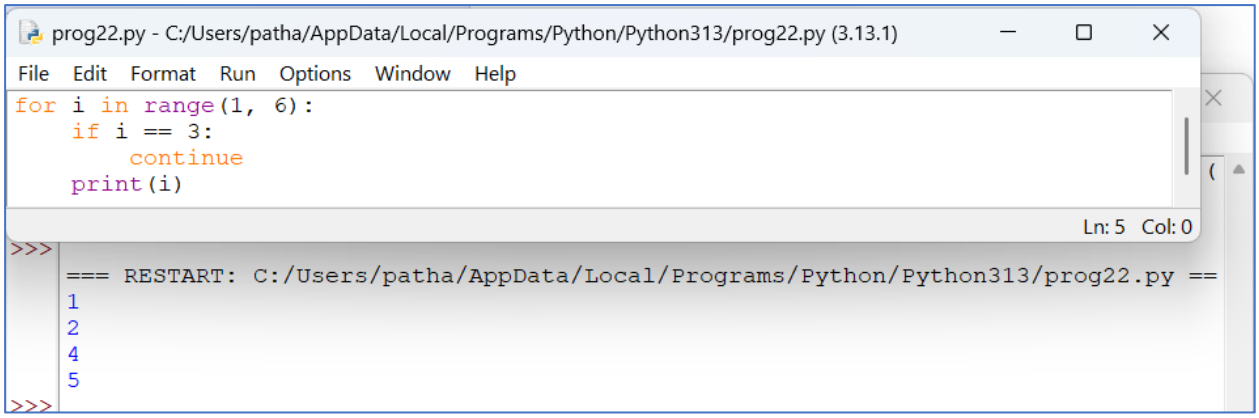
Practical No. 6: Write Python program to demonstrate use of loop control statements: continue, pass, break

1. Describe Keyword "continue" with example.

In Python, the continue keyword is used within loops (like for and while) to skip the remaining code in the current iteration and move on to the next iteration.

Syntax :

```
while condition:
# code block
        if some_condition:
        continue
# more code block
```



2. Describe pass with example.

Pass statement in Python is a null operation or a placeholder. It is used when a statement is syntactically required but we don't want to execute any code. It does nothing but allows us to maintain the structure of our program.

Syntax :

```
Pass
```

Acts as a placeholder.

It does nothing, allowing you to create an empty block of code without causing errors.

```
def my_empty_function():
  pass  # Placeholder for future implementation
my_empty_function()
```

3. Write program to demonstrate use of break.

The break statement in Python is used to exit or "break" out of a loop (either a for or while loop) prematurely. When the break statement is executed, the program immediately exits the loop, and the control moves to the next line of code after the loop.

prog23.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog23.py (3.13.1)

File Edit Format Run Options Window Help

```
# Example: Searching for an element in a list
a = [1, 3, 5, 7, 9, 11]
val = 7
for i in a:
    if i == val:
        print(f"Found at {i}!")
        break
else:
    print(f"not found")
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog23.py ==
Found at 7!
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog23.py ==
not found
>>>
```

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

prog24.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog24.py (3.13.1)

File Edit Format Run Options Window Help

```
def check_number(num):
    if num > 0:
        print("Number is positive")
        if num % 2 == 0:
            print("Number is also even")
        else:
            print("Number is odd")
    elif num < 0:
        print("Number is negative")
    else:
        print("Number is zero")

check_number(10)
check_number(-5)
check_number(0)
check_number(7)
```

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog24.py ==
Number is positive
Number is also even
Number is negative
Number is zero
Number is positive
Number is odd
>>>
```

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

A list is a type of data structure that holds an ordered collection of items, which means you can store anything from integers to strings, to other lists, and so on. They are mutable, allowing you to change their content without changing their identity.

list.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/list.py (3.13.1)

File Edit Format Run Options Window Help

```
# Create a list
my_list = [10, 20, 30, 40, 50]
print("Created List:", my_list)

# Access list elements
print("Element at index 0:", my_list[0])
print("Element at index 3:", my_list[3])

# Update list: Add item
my_list.append(60)  # Add to the end
print("List after appending 60:", my_list)

my_list.insert(2, 25)  # Insert 25 at index 2
print("List after inserting 25 at index 2:", my_list)

# Update list: Remove item
my_list.remove(30)  # Remove the value 30
print("List after removing 30:", my_list)

popped_element = my_list.pop(3)  # Remove element at index 3 and return it
print("List after popping element at index 3:", my_list)
print("Popped element:", popped_element)

# Delete List
del my_list  # Delete the entire list
# The following line would cause an error because my_list no longer exists
# print("List after deletion:", my_list)
```

```
>>>
==== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/list.py ===
Created List: [10, 20, 30, 40, 50]
Element at index 0: 10
Element at index 3: 40
List after appending 60: [10, 20, 30, 40, 50, 60]
List after inserting 25 at index 2: [10, 20, 25, 30, 40, 50, 60]
List after removing 30: [10, 20, 25, 40, 50, 60]
List after popping element at index 3: [10, 20, 25, 50, 60]
Popped element: 40
>>>
```

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

The syntax for sorting a list in Python using the sort() method is as follows:

Syntax : **list_name.sort(key=None, reverse=False)**

Example :

```
my_list = [5, 2, 9, 1, 5, 6]
sorted_list = sort_list(my_list)
print(sorted_list)  # Output: [1, 2, 5, 5, 6, 9]
```

- list_name: This is the name of the list that needs to be sorted.
- key: An optional argument that specifies a function to be called on each list element prior to making comparisons.

- reverse: An optional argument. If set to True, the list is sorted in descending order. By default, it is False, which sorts the list in ascending order.

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

A list is considered "mutable" because it can be changed or modified after it is created, meaning you can add, remove, or update individual elements within the list without needing to create a new list entirely

Key points supporting the statement "Lists are mutable":

- Direct element modification
- Adding and removing elements
- In-place changes

Example :

```
my_list = [1, 2, 3]
# Change an element
my_list[1] = 5
print(my_list)  # Output: [1, 5, 3]
# Add an element
my_list.append(4)
print(my_list)  # Output: [1, 5, 3, 4]
# Remove an element
my_list.remove(3)
print(my_list)  # Output: [1, 5, 4]
```

3. Describe various list functions.

Here's a description of various list functions in Python:

- **append(element):** Adds an element to the end of the list.
- **extend(iterable):** Appends elements from an iterable (like another list, tuple, or string) to the end of the list.
- **insert(index, element):** Inserts an element at a specific index in the list.
- **remove(element):** Removes the first occurrence of an element from the list.
- **pop(index):** Removes and returns the element at a specific index
- **clear():** Removes all elements from the list, making it empty.
- **index(element):** Returns the index of the first occurrence of an element.
- **count(element):** Returns the number of times an element appears in the list.
- **sort():** Sorts the elements of the list in ascending order (can be modified to sort in descending order).
- **reverse():** Reverses the order of elements in the list.
- **copy():** Returns a shallow copy of the list.
- **len(list):** Returns the number of elements in the list.
- **sorted(list):** Returns a new sorted list from the elements of the original list without modifying the original list.
- **min(list):** Returns the smallest element in the list.
- **max(list):** Returns the largest element in the list.
- **sum(list):** Returns the sum of all elements in the list (only works for numerical lists).

Example :

```
my_list = [3, 1, 4, 1, 5, 9]
# Using core functions
print(len(my_list))          # Output: 6
print(sorted(my_list))       # Output: [1, 1, 3, 4, 5, 9]
print(min(my_list))          # Output: 1
print(max(my_list))          # Output: 9
# Using list methods
my_list.append(2)
print(my_list)               # Output: [3, 1, 4, 1, 5, 9, 2]
my_list.insert(2, 7)
print(my_list)               # Output: [3, 1, 7, 4, 1, 5, 9, 2]
my_list.remove(1)            # Removes the first occurrence of 1
print(my_list)               # Output: [3, 7, 4, 1, 5, 9, 2]
my_list.sort()
print(my_list)               # Output: [1, 2, 3, 4, 5, 7, 9]
my_list.reverse()
print(my_list)               # Output: [9, 7, 5, 4, 3, 2, 1]
```

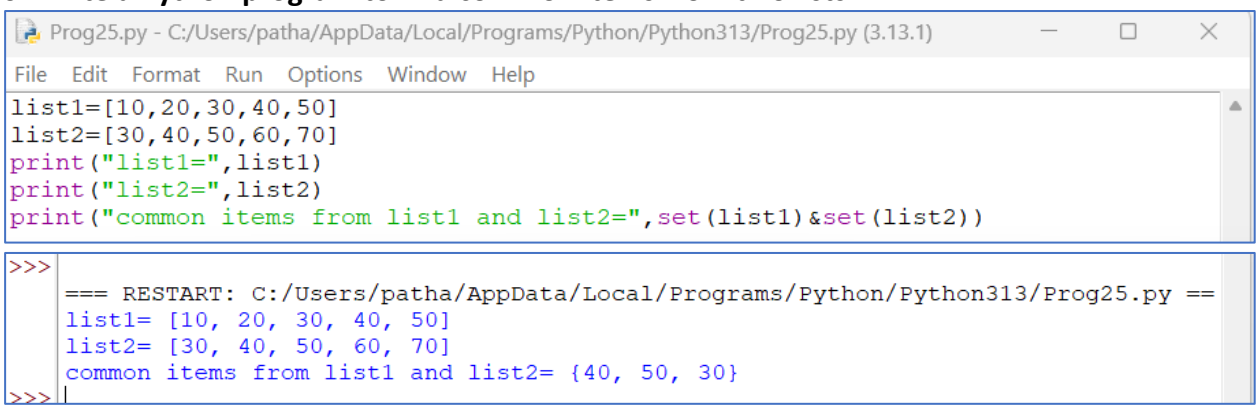
4. Describe the use pop operator in list.

The pop() operator in Python lists removes an element from a specified position and returns it. If no index is specified, it removes and returns the last element of the list. The pop() method modifies the list in place.

Example :

```
my_list = [10, 20, 30, 40, 50]
# Remove and return the element at index 2
popped_element = my_list.pop(2)
print(f"Popped element: {popped_element}")# Popped element: 30
print(f"Updated list: {my_list}")   # Updated list: [10, 20, 40, 50]
# Remove and return the last element
popped_element = my_list.pop()
print(f"Popped element: {popped_element}")#  Popped element: 50
print(f"Updated list: {my_list}")   #  Updated list: [10, 20, 40]
```

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



The & operator or intersection() method is the most efficient way to find common elements between two lists.

7. Write a Python program to reverse a list.

revlist.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/revlist.py (3.13.1)

File Edit Format Run Options Window Help

```
# Reverse the list in-place
a = [1, 2, 3, 4, 5]
a.reverse()
print(a)

# Create a new list that is a reversed list of 'a' using slicing
a = [10, 20, 30, 40, 50]
rev = a[::-1]
print(rev)

# Use reversed() to create an iterator and convert it back to a list
a = [11, 22, 33, 44, 55]
rev = list(reversed(a))
print(rev)

# Loop through each item and insert it at the beginning of new list
a = [101, 201, 301, 401, 501]
# Initialize an empty list to store reversed element
res = []
for val in a:
    res.insert(0, val)
print(res)
```

```
>>>
== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/revlist.py ==
[5, 4, 3, 2, 1]
[50, 40, 30, 20, 10]
[55, 44, 33, 22, 11]
[501, 401, 301, 201, 101]
>>>
```

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

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

tuple.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/tuple.py (3.13.1)

File Edit Format Run Options Window Help

```
# Create a tuple
my_tuple = ("apple", "banana", "cherry")
print("Created Tuple:", my_tuple)

# Access elements
print("First element:", my_tuple[0])
print("Last element:", my_tuple[-1])

# Print the entire tuple
print("Tuple:", my_tuple)

# Attempt to delete an element (demonstrates immutability)
try:
    del my_tuple[1]  # This will raise an error
except TypeError:
    print("Tuples are immutable. You cannot delete individual elements.")

# Delete the entire tuple
del my_tuple

# Attempt to print the deleted tuple (should raise an error)
try:
    print("Tuple:", my_tuple)
except NameError:
    print("Tuple has been deleted.")

my_tuple = ("apple", "banana", "cherry")
# Convert tuple to list
my_list = list(my_tuple)
print("Converted to list:", my_list)

# Convert list to tuple
my_list = [1, 2, 3]
my_tuple1 = tuple(my_list)
print("Converted to tuple:", my_tuple1)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/tuple.py ===
Created Tuple: ('apple', 'banana', 'cherry')
First element: apple
Last element: cherry
Tuple: ('apple', 'banana', 'cherry')
Tuples are immutable. You cannot delete individual elements.
Tuple has been deleted.
Converted to list: ['apple', 'banana', 'cherry']
Converted to tuple: (1, 2, 3)
>>>
```

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

An empty tuple is a tuple that contains no elements. It is represented by an empty set of parentheses. Empty tuples are useful as placeholders

empty_tuple = ()

or

empty_tuple = tuple() # Creating an empty tuple using the tuple() constructor

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

```
original_tuple = (10, 20, 30, 40, 50)
indices_to_copy = [0, 2, 4]
new_tuple = tuple(original_tuple[i] for i in indices_to_copy)
print(new_tuple)  # Output: (10, 30, 50)
```

Note :

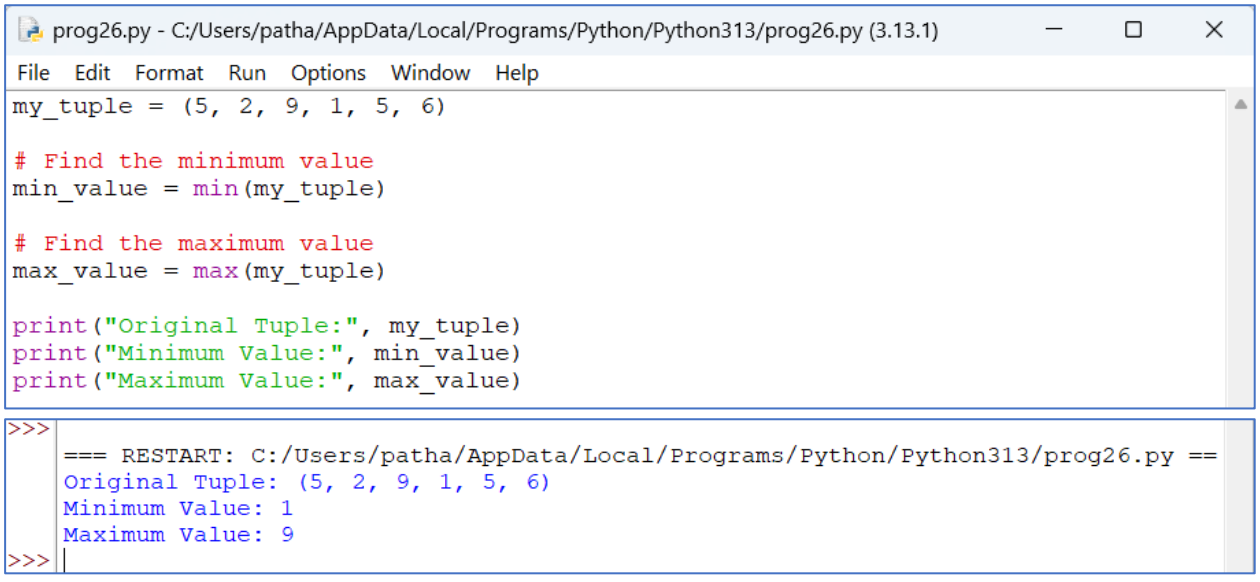
new_tuple = tuple(original tuple[i] for i in indices_to_copy):

- This is the core of the code. It's a list comprehension:
  - for i in indices_to_copy: Iterates through each index in the indices_to_copy list.
  - original_tuple[i]: Accesses the element at the current index i within the original_tuple.
- The resulting list of selected elements is then converted into a tuple using the tuple() constructor.

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

Feature	List	Tuple
Mutability	Mutable (changeable)	Immutable (unchangeable)
Syntax	[item1, item2, item3]	(item1, item2, item3)
Use Cases	Collections that require modification	Collections that should remain constant
Performance	Slightly slower, more memory overhead	Slightly faster, less memory overhead
Methods	More built-in methods for manipulation	Fewer built-in methods due to immutability

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



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

prog26.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog26.py (3.13.1)

File Edit Format Run Options Window Help

```
t=(1,1,2,3,4,5,5,5,6,7)
print("tuple=",t)
print("printing number of 5's in tuple")
count=t.count(5)
print(count)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog26.py ==
tuple= (1, 1, 2, 3, 4, 5, 5, 5, 6, 7)
printing number of 5's in tuple
3
>>>
```

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

numtoword.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/numtoword.py (3.1...

File Edit Format Run Options Window Help

```
num = int(input("Enter any number:"))
t = list()
while num > 0:
    rem = num %10
    if rem == 0:
        t.append("Zero")
    elif rem == 1:
        t.append("One")
    elif rem == 2:
        t.append("Two")
    elif rem == 3:
        t.append("Three")
    elif rem == 4:
        t.append("Four")
    elif rem == 5:
        t.append("Five")
    elif rem == 6:
        t.append("Six")
    elif rem == 7:
        t.append("Seven")
    elif rem == 8:
        t.append("Eight")
    elif rem == 9:
        t.append("Nine")
    num = num // 10

for i in range(len(t)-1,-1,-1):
    print(t[i],end=" ")
```

```
>>>
= RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/numtoword.py =
Enter any number:1234
One Two Three Four
>>>
```

```
>>>
= RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/numtoword.py =
Enter any number:65874
Six Five Eight Seven Four
>>>
```

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

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed). A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set(). It can have any number of items and they may be of different types (integer, float, tuple, string etc.).

sets.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/sets.py (3.13.1)

File Edit Format Run Options Window Help

```
# Creating a set
my_set = {1, 2, 3}
print("Created Set:", my_set)

# Accessing set elements
print("Set elements are accessed using a loop:")
for element in my_set:
    print(element)

# Updating the set
my_set.add(4)
print("Set after adding an element:", my_set)

my_set.update([5, 6, 7])
print("Set after updating with multiple elements:", my_set)

# Deleting from the set
my_set.remove(1)  # remove() will raise an error if the element is not present
print("Set after removing an element:", my_set)

my_set.discard(2)  # discard() will not raise an error if the element is not pre
print("Set after discarding an element:", my_set)

# Deleting the set
del my_set
try:
    print(my_set)
except NameError as e:
    print(f"Error: {e} (Set is deleted)")
```

```
>>>
==== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/sets.py ===
Created Set: {1, 2, 3}
Set elements are accessed using a loop:
1
2
3
Set after adding an element: {1, 2, 3, 4}
Set after updating with multiple elements: {1, 2, 3, 4, 5, 6, 7}
Set after removing an element: {2, 3, 4, 5, 6, 7}
Set after discarding an element: {3, 4, 5, 6, 7}
Error: name 'my_set' is not defined (Set is deleted)
>>>
```

1. Describe the various set operations.

Here is a description of the set operations available in Python:

- **Union:** Combines elements from two sets, removing duplicates. It can be performed using the | operator or the union() method.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1 | set2 # or set1.union(set2)
# union_set will be {1, 2, 3, 4, 5}
```

- **Intersection:** Returns common elements present in both sets. It can be performed using the & operator or the intersection() method.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1 & set2 # or set1.intersection(set2)
# intersection_set will be {3}
```

- **Difference:** Returns elements present in the first set but not in the second set. It can be performed using the - operator or the difference() method.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1 - set2 # or set1.difference(set2)
# difference_set will be {1, 2}
```

- **Symmetric Difference:** Returns elements present in either the first set or the second set, but not in both. It can be performed using the ^ operator or the symmetric_difference() method.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
symmetric_difference_set = set1 ^ set2
# symmetric_difference_set will be {1, 2, 4, 5}
```

- **isdisjoint():** Checks if two sets have no elements in common. Returns True if they are disjoint, False otherwise.

```
set1 = {1, 2, 3}
set2 = {4, 5, 6}
disjoint_check = set1.isdisjoint(set2)
# disjoint_check will be True
```

These set operations offer efficient ways to manipulate and analyze data stored in sets, leveraging their inherent properties of uniqueness and unordered nature.

2. Describe the various methods of set.

- **add(element):** Adds a single element to the set. If the element already exists, it has no effect.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

- **update(iterable):** Adds multiple elements from an iterable (like a list, tuple, or another set) to the set.

```
my_set = {1, 2, 3}
my_set.update([4, 5], {6, 7})
print(my_set)  # Output: {1, 2, 3, 4, 5, 6, 7}
```

- **remove(element):** Removes a specific element from the set. Raises a KeyError if the element is not found.

```
my_set = {1, 2, 3, 4}
my_set.remove(2)
print(my_set)  # Output: {1, 3, 4}
```

- **discard(element):** Removes a specific element from the set. If the element is not found, it does nothing.

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

```
my_set.discard(5)  # No error raised
print(my_set)  # Output: {1, 2, 3, 4}
```

- **pop():** Removes and returns an arbitrary element from the set.

```
my_set = {1, 2, 3}
removed_element = my_set.pop()
print(my_set)  # Output: {1, 2} or {2, 3} (order is not guaranteed)
print(removed_element)
```

- **clear():** Removes all elements from the set.

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set)  # Output: set()
```

- **union(other_set):** Returns a new set containing all elements from both sets.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5}
```

- **intersection(other_set):** [1] Returns a new set containing only the elements

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {3}
```

- **difference(other_set):** Returns a new set containing elements in the first set but not in the second.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1.difference(set2)
print(difference_set)  # Output: {1, 2}
```

- **symmetric_difference(other_set):** Returns a new set containing elements in either set, but not in both.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)  # Output: {1, 2, 4, 5}
```

- **issubset(other_set):** Checks if the set is a subset of another set.

```
set1 = {1, 2}
set2 = {1, 2, 3}
print(set1.issubset(set2))  # Output: True
```

- **issuperset(other_set):** Checks if the set is a superset of another set.

```
set1 = {1, 2, 3}
set2 = {1, 2}
print(set1.issuperset(set2))  # Output: True
```

- **isdisjoint(other_set):** Checks if the set has no elements in common with another set.

```
set1 = {1, 2}
set2 = {3, 4}
print(set1.isdisjoint(set2))  # Output: True
```

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

prog27.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog27.py (3.13.1)

File Edit Format Run Options Window Help

```
# Create an empty set
my_set = set()

# Add members to the set
my_set.add(1)
my_set.add(2)
my_set.add(3)
my_set.add(2)

print("Set after adding elements:", my_set)

# Add multiple members at once using update()
my_set.update({4, 5, 6})
print("Set after updating elements :", my_set)

# Remove an item from the set using remove()
my_set.remove(5)
print("Set after removing 5:", my_set)

# Remove an item using discard()
my_set.discard(3)
print("Set after discarding 3:", my_set)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog27.py ==
Set after adding elements: {1, 2, 3}
Set after updating elements : {1, 2, 3, 4, 5, 6}
Set after removing 5: {1, 2, 3, 4, 6}
Set after discarding 3: {1, 2, 4, 6}
>>>
```

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

prog28.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog28.py (3.13.1)

File Edit Format Run Options Window Help

```
s1={1,2,3,4,5}
print("set=",s1)
print("Maximum value=",max(s1))
print("Minimum value=",min(s1))
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog28.py ==
set= {1, 2, 3, 4, 5}
Maximum value= 5
Minimum value= 1
>>>
```

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

prog28.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prog28.py (3.13.1)

File Edit Format Run Options Window Help

```
set={0,1,2,3,4,5}
print("set=",set)
print("length=",len(set))
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog28.py ==
set= {0, 1, 2, 3, 4, 5}
length= 6
>>>
```

Practical No. 11: Write python program to perform following functions on Set: Union, Intersection, Difference, Symmetric Difference

1. Explain mutable and immutable data structures.

Mutable data structures allow modification of their contents after they are created. Common mutable data structures in Python include:

- **Lists:** Ordered collections of items, allowing addition, removal, and modification of elements.
- **Dictionaries:** Collections of key-value pairs, where values can be modified, added, or removed.
- **Sets:** Unordered collections of unique elements, allowing addition and removal of items.

Immutable data structures, do not allow modification after creation. Any operation that appears to modify an immutable data structure actually creates a new object with the modified value. Common immutable data structures in Python include:

- **Strings:** Sequences of characters that cannot be changed after creation.
- **Tuples:** Ordered collections of items, similar to lists, but immutable.
- **Numeric types (int, float, bool):** Represent numerical values and boolean values that cannot be changed.

Mutability affects how data structures are handled in memory and how they behave when passed as arguments to functions or assigned to new variables. Mutable objects can be modified in place, affecting all references to that object, while immutable objects, when "modified," result in a new object, leaving the original unchanged.

2. Explain any six set function with example.

Explain in above Practical

3. Write a Python program to perform following operations on set: intersection of sets

Explain in above Practical

4. union of sets, set difference, symmetric difference, clear a set.

Explain in above Practical

5. Difference between set & tuple

Feature	Set	Tuple
Mutability	Mutable (items can be added or removed)	Immutable (items cannot be changed after creation)
Ordering	Unordered (items have no specific position)	Ordered (items maintain their position)
Duplicates	No duplicates allowed	Duplicates allowed
Syntax	Enclosed in curly braces {}	Enclosed in parentheses ()

Practical No. 12: Write python program to perform following operations on the Dictionary: Create, Access, Update, Delete, Looping through Dictionary, Create Dictionary from list

Python dictionary is a container of key-value pairs. It is mutable and can contain mixed types. A dictionary is an unordered collection. Python dictionaries are called associative arrays or hash tables in other languages. The keys in a dictionary must be immutable objects like strings or numbers. They must also be unique within a dictionary.

dictionary.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dictionary.py (3.13.1)

File Edit Format Run Options Window Help

```
# Create a dictionary
my_dict = {"name": "Alice", "age": 30, "city": "New York"}

# Access elements
print("Name:", my_dict["name"])
print("Age:", my_dict["age"])
print("City:", my_dict["city"])

# Update elements
my_dict["age"] = 31
my_dict["city"] = "Los Angeles"
print("Updated dictionary:", my_dict)

# Add new element
my_dict["occupation"] = "Engineer"
print("Dictionary with new element:", my_dict)

# Delete element
del my_dict["city"]
print("Dictionary after deleting 'city':", my_dict)

# Delete all elements
my_dict.clear()
print("Dictionary after clearing:", my_dict)

# Delete the dictionary
del my_dict
try:
    print(my_dict)
except NameError as e:
    print(f"Error: {e}")
# Looping through a dictionary
my_dict = {"a": 1, "b": 2, "c": 3}
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")

# Creating a dictionary from a list (using zip for key-value pairs)
keys = ["x", "y", "z"]
values = [10, 20, 30]
new_dict = dict(zip(keys, values))
print(new_dict)

# Creating a dictionary from a list (all values the same)
keys2 = ["p", "q", "r"]
new_dict2 = dict.fromkeys(keys2, 0)
print(new_dict2)
```

```
Name: Alice
Age: 30
City: New York
Updated dictionary: {'name': 'Alice', 'age': 31, 'city': 'Los Angeles'}
Dictionary with new element: {'name': 'Alice', 'age': 31, 'city': 'Los Angeles',
 'occupation': 'Engineer'}
Dictionary after deleting 'city': {'name': 'Alice', 'age': 31, 'occupation': 'En
gineer'}
Dictionary after clearing: {}
Error: name 'my_dict' is not defined
Key: a, Value: 1
Key: b, Value: 2
Key: c, Value: 3
{'x': 10, 'y': 20, 'z': 30}
{'p': 0, 'q': 0, 'r': 0}
>>>
```

1. Write a Python script to sort (ascending and descending) a dictionary by value.

dictsort.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dictsort.py (3.13.1)

File Edit Format Run Options Window Help

```
# Key, Value of the dictionary defined
d = {'watermelon': 5, 'apple': 12, 'banana': 3}
#d = {5:'Five',2:'Two',10:'Ten',4:'Four',1:'One'}
print('Dictionary is :',d)
# Sort based on Values in asceding order
val_based = {k: v for k, v in sorted(d.items(), key=lambda item: item[1])}
# item[1] represents the sorting based on value

# Sort based on on Values in descending order
val_based_rev = {k: v for k, v in sorted(d.items(), key=lambda item: item[1], re

# Print sorted dictionary
print("Ascending order :",val_based)
print("Descending order :",val_based_rev)
```

```
>>>
== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dictsort.py =
Dictionary is : {'watermelon': 5, 'apple': 12, 'banana': 3}
Ascending order : {'banana': 3, 'watermelon': 5, 'apple': 12}
Descending order : {'apple': 12, 'watermelon': 5, 'banana': 3}
>>>
```

```
>>>
== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dictsort.py =
Dictionary is : {5: 'Five', 2: 'Two', 10: 'Ten', 4: 'Four', 1: 'One'}
Ascending order : {5: 'Five', 4: 'Four', 1: 'One', 10: 'Ten', 2: 'Two'}
Descending order : {2: 'Two', 10: 'Ten', 1: 'One', 4: 'Four', 5: 'Five'}
>>>
```

https://sparkbyexamples.com/python/how-to-sort-dictionary-by-value-in-python/

2. Write a Python script to concatenate following dictionaries to create a new one.

Sample Dictionary: dic1 = {1:10, 2:20} dic2 = {3:30, 4:40} dic3 = {5:50,6:60}

dices.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py (3.13.1)

File Edit Format Run Options Window Help

```
dic1 = {1:10, 2:20}
dic2 = {3:30, 4:40}
dic3 = {5:50, 6:60}

dic4 = {}

print("dic1=",dic1,"\ndic2=",dic2,"\ndic3=",dic3,)
for d in (dic1,dic2,dic3):
    dic4.update(d)

print("dic4",dic4)
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
dic1= {1: 10, 2: 20}
dic2= {3: 30, 4: 40}
dic3= {5: 50, 6: 60}
dic4 {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
>>>
```

https://www.w3resource.com/python-exercises/dictionary/python-data-type-dictionary-exercise-19.php

3. Write a Python program to perform following operations on set: intersection of sets, union of sets, set difference, symmetric difference, clear a set.

See in SET practical

4. Write a Python program to combine two dictionary adding values for common keys.
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}

dices.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py (3.13.1)

File Edit Format Run Options Window Help

```
d1 = {'a': 100, 'b': 200, 'c': 300}
d2 = {'a': 300, 'b': 200, 'd': 400}

# Combine the dictionaries
result = {}
print("d1 :",d1,"\nd2 :",d2)
# Add values for common keys from both dictionaries
for key in d1.keys() | d2.keys():  # Union of keys from both dictionaries
    result[key] = d1.get(key, 0) + d2.get(key, 0)

print("Result :",result)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
d1 : {'a': 100, 'b': 200, 'c': 300}
d2 : {'a': 300, 'b': 200, 'd': 400}
Result : {'a': 400, 'd': 400, 'b': 400, 'c': 300}
>>>
```

6. Write a Python program to find the highest 3 values in a dictionary.

dices.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py (3.13.1)

File Edit Format Run Options Window Help

```
d = {'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
print("d :",d)
# Sort the dictionary by value and get the top 3 items (key-value pairs)
top_3_items = sorted(d.items(), key=lambda item: item[1], reverse=True)[:3]

# Print the highest 3 key-value pairs
print("The highest 3 key-value pairs are:", top_3_items)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
d : {'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
The highest 3 key-value pairs are: [('e', 500), ('d', 400), ('c', 300)]
>>>
```

OR : Write a Python program to find the Largest of 3 values in a dictionary.

dices.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py (3.13.1)

File Edit Format Run Options Window Help

```
# Sample dictionary
d = {'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}

print("d :",d)

# Find the largest of 3 values
largest_value = max(d.values())

# Print the largest value
print("The largest value is:", largest_value)
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
d : {'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
The largest value is: 500
>>>
```

6. What is the output of the following program?

dictionary1 = {'Google' : 1, 'Facebook' : 2, 'Microsoft' : 3}

dictionary2 = {'GFG' : 1, 'Microsoft' : 2, 'Youtube' : 3 }

dictionary1.update(dictionary2)

for key, values in dictionary1.items(): print(key, values)

Output :

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
Google 1
Facebook 2
Microsoft 2
GFG 1
Youtube 3
>>>
```

Practical No. 13: Write a user define function to implement following features: Function without argument, Function with argument, Function returning value

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.

dices.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py (3.13.1)

File Edit Format Run Options Window Help

```
# Function without argument
def greet():
    print("Hello, welcome to the Python world!")

# Function with argument
def square(number):
    return number * number

# Function returning value
def add_numbers(a, b):
    return a + b

# Calling the functions
greet()  # Function without argument

num = 5
result_square = square(num)  # Function with argument
print(f"The square of {num} is {result_square}")

x, y = 10, 20
result_sum = add_numbers(x, y)  # Function returning value
print(f"The sum of {x} and {y} is {result_sum}")
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/dices.py ===
Hello, welcome to the Python world!
The square of 5 is 25
The sum of 10 and 20 is 30
>>>
```

1. Write a Python function that takes a number as a parameter and check the number is prime or not. For more practice : https://www.w3resource.com/python-exercises/

prime.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/prime.py (3.13.1)

File Edit Format Run Options Window Help

```
def is_prime(n):
    if n <= 1:
        return False
    factors = 0
    for i in range(1, n + 1):
        if n % i == 0:
            factors += 1
        if factors > 2:
            return False
    return True

num = int(input("Enter a number: "))
is_prime_num = is_prime(num)
print("Is", num, "prime?", is_prime_num)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prime.py ===
Enter a number: 10
Is 10 prime? False
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prime.py ===
Enter a number: 11
Is 11 prime? True
>>>
```

2. What is the output of the following program?

```
def myfunc(text, num):
    while num > 0:
    print(text)
    num = num - 1
myfunc('Hello', 4)
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/funct1.py ==
Hello
Hello
Hello
Hello
>>>
```

3. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

factorial.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/factorial.py (3.13.1)

File Edit Format Run Options Window Help

```
def factorial(n):
    product = 1
    for i in range(1, n + 1):
        product *= i
    return product

n = int(input("Enter a number: "))
fact = factorial(n)
print("The factorial of", n, "is", fact)
```

```
= RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/factorial.py =
Enter a number: 5
The factorial of 5 is 120
>>>
```

4. Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

uplow.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/uplow.py (3.13.1)

File Edit Format Run Options Window Help

```
def count_upper_lower(text):

    upper_count = 0
    lower_count = 0

    for char in text:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1

    return upper_count, lower_count

text = input("Enter any text :")
upper,lower = count_upper_lower(text)
print("Upper case letters :",upper)
print("Lower case letters :",lower)
```

```
>>>
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/uplow.py ===
Enter any text :Jamia Polytechnic Akkalkuwa
Upper case letters : 3
Lower case letters : 22
>>>
```

5. What is the output of the following program?

```
num = 1
def func():
        num = 3
        print(num)
func()
print(num)
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/uplow.py ===
3
1
>>>
```

Practical No. 14: Write a user define function to implement for following problem: Function positional/required argument, Function with keyword argument, Function with default argument, Function with variable length argument

Function with Positional/Required Arguments

Positional arguments are those that must be passed in the correct position when calling the function.

```
    def greet(name, age):
        print(f"Hello {name}, you are {age} years old.")
    # Calling the function
    greet("Aziz", 25)
O/p: Hello Aziz, you are 25 years old
```

Function with Keyword Arguments

Keyword arguments are those where you specify the argument by name, rather than position. This can make the code clearer.

```
    def greet(name, age):
        print(f"Hello {name}, you are {age} years old.")
    # Calling the function using keyword arguments
    greet(age=30, name="Asif")
O/p: Hello Asif, you are 30 years old.
```

Function with Default Arguments

Default arguments are values that the function uses if no value is provided for them when the function is called.

```
    def greet(name, age=30):
        print(f"Hello {name}, you are {age} years old.")
    # Calling the function with one argument
    greet("Afzal ")  # age will default to 30
    # Calling the function with both arguments
    greet("Khan ", 40)
O/p: Hello Afzal, you are 30 years old.
    Hello Khan, you are 40 years old
```

Function with Variable Length Arguments

A function can accept a variable number of arguments using *args for positional arguments and **kwargs for keyword arguments.

```
    # Function with variable-length positional arguments (*args)
    def sum_numbers(*args):
        total = sum(args)
    print(f"The sum is: {total}")
    # Calling the function with different number of arguments
    sum_numbers(1, 2, 3)
    sum_numbers(4, 5, 6, 7, 8)
    # Function with variable-length keyword arguments (**kwargs)
    def display_info(**kwargs):
        for key, value in kwargs.items():
          print(f"{key}: {value}")
```

```
    # Calling the function with different keyword arguments
    display_info(name="Akmal", age=22, city="New York")
O/p: The sum is: 6
    The sum is: 30
    name: Akmal
    age: 22
    city: New York
```

1. What are positional arguments?

Positional arguments in Python are the arguments that are passed to a function in a specific order. The values are assigned to parameters based on their position in the function call. The first argument is assigned to the first parameter, the second argument to the second parameter, and so on.

2. Write program to demonstrate use of variable length arguments.

funargu1.py - C:/Users/patha/AppData/Local/Programs/Python/Python313/funargu1.py (3.13.1)

File Edit Format Run Options Window Help

```
# Function with variable-length positional arguments (*args)
def print_numbers(*args):
    print("The numbers are:")
    for num in args:
        print(num)

# Calling the function with different number of arguments
print_numbers(1, 2, 3)
print_numbers(10, 20, 30, 40, 50)

# Function with variable-length keyword arguments (**kwargs)
def display_info(**kwargs):
    print("User Information:")
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Calling the function with different keyword arguments
display_info(name="John", age=28, city="New York")
display_info(name="Alice", country="Canada", profession="Engineer", age=30)
```

```
The numbers are:
1
2
3
The numbers are:
10
20
30
40
50
User Information:
name: John
age: 28
city: New York
User Information:
name: Alice
country: Canada
profession: Engineer
age: 30
>>>
```

3. What is a Function Argument in Python?

In Python, a **function argument** is a value you pass into a function when you call it. These values are assigned to parameters in the function and are used inside the function.

Types:

- **Positional Arguments**: Passed in a specific order.
- **Keyword Arguments**: Passed with the argument name, order doesn't matter.
- **Default Arguments**: Parameters with a default value, used if no value is provided.
- **Variable-Length Arguments**: Allow you to pass a variable number of arguments using *args (for positional) or **kwargs (for keyword).

4. Explain function with keyword argument.

A **function with keyword arguments** in Python allows you to pass arguments to a function by explicitly specifying the parameter names. This means the order of the arguments doesn't matter, as long as you provide the correct name for each argument.

```
    def greet(name, age):
        print(f"Hello {name}, you are {age} years old.")
    greet(age=30, name="Sameer")
O/p: Hello Sameer, you are 30 years old.
```

5. What will be the output of following code?

```
def print_animals(*animals):
    for animal in animals:
        print(animal)
print_animals("Lion", "Elephant", "Wolf", "Gorilla")
```

```
Lion
Elephant
Wolf
Gorilla
>>>
```

6. What will be the output of following code

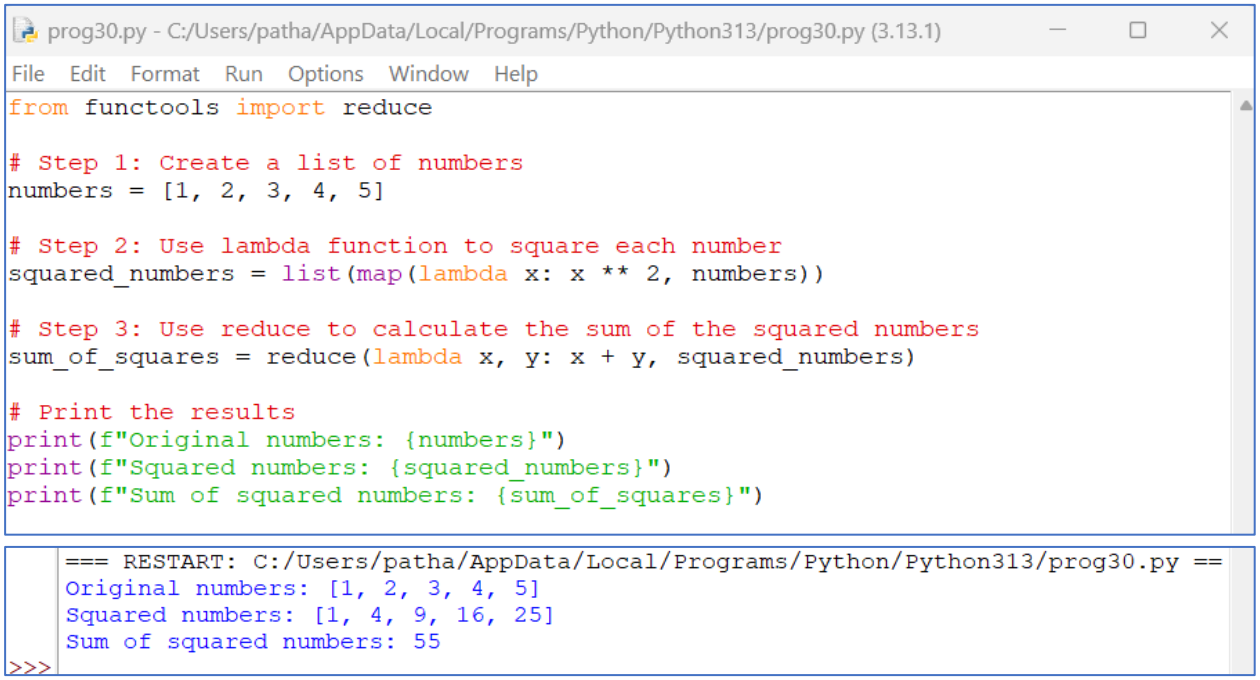
```
def print_food(**foods):
    for food in foods.items():
        print(food)
print_food(Lion="Carnivore", Elephant="Herbivore", Gorilla="Omnivore")
```

```
('Lion', 'Carnivore')
('Elephant', 'Herbivore')
('Gorilla', 'Omnivore')
>>>
```

Practical No. 15: Write Python program to demonstrate use of following advanced functions: lambda, map, reduce

Python is a dynamic yet straightforward typed language, and it provides multiple libraries and inbuilt functions. There are different methods to perform the same task and in Python lambda function proves to be king of all, which can be used everywhere in different ways.

- **lambda** is use to perform a quick operation (e.g., square each number).
- **map** is use to apply this operation to each element in the list.
- **reduce** is use to calculate the sum of all squared numbers.



1. What does the filter() function in Python do?

The filter() function in Python is used to filter elements from an iterable (like a list or a tuple) based on a condition specified by a function. It returns an iterator that contains only the elements for which the function returns True.

Syntax : filter(function, iterable)

```
    # List of numbers
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    # Use filter() to get only even numbers
    even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
    # Print the result
    print(even_numbers)
O/p: [2, 4, 6, 8]
```

2. What is the purpose of the map() function in Python?

The map() function in Python applies a given function to all items in an iterable (like a list) and returns a map object (an iterator) that yields the results. It's useful when you want to perform a transformation or operation on each element of an iterable without using an explicit loop.

Syntax : map(function, iterable)

```
    # Function to square a number
    def square(x):
        return x * x
    numbers = [1, 2, 3, 4]
    # Using map() to square each number
    squared_numbers = map(square, numbers)
    # Convert the map object to a list to view the results
    print(list(squared_numbers))
O/p: [1, 4, 9, 16]
```

3. What will be the output of following code?

```
from functools import reduce
def multiply(x, y):
    return x * y

numbers = [1, 2, 3, 4, 5]
result = reduce(multiply, numbers)
print(result)
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/Prog31.py ==
120
>>>
```

4. What will be the output of following code

```
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x**2,numbers)
print(list(squared))
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog32.py ==
[1, 4, 9, 16, 25]
>>>
```

5. What will be the output of following code?

```
def even_check(num):
    if num % 2 == 0:
        return True
    else:
        return False
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = filter(even_check, numbers)
print(list(even_numbers))
```

```
=== RESTART: C:/Users/patha/AppData/Local/Programs/Python/Python313/prog32.py ==
[2, 4, 6, 8, 10]
>>>
```