RESEARCH INNOVATION DISCOVERY

RID संस्था समस्या का समाधान

Reg.No: 048884
Foundation Day
05-09-2024

RUN BY TWKSAA WELFARE FOUNDATION

Core Python

E-Book

Er. Rajesh Prasad(B.E, M.E)
Founder: RID Organization

- **RID ORGANIZATION** यानि **Research, Innovation Discovery** संस्था जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इस RID संस्था** का स्थपना किया हूँ |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

"त्वक्सा कोर पाइथन के इस ई-पुस्तक में आप पाइथन से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। । मुझे आशा है कि इस ई-पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी"

"In this E-book of TWKSAA Core Python you will learn all the basic concepts related to python. I hope after reading this book your knowledge will be improve and you will get more interest to know more thing about computer Science".

RID हमें क्यों करना चाहिए?

(Research)

अनुसंधान हमें क्यों करना चाहिए ?

Why should we do research?

1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge)
2. समस्याओं का समाधान(To Solving problems)
3. सामाजिक प्रगति (To Social progress)
4. विकास को बढ़ावा देने(To promote development)
5. तकनीकी और व्यापार में उन्नति(To advances in technology & business)
6. देश विज्ञान और प्रौद्योगिकी के विकास(To develop the country's science & technology)

(Innovation)

नवीनीकरण हमें क्यों करना चाहिए ?

Why should we do Innovation?

1. प्रगति के लिए(To progress)
2. परिवर्तन के लिए(For change)
3. उत्पादन में सुधार(To Improvement in production)
4. समाज को लाभ(To Benefit to society)
5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition)
6. देश विज्ञान और प्रौद्योगिकी के विकास। (To develop the country's science & technology)

(Discovery)

खोज हमें क्यों करना चाहिए?

Why should we do Discovery?

1. नए ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. अविष्कारों की खोज(To Discovery of inventions)
3. समस्याओं का समाधान(To Solving problems)
4. ज्ञान के विकास में योगदान(Contribution to development of knowledge)
5. समाज के उन्नति के लिए (for progress of society)
6. देश विज्ञान और तकनीक के विकास (To develop the country's science & technology)

❖ **Research(अनुसंधान):**

- अनुसंधान एक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोधोगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उचित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ **Innovation(नवीनीकरण): -**

- Innovation एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ **Discovery (आविष्कार):**

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

नोट : अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिस्टमैटिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

सुविचार:	
1.	**समस्याओं का समाधान करने का उतम मार्ग हैं |** → शिक्षा ,RID, प्रतिभा, सहयोग, एकता एवं समाजिक-कार्य
2.	**एक इंसान के लिए जरूरी हैं |** → रोटी, कपड़ा, मकान, शिक्षा, रोजगार, इज्जत और सम्मान
3.	**एक देश के लिए जरूरी हैं |** - → संस्कृति-सभ्यता, भाषा, एकता, आजादी, संविधान एवं अखंडता
4.	**सफलता पाने के लिए होना चाहिए |** → लक्ष्य, त्याग, इच्छा-शक्ति, प्रतिबद्धता, प्रतिभा, एवं सतता
5.	**मरने के बाद इंसान छोडकर जाता हैं |** → शरीर, अन-धन, घर-परिवार, नाम, कर्म एवं विचार
6.	**मरने के बाद इंसान को इस धरती पर याद किया जाता हैं उनके**
➔ नाम, काम, दान, विचार, सेवा-समपर्ण एवं कर्मो से...	

आशीर्वाद (बड़े भैया जी)

Mr. RAMASHANKAR KUMAR

मार्गदर्शन एवं सहयोग

Mr. GAUTAM KUMAR

... सोच है जिनकी नई कुछ कर दिखाने की, खोज है रीड संस्था को उन सभी इंसानों की...

"अगर आप भी **Research, Innovation and Discovery** के क्षेत्र में रूचि रखतें हैं एवं अपनी प्रतिभा से दुनियां को कुछ नया देना चाहतें एवं अपनी समस्या का समाधान **RID** के माध्यम से करना चाहतें हैं तो **RID ORGANIZATION (रीड संस्था)** से जरुर जुड़ें" || **धन्यवाद** || **Er. Rajesh Prasad (B.E, M.E)**

S. No:	Topic Name	Page No:
1	What is program and programming language? Why should we learn?	4
2.	Example of programming language	4
3	Why we should learn python?	4
4	How we can learn any programming language?	5
5	Why we should learn python?	6
6	What is python?	7
7	Which type of application we can develop by using python?	7
8	History of python	7
9	Features of python	8
10	What is compiler and interpreter?	9
11	Python version	11
12	Python execution environment	12
13	Keyword symbols with names	13
14	Number system	14
15	Reserved keywords	17
16	Identifiers	18
17	Variable	19
18	Comments	21
19	Data types	22
20	Inbuilt functions	23
21	Base conversions	25
21	Type casting	33
22	Escape characters	39
23	Constants & literals	39
24	Operator	40
25	Operator precedence	53
26	Input output statements	54
27	Control statement (if and while)	59
28	Problem based on control statement	73
29	110 Pattern program	80
30	List	95
31	List-based problem	116
32	Tuple	125
33	Set	132
34	Dictionary	144
35	String	158
36	String based problem	172
37	Data Structure	180
38	Function	185
39	Modules	205
40	Package	217
41	File Handling	223
42	Top 50 interview question and answer	238
43	What is RID?	242

Basic Information

- **Software:** A collection of integrated programs designed to perform specific tasks.
- **Program:** A set of instructions written to execute a particular operation.
- **Code:** Instructions written in a programming language.
- **Programming Language:** A system of notation used to write computer programs.

Software
- Program
  - Code
    - Programming Language
      - **Examples:** C, C++, Java, Python, etc.
  - **Purpose:** Calculation, Processing, Problem-solving

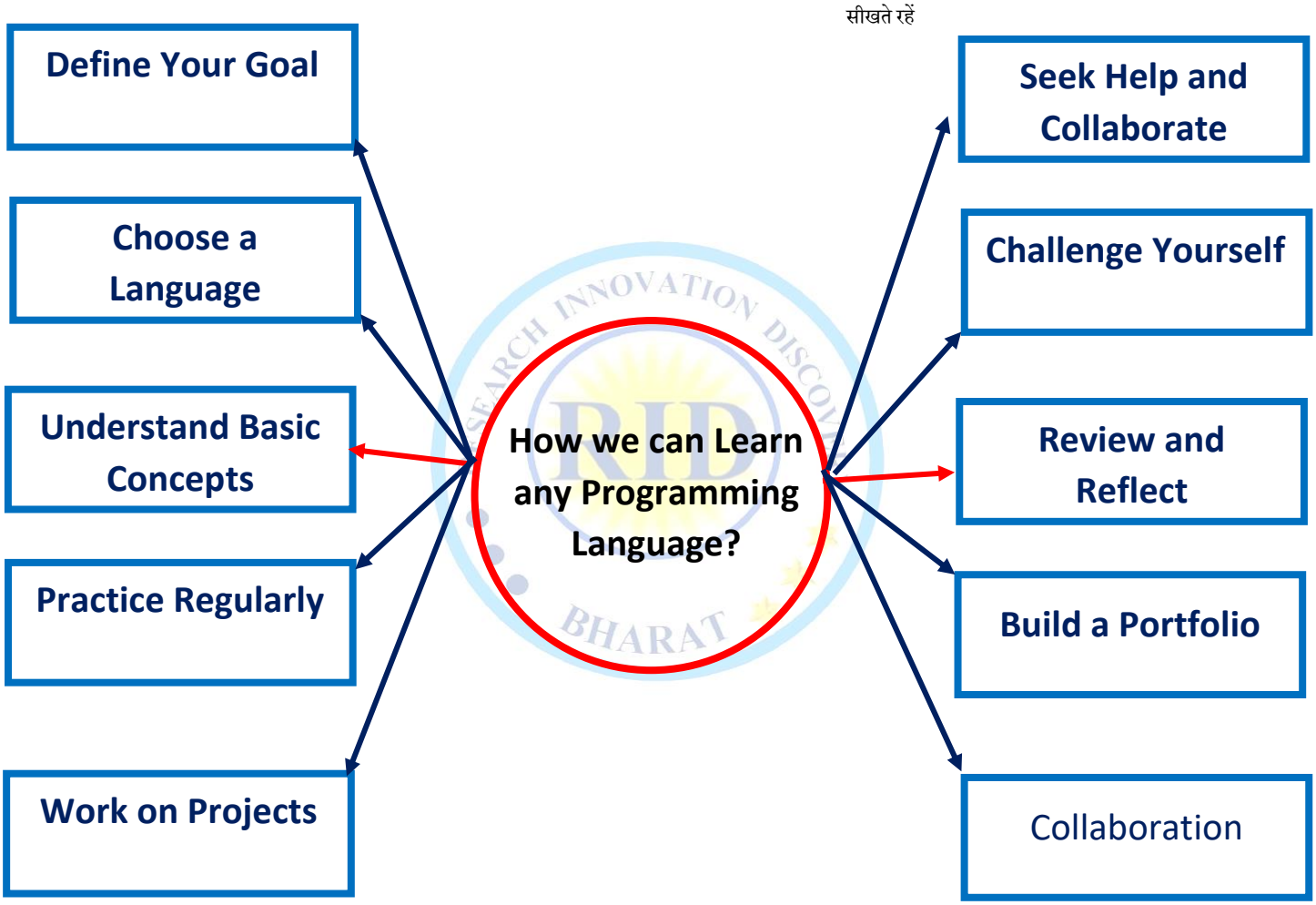
Why should we learn a programming language?

- ✓ **communicate with computers:** interacting with computers and giving them instructions.
- ✓ **Problem Solving:** Programming develops ability to solve problems and think logically.
- ✓ **Career Opportunities:** Programming skills are in high demand across industries.
- ✓ **Automation:** Programming allows you to automate tasks, making workflows more efficient.
- ✓ **Creativity:** Programming enables to bring creative ideas to life through software.
- ✓ **Technological Understanding:** Programming helps you understand and interact with technology effectively.
- ✓ **Personal Projects:** Programming allows you to create personal projects and tools.
- ✓ **Collaboration:** Programming facilitates collaboration with others on software development.
- ✓ **Critical Thinking:** Programming hones your critical thinking and debugging skills.
- ✓ **Future-Proofing:** As technology advances, programming skills become increasingly essential for everyday tasks and career success. (भविष्य सुरक्षित बनाना" या "भविष्य के लिए तैयार करना")
- ✓ **Data Analysis:** It refers to the process of examining and interpreting data to extract meaningful insights, patterns, and information from it

❖ How we can Learn any Programming Language?

- Define Your Goal
- Choose a Language
- Find Learning Resources
- Understand Basic Concepts
- Practice Regularly
- Work on Projects
- Seek Help and Collaborate
- Challenge Yourself
- Review and Reflect & Build a Portfolio and Keep Learning

- अपने लक्ष्य परिभाषित करें
- एक भाषा चुनें
- सीखने के संसाधन खोजें
- बुनियादी अवधारणाओं को समझें
- नियमित रूप से अभ्यास करें
- परियोजनाओं पर काम करें
- सहायता लें और सहयोग करें
- आपने आप को चुनौती दो
- समीक्षा करें और चिंतन करें तथा एक पोर्टफोलियो बनाएं और सीखते रहें



1. **Define Your Goal:**
   - Clarify why you want to learn a programming language to set a clear direction for your learning journey.
2. **Choose a Language:**
   - Select a language based on your goal and preferences; popular choices for beginners include Python, JavaScript, or Ruby.
3. **Find Learning Resources:**
   - Explore online courses, tutorials, and books to gather resources that suit your learning style and pace.
4. **Understand Basic Concepts:**

- Grasp fundamental programming concepts such as variables, loops, and conditionals to build a solid foundation.

5. **Practice Regularly:**
   - Consistently write code to reinforce your understanding and improve your programming skills.
6. **Work on Projects:**
   - Apply your knowledge by working on real-world projects to gain practical experience and showcase your abilities.
7. **Seek Help and Collaborate:**
   - Engage with coding communities, forums, and collaborate with others to learn from different perspectives and solve challenges.
8. **Challenge Yourself:**
   - Tackle progressively more complex problems and projects to push your boundaries and enhance your problem-solving skills.
9. **Review and Reflect:**
   - Regularly review your code, analyze your mistakes, and reflect on your progress to identify areas for improvement.
10. **Build a Portfolio:**
    - Showcase your projects in a portfolio to demonstrate your skills and make a compelling case to potential employers or collaborators.
11. **Keep Learning:**
    - Embrace a mindset of continuous learning to stay updated with new technologies and advancements in the programming world.

❖ Why we Should Learn Python?

❖ Learning Python several simple and compelling reasons.

- ✓ **Easy to Learn:** Python's simple and readable syntax makes it beginner-friendly.
- ✓ **Versatile:** Python can be used for web development, data analysis, machine learning, automation, and more.
- ✓ **Rich Ecosystem:** Python's vast library of modules and packages simplifies coding tasks.
- ✓ **Community Support:** Python has an active and supportive community.
- ✓ **Rapid Prototyping:** Python's quick development cycle allows you to experiment, test ideas, and build functional prototypes efficiently.
- ✓ **Future-Proof:** As technology advances, Python's relevance is expected to grow.
- ✓ **Scripting and Automation:** Python's scripting capabilities make it ideal for automating repetitive tasks, saving time and effort.
- ✓ **Entry Point to Programming:** Learning Python can serve as a stepping stone to understanding other programming languages and concepts.
- ✓ **Educational Value:** Python is widely used in educational settings.
- ✓ **High Demand:** Python is in high demand in the job market.

WHAT IS PYTHON?

- Python is a **General Purpose**, **high-level, Object-Oriented**, **interpreted**, **versatile** and **dynamically typed** programming language. (पायथन एक सामान्य प्रयोजन, उच्च-स्तरीय, ऑब्जेक्ट-ओरिएंटेड, व्याख्या की गई, बहुमुखी और गतिशील रूप से टाइप की गई प्रोग्रामिंग भाषा है।)
- Python is **case sensitive** programming language.

❖ Which Type of Application we can develop by using Python?

1) Web Applications
2) Data Analysis and Visualization
3) Machine Learning and AI
4) Scientific Computing
5) Desktop Applications
6) Automation and Scripting
7) Game Development
8) Mobile Applications
9) Networking and Network Programming
10) IoT (Internet of Things) Applications
11) Image Processing and Computer Vision
12) Natural Language Processing (NLP)

1) वेब अनुप्रयोग
2) डेटा विश्लेषण और विज़ुअलाइज़ेशन
3) मशीन लर्निंग और एआई
4) वैज्ञानिक कंप्यूटिंग
5) डेस्कटॉप एप्लीकेशन
6) स्वचालन और स्क्रिप्टिंग
7) खेल विकास
8) मोबाइल एप्लीकेशन
9) नेटवर्किंग और नेटवर्क प्रोग्रामिंग
10) IoT (इंटरनेट ऑफ थिंग्स) अनुप्रयोग
11) इमेज प्रोसेसिंग और कंप्यूटर विज़न
12) प्राकृतिक भाषा प्रसंस्करण (एनएलपी)

1) **Web Applications:** Python web frameworks like Django, Flask, and Pyramid.
2) **Data Analysis and Visualization:** Python's libraries like Pandas, NumPy, and Matplotlib allow for data manipulation, analysis, and visualization.
3) **Machine Learning and AI:** Python is used in ML & AI applications with libraries like TensorFlow, PyTorch, and scikit-learn.
4) **Scientific Computing:** Python, along with libraries like SciPy, is used for mathematical and scientific computations.
5) **Desktop Applications:** Python is used to build desktop applications with (GUI) using libraries like Tkinter, PyQt, and wxPython.
6) **Automation and Scripting:** Python's make for automating repetitive tasks and scripting.
7) **Game Development:** Python can be used for game development.
8) **Mobile Applications:** Python can be used to build mobile applications using frameworks like Kivy or through hybrid app development tools like BeeWare.
9) **Networking & Network Programming:** it is suitable for developing networked applications.
10) **IoT (Internet of Things) Applications:** Python can be used to interact with hardware and develop applications for IoT devices.
11) **Image Processing and Computer Vision:** Python's libraries like OpenCV are commonly used for image processing and computer vision tasks.

❖ **Natural Language Processing (NLP):** Python has libraries like NLTK and spaCy

❖ History of Python:

- Guido van Rossum developed Python while working at the Centrum Wiskunde & Informatica (CWI), **a research center for mathematics and computer science** located in the Netherlands. during his Christmas holidays in December 1989. He named it "Python" after being inspired by the British comedy television show "Monty Python's Flying Circus."
- its first implementation, Python 0.9.0, was released on February 20, 1991.

FEATURES OF PYTHON

1) Simple and easy to learn
2) Free and Open Source
3) Dynamic Type Programming Language
4) Platform independent Programming Language
5) Interpreted Programming Language
6) High level Programming Language
7) Procedural and Object-Oriented
8) Robust
9) Portable
10) Extensible Programming Language
11) Embedded Programming Language
12) Support third party API'S

1) सरल और सीखने में आसान
2) मुफ़्त और खुला स्रोत
3) डायनामिक टाइप प्रोग्रामिंग लैंग्वेज
4) प्लेटफार्म स्वतंत्र प्रोग्रामिंग भाषा
5) व्याख्या की गई प्रोग्रामिंग भाषा
6) उच्च स्तरीय प्रोग्रामिंग भाषा
7) प्रक्रियात्मक और वस्तु-उन्मुख
8) मजबूत
9) पोर्टेबल
10) एक्स्टेंसिबल प्रोग्रामिंग लैंग्वेज
11) एंबेडेड प्रोग्रामिंग लैंग्वेज
12) तृतीय पक्ष एपीआई का समर्थन करें

1) Simple and easy to learn:

- Python is a simple Programming Language when we read python program, we will feel like reading English statement.

❖ **Python is Simple and easy to learn because of these three reasons.**

1. Python Provide rich set of Package, Module and Library.
   - **Function:** a group of lines with some name is called a function
   - **Module:** a group of function saved to a file, is called module
   - **Library:** a group of modules is called Library.
2. Python provide in built facility called garbage collector.

- Garbage collector is one of software component in python software which is running in the background of regular python program whose role is too called remove un-used memory space.

➢ **Why python provide garbage collector?**

- It's collected un-used memory space and improve the performance of python-based application.

3. Python provides developer friendly syntaxes.

2) Free and Open Source:

- We can use python without any licence and it is free.
- Its source code is open so that we can customized based on our requirements.

Note:

- The father of python Guido van Rossum developed CPYTHON
- This CPYTHON is customised by many companies called "Python Distribution"

❖ **There are lots of python distribution**

Example:

- **Jython or Jpython:** used for running java-based application
- **Iron Python or Ipython:** used for running C#, .Net application.
- **Micro Python:** used for developing Microntroller
- **Anaconda python:** used for running bigdata, Hadoop application.
- **Ruby Python:** used for running ruby-based application.
- Pypy etc…

3) Dynamic Type Programming Language:

- In python not required to declare type for variables. Whenever we are assigning the value, based on value type will be allocated automatically, so that python is Known as dynamically typed programming language.
- But in C, Java and other programming language are Statically typed language because we have to provide type for variable at the beginning only.
- This dynamic typing nature will provide more flexibility to the programmer.

4) Platform independent Programming Language:

- Once we write a python program, it can run on any platform without rewriting once again.
- Internally PVM is responsible to convert into machine understandable form.
- Python is one of the platforms independent languages because in python programming execution environment all values are stored in the form of "object".
- All the object can store unlimited number of values and they will not depend on any OS.

Note:

- Platform "means" where software is running, Example any types of operating system.
- Data Types: it is allocating memory space to our inputs.
- **More example** of platform independent programming language: java, c#, go, ruby, swift, Rust, Scala, Kotlin etc.

➢ **Platform Dependent:**

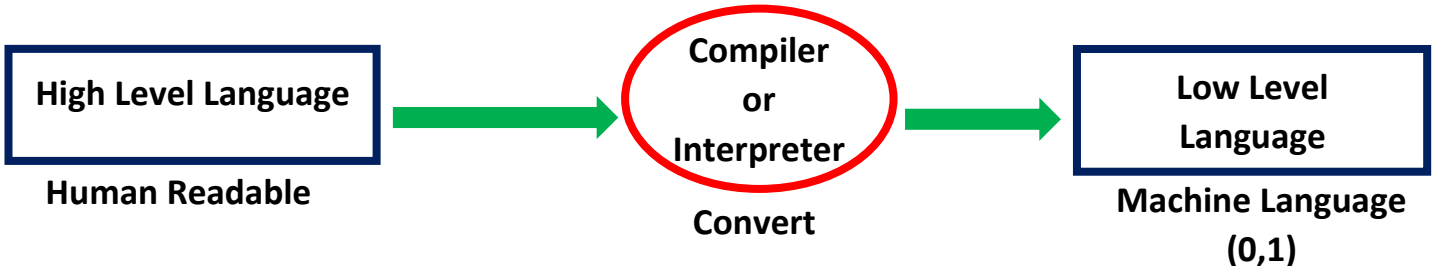
- whose corresponding data types varying their memory spaces from one operating system to another operating system, therefore there are platform dependent. **Example:** C, C++
- As java data types same memory space of all types of Operating System and hence java is platform independent language.

5) Interpreted Programming Language.

- where the source code is executed line by line by an interpreter at runtime is known as interpreted programming language.
- We are not required to compile python programs explicitly. Internally python interpreter will take care that compilation.
- If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python virtual Machine) is responsible to execute.

➢ **Compiled Programming language:** Mean's where the source code is translated into machine code or an intermediated code by a compiler before execution. This compilation process creates an executable file which can be run independently of the original source code.

❖ What is compiler and interpreter?



❖ Compiler:

- A compiler takes the entire program in one go.
- Compiler generates an intermediate object code.
- High memory required.
- Error is displayed after entire program is checked
- Computer is used by C, C++, C#, JAVA etc.

❖ Interpreter:

- An interpreter takes single line of code at a time.
- interpreter never produces any intermediate object code
- Interpreter less memory required
- Errors are displayed line by line
- Interpreter used by python, php Ruby etc.

6) High level Programming Language:

- A high-level programming language is a type of programming language that is designed to be more human-readable and user-friendly.

7) Procedural and Object-Oriented Programming Language:

- Python is a programming language that allows to write code using step-by-step instructions (**procedural)** or by creating reusable objects with specific abilities **(object-oriented).**
- Python Supports both procedure oriented (like c, pascal etc) and object oriented (like C++, Java) features. Hence, we can get benefits of both like security and reusability etc.

➢ Procedure Oriented programming:

- Procedure-oriented programming is like giving a computer a list of tasks to do step by step.
- It is a programming paradigm that involves breaking down a program into smaller, self-contained procedures or functions, each responsible for performing a specific task or action.
- Its focus is on sequence of steps or procedures that need to be executed to solve a problem.

❖ Characteristics of procedural Oriented programming:

- **Functions:** Programs are divided into smaller functions each handling a specific task.
- **Sequential Execution:** Code is executed in a top-down, step-by-step manner, following the order in which it's written.
- **Global Data:** Data is often shared among functions, which can lead to potential issues with data integrity.
- **Modularity:** Emphasizes breaking down complex problems into smaller, manageable parts.
- **Reusability:** Functions can be reused, but the focus is less on creating reusable objects.
- **Less Complex:** Well-suited for simpler programs where step-by-step instructions are sufficient.
- **Limited Encapsulation:** Limited ability to hide implementation details and protect data.

❖ **Examples:** like C, Pascal, and Fortran are commonly used for procedural programming.

➢ Object Oriented Programming:

- To write the code by using creating reusable objects with specific abilites.
- OOP is like creating and using toys. Each toy is an "object" that has a name (attributes) and things it can do (actions). You can make more toys based on same "blueprint" (class) and even make new toys that are similar to existing ones but with their unique features. OOP helps us build programs in a way that's similar to how we organize & interact with things in real world.

❖ Characteristics of object-oriented programming:

- **Objects:** Objects are instances of classes
- **Classes:** A class is a blueprint that defines the structure and behaviour of objects.
- **Encapsulation:** Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on the data within a single unit (object), while hiding the internal details from the outside.
- **Inheritance:** Inheritance allows a new class (subclass) to inherit attributes and methods from an existing class (superclass)
- **Polymorphism:** Polymorphism is the ability of objects of different classes to be treated as objects of a common parent class. It enables flexibility and dynamic behaviour based on the actual object type.
- **Abstraction:** Abstraction involves simplifying complex reality by modelling classes based on relevant attributes and behaviours, ignoring unnecessary details.

8) Robust:

- Robust means "**strong"**, python is a Robust programming language because of error handling concept.
- A programming is said to be Robust if and only if it always provides user-friendly error messages when and user commits mistakes at implementation level.
- To get user friendly error message we used exception handling features.
- Since python provide exception handling facility it is a Robust programming language.

9) Portable:

- Portable means python can be easily moved or adapted to different computing environments or platforms means "OS" here without requiring significant modifications.
- computing environment means set of hardware and software resources that support the execution of programs written in a particular language.

10) Extensible Programming Language:

- Extensible Programming means we can use other programming language in python.

11). Embedded Programming Language:

- Embedded programming language means we can use python programs in any other programming language program i.e., we can embed python program anywhere.

12). Support third party API'S

- Python supports extensively for making use of third party API'S (Modules) for developing the real time application with high performance with precise code.

Note: - **First Party API:** In built Library python environment.

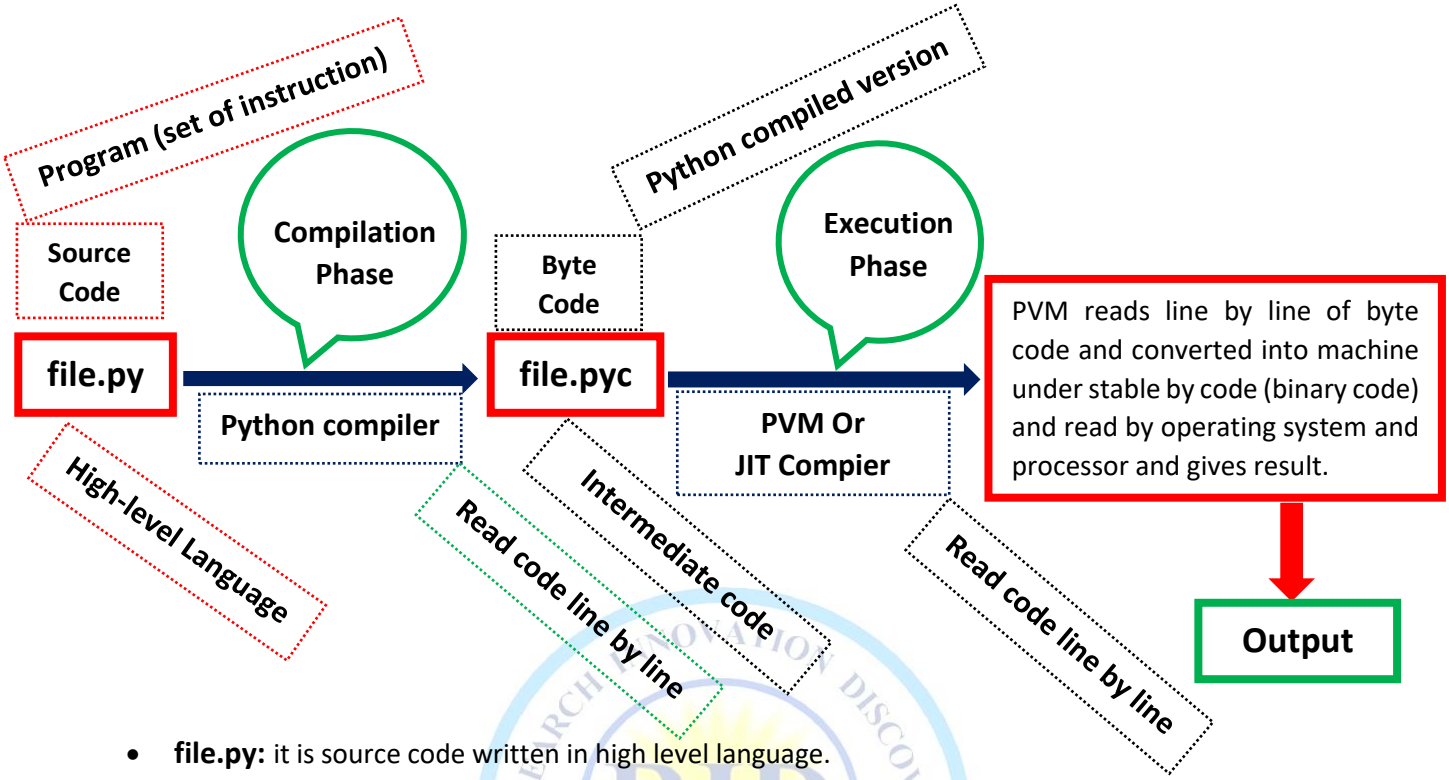
- **Second Party API:** Operating System Library.
- **Third Party API:** Vender, Developer Ex: NumPy, Pandas, SciPy, Scikit etc.

❖ Python Version:

- **Python 1.0 (January 26, 1994):** This was the initial release of Python.
- **Python 2.0 (October 16, 2000):** This version introduced list comprehensions, garbage collection improvements, & Unicode support. Python 2.0 marked a significant step forward language's evolution.
- **Python 2.7 (July 3, 2010):** This was last release of Python 2.x series and included many improvements bug fixes. Python 2.7 provided a bridge for developers transitioning from Python 2 to Python 3.
- **Python 3.0 (December 3, 2008):** Also known as "Python 3000" or "Py3K," this version introduced numerous breaking changes to the language in order to improve consistency, eliminate redundancies, and modernize the language's design.
- **Python 3.5 (September 13, 2015):** This release included features like asynchronous programming support with the "async" and "await" keywords, the "typing" module for type hints
- **Python 3.6 (December 23, 2016):**in this version included formatted string literals (f-strings), the "secrets" module for secure random number generation, and improvements in dictionary ordering.
- **Python 3.7 (June 27, 2018):** This version introduced data classes, the "contextvars" module for managing context-local data, and various syntax enhancements.
- **Python 3.8 (October 14, 2019):** Features included the "walrus operator" (:=), the "typing" module enhancements, and improvements in performance and security.
- **Python 3.9 (October 5, 2020):** This release added features like dictionary merge and update operators, the "zoneinfo" module for time zone support
- **Python 3.10 was released on October 4, 2021.**
- **Python 3.11 was released on Oct. 24th, 2022.**

Note: Python 3 won't provide backward compatibility to python 2 i.e., there is no guarantee that python 2 programs will run in python 3.

PYTHON EXECUTION ENVIRONMENT



- **file.py:** it is source code written in high level language.
- **Python compiler:** read code line by line.
- **file.pyc:** it is byte code it is also called python compiled version.
- **PVM:** Python virtual machine is machine dependent means for (window, mac, Linux) have different PVM.
- **JIT:** Just in Time compiler read line by line code not wait next line execution. Python used JIT compiler for improve performance.

Note:

- When we execute any python program internally two phases are occurring.
  1) Compilation phase
  2) Execution phase

1) Compilation phase:

- Python compiler reads the source code line by line and converted into intermediate code of python called byte code
- Since python compiler converting the source code into byte code line by line it is interpreted.

2) Execution phase:

- PVM (python virtual machine) reads line by line of byte code and converted into machine understandable code and it is read by operating system and processor and gives final result of the program.

KEYWORD SYMBOLS WITH NAMES

`	back quote	[	open square-bracket
~	tilde	]	close square-bracket
!	exclamation point	{	open curly-bracket/brace
@	at-sign	}	close curly-bracket/brace
#	pound-sign, number-sign, hash-tag	\	back-slash
$	dollar sign	|	pipe, bar
%	percent	;	semi colon
^	carat	:	colon
&	and-sign, ampersand	'	apostrophe, single-quote
*	asterisks	"	double-quote
(	open-parentheses	,	comma
)	close-parentheses	<	less-than, open angle-bracket
-	dash, minus-sign	.	period, dot, full-stop
_	underscore	>	greater-than, close angle-bracket
=	equals	/	slash
+	plus	?	question-mark

NUMBER SYSTEM

1) Binary Number:

- Binary number system, is a base-2 numeral system that uses two symbols: **0 and 1.**
- It's the foundation of digital technology and computing.
- Binary is fundamental in modern computing because digital devices, such as computers and microcontrollers, use binary to process and store data.
- All data, including text, images, sound, and videos, is ultimately represented in binary form within these devices.
- it directly corresponds to the on and off states of electronic switches and represents information using two distinct states.
- (0, 1) $( )_2$ {Base or Radix}

  Example: 0, 1, 01, 10, 1110, 10101011, 111001110101 etc.

2) Decimal Number:

- The decimal number system, also known as the base-10 number system, is a positional numeral system that uses ten symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) to represent numbers.
- It's the most common number system used by humans in everyday life.
- The decimal system is essential for everyday arithmetic, commerce, science, and a wide range of applications.
- (0,1…..9) $( )_{10}$ {Base or radix}

  Example: 0,1,2,4,5,6,7,8,9,10,11,12,13,14, 15……

3) Octal Number:

- The octal number system, also known as base-8, is a positional numeral system that uses eight symbols (0, 1, 2, 3, 4, 5, 6, and 7) to represent numbers.
- The octal system was more commonly used in computing systems that were based on multiples of 3 (as opposed to the binary system's base-2).
- However, octal has largely been replaced by hexadecimal (base-16) in modern computing due to its compatibility with binary and its more compact representation.
- **(0,1,2…….7) $( )_{8}$ {Base or radix}**

Example: 0,1,2,3,4,5,6,7,10, 11,…17,20,21……..27,30,31……..

4) Hexadecimal Number:

- Hexadecimal number system, often referred to as "hex" or base-16, is a positional numeral system that uses sixteen symbols: 0-9 for values 0 to 9, and A-F (or a-f) for values 10 to 15.
- The hexadecimal system is widely used in computing and digital systems as a concise way to represent binary data and memory addresses.
- Hexadecimal is often used in programming and computer science because it provides a more compact representation of binary data, and it's easier to work with when converting between binary and other number systems.
- (0 …….9, A,B,C,D,E,F) $( )_{16}$ {Base or Radix}

Example: 0, 1, 2, 3, 4, 5, 6, 7,8, 9, A, B, C, D, E, F, 1a,1b,1c …….. etc

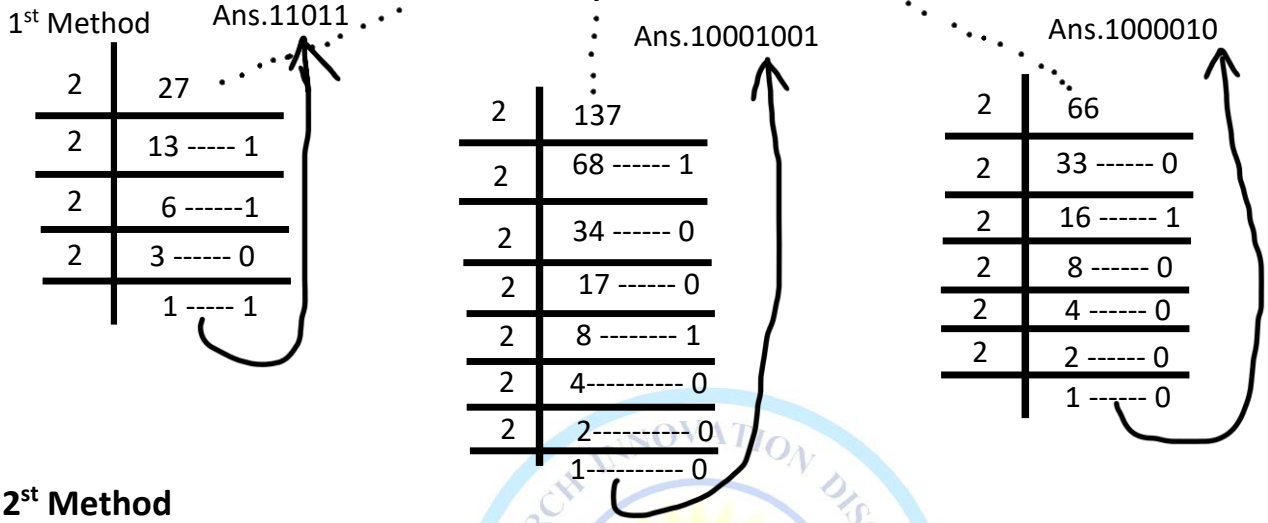
NUMBER SYSTEM CONVERSION

- Number system conversion is the process of converting a value from one numeral system (base) to another.

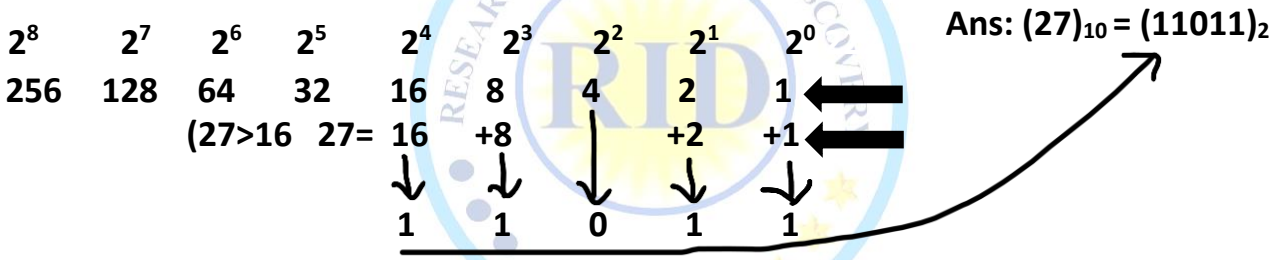
❖ Decimal Number to Binary Number Conversion

- **Question-1: Convert $(27)_{10}$, $(137)_{10}$ and $(66)_{10}$ Decimal into binary number**

1st Method

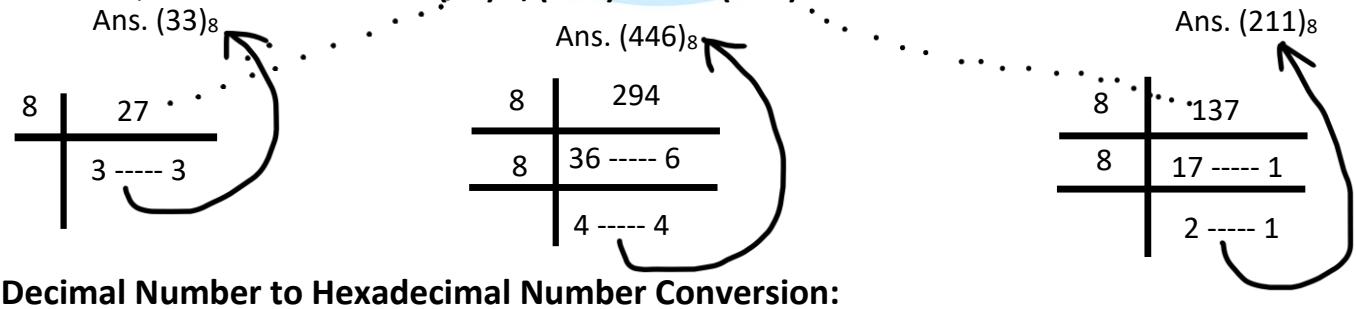


2st Method



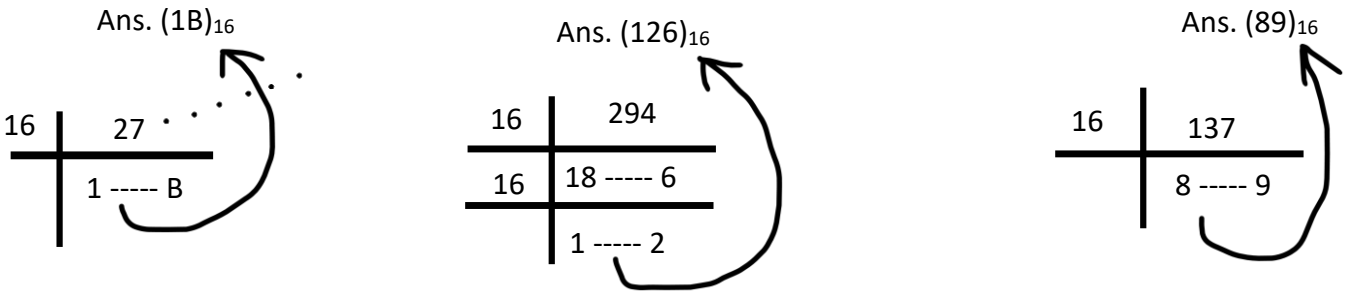
❖ Decimal Number to Octal Number Conversion:

- **Question-2: Convert $(27)_{10}$, $(294)_{10}$ and $(137)_{10}$ Decimal into octal number**



❖ Decimal Number to Hexadecimal Number Conversion:

- **Question-3: Convert $(27)_{10}$, $(294)_{10}$ and $(137)_{10}$ Decimal into Hexadecimal number**



❖ Binary to Decimal:

1. $(100010010)_2 = (\ )_{10}$
   - Solution: $1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
     $256+0+0+0+16+0+0+2+0=274 \quad (274)_{10}$ Ans.
2. $(10010)_2 = (\ )_{10}$
   - Solution: $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
     $16+0+0+2+0=18 \quad (18)_{10}$ Ans.
3. $(1110100)_2 = (\ )_{10}$
   - Solution: $1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
     $64+32+16+0+4+0+0= 116 \quad (116)_{10}$ Ans.

❖ Octal to Decimal:

1. $(422)_8 = (\ )_{10}$
   - Solution: $4 \times 8^2 + 2 \times 8^1 + 2 \times 8^0$
     $4 \times 64 + 2 \times 8 + 2 \times 1$
     $256 + 16 + 2 = 274 \quad (274)_{10}$ Ans.
2. $(4211)_8 = (\ )_{10}$
   - Solution: $4 \times 8^3 + 2 \times 8^2 + 1 \times 8^1 + 1 \times 8^0$
     $4X\ 512 + 2 \times 64 + 1 \times 8 + 1$
     $2048 + 128 + 8 + 1 = 2185 \quad (2185)_{10}$ Ans.
3. $(333)_8 = (\ )_{10}$
   - Solution: $3 \times 8^2 + 3 \times 8^1 + 3 \times 8^0$
     $3 \times 64 + 3 \times 8 + 3 \times 1$
     $192 + 24 + 3 = 219 \quad (219)_{10}$ Ans.

❖ Hexadecimal to decimal:

1. $(422)_{16} = (\ )_{10}$
   - Solution: $4 \times 16^2 + 2 \times 16^1 + 2 \times 16^0$
     $4 \times 256 + 2 \times 16 + 2 \times 1$
     $1024 + 32 + 2 = 1058 \quad (1058)_{10}$ Ans.
2. $(4211)_{16} = (\ )_{10}$
   - Solution: $4 \times 16^3 + 2 \times 16^2 + 1 \times 16^1 + 1 \times 16^0$
     $4 \times 4096 + 2 \times 256 + 1 \times 16 + 1$
     $16384 + 512 + 16 + 1 = 16913 \quad (16913)_{10}$ Ans.
3. $(333)_{16} = (\ )_{10}$
   - Solution: $3 \times 16^2 + 3 \times 16^1 + 3 \times 16^0$
     $3 \times 256 + 3 \times 16 + 3 \times 1$
     $768 + 48 + 3 = 819 \quad (819)_{10}$ Ans.
4. $(333AB)_{16} = (3 \times 16^4) + (3 \times 16^3) + (3 \times 16^2) + (10 \times 16^1) + (11 \times 16^0) = (209835)_{10}$
5. $(DCF39)_{16} = (13 \times 16^4) + (12 \times 16^3) + (15 \times 16^2) + (3 \times 16^1) + (9 \times 16^0) = (905017)_{10}$
6. $(AB23F34)_{16} = (10 \times 16^6) + (11 \times 16^5) + (2 \times 16^4) + (3 \times 16^3) + (15 \times 16^2) + (3 \times 16^1) + (4 \times 16^0) = (179453748)_{10}$

RESERVED KEYWORDS

- In python some words are reserved to represent some meaning or functionality Such types of words are called reserved words.
- There are following reserved keywords in python

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

Note:

- All reserved words in python contain only alphabet symbols.
- Except the following 3 reserved words, all contain only lower-case alphabet symbols.
  1) True
  2) False
  3) None

❖ How to check all python reserved keyword through python program:

```
>>> import keyword
>>> keyword. kwlist
```

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

❖ The importance of reserved keywords in Python includes:

- **Defining Language Features:** Reserved keywords define the core features and functionalities of the Python programming language.
- **Enforcing Syntax Rules:** Keywords help enforce the syntactical rules of the language.
- **Facilitating Control Flow:** Keywords like if, else, elif, while, and for are crucial for controlling the flow of program execution.
- **Defining Functions and Classes:** Keywords like def and class are used to define functions and classes, respectively.
- **Handling Exceptions:** Keywords like try, except, raise, and finally are used for exception handling. They allow developers to catch and manage runtime errors.
- **Boolean Logic:** Keywords like True, False, and not are essential for Boolean logic and conditional statements. They help in evaluating conditions and making logical comparisons.
- **Variable Scope and Namespaces:** Keywords like global and nonlocal are used to control variable scope and namespaces.
- **Concurrency and Asynchronous Programming:** Keywords like async and await are used in asynchronous programming to define asynchronous functions and await asynchronous operations.
- **Data Importing:** Keywords like import and from are used to import modules and packages
- **Loop Control:** Keywords like break and continue provide control within loops.
- **Context Management:** Keywords like with and as are used in context management to simplify the process of acquiring and releasing resources, such as files or network connections.

IDENTIFIER

- A Name in python program is Identifier.
- It can be Variable name, function name, class name, or module name.
- Identifiers are essential for developers to give meaningful and recognizable names.
- These names are used to access and manipulate the associated values in the code.

Example:

a=6

❖ Rules to define identifiers:

1) Name Rules:

- Identifiers can consist of letters (both uppercase and lowercase), digits, and underscores (_).
- They must start with a letter (uppercase or lowercase) or an underscore.
- Identifiers are case-sensitive, meaning that **name** and **Name** are treated as two different identifiers.

2) Allowed Characters:

- Letters: A-Z, a-z
- Digits: 0-9
- Underscore: _

3) Length:

- There is typically no fixed limit on the length of identifiers.

4) Identifier cannot start with any digit, special symbol and dot.

5) We cannot use reserved words as identifiers.

6) Dollar ($) symbol is not allowed in python.

7) Space is not allowed in between to character

Note:

❖ If identifier starts with **_(underscore)** symbol then it indicates that is private.

❖ If identifier starts with **__(Two under score symbols)** indicating that strongly private identifiers.

❖ If the identifier starts and ends with two underscores **(e.g., __add__)** symbols then the identifier is python language defined special name, which is also known as magic methods or constructor.

Valid Identifiers

1) A
2) _a
3) _36
4) Abt30
5) _6_3_a
6) t3_3a
7) t3abt
8) skills
9) stu_name
10) twksaa_skills_center

Invalid Identifiers

11) 3A
12) $_a
13) 39
14) =Abt30
15) 6_3_a
16) %t3_3a
17) .t3abt
18) if
19) def
20) 3twksaa_skills_center
21) a b

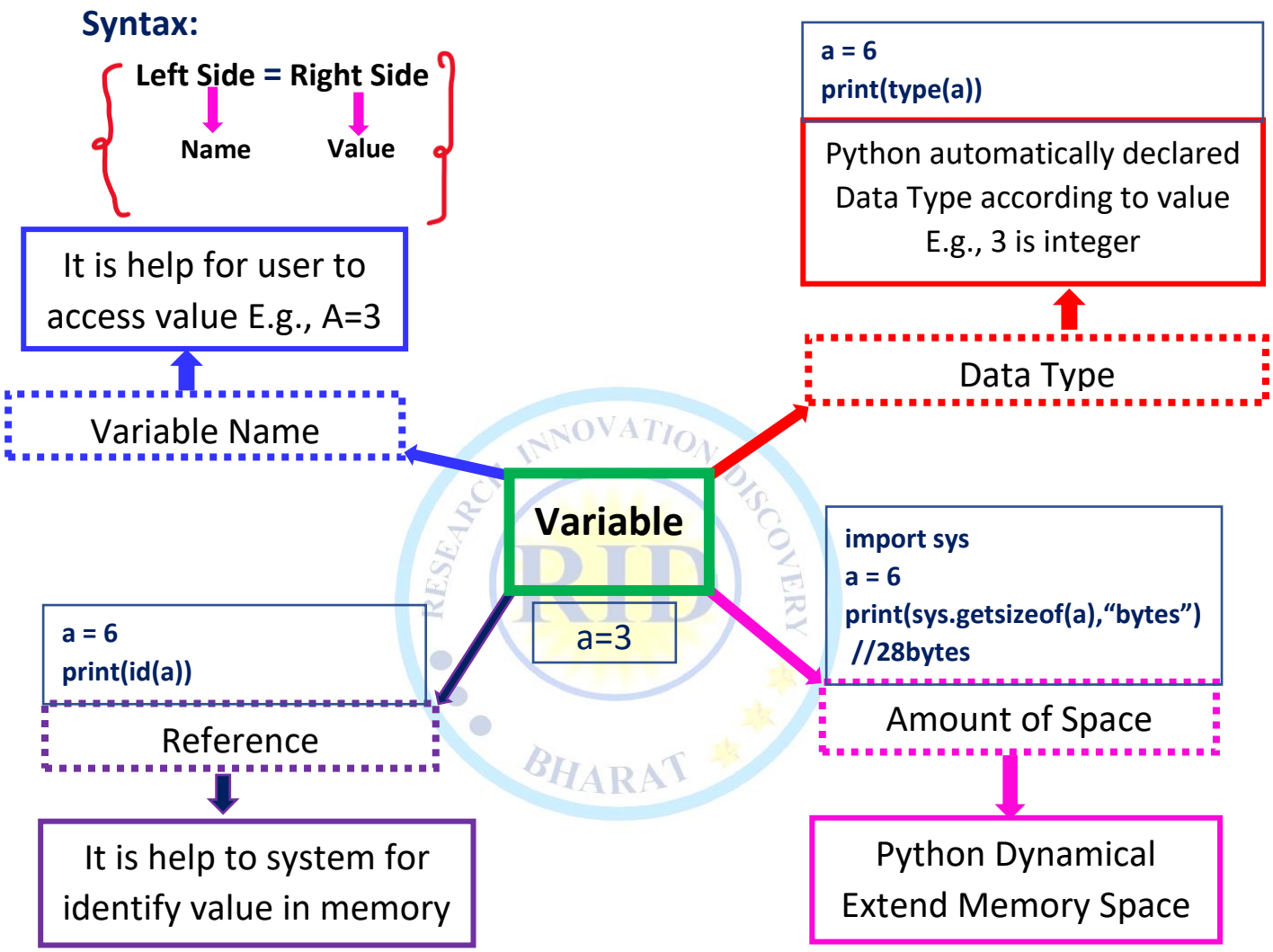
VARIABLE

- variable is a symbolic name that represents a value stored in the computer memory. Or Memory location Named are called variable.
- Its value will change time to time so it is known as a variable.
- A variable is an identifier whose values can be changed during execution of the program.

Syntax:

Left Side = Right Side

Name **Value**

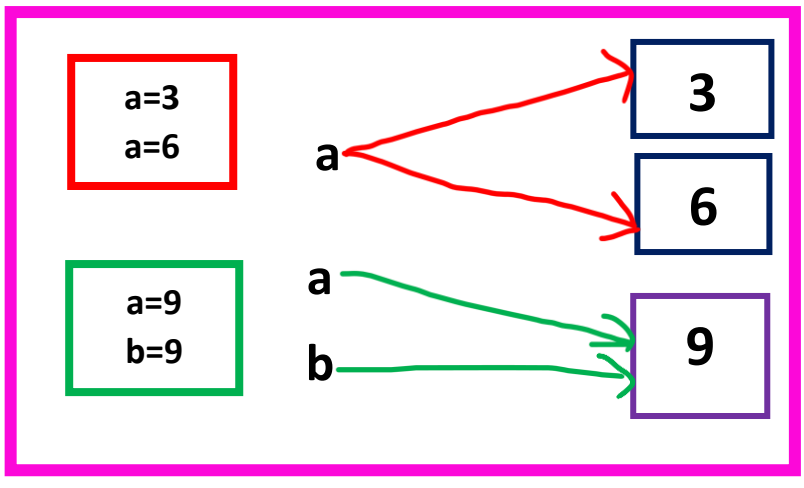


- **Python Variable is Immutable**

Note: if we change the variable, it does not modify in the exit it will be create new object.

Example:

```
>>> a=6
>>> print(id(a))
140716071969736
>>> b=6
>>> print(id(b))
140716071969736
>>> a=10
>>> print(id(a))
140716071969864
>>> a=12
>>> print(id(a))
140716071969928
```



Variable name and its value Execution:

Case-1: if two variable name is same and its value is also same then its memory address will be same.

Example:

a=6
print(id(a)) → 140051812876520
a=6
print(id(a)) → 140051812876520

Case-2: if two variable name is same and its value is different than its memory address will be different.

Example:

a=12
print(id(a)) → 140225080467880
a=20
print(id(a)) → 140225080468136

Case-3: if two variable name is different and its value is same than its memory address will be same.

Example:

a=3
print(id(a)) → 140551177728136
b=3
print(id(b)) → 140551177728136

Case-4: if two variable name is different and its value is also different than its memory address will be different.

Example:

a=6
print(id(a)) → 140288508899560
b=9
print(id(b)) → 140288508899656

Note-1: Here a, b is a variable name and 6 and 9 is its value where print() function is used for display the output and id() function is used for display the memory address of variable.

Note-1: During execution your code memory address will difference to given above address.

❖ Important of variable in python:

- In Python, variables play a crucial role in programming by serving as named containers for storing and manipulating data. They allow programmers to assign values to symbols, enabling creation of dynamic and flexible programs. Variables facilitate code readability, reusability, and maintenance by providing meaningful names to data.

COMMENTS

- Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program.
- Comments can be used to explain Python code. Comments can be used to make the code more readable.

➢ common uses for comments:

- Readability of the Code
- Restrict code execution
- Provide an overview of the program or project metadata
- To add resources to the code

❖ Types of Comments:

1) Single-Line Comments
2) Multi-Line Comments
3) docstring comments.

1) Single-Line Comments:

- A single-line comment starts and ends in the same line.
- We use the # symbol to write a single-line comment.

Example:

```
# Welcome message for everyone
Print("welcome to T3 Skills Center")
```

2) Multi-Line Comments:

- use hashtags (#) multiple times to construct multiple lines of comments.

Example:

```
# it is a
# comment
# extending to multiple lines
```

3) docstring comments:

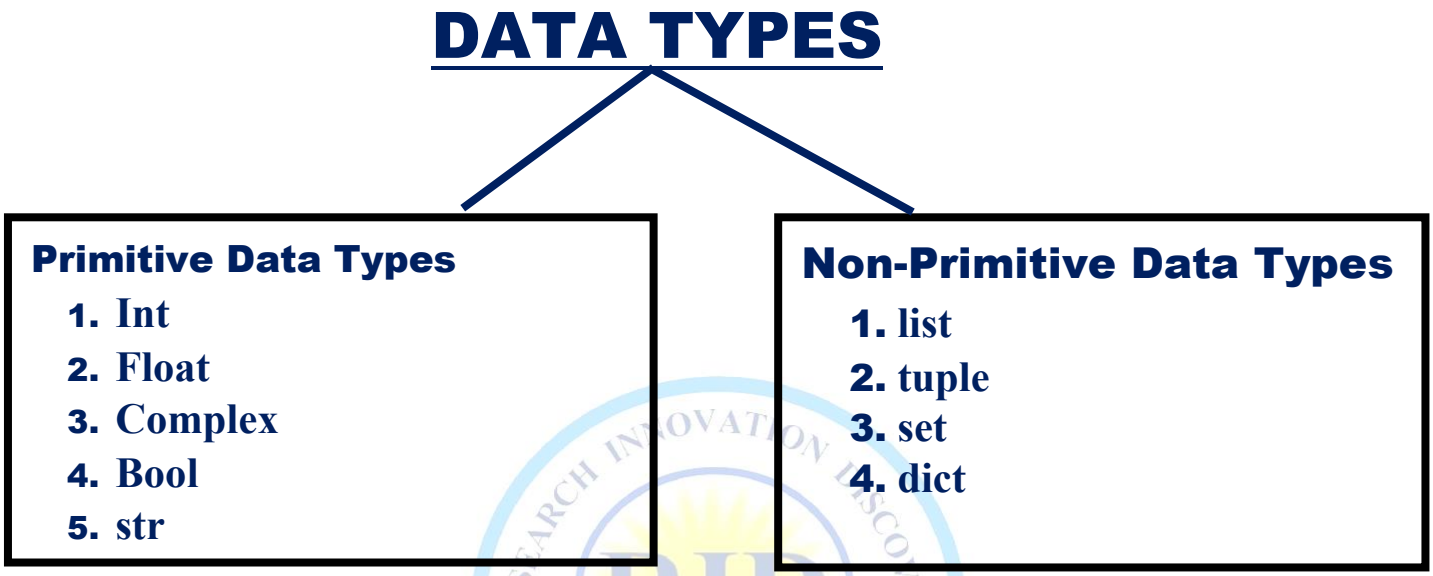
- The strings enclosed in triple quotes that come immediately after the defined function are called Python docstring.

Example:

```
# Code to show how we use docstrings in Python
def add(x, y):
   """This function adds the values of x and y"""
   return x + y
# Displaying the docstring of the add function
print( add.__doc__ )
```

DATA TYPES

- **Data Type**: A classification that defines how data is used in a program.
- **Purpose**: Represents the type of data stored in a variable.
- **Python Data Types**: No need to specify explicitly; assigned automatically.
- **Dynamic Typing**: Python determines the type based on the provided value.



❖ Python Contains the following inbuilt data types:

- **Integer (int):** Represents whole numbers. **Example:** age = 30
- **Floating-Point (float):** Represents real numbers with decimal points. **Ex:** pi = 3.14
- **String (str):** Represents sequences of characters. **Example:** name = "skills"
- **Boolean (bool):** Represents true or false values. **Example:** a = True
- **Complex:** Represents complex numbers with real and imaginary parts. **Ex:** cn = 3 + 6j
- **List (list):** Represents an ordered collection of items. **Example:** l = [1, 2, 3, 4, 5,6]
- **Tuple (tuple):** Represents an ordered, immutable collection of items. **Ex:** t = (10, 20,30)
- **Dictionary (dict):** Represents key-value pairs. **Ex:** d = {"name": "Sangam", "age": 12}
- **Set (set):** Represents an unordered collection of unique items. **Ex**: s = {1, 2, 3, 4,5,6}
- **None (NoneType):** Represents the absence of a value. **Example**: r = None
- **Range (range):** Represents a sequence of numbers within a range. **Ex:** R = range(1, 6)
- **Frozenset:** Represents an immutable set.
  Example: frozen_numbers = frozenset([4, 5, 6])
- **Bytes (bytes):** Represents binary data.
  Example: binary_data = b'\x00\x01\x02'
- **Bytearray (bytearray):** Similar to bytes, but mutable.
  Example: mutable_data = bytearray([0, 1, 2])

PYTHON INBUILT FUNCTIONS

1) Print (): It is used to print the value in python everything is an object.

Example:

- **Printing a single string:**

```
print ("TWKSAA SKILLS CENTER")
    TWKSAA SKILLS CENTER
```

- **Printing variables and literals:**

```
name = "Sangam"
age = 12
print ("Name:", name, "Age:", age)
output: Name: Sangam Age: 12
```

- **Printing with formatted strings:**

```
name = "Sangam Kumar"
age = 12
print (f"Name: {name}, Age: {age}")
Output: Name: Sangam Kumar, Age: 12
```

- **Printing multiple items:**

```
Print ("TWKSAA skills center", "Foundation Day", "30-09-2023", sep=", ", end="!\n")
```

Output:

TWKSAA skills center, Foundation Day, 30-09-2023!

2) type ():- This inbuilt function is used for check the type of variable.

Example:

```
>>> a=3
>>> type(a) or print(type(a))
<class 'int'>
>>> b='twksaa'
>>> type(b) or print(type(b))
<class 'str'>
>>> c=3.3
>>> type(c) or print(type(c))
<class 'float'>
>>> d=True
>>> type(d) or print(type(d))
<class 'bool'>
>>> e=[1,2,3]
>>> type(e) or print(type(e))
<class 'list'>
>>> f=1,2,3
>>> type(f) or print(type(f))
<class 'tuple'>
>>> d={ "name":"Sangam", "age":12}
>>> type(d) or print(type(d))
<class 'dict'>
>>> s = {1, 2, 3, 4,5,6}
>>> type(s) or print(type(s))
<class 'set'>
>>> r = None
>>> type(r) or print(type(r))
<class 'NoneType'>
>>> binary_data = b'\x00\x01\x02'
>>> type(binary_data)
<class 'bytes'>
>>>mutable_data = bytearray([0, 1, 2])
>>> type(mutable_data)
<class 'bytearray'>
>>> R = range(1, 6)
>>> type(R) or print(type(R))
<class 'range'>
>>> cn = 3 + 6j
>>> type(cn) or print(type(cn))
<class 'complex'>
```

3) id (): - This inbuilt function is used for get the memory address of an object or variable.

Example:

```
>>> a=3
>>> id(a)
140715871363944
>>> b='skills'
>>> id(b)
2784336905200
```

```
a=3
print(id(a))
140715871363944
b='skills'
print(id(b))
2784336905200
```

INT DATA TYPES

- we can use int type to represent whole numbers (integral values)
- Integers in Python can be positive, negative, or zero.
- Integer data is immutable, meaning their value cannot be changed after creation.
- It is used in mathematical operations, counting, indexing & various other programming tasks.

Example: a=6, n=9, abc=333 etc.

```
a=6
print(type(a))
<class 'int'>
n=9
print(type(n))
<class 'int'>
abc=333
print(type(abc))
<class 'int'>
```

❖ We can represent int values in the following ways:

1) Decimal form
2) Binary form
3) Octal form
4) Hexa decimal form

1) Decimal form (Base-10):

- It is the default number system in python
- The allowed digits are (0 – 9)

E.g., 0,1,2,3,4,5,6,7,8,9

2) Binary Form (Base-2):

- The allowed digits are (0 & 1)
- Literal value should be prefixed with 0b or 0B

E.g., a=0B10101, a=b01010, a=0b111 etc.

3) Octal form (Base-8):

- The allowed digits are (0 – 7)
- Literal value should be prefixed with 0o or 0O

E.g., a=0o123, a=0O432, a=0o345 etc.

4) Hexa decimal form (Base-16):

- The allowed digits are (0 – 9, a-f or A-F)
- Literal value should be prefixed with 0x or 0X

E.g., a=0X12abf, a=0xfac120, a=0xAB120 etc.

Note:

Being a programmer, we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only decimal form.

Example:

```
a=6
b=0o6
c=0X6
d=0B101010
print(a) 6a
print(b) 6
print(c) 6
print(d) 42
```

```
a=30
b=0b1010111
c=0o165
d=0x15a
print("Value of a=",a)
print("Value of b=",b)
print("Value of c=",c)
print("Value of d=",d)
```

Output:

```
Value of a= 30
Value of b= 87
Value of c= 117
Value of d= 346
```

BASE CONVERSIONS

- Python provide the following in-built function for base conversion.
    1) bin ()
    2) oct ()
    3) hex ()
    4) int ()

1) bin ():

➢ bin() is used for convert any number system to binary number system .

Example:

```
bin (6)
'0b110'
bin(0o6)
'0b110'
bin(0x112ab)
'0b10001001010101011'
bin (333)
'0b101001101'
bin(0O1203)
'0b1010000011'
```

```
a=30
b=0b1010111
c=0o165
d=0x15a
print("Value of a=",a)
print("Value of b=",b)
print("Value of c=",c)
print("Value of d=",d)
```

Example:

```
a=6 #decimal number
b=0o165 #octel number
c=0x15a #Hexadecimal
Binary_Number_of_a=bin(a)#decimal Number
Binary_Number_of_b =bin(b) #Octel Number
Binary_Number_of_c=bin(c) #Hexadecimal Number
print("Binary Number of a=",Binary_Number_of_a)
print("Binary Number of b=",Binary_Number_of_b)
print("Binary Number of c=",Binary_Number_of_c)
```

Output:

```
Binary Number of a= 0b110
Binary Number of b= 0b1110101
Binary Number of c= 0b101011010
```

2) oct ():

➢ oct() is used for convert any number system to octal number system ..

Example:

```
oct (6)
'0o6'
oct (333)
'0o515'
oct(0B111)
'0o7'
oct(0X39)
'0o71'
```

Example:

```
a=6 #decimal number
b= 0b1110101 #Binary number
c=0x15a #Hexadecimal
x=oct(a)
y=oct(b)
z=oct(c)
print("Octal Number of a=",x)
print("Octal Number of b=",y)
print("Octal Number of c=",z)
```

Output:

```
Octal Number of a= 0o6
Octal Number of b= 0o165
Octal Number of c= 0o532
```

3) hex ():

- hex() is used for convert any number system to hexadecimal number system ..

Example:

```
hex (2039)
'0x7f7'
hex(0B1011001)
'0x59'
hex(0o32120)
'0x3450'
```

Example:

```
a=39 #decimal number
b= 0b1110101 #Binary number
c=0o1657 #octal number
x=hex(a)
y=hex(b)
z=hex(c)
print("Hexadecimal Number of a=",x)
print("Hexadecimal Number of b=",y)
print("Hexadecimal Number of c=",z)
```

Output:

```
Hexadecimal Number of a= 0x27
Hexadecimal Number of b= 0x75
Hexadecimal Number of c= 0x3af
```

4).int(): int() is used for convert any number system to Decimal number system .

```
print(int("1010", 2))  # Binary 1010 to Decimal
# Octal to Decimal
print(int("12", 8))    # Octal 12 to Decimal
# Hexadecimal to Decimal
print(int("A", 16))    # Hexadecimal A to Decimal
```

SIMPLE APPLICATION ON BASE CONVERSION

❖ Program:

```
print("Welcome to TWKSAA Base Conversion Application:")
print("For Enter Decimal Number press-0\nFor Enter Binary Number press-1\nFor Enter Octal Number press-2\nFor Enter Hexadecimal Number press-3")
choice=int(input())
if choice==0:
   number = int(input("Enter the Decimal Number: "))
   print("For Convert Binary Number press-1 \nFor Convert Octal Number press-2 \nFor Convert Hexadecimal Number press-3")
   user_choice = int(input("Enter your choice 1,2,3="))
elif choice==1:
   number = int(input("Enter the Binary Number: "),2)
   print("For Convert Decimal Number press-0\nFor Convert Octal Number press-2 \nFor Convert Hexadecimal Number press-3")
   user_choice = int(input("Enter your choice 0,2,3="))
elif choice==2:
   number = int(input("Enter the Octal Number: "),8)
   print("For Convert Decimal Number press-0\nFor Convert Binary Number press-1 \n\nFor Convert Hexadecimal Number press-3")
   user_choice = int(input("Enter your choice 0,1,3="))
elif choice==3:
   number = int(input("Enter the Hexadecimal Number: "),16)
   print("For Convert Decimal Number press-0\nFor Convert Binary Number press-1 \nFor Convert Octel Number press-2")
   user_choice = int(input("Enter your choice 0,1, 2="))
else:
   print("Invalid Number")
if user_choice==0:
   print("Decimal Number is:", number)
elif user_choice == 1:
   result = bin(number)[2:]  # Remove the '0b' prefix
   print("Binary Number is:", result)
elif user_choice == 2:
   result = oct(number)[2:]  # Remove the '0o' prefix
   print("Octal Number is:", result)
elif user_choice == 3:
   result = hex(number)[2:]  # Remove the '0x' prefix
   print("Hexadecimal Number is:", result)
else:
   print("Invalid Number")
```

Output:

```
Welcome to TWKSAA Base Conversion Application:
For Enter Decimal Number press-0
For Enter Binary Number press-1
For Enter Octal Number press-2
For Enter Hexadecimal Number press-3
1
Enter the Binary Number: 11010110
For Convert Decimal Number press-0
For Convert Octal Number press-2
For Convert Hexadecimal Number press-3
Enter your choice 0,2,3=3
Hexadecimal Number is: d6

Welcome to TWKSAA Base Conversion Application:
For Enter Decimal Number press-0
For Enter Binary Number press-1
For Enter Octal Number press-2
For Enter Hexadecimal Number press-3
3
Enter the Hexadecimal Number: 3abc39
For Convert Decimal Number press-0
For Convert Binary Number press-1
For Convert Octal Number press-2
Enter your choice 0,1, 2=1
Binary Number is: 11101010111100000111001

Welcome to TWKSAA Base Conversion Application:
For Enter Decimal Number press-0
For Enter Binary Number press-1
For Enter Octal Number press-2
For Enter Hexadecimal Number press-3
0
Enter the Decimal Number: 393
For Convert Binary Number press-1
For Convert Octal Number press-2
For Convert Hexadecimal Number press-3
Enter your choice 1,2,3=3
Hexadecimal Number is: 189
Welcome to TWKSAA Base Conversion Application:
For Enter Decimal Number press-0
For Enter Binary Number press-1
For Enter Octal Number press-2
For Enter Hexadecimal Number press-3
6
Invalid Number
```

FLOAT DATA TYPE

- We can use float data type to represent floating point values (decimal values)

Example: a=3.3, b=6.63, a=3.9.3 etc.

```
a=3.3
b=6.63
c=0.03
print(type(a))
<class 'float'>
print(type(b))
<class 'float'>
print(type(c))
<class 'float'>
```

- We can also represent floating point values by using exponential form (Scientific Notation).

Example: f=3.3e3 ---> instead of 'e' we can use 'E'

```
f=3.3e3
a=0.e0
c=6.E33
print(type(f))
<class 'float'>
print(type(a))
<class 'float'>
print(type(c))
<class 'float'>
d=3e6
print(type(d))
<class 'float'>
```

- The main advantage of exponential form is we can represent big values in less memory.

Note: We can represent int values in decimal, binary, octal and hexa decimal forms. But we cannot represent float values.

Example: f=0B11.01

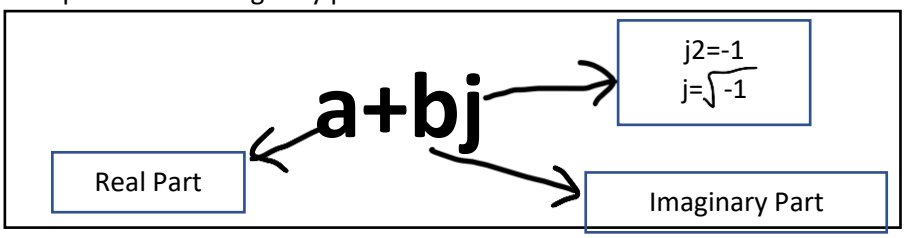
```
SyntaxError: incomplete input
a=0x3.3
SyntaxError: incomplete input
b=0O6.6
SyntaxError: incomplete input
d=0xab.123
SyntaxError: incomplete input
```

Example:

```
a=0b10101110
print(a) #174
b=0b1010111.10
print(b) #SyntaxError: invalid syntax
b=0o127.45
print(b) #SyntaxError: invalid syntax
f=0x124abc.bc223
print(f) #AttributeError: 'int' object has no attribute 'bc223
```

COMPLEX DATA TYPE

- Complex data type is used to represent complex numbers, which are numbers that have both a real part and an imaginary part.



- Where 'a', and 'b', contain integers or floating-point values.
  E.g., 3+6j, 0.3+6j ,0.6+0.3j etc.
- In the real part if we use int value then we can specify that either by decimal, octal, binary or hexadecimal form.
- But imaginary part should be specified only by using decimal form.

Example:
```
a=0B11+3j, 0o12+6j, 0X1ab34+9j etc.
a=3+6j
print(type(a))
<class 'complex'>
b=0B101+3j
print(type(b))
<class 'complex'>
c=0o123+9j
print(c)
(83+9j)
d=0x12abf+3j
print(type(d))
<class 'complex'>
e=3+0x23j
SyntaxError: invalid hexadecimal literal
```

- We can perform operations on complex type values.

Example:
```
a=3+6j
b=33+69j
c=a+b
print(c)
(36+75j)
print(type(c))
<class 'complex'>
```

Note: Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

Example:
```
a=3+39j
print(a.real)
3.0
print(a.imag)
39.0
```

Example:
```
a=3+6j
print("value of a=",a)
b=3.3+6j
print("value of b=",b)
c=3.3+6.6j
print("value of c=",c)
d=0b1101+4j
print("value of d=",d)
e=0o123+3.4j
print("value of e=",e)
f=0x123a+6.4j
print("value of e=",f)
g=0x123a.5+6.4j #SyntaxError: invalid syntax
print("value of e=",g)
h=3+0b111j #SyntaxError: invalid binary literal
print("value of h=",h)
h=3+7j
print("value of h=",h)
```

output:
```
value of a= (3+6j)
value of b= (3.3+6j)
value of c= (3.3+6.6j)
value of d= (13+4j)
value of e= (83+3.4j)
value of e= (4666+6.4j)
value of h= (3+7j)
```

BOOL DATA TYPES

- You can use this data types to represent Boolean values.
- The only allowed values for this data type are:
    1) True
    2) False
- Internally python represents True as 1 and False as 0

Example:

```
a=True
b=False
print(type(a))
<class 'bool'>
print(type(b))
<class 'bool'>
```

Example:

```
a=3
b=6
c=a<b
print(c)
True
```

STRING

- str represents String data type. A string is a sequence of characters enclosed within single quotes or double quotes.
- String can be created by using '' or "" or ''' ''' or """ """
- Triple quotes can be used for multiline string. String is immutable, that can't be modifying directly.

Example:

```
a='RID' #single quotes
print(a)
print(type(a))
s1='skill\'s'
print(s1)
s2="T SKILLS CENTER" #double quotes
print(s2)
print(type(s2))
s3='''This is Learning Earning Development skills center''' #triple quotes
print(s3)
print(type(s3))
s4="""Foundation Day of TWKSAA SKILLS CENTER is 30-09-2023. """ #double quotes
at three times
print(s4)
print(type(s4))
```

Output:

```
RID
<class 'str'>
skill's
T3 SKILLS CENTER
<class 'str'>
This is Learning Earning & Development skills center
<class 'str'>
Foundation Day of TWKSAA SKILLS CENTER is 30-09-2023.
<class 'str'>
```

Note:

- In python the following data types are considered as fundamental data types
  1. int
  2. float
  3. complex
  4. bool
  5. str
- In python, we can represent char values also by using str type and explicitly char type is not available.
  E.g., c='a'
  type(c)
  <class 'str'>
- long data type is available in python 2 but not in python3. In python 3 long values also represent by using int type only.

TYPE CASTING

- Conversion from one data type value to another data type value, this conversion or process is called typecasting or type coercion.
- The following are various inbuilt function for type casting.
    1) int()
    2) float()
    3) complex()
    4) bool()
    5) str()

❖ int ():

- we can use this function to convert values from other types to int.

Example:

```
>>>int(3.33)
>>>3
>>>int(True)
>>>1
>>>int(False)
>>>0
>>>int("393")
>>>393
>>>int("3.3")
>>>ValueError: invalid literal for int() with base 10: '3.3'
ValueError: invalid literal for
 int() with base 10: 'six'
>>>int(0B1101)
>>>13
>>>int("0B1101")
>>>ValueError: invalid literal for int() with base 10: '0B1101'
>>>int(3+6j)
>>>TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

Ex-1

```
A=6.33 #foat value
B=int(A)
print(B)
print(type(B))
```

Output:

```
6
<class 'int'>
```

Ex-2

```
A=True #bool value
B=int(A)
print(B)
print(type(B))
```

Output:

```
1
<class 'int'>
```

Ex-3

```
A= "skills" #string value
B=int(A)
print(B)
print(type(B))
```

Output:

```
ValueError: invalid literal for int() with base 10: 'skills'
```

Ex-4

```
A= "6.33" #string value
B=int(A)
print(B)
print(type(B))
```

Output:

```
ValueError: invalid literal for int() with base 10: '6.33'
```

Ex-5

```
A=0b10111#Binary_No
B=int(A)
print(B)
print(type(B))
```

Output:

```
23
<class 'int'>
```

Ex-6

```
A= "0b101"#string
B=int(A)
print(B)
print(type(B))
```

Output:

```
ValueError: invalid literal for int() with base 10: '0b10111'
```

Ex-7

```
A= 3+6j  #complex value
B=int(A)
print(B)
print(type(B))
```

Output:

```
TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

Note:

- we can convert from any type to int except complex type.
- If we want to convert str type to int type, compulsory str should contain only integral value and should be specified in base-10.

FLOAT ()

- we can use float () function to convert other type values to float type.

Example:

```
float(3)
3.0
float(True)
1.0
float(False)
0.0
float(3+6j)
TypeError: float() argument must be a string or a real number, not 'complex'
float("three")
ValueError: could not convert string to float: 'three'
float(OB111)
NameError: name 'OB111' is not defined
float(0Xab12)
43794.0
float(0b11)
3.0
float(0O123)
83.0
```

Note:

- we can convert any type value to float type except complex type.
- Whenever we are trying to convert str type to float type compulsory str should be either integral or floating-point literal and should be specified only in base-10.

COMPLEX ()

- We can use complex () function to convert other types to complex type.

Form-1: Complex (x)

- We can use this function to convert x into complex number with real part x and imaginary part 0.

Example:

```
complex(3)
(3+0j)
complex(3.6)
(3.6+0j)
complex(True)
(1+0j)
complex(False)
0j
complex("6.3")
(6.3+0j)
complex("three")
 File "<pyshell#15>", line 1, in <module>
complex("three")
ValueError: complex() arg is a malformed string
```

Form-2: Complex (x, y)

- We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

Example:
(3+6j)
complex (True, False)
(1+0j)

BOOL ()

- We can use this function to convert other values to bool type.

Example:
bool(0)
False
bool(1)
True
bool(3)
True
bool(0.0)
False
bool("6")
True
bool(3.6)
True

❖ **str ():** we can use this method to convert other values to str type.

Example:
str(3)
'3'
str(3.3)
'3.3'
str(3+6j)
'(3+6j)'
str(True)
'True'

❖ **bytes Data Type:** bytes data type represents a group of byte number just like an array.

Example:

```
a=[3,9,3, 139,255]
b=bytes(a)
print(type(b))
print(b[0])
print(b[-1])
for i in b:
    print(i)
```

Output

```
<class 'bytes'>
3
255
3
9
3
139
255
```

Conclusion-1:

- The only allowed values for byte data type are 0 to 256. If you will provide other then we get value error.
- One we create bytes data types; cannot change its value. Otherwise, will get typeError.

❖ Byte array Data types:

- It is exactly same as bytes data types except that its elements can be modified.

Example:

```
a=[3,9,3, 139,255]
b=bytearray(a)
print(type(b))
for i in b:
   print(i)
b[0]=100
for i in b:
   print(i)
```

Output

```
<class 'bytearray'>
3
9
3
139
255
100
9
```

LIST DATA TYPES

- If we want to represent a group of value as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.
- List is growable in nature. Means based on our requirement we can increase or decrease the list size.
- List is ordered collection of elements. Means it's allowed indexing.
- Values should be enclosed within square brackets. Means We can create list by using [].
- List will allow duplicate elements.
- Heterogeneous objects are allowed. means List will allow different data type elements
- List is mutable, once we create a list that can be modified.

Example:

```
l= [3,6,"twksaa",30.9,23,'A']
print(l)
print(type(l))
```

Output

```
[3, 6, 'twksaa', 30.9, 23, 'A']
<class 'list'>
```

TUPLE DATA TYPES

- tuple data type is exactly same as list data types except that is immutable means we cannot modify values.
- Tuple is ordered collection of elements same as list.
- We can create tuple by using () but brackets are optional.
- Tuple will allow different data type elements and Tuple will allow duplicate elements.
- Creating a tuple with one element is bit different, to create a tuple with one element, after element we have to give comma (,)

Example:

```
t=(6,3,"twksaa",30.9,2023,'foundation day')
print(t)
print(type(t))
#Ex2:
t=(6,)
print(t)
print(type(t))
t=6,
print(t)
print(type(t))
```

Output

```
(6, 3, 'twksaa', 30.9, 2023,
'foundation day')
<class 'tuple'>
(6,)
<class 'tuple'>
(6,)
<class 'tuple'>
```

Note: tuple is the read only version of list.

RANGE DATA TYPES

- range data types represent a sequence of numbers.
- Range is used to generate range of values.
- By default, range starts from 0
- Range is immutable the elements present in range data type are not modifiable. Means we cannot change the range values.

Example:

```
r=range(9)
print(r)
print(type(r))
r=range(6)
for i in r:
    print(i)
r1=range(10,15)
print(r1)
r1=range(10,13)
for i in r1:
    print(i)
r3=range(15,20,3)
for i in r3:
    print(i)
```

Output

```
range(0, 9)
<class 'range'>
0
1
2
3
4
5
6
range(10, 15)
10
11
12
15
18
```

SET DATA TYPES

- If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.
- Insertion order is not preserved. Means it is not allowed indexing.
- Heterogeneous objects are allowed. means set will allow different data type elements
- Set is unordered collection of unique elements.
- We can create by using {}
- Set will not allow duplicate elements.
- Set is mutable, we can modify the set.
- Growable in nature. Means based on our requirement we can increase or decrease the size.
- To create an empty set then we use set() function.

Example:

```
s=set()
print(s)
print(type(s))
s1={10,"twksaa",30.6,23,'foundation day'}
print(s1)
print(type(s1))
```

Output

```
set()
<class 'set'>
{30.6, 23, 10, 'foundation day', 'twksaa'}
<class 'set'>
```

DICT DATA TYPES

- If we want to represent a group of values as a key-value pairs then we should go for dictionary data type
- Dict is a collection of items.
- In dict each item can be a pair i.e. key and value.
- We can create dict by using {}
- In dict keys are immutable and must be unique.
- In dict values are mutable and no need to be unique.
- Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.
- In dict keys and values can be of any data type.
- In dict keys cannot be modified but values can be modified.

Example-1:

```
d={301: 'twksaa',302: 'twf', 303: 'skills'}
```

Example-2:

```
d={}
print(d)
print(type(d))
d={1:"twksaa",2:"skills",'a':'apple',3:30.9,1:"center"}
print(d)
print(type(d))
```

Output

```
{}
<class 'dict'>
{1: 'center', 2: 'skills', 'a': 'apple', 3: 30.9}
<class 'dict'>
```

Note: dict is mutable and the order won't be preserved.

❖ forzenset Data Type:

- it is exactly same as set except that it is immutable.
- Hence, we cannot use add or remove function.

Example:

```
s={3,6,9,12,15,18}
fs=frozenset(s)
print(type(fs))
fs
for i in fs:
    print(i)
fs.add(70)
```

Output

```
<class 'frozenset'>
18
3
6
9
12
15
AttributeError: 'frozenset'
```

❖ None Data Type:

- None means nothing or No value associated.
- If the value is not available, then to handle such type of case None introduced.
- It is something like null value in java.

Note:

- In general, we can use bytes and bytearray data types to represent binary information like images, video files etc.
- In python 2 long data types is available. but in python 3 it is not available and we can represent long values also by using int type only.
- In python there is no char data type. Hence, we can represent char values also by using str type.

ESCAPE CHARACTERS

- In string literals, we can use escape characters to associated a special meaning.
- The following are various important .
    1) \n → new line
    2) \t → Horizontal tab
    3) \r →carriage Return
    4) \b →back space
    5) \f →form feed
    6) \v →vertical tab
    7) \' → single quote
    8) \'' →Double Quote
    9) \\ →back slash symbol

Example:

```
print("SKILLS\nWIT\nRID")
print("Bharat\t Bihar \t Patna")
print("Bharat\r Bihar \r Patna")
print("Bharat\b Bihar\b Patna")
print("Bharat\f Bihar\f Patna")
print("Bharat\v Bihar\v Patna")
print("Bharat \'Bihar\' Patna")
print("Bharat \"Bihar\" Patna")
print("Bharat \\Bihar\\ Patna")
```

Output

```
SKILLS
WIT
RID
Bharat      Bihar          Patna
Patna
Bhara Biha Patna
Bharat Bihar Patna
Bharat Bihar Patna
Bharat 'Bihar' Patna
Bharat "Bihar" Patna
Bharat \Bihar\ Patna
```

CONSTANTS & LITERALS

- Constant's concept is not applicable in python.
- But it is convention to use only uppercase characters if we don't want to change value.
- A=6
- It is convention but we can change the value.
- A constant is a variable whose value cannot be modified. Literals are raw values or data that are stored in a variable or constant.
- Constants or literals both are same. It is fixed.

❖ Types of Literals in Python:

- Python supports various types of literals, such as numeric literals, string literals, Boolean literals, and more. Let's explore different types of literals in Python with examples:
    1) String literals
    2) Character literal
    3) Numeric literals
    4) Boolean literals
    5) Literal Collections
    6) Special literals
    7) None literals
- Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, 'Hello, World!' , 12 , 23.0 , 'C' , etc.

OPERATOR

- operator is a symbol that represents an operation to be performed on one or more operands.
- It is used to manipulate data and perform various calculations or comparisons within expressions.

Example: arithmetic, comparison, logical, assignment, membership, identity, and bitwise operator.

1) Arithmetic Operators: Used for basic mathematical calculations.

1. **+** → (addition)
2. **-** → (subtraction)
3. ***** → (multiplication)
4. **/** → (division)
5. **%** → (modulo, remainder)
6. ****** → (exponentiation or power operator)
7. **//** → (floor division)

2) Comparison Operators or Relational Operators: Used to compare values.

1. **==** → (equal to)
2. **!=** → (not equal to)
3. **<** →(less than)
4. **>** →(greater than)
5. **<=** →(less than or equal to)
6. **>=** →(greater than or equal to)

3) Logical Operators: Used to combine and manipulate Boolean values.

1. **And** → (logical AND)
2. **or** → (logical OR)
3. **not** → (logical NOT)

4) Assignment Operators: Used to assign values to variables.

1. **=** → (assignment)
2. **+=** → (addition assignment)
3. **-=** → (subtraction assignment)
4. ***=** → (multiplication assignment)
5. **/=** → (division assignment)
6. **%=, **=, //=** → (other compound assignments)

5) Membership Operators: Used to test if a value is a member of a sequence (list, tuple, string).

1. **in**
2. **not in**

6) Identity Operators: Used to compare the memory locations of objects.

1. **is**
2. **is not**

7) Bitwise Operators: Used to perform bitwise operations on integers.

1. **&** → (bitwise AND)
2. **|** → (bitwise OR)
3. **^** → (bitwise XOR)
4. **~** → (bitwise NOT)
5. **<<** → (left shift)
6. **>>** → (right shift

ARITHMETIC OPERATORS

(+ - / % // * **)

Example:

```
a=3
b=6
print(a+b)  → 9
print(a-b)  → -3
print(a*b)  → 18
print(a/b)  → 0.5
print(a//b) → 0
print(a%b)  → 3
print(a**b) →729
```

Output

```
a=3.3
b=6
print(a+b)  → 9.3
print(a-b)  → -2.7
print(a*b)  → 19.79
print(a/b)  → 0.549
print(a//b) → 0.0
print(a%b)  → 3.3
print(a**b) → 1291.4679
```

Output

Note: + operator is also used to add two string that is known as concentration and * are used to multiple string.

Example:

```
a='twk'
b='saa'
abc=3
c=a+b
d=a*abc
print(c)
print(d)
```

Output

```
twksaa
twktwktwk
```

Example:

```
a='bharat'
print(a*3,'Bihar')
```

Output

```
BharatBharatBharat Bihar
```

Example:

```
a='twk'
b='saa'
c=a*b
print(c)
```

Output

TypeError: can't multiply sequence by non-int of type 'str'

Note:

- / Operator always performs floating point arithmetic. Hence it will always return float value.
- But floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If at least one argument is float type, then result is float type.

Example:

```
Print (6/3)
Print (6//3)
Print (6.0/3)
Print (6.0//3)
```

Output

```
2.0
 2
2.0
2.0
```

Note:

- For any number a, a/0 and a%0 always raise "ZeroDivisionError''

Example:

```
>>> 6/0  → ZeroDivisionError: division by zero
>>> 6%0  → ZeroDivisionError: division by zero
```

Example:

```
Num1=eval(input("Enter the First Number: "))
Num2=eval(input("Enter the Second Number: "))
sum=Num1+Num2
print("Sum of two number=",sum)
sub=Num1-Num2
print("Subtraction of two number=",sub)
Mul=Num1*Num2
print("Multiplication of two number=",Mul)
Div=Num1/Num2
```

```
print("Division of Num1 and Num2=",Div)
Modulo=Num1%Num2
print("Remainder of Num1 and Num2=",Modulo)
power=Num1**Num2
print("Power of Num1 and Num2=",power)
floor_division=Num1//Num2
print("Floor division of Num1 and Num2=",power)
Cube1=Num1**3
print("Cube of Num1=",Cube1)
Cube2=Num2**3
print("Cube of Num2=",Cube2)
```

Output:

```
Enter the First Number: 3
Enter the Second Number: 6
Sum of two number= 9
Substation of two number= -3
Multiplication of two number= 18
Division of Num1 and Num2= 0.5
Remainder of Num1 and Num2= 3
Power of Num1 and Num2= 729
Floor division of Num1 and Num2= 729
Cube of Num1= 27
Cube of Num2= 216

Enter the First Number: 3.5
Enter the Second Number: 9
Sum of two number= 12.5
Substation of two number= -5.5
Multiplication of two number= 31.5
Division of Num1 and Num2= 0.3888888888888889
Remainder of Num1 and Num2= 3.5
Power of Num1 and Num2= 78815.638671875
Floor division of Num1 and Num2= 78815.638671875
Cube of Num1= 42.875
Cube of Num2= 729

Enter the First Number: 30
Enter the Second Number: 0
Sum of two number= 30
Substation of two number= 30
Multiplication of two number= 0
ZeroDivisionError
Cell In[10], line 9
     7 Mul=Num1*Num2
     8 print("Multiplication of two number=",Mul)
----> 9 Div=Num1/Num2
    10 print("Division of Num1 and Num2=",Div)
    11 Modulo=Num1%Num2
ZeroDivisionError: division by zero
```

COMPARISON OPERATORS

(== != >, >=, <, <=)

Example:

```
a=3
b=6
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a>=b)
print(a<=b)
```

Output

→ False
→ True
→ False
→ True
→ False
→ True

- We can use apply relational operators for str types also.

Example:

```
a='twksaa'
b='twksaa'
print(a>b)
print(a>=b)
print(a<b)
print(a>=b)
```

Output

→ False
→ False
→ True
→ False

Example:

```
print(True>True)
print(True>=True)
print(True<True)
print(True>=True)
print(False>False)
print(False>=False)
print(False<False)
print(False>=False)
print(True>False)
print(True>=False)
print(True<False)
print(True>=False)
print(False>True)
print(False>=True)
print(False<True)
print(False>=True)
```

Output

→ False
→ True
→ False
→ True
→ False
→ True
→ False
→ True
→ True
→ True
→ False
→ True
→ False
→ False
→ True

Note: Chaining of relational operators is possible. in the chaining if all comparisons return True then only result is True. if at least one comparison returns False then the result is False.

Example:

```
>>> 3<6
True
>>> 3<6<9
True
>>> 3<6<9>12
False
```

LOGICAL OPERATORS

(and, or, not)

- Logical operators are used to perform logical operations on Boolean values (True or False).
- Python has three main logical operators:
  1. **and** →Returns True if both statements are true Ex: x < 5 and x < 10 (True)
  2. **or** →Returns True if one of the statements is true Ex: x < 5 or x < 4 (True)
  3. **not** →Reverse the result, returns False if the result is true Ex: not (x < 5 and x < 10)

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Truth Table

A	B
0	1
1	0

Note: These all-tree operators allow to combine Boolean values or expressions to create more complex conditions.

➢ **For Boolean types behavior:**
- True and False → False
- True or False → True
- not False →True

➢ **For non-Boolean types behavior:**
- 0 means False
- Non-zero means True
- Empty string is always treated as False

▪ **Details Explanation with example:**

1. and Operator:
- Syntax: expression1 and expression2
- Returns True if both expression1 and expression2 are True, otherwise returns False.

Example:

```
a, b=18,25
result=a>10 and b<30
print(result) #True because both conditions are true
result=a>10 and b<20
print(result)#False because b conditions is False
result=a>20 and b<20
print(result)#False False because both conditions are False
result=a>20 and b<30
print(result)#False because a condition is False
```

Output:
```
True
False
False
False
```

Example:

```
a=True
b=False
print(a and b)
c=True
d=True
print(c and d)
e=False
f=True
print(e and f)
g=False
h=False
print(g and h)
```

Output:

```
False
True
False
False
```

Example:

```
a=1
b=1
print(a and b)
c=1
d=0
print(c and d)
e=0
f=1
print(e and f)
g=0
h=0
print(g and h)
```

Output:

```
1
0
0
0
```

2. or Operator:

- Syntax: expression1 or expression2
- Returns True if either expression1 or expression2 (or both) is True, otherwise returns False.

Example:

```
a, b=18,25
result=a>10 or b<30
print(result)
result=a>10 or b<20
print(result)
result=a>20 or b<20
print(result)
result=a>20 or b<30
print(result)
```

Output:

```
True
True
False
True
```

Example:

```
a=1
b=0
print(a or b)
c=1
d=0
print(c or d)
e=0
f=1
print(e or f)
g=0
h=0
print(g or h)
```

Output:

```
1
1
1
0
```

3. not Operator:

- Syntax: not expression
- Returns the opposite of the Boolean value of expression.

Example:

```
x = 6
result = not (x > 0)  # False because the condition is true, but 'not' makes it false
```

Note: Logical operators are fundamental tools for building complex decision-making and control structures in Python programs.

ASSIGNMENT OPERATORS

(=, +=, -=, *=, /=, //=, %=, **=)

- Assignment operator is a symbol used to assign a value to a variable.
- It allows to modify value of a variable by performing an operation on existing value.
- It combines assignment operation with another operation, such as addition, subtraction, etc.

1). = (Assignment Operator):

- **Syntax:** variable = expression
- Assigns the value of the expression to the variable.

Example: x = 6

2). += (Add and Assign Operator):

- **Syntax:** variable += expression
- Adds value of expression to current value of variable and assigns result back to variable.

Example: x = 6
x += 3 →Equivalent to x = x + 3

3). -= (Subtract and Assign Operator):

- **Syntax:** variable -= expression
- Subtracts value of expression from current value of variable assigns result back to variable.

Example: y = 10
y -= 2 →Equivalent to y = y - 2

4). *= (Multiply and Assign Operator):

- Syntax: variable *= expression
- Multiplies current value of variable by value of expression and assigns result back to variable

Example: z = 3
z *= 4 → Equivalent to z = z * 4

5). /= (Divide and Assign Operator):

- **Syntax:** variable /= expression
- Divides current value of **variable** by value of `expression` and assigns result back to variable

Example: a = 15
a /= 5 → Equivalent to a = a / 5

6). //= (Floor Division and Assign Operator):

- **Syntax:** variable //= expression
- Performs floor division on the current value of `variable` by the value of `expression` and assigns the result back to `variable`.

Example: b = 20
b //= 3 → Equivalent to b = b // 3

7). %= (Modulus and Assign Operator):

- **Syntax:** variable %= expression
- Computes the modulus (remainder) of the current value of `variable` divided by the value of `expression` and assigns the result back to `variable`.

Example: c = 13
c %= 5 →Equivalent to c = c % 5

8). **= (Exponentiation and Assign Operator):

- **Syntax:** variable **= expression

- Raises current value of variable to power of `expression` and assigns result back to variable

Example: d = 2
d **= 3 →Equivalent to d = d ** 3

Example:

```
a=10
a+=20 # a=a+20
print("value of a=",a)
b=10
b-=20 # b=b-20
print("value of b=",b)#-10
c=10
c*=20 #c=c*20
print("value of c=",c) #200
d=10
d/=5 #d=d/5
print("value of d=",d) #2.0
e=10
e//=5 #e=e//5
print("value of e=",e) #2
f=10
f%=5 #f=f%5
print("value of f=",f) #0
g=12
g%=5 #g=g%5
print("value of g=",g) #2
h=3
h**=2 # h=h**2
print("value of h=",h) #9
```

Output:

```
value of a= 30
value of b= -10
value of c= 200
value of d= 2.0
value of e= 2
value of f= 0
value of g= 2
value of h= 9
```

Example:

```
# Assignment operator =
x = 5
print("x =", x)
# Addition assignment operator +=
x += 3
print("x += 3:", x)
# Subtraction assignment operator -=
x -= 2
print("x -= 2:", x)
# Multiplication assignment operator *=
x *= 4
print("x *= 4:", x)
# Division assignment operator /=
x /= 2
print("x /= 2:", x)
# Modulus assignment operator %=
x %= 3
print("x %= 3:", x)
# Exponentiation assignment operator **=
x **= 2
print("x **= 2:", x)
# Floor division assignment operator //=
x //= 2
print("x //= 2:", x)
```

Output:

```
x = 5
x += 3: 8
x -= 2: 6
x *= 4: 24
x /= 2: 12.0
x %= 3: 0.0
x **= 2: 0.0
x //= 2: 0.0
```

Note:

- Assignment operators in Python are used to assign values to variables. They allow you to update the value of a variable based on its current value and the value on the right-hand side of the operator. Assignment operators are a fundamental part of programming and are commonly used to store and manipulate data.

MEMBERSHIP OPERATORS

(in, not in)→ used to test a value

- Membership operators (in and not in) are used to test if a value is present or absent in a sequence or collection.
- These operators are used with strings, lists, tuples, dictionaries, and other iterable objects.
- **in** →it is return True if the given object presents in the specified collection.
- **not in** → it is return True if the given object not present in the specified collection.

Example

```
s='twksaa skills center'
print('t' in s)              → True
print('skills' in s)         → True
print('A' in s)              → False
print('cen' not in s)        → False
print('t' not in s)          → False
print('center' not in s)     → False
```

Output

IDENTITY OPERATOR

(is, is not)→ compare the memory locations

- identity operators are used to compare the memory locations of two objects to determine if they are the same object or not.
- The two main identity operators are is and is not.
  1. **is operator:** Returns True if both operands refer to the same object in memory, and False otherwise.
  2. **is not operator:** Returns True if both operands do not refer to the same object in memory, and False otherwise.

Example-1

```
a=6
b=6
print(id(a)) →140710794814408
print(id(b)) →140710794814408
print(a is b)→True
```

Output

Example-2

```
a='TWKSAA'
b='TWKSAA'
print(id(a)) →2356843061872
print(id(b)) →2356843061872
print(a is b) →True
```

Output

Example-3

```
a='skills'
b='skills'
print(id(a)) →1917183861360
print(id(b)) →1917183861360
print(a is not b) →False
```

Output

Example-4

```
l1=[30,9,23]
l2=[30,9,23]
print(id(l1)) →1666708898944
print(id(l2)) →1666664015488
print(l1 is l2) →False
print(l1 is not l2) →True
print(l1 == l2) →True
```

Output

Example-5

```
l1=['one', 'two', 'three']
l2=['one', 'two', 'three']
print(id(l1)) →2387608831168
print(id(l2)) →2387563947648
print(l1 is l2) →False
print(l1 is not l2) →True
print(l1 == l2) →True
```

Output

Note: we can use is operator for address comparison whereas == operator for content comparison.

Example:

```
s="Bharat is a great country"
l=len(s)
print(l)
print("is" in s)#True
print("is"not in s)#False
print("Is" in s)#False
print("Is"not in s)#True
c= "gre" in s
print(c)
d="great" in s
print(d)
e="country" not in s
print(e)
e="countryr" not in s
print(e)
```

Output:

```
25
True
False
False
True
True
True
False
True
```

Example:

```
List=[1,2,3,4,5,'skils', "good"]
l=len(List)
print(l)
print(1 in List)
print(6 not in List)
print(3 not in List)
print('s' in List)
print('skils' in List)
print('g' in List)#False
print('good' in List)
```

Output:

```
7
True
True
False
False
True
False
True
```

Example:

```
s1="good"
print('g' in s1)
print('o' in s1)
print(len(s1))
print(type(s1))
l1=["good"]
print('g' in l1)
print('o' in l1)
print(len(l1))
print(type(l1))
```

Output:

```
True
True
4
<class 'str'>
False
False
1
<class 'list'>
```

Example:

```
abc=25
ecr=25
pnb=25
print(id(abc), id(ecr),id(pnb))
result= abc is ecr
print(result)
result= abc is pnb
print(result)
result= pnb is ecr
print(result)
result= pnb is not ecr
print(result)
```

Output:

```
140255737152496
140255737152496
140255737152496
True
True
True
False
```

Example:

```
abc1=25
ecr1=250
pnb1=125
print(id(abc1), id(ecr1),id(pnb1))
print(abc1 is not ecr1)
print(abc1 is ecr1)
```

Output: 737152496 737159696 7155696

```
True
False
```

BITWISE OPERATOR

(&, |, ^, ~, <<, >>)

- Bitwise operators are used to perform operations at the bit level on integers.
- These operators are applicable only for int and Boolean types.
- By mistake if we are trying to apply for any other type then we will get Error.
- **&** → if both bits are 1 then only result is 1 otherwise result is 0.
- **|** → if at least one bit is 1 then result is 1 otherwise result is 0.
- **^** → if bits are different then only result is 1 otherwise result is 0.
- **~** → bitwise complement operator 1 →0 & 0→1
- **<<** →Bitwise left shift
- **>>** →Bitwise right shift

BITWISE OPERATOR TRUTH TABLES

AND "&"

INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

OR "|"

INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

XOR "^"

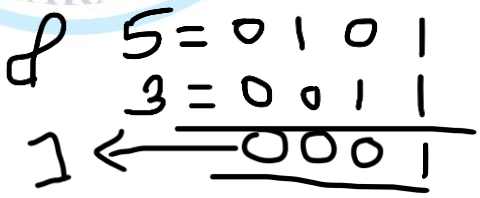
INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

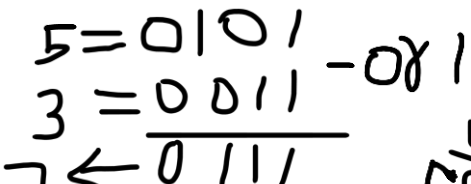
NOT "~"

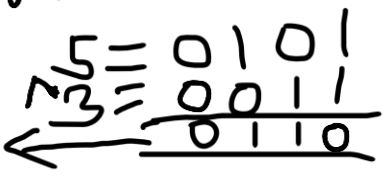
INPUT	OUTPUT
0	1
1	0

Example-1

a = 5 →0101 in binary
b = 3 → 0011 in binary
print(a & b) →Bitwise AND: 0001 (1 in decimal)
print(a | b) → Bitwise OR: 0111 (7 in decimal)
print(a ^ b) →Bitwise XOR: 00110 (6 in decimal)
print(~a) → Bitwise NOT: 1010 (-6 in decimal)
print(a << 1) → Left Shift: 1010 (10 in decimal)
print(b >> 1) →Right Shift: 0001 (1 in decimal)







Note: How to convert Binary to decimal numbre

- Binary_Numbar=int("1010101",2)
- Print("Decimal_Numbar=",Binary_Number)

➢ Bitwise Complement Operator (~):

- We have to apply complement for total bits.

Example	Output
print(~3)	→ -4
print(~5)	→ -6
print(~6)	→ -7
print(~0)	→ -1
print(~1)	→ -2
print(~2)	→ -3

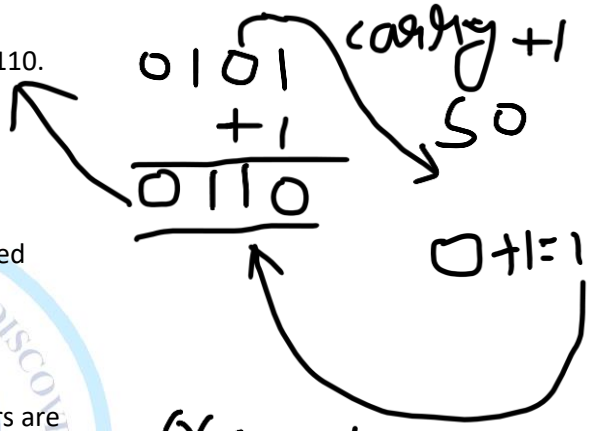
Note:

- the most significant bits acts as sign bit. 0 value represents **+Ve** number where as 1 represents **-Ve** value.
- Positive numbers will be represented directly in the memory whereas negative number will be represented indirectly in 2's complement for

❖ print(~5) → -6

❖ Explanation:

- The binary representation of 5 is 0101.
- after apply the bitwise NOT operator you get 1010.
- Python uses two's complement representation for signed integers.
- note: (signed integers are a way to represent both positive and negative whole numbers. They include zero, all the positive whole numbers, and all the negative whole numbers.
- range -2,147,483,648 to 2,147,483,647.)
- In two's complement, inverting the bits also involves negating the number. So, when you **invert 1010, you get -1010 in binary.**
- convert this binary representation to decimal, you can use the two's complement method:
    1. Invert all the bits: -1010 becomes -0101.
    2. Add 1 to the inverted number: -0101 + 1 = -0110.
- So, the final result is 0110, which is 6 in decimal.
- Therefore, the code print(~a) will output -6



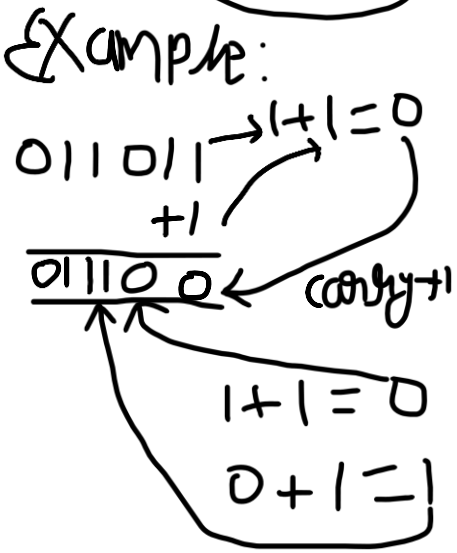
➢ In digital computing, one's complement and two's complement are two different ways to represent signed integers (positive and negative numbers) using binary numbers.

❖ One's Complement:

- In one's complement representation, positive numbers are represented as usual in binary, but negative numbers are obtained by inverting (changing 0s to 1s and 1s to 0s) all the bits of the corresponding positive number.
- For example, if you have the 8-bit binary representation of 5 as 00000101, then the one's complement representation of -5 would be 11111010.

❖ Two's Complement:

- Two's complement is the most common way to represent signed integers in modern digital computers.
- In two's complement representation, positive numbers are represented as usual in binary, but negative numbers are obtained by taking the one's complement of the positive number and then adding 1 to the result.
- For example, if you have the 8-bit binary representation of 5 as 00000101, then the two's complement representation of -5 would be 11111011.



Note:

- If You want to add 1 to this binary number. When adding binary numbers, you start from the rightmost bit (the least significant bit) and move towards the left, just like when you add decimal numbers.
- The rightmost bits 1 + 1 equals 0, with a carry of 1.

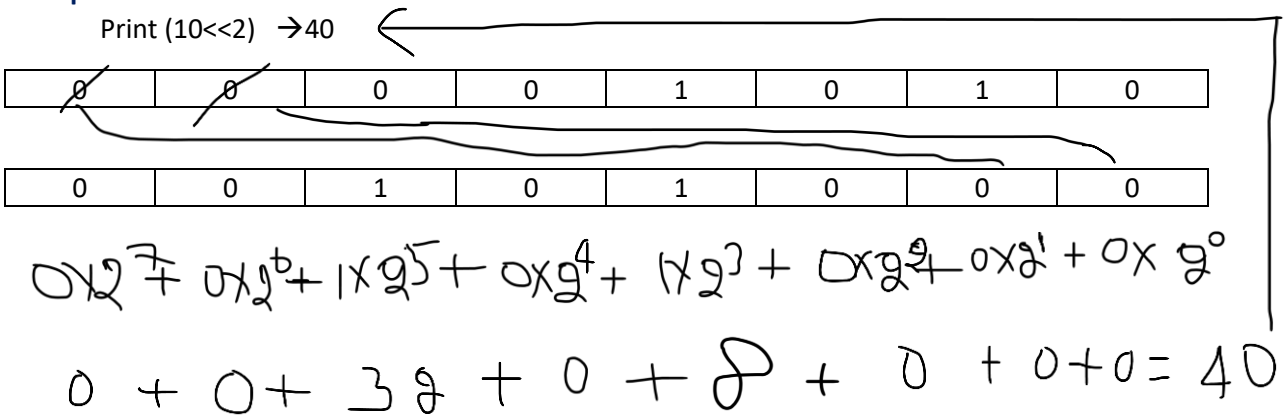
SHIFT OPERATOR

- **<< Left shift Operator:**

- After shifting the empty cells, we have to fill with zero.

Example:

Print (10<<2) →40

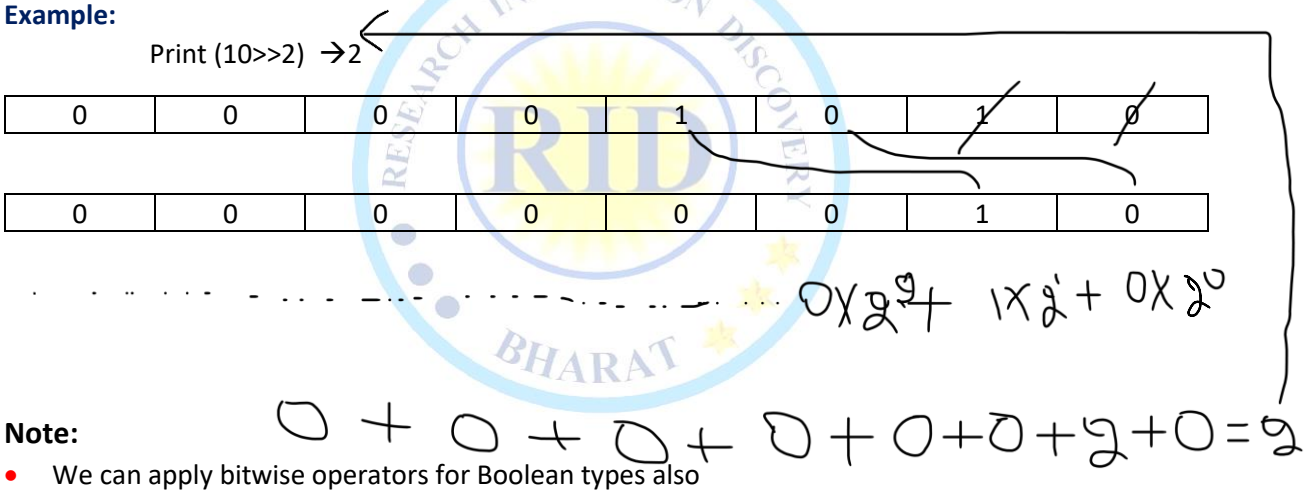


- **>> Right shift Operator:**

After shifting the empty cells, we have to fill with sign bit. (0 for **+Ve** and 1 for **-Ve**)

Example:

Print (10>>2) →2



Note:

- We can apply bitwise operators for Boolean types also

Example	Output
print(True & False)	→ False
print(True | False)	→ True
print(True ^ False)	→ True
print(~True)	→ -2
print(True <<2)	→ 4
print(True >>2)	→ 0

❖ Ternary Operator or Conditional operator:

- Syntax: a=first value if condition else second value.
- If condition is True then first value will be considered else second value will be considered.

Example:

```
a, b=10,20
x=30 if a<b else 40
print(x)  →30
```

OPERATOR PRECEDENCE

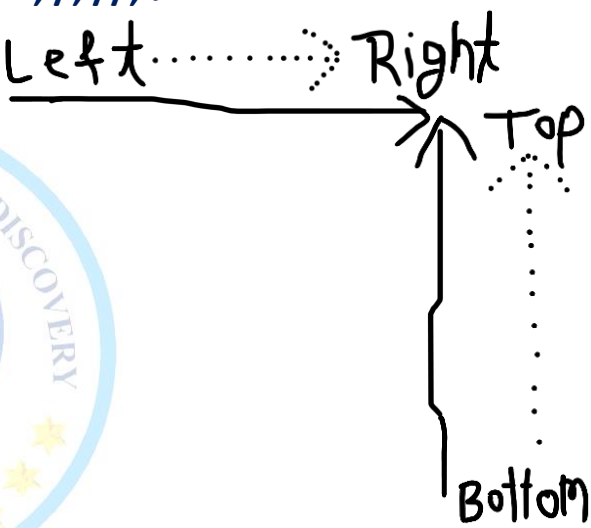
- If multiple operators present then which operator will be evaluated first is decided by operator precedence.

  Example:

  - print (3+6*2) →15
  - print ((3+6)*2) → 18

- The following list describes operator precedence in python:

1. Parentheses: **()**
2. Exponentiation: ******
3. Bitwise Complement operator, unary Minus Operator: **~, -**
4. Multiplication, Division, Floor Division, Modulus: ***, /, //, %**
5. Addition and Subtraction: **+, -**
6. Bitwise Shifts: **<<, >>**
7. Bitwise AND: **&**
8. Bitwise XOR: **^**
9. Bitwise OR: **|**
10. Comparison Operators: **<, <=, >, >=, ==, !=**
11. Membership Operators: **in, not in**
12. Identity Operators: **is, is not**
13. Logical NOT: **not**
14. Logical AND: **and**
15. Logical OR: **or**
16. Conditional Expression (Ternary Operator): **x if c else y**
17. Assignment Operators: **=, +=, -=, *=, /=, //=, %=, **=, &=, |=, ^=, <<=, >>=**
18. Comma: **,**



Note: parentheses can always be used to control the order of evaluation and explicitly specify how expressions should be grouped.

Example

```
a,b,c,d=30,20,10,5
print((a+b)*c/d)      → 100.0
print((a+b)*(c/d))    → 100.0
print(a+(b*c)/d)      → 70.0
```

Output

INPUT OUTPUT STATEMENT

INPUT STATEMENT

- Reading dynamic input from the keyboard:
- To read the input from user, input () used.

1. **input ()** is reading entire line of data in string format.

Example

```
a=input("enter the number: ")
print(type(a),a)
b=int(input("Enter the number: "))
print(type(b),b)
c=float(input("Enter the number: "))
print(type(c),c)
```

Output

```
→ enter the number: 3
<class 'str'> 3
→Enter the number: 6
<class 'int'> 6
→Enter the number: 30.9
<class 'float'> 30.9
```

2. **eval ()**:- this function is worked for both int and float data types of value.

Example

```
a=input("Enter the number: ")
print(type(a),a)
b=eval(input("Enter the number: "))
print(type(b),b)
c=eval(input("Enter the number: "))
print(type(c),c)
```

Output

```
→ Enter the number: 6
<class 'str'> 6
→ Enter the number: 12
<class 'int'> 12
→ Enter the number: 30.9
<class 'float'> 30.9
```

3. split ()

- split () method is used to split a string into a list of substrings based on a specified delimiter.
- delimiter is a character or sequence of characters that determines where string should be split.
- It provides a convenient way to break down a string into meaningful
- The split () method is commonly used when processing text data, such as reading and parsing CSV files, log files, and other structured or semi-structured text formats.

Example

```
data=input("Enter Some Data: ")
print(data)
c=data.split()
print(type(c), c)
data=input("Enter Some Data: ")
print(data)
c=data.split()
print(type(c), c)
```

Output

```
→ Enter Some Data: 3
3
<class 'list'> ['3']
→Enter Some Data: TWKSAA
TWKSAA
<class 'list'> ['TWKSAA']
```

4. Map ()

- The map () function in Python is a built-in function that allows you to apply a given function to every item in an iterable (such as a list, tuple, or string) and returns an iterator (map object) containing the results.

Syntax:

- map (function, iterable, ...)

function: This is the function that you want to apply to each element of the iterable. It can be a built-in function, a user-defined function, or a lambda function.
iterable: This is the collection of items that you want to process using the provided function.

Use:

- map () is used to apply specific function to every element to within a sequence.
- It is also used for take multiple input from user in same line.

❖ How to read multiple values from the keyboard in a single line:

➢ By using map() and split() you can read multiple values from user.

Example:

```
a, b, c=map (int, input ("Enter the Number: "). split ())
print(a)
print(b)
print(c)print(type(c), c)
```

Output

```
→ Enter the Number: 30 9 2023
30
9
2023
```

Note: Split () function can take space as separator by default. But you can pass anything as separator.

Example-1:

```
a=input("enter the number-1")
b=input("enter the number-2")
print(type(a))
print(type(b))
sum=a+b
print(sum)
```

Output:

```
enter the number-1 10
enter the number-2 20
<class 'str'>
<class 'str'>
1020
```

Example-2:

```
a=input("enter the number-1")
b=input("enter the number-2")
sum=a+b
print(sum)
```

Output:

```
enter the number-1 twk
enter the number-2 saa
twksaa
```

Example-3:

```
a=int(input("enter the number-1"))
b=int(input("enter the number-2"))
print(type(a))
print(type(b))
sum=a+b
print(sum)
```

Output:

```
enter the number-1 10
enter the number-2 20
<class 'int'>
<class 'int'>
30
```

Example-4:

```
a=input("enter things").split()
print(a)
print(len(a))
print(type(a))
```

Output:

```
enter things 10 10.3 0b100 skills
['10', '10.3', '0b100', 'skills']
4
<class 'list'>
```

Example-5:

```
# a,b,c=eval(input("enter the 3 number"))
# print(a,b,c) #SyntaxError: invalid syntax
a,b,c=map(int,input("enter the  3
number:").split())
print("value a=",a)
print("value b=",b)
print("value c=",c)
sum=a+b+c
print("sum=",sum)
```

Output:

```
enter the  3 number: 10 20 30
value a=10
value a=20
value a=30
sum=60
```

Example-6:

```
a=eval(input("enter the number-1"))
b=eval(input("enter the number-2"))
print(type(a))
print(type(b))
sum=a+b
print(sum)
```

Output:

```
enter the number-1 10
enter the number-2 20.5
<class 'int'>
<class 'float'>
30.5
```

OUTPUT STATEMENT

- We can use print() function to display output.

5. Print ():

- print () function in Python is a built-in function used to display output on console or terminal.
- By default, print() will be terminated current line with new line to change it, you need to used **"end"** option.

Example

```
a,b,c=30,9,23        Output
print(a)             → 30
print(b)             → 9
print(c)             → 23
print(a,end=" ")     → 30 9
print(b)             → 23
print(c)
```

Example:

```
print("skills",end=" ")
print("center",end=" ")
print("wit",end='*')
print("rid")
```

Output:

skills center wit*rid

Example:

```
a,b,c=10,20,30
print(a,b,c,)
print(a,b,c,sep="- ")
print(a,b,c,sep="=")
```

Output:

10 20 30
10- 20- 30
10=20=30

- By default within print() all the arguments will print() separated with spaces to change it **"sep"** option will used.

Example

```
a,b,c=30,9,2023
print(a,b,c, sep="-")   → 30-9-2023
```

Output

6. Form-1:

- Print() without any argument. Just it prints new line character

Example

```
print("TWKSAA")
print('twksaa skills center')
#we can use escape characters also
print("Foundation Day\n30-09-2023")
print("Foundation Day\t30-09-2023")
#we can use repetition operator (*) in the string
print(3*'TWKSAA')
print('SKILLS'*3)
print("CENTER"*3)
#we can use + operator also
print("TWK" + "SAA")
```

Output

→ TWKSAA
→twksaa skills center
→Foundation Day
→30-09-2023
→Foundation Day 30-09-2023
→TWKSAATWKSAATWKSAA
→SKILLSSKILLSSKILLS
→CENTERCENTERCENTER
→TWKSAA

Note:

- If the both arguments are string then + operator acts as concatenation operator.
- If one argument is string type and second is any other type like int then we will get Error.
- If both arguments are number type, then + operator acts as arithmetic addition operator.

Note:

```
print('TWK' + 'SAA')
print("TWK" , "SAA")
```

TWKSAA
TWK SAA

7. Form-2: print() with variable number of arguments.

Example

```
a, b, c=30,9,23
print("The value are:", a, b, c)   → The value are: 30 9 23
```

Output

8. **Form-3:** print() with end attribute

Example **Output**

print("SKILLS") → SKILLS
print("WIT") →WIT
print("RID") →RID

- If we want output in the same line with space.

Example

```
print("TWKSAA", end=' ')
print("SKILLS", end=' ')
print("CENTER")
```

Output

→ TWKSAA SKILLS CENTER

Note: The default value for end attribute is \n, which is nothing but new line character.

9. **Form-4:** print(object) statement

➔ We can pass any objects (like list, tuple, set etc.) as argument to the print() statement.

Example

```
l=[10,20,30,40,10]
t=(10,20,30,40,10)
s={10,20,30,40,10}
```

Output

print(l) → [10, 20, 30, 40, 10]
print(t) →(10, 20, 30, 40, 10)
print(s)) →{40, 10, 20, 30}

10. **Form-5:** print(string, variable list)

- We can print() statement with string and any number of arguments

Example

```
s="skills center"
a=30_9_23
s1="HTML"
s2="Pyhton"
print("twksaa", s, "Foundation day is", a)
print("you are techning", s1, "and", s2)
```

Output

→ twksaa skills center Foundation Day is 30923
→you are techning HTML and Pyhton

11. **Form-6:** print(formatted string)

1) %i →int
2) %d → int
3) %f → float
4) %s → string type

Syntax: print("formatted string", %(variable list)

Example-1

```
a=3
b=6
c=9
print("a value is %i" %a)
print("b value is %d and c value is %d" %(b,c))
```

Example-2

```
s="TWKSAA"
list=[10,20,30]
print("Hello %s...the list of items are %s" %(s,list))
```

Output

→ a value is 3
→b value is 6 and c value is 9
→Hello TWKSAA...the list of items are [10, 20, 30]

12. Form-7: print() with replacement operator{}

Example:

```
name="RAM"
salary=10000
w="developer"
print("Hello {0} your salary is {1} and your friend{2} is wating". format(name,salary,w))
print("Hello {x} your salary is {y} and your frined {z} is waiting".format(x=name,y=salary,z=w))
```

Output

```
Hello RAM your salary is 10000 and your frienddeveloper is wating
Hello RAM your salary is 10000 and your frined developer is waiting
```

Example-1:

```
print("skills") →skills
a=int(input("enter the number"))
print(a)
```

Output:

```
enter the number 39
39
```

Example-2:

```
name=input("enter the name")
print(name)
print("Name=",name, "is a student")
```

output:

```
enter the name Sangam Kumar
Sangam Kumar
Name= Sangam Kumar is a student
```

Example-3:

```
print("Name",end=" ")
print("Roll_No",end=" ")
print("Marks")
```

output:

```
Name Roll_No Marks
```

Example-4:

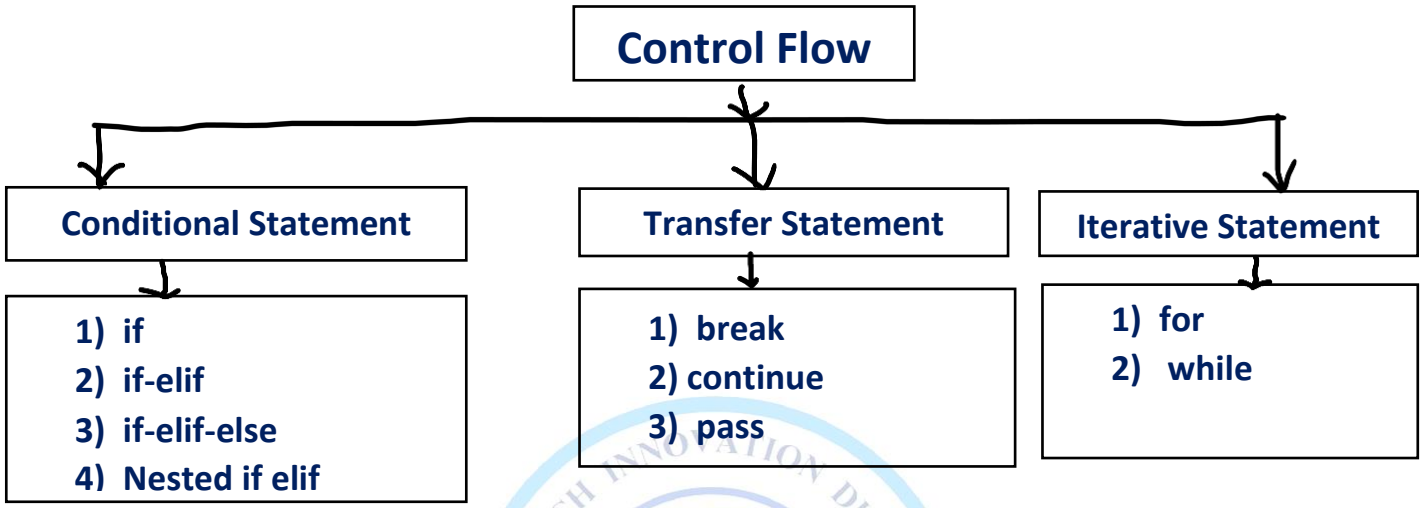
```
Name="Sangam Kumar"
age="15"
Roll_no="30"
marks=99
print("{} is a student his age is {} and his roll number is {} and marks is
{}".format(Name,age,Roll_no,marks))
print("{0} is a student his age is {1} and his roll number is {2} and marks is
{3}".format(Name,age,Roll_no,marks))
print("{a} is a student his age is {b} and his roll number is {c}".format(a=Name,b=age,c=Roll_no))
print("{x} is a student his age is {y} and his roll number is {z}".format(x=Name,y=age,z=Roll_no))
```

Output:

```
Sangam Kumar is a student his age is 15 and his roll number is 30 and marks is 99
Sangam Kumar is a student his age is 15 and his roll number is 30 and marks is 99
Sangam Kumar is a student his age is 15 and his roll number is 30
Sangam Kumar is a student his age is 15 and his roll number is 30
```

CONTROL STATEMENT

- Control Statement or Flow control describe the order in which statements will be executed at runtime.
- To change the programing follow based on the condition we will used control statement.
- This is used for Decision making.



❖ Indentation in Python:

- python doesn't allow the use of Curly Braces {} for the block level code.
- In Python, indentation (:) is used to declare a block.
- If two statements are at same indentation level, then they are the part of the same block.
- Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation.

❖ Sequential statement:

- In case of sequential statement in which ordered you represent in the program that same those statements will get execute.

Example in Python block of code

```
def greet():
    print("Hello!")
    print("Welcome to Python.")  # Indented block

if True:
    print("Condition is True")
    print("Executing the block of code")  # Indented block

for i in range(3):
    print("Loop iteration:", i)
    print("Inside the loop")  # Indented block
```

Example in C block of code

```
#include <stdio.h>
void greet() {
    printf("Hello!\n");
    printf("Welcome to C.\n");  // Block enclosed in {}
}
int main() {
    if (1) {
        printf("Condition is True\n");
 printf("Executing the block of code\n");  // Block enclosed in {}
    }
    for (int i = 0; i < 3; i++) {
        printf("Loop iteration: %d\n", i);
        printf("Inside the loop\n");  // Block enclosed in {}
    }
```

CONDITIONAL STATEMENT

Based on some condition program will execute a block of statement or you may skip a block of statement.

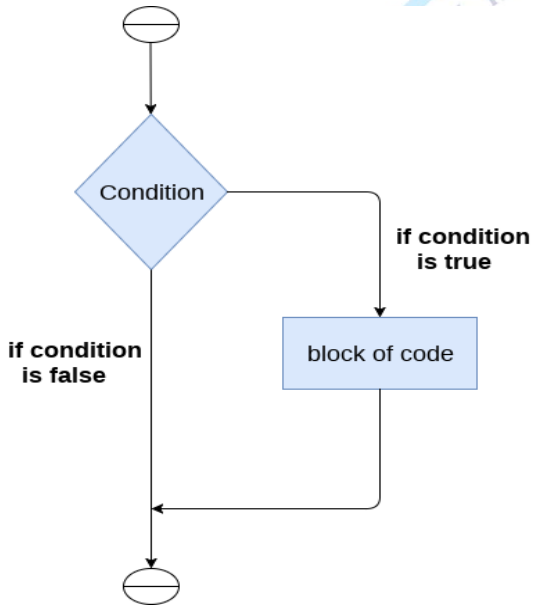
❖ Types of Conditional Statements:

1) Simple if.
2) if else.
3) elif ladder.
4) nested if else.

1). Simple if:

- If the condition is true block of statement will execute otherwise skip.
- if statement is used to test a particular condition and if the condition is true.
- it executes a block of code known as if-block.
- The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

13. Flow Chart of simple if:



Syntax:

```
if expression:
    statement-1
    statement-2
    statement-3
    :
    statement-n
```

Example-1:

```
if True:
   print("TWKSAA SKILLS CENTER")
```

Output:

TWKSAA SKILLS CENTER

Example-2:

```
if False:
   print("TWKSAA SKILLS CENTER")
print("TWKSAA RID CENTER")
```

Output:

TWKSAA RID CENTER

➢ **Problem:** write a program to find the greatest among two numbers.

Program:

```
a=int(input("Enter the value of a: "))
b=int(input("Enter the value of b: "))
if a>b:
   print("value of a=",a)
if b>a:
   print("value of b=",b)
if a==b:
   print("value of a and b are equal")
```

Output:

```
Enter the value of a: 3
Enter the value of b: 6
value of b= 6

Enter the value of a: 6
Enter the value of b: 3
value of a= 6

Enter the value of a: 3
Enter the value of b: 3
value of a and b are equal
```

➢ **Problem:** write a program to check given number positive or negative.

Program:

```
a=int(input("Enter the value of a: "))
if a>0:
   print("A is positive number")
if a<0:
   print("A is Negative number")
if a==0:
   print("A is note positive & Negative")
```

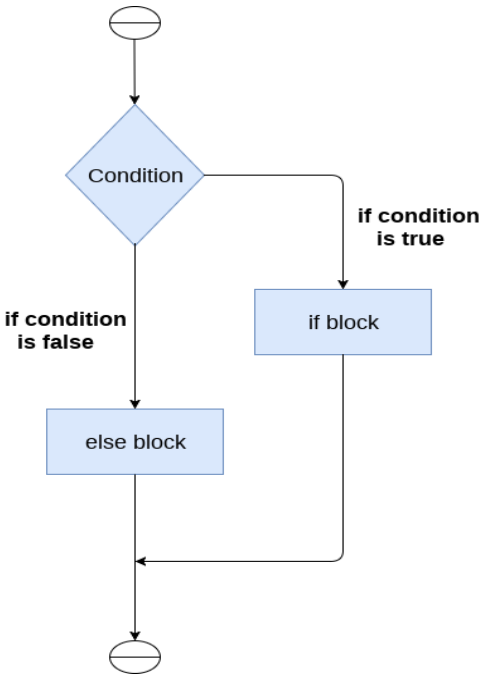
Output:

```
Enter the value of a: 6
A is positive number

Enter the value of a: -6
A is Negative number

Enter the value of a: 0
A is note positive & Negative
```

2). if-else statement:

- if-else statement provides an else block combined with the if statement.
- If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

❖ **Flow Chart of if-else Statement:**



Syntax:

```
if condition:  #(if-block)
   statement-1
   statement-2
   statement-3

   statement-n
else:  #(else-block)
   statement-1
   statement-2
   statement-3

   statement-n
```

Example-1: Write a Program to check greater among two number.

```
a=int(input("Enter the value of a= "))
b=int(input("Enter the value of b= "))
if a>b:
   print("a is greater")
else:
   print("b is greater")
```

Output:

```
Enter the value of a= 6
Enter the value of b= 3
a is greater
Enter the value of a= 3
Enter the value of b= 6
```

Example-2: Write a Program to check whether given number is even or not.

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

Output:

```
Enter the value of a= 39
39 is odd number
Enter the value of a= 30
30 is even number
```

Example-3: Write a Program to check whether given number is divisible by 3 or not

```
a=int(input("Enter the value of a= "))
if a%3==0:
   print(a,"is divisible by 3 ")
else:
   print(a, "is not divisible by 3")
```

Output:

```
Enter the value of a= 39
39 is divisible by 3
Enter the value of a= 53
53 is not divisible by 3
```

Example-4: Write a Program to check whether given number is divisible by 5 and 7.

```
a=int(input("Enter the value of a= "))
if a%5==0 and a%7==0:
   print(a,"is divisible by 5 and 7 ")
else:
   print(a, "is not divisible by 5 and 7")
```

Output:

```
Enter the value of a= 35
35 is divisible by 5 and 7
Enter the value of a= 49
49 is not divisible by 5 and 7
```

Example-5: Write a Program to check whether a person is eligible for vote or not (age is input)

```
age=int(input("Enter the age of a person: "))
if age>=18:
   print("Person is eligible for vote")
else:
   print("Person is not eligible for vote")
```

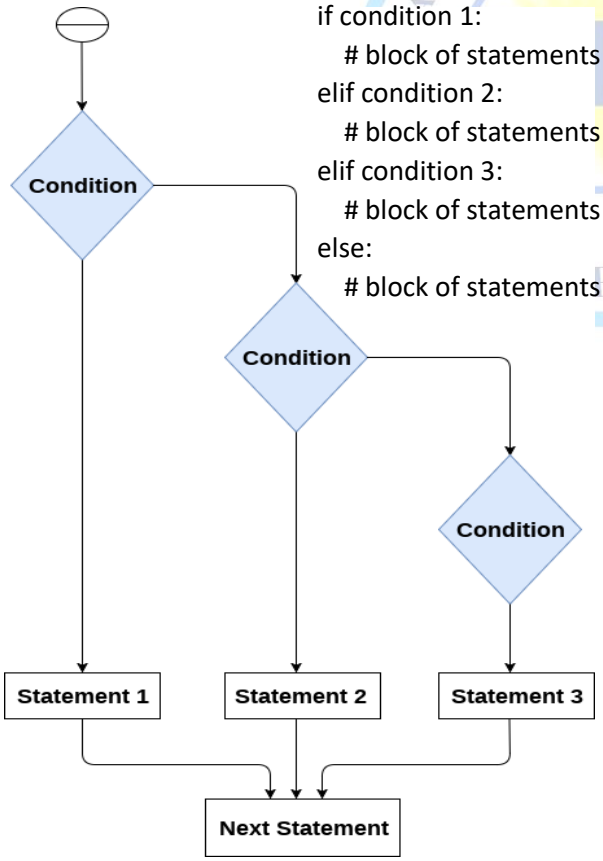
Output:

```
Enter the age of a person: 18
Person is eligible for vote
Enter the age of a person: 16
Person is not eligible for vote
```

3). elif ladder statement:

- The elif ladder statement check multiple conditions and execute the specific block of statements depending upon the true condition among them.
- elif statement works like an if-else-if ladder statement in C.

❖ Flow Chart of if-else Statement:



Syntax:

```
if condition 1:
   # block of statements
elif condition 2:
   # block of statements
elif condition 3:
   # block of statements
else:
   # block of statements
```

Example-1: Write a Program to find the greatest among three numbers.

```
a=int(input("Enter the value a= "))
b=int(input("Enter the value b= "))
c=int(input("Enter the value c= "))
if a==b and b==c:
   print("a b and c are equal ")
elif a>b and a>c:
   print(a," is greater")
elif b>a and a>c:
   print(b," is greater")
else:
   print(c,"is greater")
```

Output:

```
Enter the value a= 3
Enter the value b= 9
Enter the value c= 6
6 is greater

Enter the value a= 3
Enter the value b= 3
Enter the value c= 3
a b and c are equal

Enter the value a= 15
Enter the value b= 20
Enter the value c= 50
50 is greater
```

Example-2: Write a Program to display the students grade by reading the students marks.

```
marks=eval(input("Enter the students marks: "))
if marks>=90:
   print("Grade-A")
elif marks>=80:
   print("Grade-B")
elif marks>=70:
   print("Grade-C")
elif marks>=60:
   print("Grade-D")
else:
   print("Fail")
```

Output:

```
Enter the student's marks: 90
Grade-A
Enter the student's marks: 59
Fail
```

Example-4: Write a Program to display the below Result.

```
age=int(input(' Enter your age'))
salary=int(input('Enter your salary'))
if age>=21 and salary>=100000:
   print("you will marry with a super model")
elif age>=21 and salary>=75000 :
   print("you will marry with a Queen girl")
elif age>=21 and salary>=50000:
   print("you will marry with a beautiful girl")
elif age>=21 and salary>=25000:
   print("you will marry with a Simple girl")
else:
   print(" Bhagwan Bharose !!")
```

Output:

```
Enter your age 27
Enter your salary 5000000
you will marry with a super model
```

Example-3: Write a Program to check given character is lower case, uppercase, digit or other.

```
char=input("Enter any data: ")
if char>='a' and char <='z':
   print(char,'is a lower-case character')
elif char>='A' and char<='Z':
   print(char, 'is an upper-case character')
elif char>='0' and char<='9':
   print(char, ' is a digit')
else:
   print(char,'is other symbol')
```

Output:

```
Enter any data: b
b is a lower-case character
Enter any data: R
R is an upper-case character
Enter any data: # # is other symbol
```

Example-5: Write a Program to display all days Name.

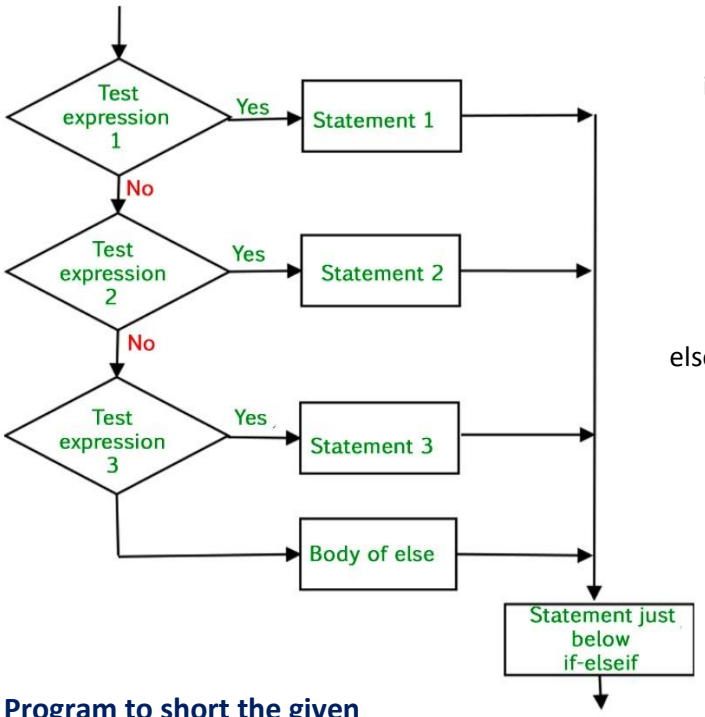
```
week = {1:"Monday", 2:"Tuesday", 3: "Wednesday", 4:
"Thursday", 5:"Friday", 6: "Saturday", 7:"Sunday"}
number=int(input("Please Enter Week Number (Between 1-7):"))
if number==1:
   print("Today is ", week[number])
elif number==2:
   print("Today is ", week[number])
elif number==3:
   print("Today is ", week[number])
elif number==4:
   print("Today is ", week[number])
elif number==5:
   print("Today is ", week[number])
elif number==6:
   print("Today is ", week[number])
elif number==7:
   print("Today is ", week[number])
else:
   print("pls enter valid credential number")
```

1. Write a Python program using an `elif` ladder to check if a number is positive,negative, or zero.
2. Write a Python program using an `elif` ladder to determine if a number is even or odd.
3. Write a Python program using an `elif` ladder to find the largest of three numbers.
4. Write a Python program using an `elif` ladder to categorize a student's score into grades:
5. Write a Python program using an `elif` ladder to check if a number is divisible by 2, divisible by 3.
6. Write a Python program using an `elif` ladder to determine if a year is a leap year or not.
7. Write a Python program using an `elif` ladder to calculate the discount on a purchase based on the following criteria: - 10% discount if the purchase amount is between 100 and 500.
   - 20% discount if the purchase amount is between 500 and 1000.
   - 30% discount if the purchase amount is above 1000.
8. Write a Python program using an `elif` ladder to determine the type of triangle based on its sides:
9. Write a Python program using an `elif` ladder to calculate the electricity bill based on the following criteria:
   - First 100 units: ₹5 per unit. Next 100 units: ₹7 per unit. Above 200 units: ₹10 per unit.

4). Nested if else statement:

- The nested if statements are the nesting of an if statement inside another if statement with or without an else statement.

❖ Flow Chart of if-else Statement:



Syntax:

```
if condition:
    statement-1
    statement-2
  if condition:
     statement-1
      statement-2
       statement-3
    if condition:
          statement-1
          statement-2
else:
    if condition:
          statement-1
           statement-2
           statement-3
        if condition:
                statement-1
                statement-2
               statement-3
```

Example-1: Write a Program to short the given three numbers is ascending order.

```
#a=int(input("Enter the value a= "))
#b=int(input("Enter the value b= "))
#c=int(input("Enter the value c= "))
a,b,c=map(int,input("Enter the tree number: ").split())
if a>b and a>c:
   if b>c:
      print(c,b,a)
   else:
      print(b,c,a)
elif b>c:
   if a>c:
      print(c,a,b)
   else:
      print(a,c,b)
else:
   if a>b:
      print(b,a,c)
   else:
      print(a,b,c)
```

Output:

```
Enter the tree number: 6 3 9
3 6 9
Enter the tree number: 50 15 20
15 20 50
```

Example-2: Write a Program to short the given three numbers is descending order.

```
a=int(input("Enter the value a= "))
b=int(input("Enter the value b= "))
c=int(input("Enter the value c= "))
if a<b and a<c:
   if b<c:
      print(c,b,a)
   else:
      print(b,c,a)
elif b<c:
   if a<c:
      print(c,a,b)
   else:
      print(a,c,b)
else:
   if a<b:
      print(b,a,c)
   else:
      print(a,b,c)
```

Output:

```
Enter the value a= 6
Enter the value b= 3
Enter the value c= 9
9 6 3
```

Example-3: Write a Program to Online Shopping Discounts:

```
total_amount = float(input("Enter the total amount of your purchase: "))
is_member = input("Are you a member? (yes/no): ").lower()

if is_member == 'yes':
    if total_amount >= 100:
        discount = 0.1  # 10% discount for members
    else:
        discount = 0.05  # 5% discount for members
else:
    discount = 0  # No discount for non-members
discounted_amount = total_amount - (total_amount * discount)
print("Your discounted amount is:", discounted_amount)
```

Output:

```
Enter the total amount of your purchase:  1000
Are you a member? (yes/no):  yes
Your discounted amount is: 900.0
```

Example-4: Write a Program to Authentication System:

```
username = input("Enter your username: ")
password = input("Enter your password: ")

if username == 'admin':
    if password == 'password123':
        print("Welcome, admin!")
    else:
        print("Incorrect password for admin.")
else:
    print("Unknown user.")
```

Output:

```
Enter your username:  admin
Enter your password:  password123
Welcome, admin!
```

nested if-else Based questions

1. Write a program to check if a number is **positive, negative, or zero**. If positive, check if it is **even or odd**.
2. Develop a program to determine whether a given **year is a leap year**. If it is a leap year, check if it is a **century year or not**.
3. Create a program to **find the largest among three numbers** using nested if-else.
4. Write a program to check if a **student has passed** based on marks. If passed, check if the grade is **A, B, or C**.
5. Develop a program to classify a **person's age group** (Child, Teen, Adult, Senior). If a child, check if they are an **infant or toddler**.
6. Write a program to determine if a **triangle is valid** based on three given sides. If valid, check whether it is **equilateral, isosceles, or scalene**.
7. Create a program to check if a **number is divisible by 5 and/or 10** using nested if-else.
8. Develop a program to determine if a **character is a vowel or consonant**. If a vowel, check if it is **uppercase or lowercase**.
9. Write a program to check if a **user can vote** (age >= 18). If eligible, check if they are **above 60** (senior citizen category).
10. Create a program to determine if an **entered username and password are correct**. If the username is correct, check if the **password is also correct or incorrect**.
11. Write a program to check whether a number is **positive, negative, or zero**. If positive, check if it is **greater than 100 or not**.
12. Develop a program to determine if a given **character is an alphabet, digit, or special character**. If an alphabet, check if it is **a vowel or consonant**.
13. Create a program to check if a **person is eligible for a job** based on their age and qualification. If eligible, check if they have **work experience or not**.
14. Write a program to determine whether a **number is prime or not**. If not prime, check if it is **even or odd**.
15. Develop a program to check whether a given **day number (1-7) corresponds to a weekday or weekend**. If it's a weekday, check if it's a **Monday or Friday**.
16. Create a program to find **roots of a quadratic equation**. If real roots exist, check they are **equal or distinct**.
17. Write a program to check if a given **password is strong or weak**. If weak, check whether it's due to **short length or missing special characters**.
18. Develop a program to **calculate the discount on a product** based on its price. If the price is high, apply a **higher discount percentage**.
19. Write a program to determine whether a **triangle is right-angled**. If it is, check whether it is also **isosceles or scalene**.
20. Create a program to determine whether a **student is eligible for a scholarship** based on their marks. If eligible, check if they qualify for **full or partial scholarship**.

ITERATIVE or LOOPING STATEMENTS

- If we want to execute a group of statements multiple times we should go for iterative statements.
- Looping statements are used to repeat a block of statements specifying number of times or until a specific event is happen.
- In python have 2 types of iterative statements
  1. **for loop**
  2. **while loop**

➢ Counter Control: it's means before entering in the loop it knows how many times loop will be repeat.

➢ Event Control: it's means in advanced it does not know how many times loop will repeat.

1. for loop:

- for is a purely counter control looping statements.
- It is a definite iteration loop because you know in advance how many times loop will run.

Syntax:

```
for variable in sequence:
    # Code of block
```

or

```
for loop_variable membership_op sequence_data_types:
    # Code of block
```

Where:

- **variable:** This is a variable that takes on the value of each item in the sequence.
- **Sequence:** This is the sequence of items over which the loop will iterate. It can be a list, tuple, string, range, or any other iterable object.
  Note: arguments for range function should be integer. real value is not allowed.
- **Code of block:** This is the block of code that you want to execute.

➢ Within the for-loop sequence will initialise starting the loop in between the loop execution you can not able to modify the sequence.

Example-1

```
n=6
for i in range(1,n+1):
   print("skills",end=" ")
```

Output:

```
skills skills skills skills skills skills
```

Example-2

```
n=6
for i in range(1,n+1):
   print(i,end=" ")
```

Output:

```
1 2 3 4 5 6
```

Example-3

```
for i in range(1,7):
   print('*',end=" ")
```

Output:

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

Problem-1: write a program to display the 1 to n number.

```
n=int(input("Enter the number: "))
for i in range(1, n+1):
   print(i, end=" ")
```

Output:

```
Enter the number: 10
1 2 3 4 5 6 7 8 9 10
```

Problem-2: write a program to display the even number for given number.

Method-1

```
n=int(input("Enter the number: "))
for i in range(0, n+1):
   if i%2==0:
      print(i,end=" ")
```

Output:

```
Enter the number: 10
0 2 4 6 8 10
```

Method-2

```
n=int(input("Enter the number: "))
for i in range(0,n+1,2):
      print(i,end=" ")
```

Output:

```
Enter the number: 10
0 2 4 6 8 10
```

Problem-3: write a program to display the character in the given string.

```
s=input("Enter the string: ")
for i in s:
   print(i)
```

Output:

```
Enter the string: TWKSAA
T
W
K
S
A
A
```

Problem-4: write a program to display the character in the given string index wise.

```
s=input("Enter the string: ")
i=0
for j in s:
   print("The character present at", i, "index is: ", j)
   i=i+1
```

Output:

```
Enter the string: SKILLS
The character present at 0 index is:  S
The character present at 1 index is:  K
The character present at 2 index is:  I
The character present at 3 index is:  L
The character present at 4 index is:  L
The character present at 5 index is:  S
```

Problem-5:

```
#To print skills 4 times
for i in range(4):
   print("skills")
#To display odd number from 0 to 10
print("..................")
for i in range(11):
   if i%2!=0:
      print(i)
#To display the numbers from 6 to 1 in descending order
print("..................")
for i in range(6,2,-1):
   print(i)
```

Output:

```
skills
skills
skills
skills
..................
1
3
5
7
9
..................
6
5
4
3
```

Problem-6: write the python program to print the multiplication table.

```
number = int(input("Enter a number: "))
# Define the range of the multiplication table
table_range = 10
# Use a for loop to generate and print the multiplication table
print(f"Multiplication table for {number}:")
for i in range(1, table_range + 1):
   product = number * i
   print(f"{number} x {i} = {product}")
```

Output:

```
Enter a number:  9
Multiplication table for 9:
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
```

TRANSFER STATEMENTS

- **There are three types of transfer statements.**
  1) **break**
  2) **continue**
  3) **pass**

1. Break:

- break statement is used to terminate current loop (such as for or while loop) prematurely. ब्रेक स्टेटमेंट का उपयोग वर्तमान लूप (जैसे फॉर या व्हाइल लूप) को समय से पहले समाप्त करने के लिए किया जाता है।
- When the break statement is encountered, the program exits the loop immediately and continues executing the code after the loop. (जब ब्रेक स्टेटमेंट सामने आता है, तो प्रोग्राम तुरंत लूप से बाहर निकल जाता है और लूप के बाद कोड निष्पादित करना जारी रखता है।)

Example:
```
for i in range (6):
    if i == 3:
        break
    print(i)
```
Output: 0 1 2

2. Continue:

- continue statement is used to skip current iteration of a loop & move on to next iteration.
- When the continue statement is encountered, the program skips the remaining code within the current loop iteration and proceeds with the next iteration.

Example:
```
for i in range (6):
    if i == 3:
        continue
    print(i)
```
Output: 0 1 2 4 5

3. Pass: The pass statement is a placeholder statement that does nothing.

Example:
```
for i in range (6):
    if i == 3:
        pass
    else:
        print(i)
```
Output: 0 1 2 4 5

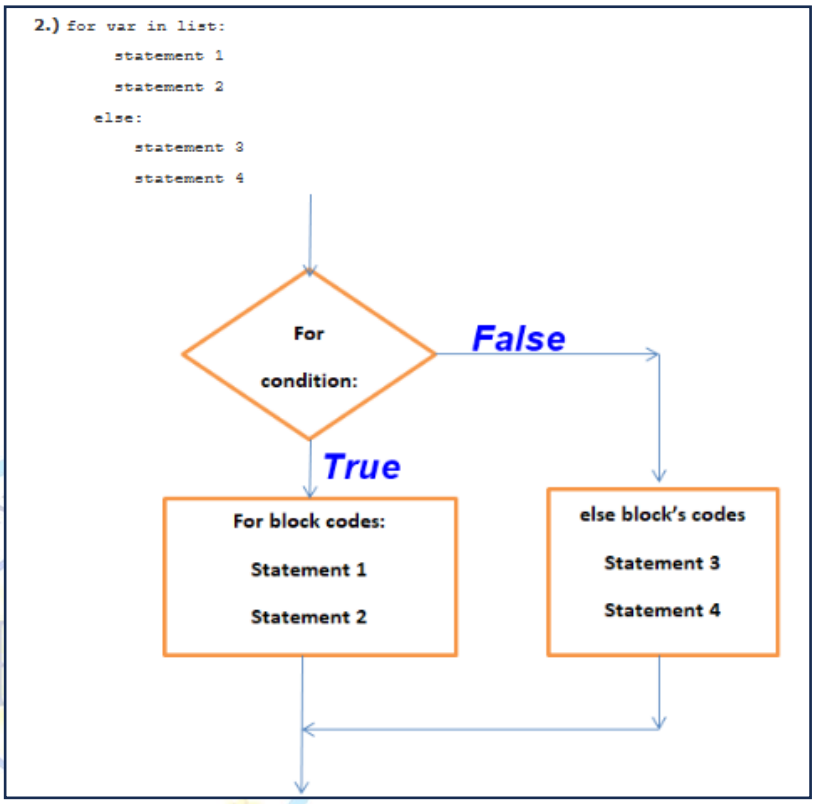
- break is used to exit a loop prematurely when a certain condition is met.
- continue is used to skip the current iteration of a loop and move to the next iteration.
- pass is used as a placeholder means create an empty block of code that doesn't raise any errors.

FOR ELSE

- for...else in Python is a unique feature that allows you to attach an else block to a for loop.
- For else is used to avoid flag variables.
- Else part will execute when the loop is terminated due to end of the sequence.
- Else part will not execute if loop is terminated due to break statement.

Syntax:

```
for iter_var in sequence:
    if condition:
        break
else:
    # Code to execute if the loop completes without encountering a 'break'
```



Example: Problem: write a program to check the given number is prime or not.

```
import math
n=int(input("Enter the number"))
for i in range(2,int(math.sqrt(n))+1):
    if n%i==0:
        print("not a prime")
        break
    else:
        print("prime")
```

output:

Enter the number 7

prime

Example-2

```
numbers = [1, 2, 4, 5]
for num in numbers:
    if num == 3:
        print("Found 3!")
        break
    print(num)
else:
    print("Loop completed without finding 3.")
```

output:

1

2

4

5

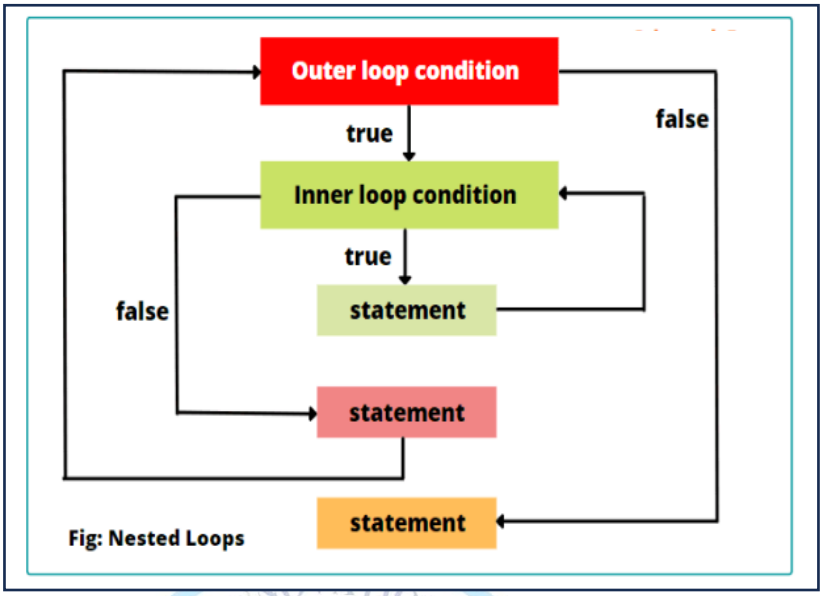
Loop completed without finding 3.

Note:

- if i is a factor for n then n is not a prime number. From 2 to square root of n if there is no factors n is a prime number.

NESTED FOR

- for within for
- for every iteration of the outer loop, inner loop will execute completely.



Syntax:

```
for outer_var in sequence:
    for inner_item in sequence:
        # Code to execute for each combination of outer_item and inner_item.
```

Example-1:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+1):
        print(j, end=" ")
    print()
```

Output

```
Enter the number 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Example-2:

```
for i in range(1, 4):
    for j in range(1, 6):
        print(i * j, end=' ')
    print()
```

Example-3:

```
for i in range(1, 4):
    for j in range(1, 6):
        print(i * j, end=' ')
    print()
```

2. while loop

- while loop in Python repeatedly executes a block of code as long as a specified condition remains true.
- while loops should be used when you don't know how many times the loop needs to run.
- If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syntax:

```
while condition:
    statement-1
    statement-2
    statement-3
   # Code to execute
```

Example:

```
i = 1
while i <= 3:
   print("TWKSAA")
   i = i + 1
```

output

```
TWKSAA
TWKSAA
TWKSAA
```

❖ Loops with else block:

- Inside loop execution, if break statement not executed, then only else part will be executed.
- Else means loop without break.

Example: Guess the secret number

```
import random
# Generate a random number between 1 and 10
secret_number = random.randint(1, 10)

attempts = 0
# Maximum number of attempts allowed
max_attempts = 3
while attempts < max_attempts:
   guess = int(input("Guess the secret number (between 1 and 10): "))
   if guess == secret_number:
      print("Congratulations! You guessed the secret number.")
      break
   elif guess < secret_number:
      print("Try a higher number.")
   else:
      print("Try a lower number.")
   attempts += 1
else:
   print(f"Sorry, you've reached maximum number of attempts. secret number was {secret_number}.")
```

Output:

```
Guess the secret number (between 1 and 10):  6
Try a lower number.
Guess the secret number (between 1 and 10):  3
Try a higher number.
Guess the secret number (between 1 and 10):  5
Congratulations! You guessed the secret number.
```

Example: print the all-days name:

```
while True:
   num=int(input("Enter the number (1-7) or 0 to exit:"))
   if num==1:
      print("Monday")
   elif num==2:
      print("Tuesday")
   elif num==3:
      print("Wednesday")
   elif num==4:
      print("Thursday")
   elif num==5:
      print("Friday")
   elif num==6:
      print("Saturday")
   elif num==7:
      print("Sunday")
   elif num==0:
      print("closed")
      print("Press Enter to continue or any other key to exit.")
      a=input()
      if a=="": # it is used press enter to continue our program
         continue # restart the loop
      else:
         print("finally your program is closed")
         break
   else:
      print("Your given number", num, "is not in range. Please enter a number from 1 to 7 or
0 to exit")
```

Output:

```
Enter the number (1-7) or 0 to exit: 3
Wednesday
Enter the number (1-7) or 0 to exit: 6
Saturday
Enter the number (1-7) or 0 to exit: 9
Your given number 9 is not in range. Please enter a number from 1 to 7 or 0 to exit
Enter the number (1-7) or 0 to exit: 0
closed
Press Enter to continue or any other key to exit.

Enter the number (1-7) or 0 to exit: 6
saturday
Enter the number (1-7) or 0 to exit: 0
closed
Press Enter to continue or any other key to exit.
 8
Finally  your program is closed
```

PROBLEM BASED 0N CONTROL STATEMENT

Problem-1: Write a Program to Check given number is Prime Number or not.
Problem-2: Write a Program to Display the multiplication Table
Problem-3: Write a Program to Check given number is Armstrong Number or not
Problem-4: Write a Program to Print all Prime Numbers in an Interval
Problem-5: Write a Program to Find the Factorial of a Number
Problem-6: Write a Program to find the Fibonacci sequence
Problem-7: Write a Program to Find Armstrong Number in an Interval
Problem-8: Write a Program to Find the Sum of Natural Numbers
Problem-9: write a program to extract the last number is even number or not
Problem-10: write a program to check the given number is divisible by 3 and 5 or not
Problem-11: write a program to check the weather given number is divisible by 4 or 7
Problem-12: write a python program to check the greatest among three number
Problem-13: write a program to perform all the bitwise operator
Problem-14: write a program to check whether the last digit of a number is divisible by 3 or not
Problem-15: write a program to display the student grade by reading the student marks
Problem-16: write a program to short three numbers ascending order or descending order
Problem-17: write a program to display the alternat even number.
Problem-18: write a program to display the alternat even number is divisible by 4.
Problem-19: write a program to display the sum of 1 to n natural numbers
Problem-20: write a program to display the sum of all even numbers between 1 to n.
Problem-21: write a program to display the list of factors for a given number.
Problem-22: write a program to display sum product & count of factor for a given number excluding n
Problem-23: write a program to check the given number is perfect(equal), abundant(less), and deficient(greater) number.
Problem-24: write a program to count the number of factors is given number excluding 1 and n and also check the number is prime or not
Problem-25: write a program to check whether given number is prime number or based on factors
Problem-26: write a program to check the given number is prime or not by using the while loop.
Problem-27: write a program to display the sum of individual digits are given number.
Problem-28: write a program to find the GCD(greatest common divisor) and LCM of the given number
Problem-29: Write a program to read unspecified number of integers count how many positive and negative numbers entered when user types zero our program should terminated
Problem-30: write a to check the given number is co-prime or not.
Problem-31: write a program to find the nth prime number.
Problem-32: write a program to display list of prime number within the given range.
Problem-33: write a program to find the prime factor of a given number.
Problem-34: write a program to display nearest prime number for given number n where p>=n.
Problem-35: write a program to check the nearest prime number is palindrome or not.
Problem-36: write a program to maximum digit form a given number.
Problem-37: write a program to display sum of individual number is given until single digit.

Problem-1 Write a Program to Check Prime Number

```
n=int(input("n="))
for i in range(2,n//2+1):
    if n%i==0:
        print("not prime number")
        break
else:
   print (n, "is a prime number ")
```

.....output.....

```
n=7
7 is a prime Number
.....2nd method...
n=int(input("n="))
f=0
for i in range(2,n):
    if n%i==0:
       print("not prime number")
       f=1
       break
if f==0:
      print(n, "is a prime number ")
.....o/p.....
n=53
53 is a prime number
```

Problem-4: Print all Prime Numbers in an Interval.

```
n1=int(input("n1= "))
n2=int(input("n2= "))
for n in range(n1,n2+1):
   for i in range(2,n):
      if n%i==0:
         break
   else:
      print(n,end=" ")
.....o/p.....
n1= 6
n2= 50
7 11 13 17 19 23 29 31 37 41 43 47
```

Problem-5: Find the Factorial of a Number

(n! = n* (n-1) * (n-2) *...1)

```
n=int(input("n="))
f=1
for i in range(1,n+1):
      f*=i
print(f)
.....o/p....
n=6
720
```

Problem-2: Display the multiplication Table

```
n=int(input("n="))
for i in range(1,11):
   print("{}*{} =
{}".format(n,i,n*i))
 .....o/p.....
n=3
3*1 = 3
3*2 = 6
3*3 = 9
3*4 = 12
3*5 = 15
3*6 = 18
3*7 = 21
3*8 = 24
3*9 = 27
3*10 = 30
```

Problem-6: find the Fibonacci sequence

```
n=int(input("n="))
a=0
b=1
print(a,end=" ")
print(b,end=" ")
for i in range(2,n+1):
   c=a+b
   print(c,end=" ")
   a=b
   b=c
.....o/p.....
n=6
0 1 1 2 3 5 8
2nd method
n=int(input("n="))
a=-1
b=1
for i in range(n+1):
   c=a+b
   print(c,end=" ")
   a=b
   b=c
....o/p.....
n=6
0 1 1 2 3 5 8
```

Problem-3: Check Armstrong Number

```
n=int(input("n="))
t=n
sum=0
while t>0:
   a=t%10
   sum=sum+a**3
   t=t//10
print(sum)
if n==sum:
   print("Armstrong Number")
else:
   print("not Armstrong Number")
.....o/p.....
n=153
153
Armstrong Number
.....2nd method
n=9926315
l=len(str(n))
t=n
r=0
while n!=0:
   a=n%10
   r=r+a**l
   n=n//10
print(r)
if r==t:
   print("Armstrong Number")
else:
   print("n")
.....o/p.....
9926315
Armstrong Number
```

Problem-7: Find Armstrong Number in an Interval

```
n1=int(input("n1="))
n2=int(input("n2="))
for n in range(n1,n2+1):
   sum=0
   temp=n
   while temp>0:
      a=temp%10
      sum=sum+a**3
      temp=temp//10
      if n==sum:
         print(n)
.....o/p.....
n1=100
n2=200
125
153
```

Problem-8: Write a Program to Find the Sum of Natural Numbers

```
n=int(input("n="))
sum=0
for i in range(1,n+1):
   sum=sum+i
print(sum)
```

.....o/p.....

```
n=5
15
```

Problem-9: write the program to extract the last number is even number or not

```
n=int(input("n="))
a=n%10
if a%2==0:
   print("last digit is even number")
else:
   print("Not")
```

....o/p....

```
n=336
last digit is even number
```

Problem-10: write the program to check the given number is divisble by 3 and 5 or not

```
n=int(input("n="))
print(n%3==0 and n%5==0)
```

.....o/p....

```
n=15
True
```

Problem-11: write a python program to check the weather given number is divisible by 4 or 7

```
n=int(input("n="))
print(n%4==0 or n%7==0)
```

.....o/p....

```
n=24
True
```

Problem-12: write a python program to check the greatest among three number

```
a,b,c=map(int,input("enter three number ").split())
if a>b and a>c:
   print(a,"is greater")
elif b>c:
   print(b," is greater")
else:
   print(c,"is greater")
```

.....o/p....

```
enter three number 9 5 6
9 is greater
```

Problem-13: write a program to perform all the bitwise operator

```
a,b=6,9
print(a&b)
print(a|b)
print(a^b)
print(a<<b)
print(a>>b)
```

....o/p.....

```
0
15
15
3072
```

Problem-14: write a program to check whether the last digit of a number is divisible by 3 or not

```
num=eval(input("Enter your number: "))
last_no=num%10
if last_no%3==0:
   print("the given number last value is divisible by 3")
else:
   print("the given number last value is not divisible by 3")
print("given number last value is: ",  last_no)
```

.....o/p.....

```
Enter your number: 56
the given number last value is divisible by 3
given number last value is:  6
```

Problem-15: write a python program to display the student grade by reading the student marks

```
marks=eval(input("marks="))
if marks>90:
   print("A")
elif marks>=80:
   print("B")
elif marks>=70:
   print("C")
elif marks>=60:
   print("D")
elif marks>=50:
   print("E")
else:
   print("fail")
```

.....O/P.....

```
marks=96
A
```

Problem-16: write a python program to short the three numbers ascending order or descending order

```
a,b,c=map(int, input("enter the a b c values:").split())
if a>b and a>c:
   if b>c:
      print(c,b,a)
   else:
      print(b,c,a)
elif b>c:
   if a>c:
      print(c,a,b)
   else:
      print(a,c,b)
else:
   if a>b:
      print(b,a,c)
   else:
      print(a,b,c)
.........o/p.......
enter the a b c values:5 7 2
```

Problem-17: write a python program to display the alternat even number.

```
n=int(input("n="))
for i in range(1,n+1):
   if (i%2==0 and i%4!=0):
      print(i,end=" ")
........o/p.....
n=20
2 6 10 14 18
```

Problem-18: write a python program to display the alternat even number is divisible by 4.

```
n=int(input("n="))
for i in range(1,n+1):
   if i%4==0:
      print(i,end=" ")
......o/p.....
n=30
4 8 12 16 20 24 28
```

Problem-19: write a python program to display the sum of 1 to n natural numbers

```
n=int(input("n="))
s=0
for i in range(1,n+1):
   s+=i
   print(i,end=" ")
print("\nsum=",s)
.........o/p....
n=6
1 2 3 4 5 6
sum= 21
```

Problem-20: write a python program to display the sum of all even numbers between 1 to n.

```
n=int(input("n="))
s=0
for i in range(1,n+1):
   if i%2==0:
      s+=i
      print(i,end=" ")
print("\nsum of even numbers=",s)
......o/p.......
n=6
2 4 6
sum of even numbers= 12
```

Problem-21: write a python program to display the list of factor for a given number.

```
n=int(input("n="))
for i in range(1,n+1):
   if n%i==0:
      print(i,end=" ")
....o/p.....
n=6
1 2 3 6
```

Problem-22: write a python program to display the sum, product and count of factor for a given number excluding n.

```
n=int(input("n="))
s=0
p=1
c=0
for i in range(1,n):
   if n%i==0:
      s+=i
      p*=i
      c+=1
      print(i,end=" ")
print("\nsum=",s)
print("product=",p)
print("count of factor=",c)
......o/p.....
n=10
1 2 5
sum= 8
product= 10
count of factor= 3
```

Problem-23: write a python program to check the given number is perfect(equal), abundent(less),and difficient(greater) number.

```
n=int(input("n="))
s=0
for i in range(1,n):
   if n%i==0:
      s+=i
print("sum=",s)
if s==n:
   print("perfect number:")
elif s<n:
   print("difficient number")
else:
   print("Abundent number")
```

....o/p...

```
n=6 (0,20,100)
sum= 6
perfect number
```

Problem-24: write a python program to count the number of factor is given number excluding 1 and n and also check the number is prime or not.

```
n=int(input("n="))
s=0
p=1
c=0
for i in range(2,n//2+1):
   if n%i==0:
      s+=i
      p*=i
      c+=1
      print(i,end=",")
print("sum=",s)
print("product=",p)
print("count of factor=",c)
if c==0:
   print(n,"is prime number")
else:
   print(n,"is not prime number")
```

.....output.....

```
n=101
sum= 0
product= 1
count of factor= 0
101 is prime number
```

Problem-25: check whether given number is prime number or based on factors

```
n=int(input("n="))
s,p,c=0,1,0
for i in range(2,n//2+1):
   if n%i==0:
      s+=i
      p*=i
      c+=1
print("sum=",s)
print("product=",p)
print("count of factor=",c)
if c==0:
   print(n,"is a prime number")
else:
   print(n,"is not a prime number")
```

.....o/p.....

```
n=131
sum= 0
product= 1
count of factor= 0
131 is a prime number
```

Problem-26: check the given number is prime or not by using the while loop.

```
n=int(input("n="))
i=2
while i<n//2+1:
   if n%i==0:
      print(n,"is a not prime number")
      break
   i=i+1
else:
   print(n,"is a prime number")
```

.....o/p.....

```
n=53
53 is a prime number
```

.....or.....

```
n=int(input("n="))
f=0
i=2
while i<n//2+1:
   if n%i==0:
      f=1
      print(n,"is a not prime number")
      break
   i=i+1
if f==0:
   print(n,"is a prime number")
```

.....o/p....

```
n=71
71 is a prime number
```

.....or.....by using the count factor

```
n=int(input("n="))
i=2
s=0
p=1
c=0
while i<n//2+1:
   if n%i==0:
      s+=i
      p*=i
      c+=1
      print(i,end=" ")
      break
   i=i+1
print("\nsum=",s)
print("product=",p)
print("count of factor=",c)
if c==0:
   print(n,"is a prime number")
else:
   print(n,"is not a prime number")
```

.....o/p.....

```
n=26
2
sum= 2
product= 2
count of factor= 1
26 is not a prime number
```

Problem-27: write a python program to display the sum of individual digits are given number.

```
n=int(input("n="))
s=0
while n!=0:
   r=n%10
   s+=r
   n=n//10
print("sum=",s)
```

.....o/p.....

```
n=153
sum= 9
```

Problem-28: write a python program to find the GCD(greatest common divisor) and LCM of the given number.

```
a=int(input("a="))
b=int(input("b="))
m=a
n=b
while b!=0:
   r=a%b
   a=b
   b=r
print("Gcd=", a)
lcm=m*n//a
print("LCM=",lcm)
```

.....O/P.....

```
a=6
b=9
```

Gcd= 3
LCM= 18

.....or.....

```
a=int(input("a="))
b=int(input("b="))
while True:
   r=a%b
   if r==0:
      print(b)
      break
   a=b
   b=r
```

.....O/P.....

```
a=24
b=34
```

GCD= 2

.....or .....

```
a=int(input("a="))
b=int(input("b="))
if a>b:
   m=a
else:
   m=b
for i in range(m,0,-1):
   if a%i==0 and b%i==0:
      print("gcd=",i)
      break
```

.....O/P.....

```
a=56
b=23
```

gcd= 1

Problem-29: Write a python program to read unspecified number of integers count how many positive and negative numbers entered when user types zero our program should terminated.

```
pc=nc=0
while True:
   n=int(input("n="))
   if n==0:
      break
   elif n>0:
      pc+=1
   else:
      nc+=1
print("postive count=",pc)
print("negative count=",pc)
```

.....o/p.....

```
n=6
n=-9
n=-6
n=6
n=6
n=0
```

postive count= 3
negative count= 2

Problem-30: write a python to check the given number is co-prime or not.

```
a=int(input("a="))
b=int(input("b="))
while b!=0:
   r=a%b
   a=b
   b=r
print("gcd=",a)
if a==1:
   print("it is co-prime number")
else:
   print("it is not a co-prime number")
```

.....o/p.....

```
a=53
b=37
```

gcd= 1
it is co-prime number

Problem-31: write a python program to find the nth prime number.

```
n=int(input("n="))
c=0
a=2
while c!=n:
   for i in range(2,a//2+1):
      if a%i==0:
         a=a+1
         break
   else:
      st=a
      a=a+1
      c=c+1
print("nth prime number of",n,"=",st)
```

.....o/p.....

```
n=6
```

6 th prime number = 13

Problem-32: write a python program to display list of prime number within the given range.

```
n=int(input("n="))
for i in range(2,n+1):
   for j in range(2,i//2+1):
      if i%j==0:
         break
   else:
      print(i,end=" ")
```

....o\p.....

```
n=21
```

2 3 5 7 11 13 17 19

Problem-33: write a python program to find the prime factor of a given number.

```
n=int(input("n="))
for i in range(2,n+1):
   if n%i==0:
      for j in range(2,i//2+1):
         if i%j==0:
            break
      else:
         print(i)
```

.....op.....

```
n=35
```

5
7

Problem-34: display nearest prime number for given number n where p>=n.

```
n=int(input("n="))
while True:
   for i in range(2,n//2+1):
      if n%i==0:
         n=n+1
         break
   else:
      print(n)
      break
```

.....op.....

```
n=6
7
```

Problem-35: check the nearest prime number is palindrome or not.

```
n=int(input())
while True:
   for i in range(2,n//2+1):
      if n%i==0:
         n=n+1
         break
   else:
      x=n
      rev=0
      while x!=0:
         r=x%10
         rev=rev*10+r
         x=x//10
         break
      if rev==n:
         print(n,"palindrome number")
         break
else:
   print("not")
```

.....o/p.....

Problem-36: write a program to maximum digit form a given number.

```
n=int(input())
m=n%10
while n!=0:
   r=n%10
   if r>m:
      m=r
   n=n//10
print(m)
```

.....o/p....

```
n=253149532
9
```

Problem-37: write a python program to display sum of individual number is given until single digit.

```
n=int(input("n="))
while n>9:
   s=0
   while n!=0:
      r=n%10
      s=s+r
      n=n//10
   print(s)
```

.....o/p.....

```
n=123456
21
```

...2nd method...

```
=int(input("n="))
r=n%9
if r==0:
   print(r)
else:
   print(r)
```

.....o/p....

```
n=123
6
```

PATTERN PROGRAM

Pattern-1

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

Program:

```
n=int(input("Enter the number:"))
for i in range(1,n+1):
      print(" * "*n)
```

Pattern-2

```
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(j, end="  ")
   print()
```

Pattern-3

```
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4
5 5 5 5 5 5
6 6 6 6 6 6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(i, end="  ")
   print()
```

Pattern-4

```
A A A A A A
B B B B B B
C C C C C C
D D D D D D
E E E E E E
F F F F F F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(chr(64+i), end="  ")
   print()
```

Pattern-5

```
A B C D E F
A B C D E F
A B C D E F
A B C D E F
A B C D E F
A B C D E F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(chr(64+j), end="  ")
   print()
```

Pattern-6

```
a b c d e f
a b c d e f
a b c d e f
a b c d e f
a b c d e f
a b c d e f
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(chr(96+j), end="  ")
   print()
```

Pattern-7

```
6 6 6 6 6 6
5 5 5 5 5 5
4 4 4 4 4 4
3 3 3 3 3 3
2 2 2 2 2 2
1 1 1 1 1 1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(n+1-i, end="  ")
   print()
```

Pattern-8

```
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(n+1-j, end="  ")
   print()
```

Pattern-9

```
F F F F F F
E E E E E E
D D D D D D
C C C C C C
B B B B B B
A A A A A A
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
   for j in range(1, n+1):
      print(chr(65+n-i), end="  ")
   print()
```

Pattern-10

F E D C B A
F E D C B A
F E D C B A
F E D C B A
F E D C B A
F E D C B A

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+1):
        print(chr(65+n-j), end=" ")
    print()
```

Pattern-11

f f f f f f
e e e e e e
d d d d d d
c c c c c c
b b b b b b
a a a a a a

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+1):
        print(chr(97+n-i), end=" ")
    print()
```

Pattern-12

f e d c b a
f e d c b a
f e d c b a
f e d c b a
f e d c b a
f e d c b a

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+1):
        print(chr(97+n-j), end=" ")
    print()
```

Pattern-13

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, i+1):
        print("*", end=" ")
    print()
```

Pattern-14

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, i+1):
        print(i, end=" ")
    print()
```

Pattern-15

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, i+1):
        print(j, end=" ")
    print()
```

Pattern-16

A
B B
C C C
D D D D
E E E E E
F F F F F F

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+1):
        print(n+1-i, end=" ")
    print()
```

Pattern-17

A
A B
A B C
A B C D
A B C D E
A B C D E F

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, i+1):
        print(chr(64+j), end=" ")
    print()
```

Pattern-18

a
a b
a b c
a b c d
a b c d e
a b c d e f

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, i+1):
        print(chr(96+j), end=" ")
    print()
```

Pattern-19

```
1 1 1 1 1 1
2 2 2 2 2
3 3 3 3
4 4 4
5 5
6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(i, end=" ")
    print()
```

Pattern-20

```
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(j, end=" ")
    print()
```

Pattern-21

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print("*", end=" ")
    print()
```

Pattern-22

```
A A A A A A
B B B B B
C C C C
D D D
E E
F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(chr(64+i), end=" ")
    print()
```

Pattern-23

```
A B C D E F
A B C D E
A B C D
A B C
A B
A
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(chr(64+j), end=" ")
    print()
```

Pattern-24

```
a b c d e f
a b c d e
a b c d
a b c
a b
a
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(chr(96+j), end=" ")
    print()
```

Pattern-25

```
     1
    2 2
   3 3 3
  4 4 4 4
 5 5 5 5 5
6 6 6 6 6 6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print("  "*(n-i), end="")
    for j in range(1, i+1):
        print(i, end=" ")
    print()
```

Pattern-26

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print("  "*(n-i), end="")
    for j in range(1, i+1):
        print("*", end=" ")
    print()
```

Pattern-27

```
     A
    B B
   C C C
  D D D D
 E E E E E
F F F F F F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print("  "*(n-i), end="")
    for j in range(1, i+1):
        print(chr(64+i), end=" ")
    print()
```

Pattern-28

```
6 6 6 6 6 6
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(n+1-i, end="  ")
    print()
```

Pattern-29

```
F F F F F F
E E E E E
D D D D
C C C
B B
A
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(chr(64+n+1-i), end="  ")
    print()
```

Pattern-30

```
f f f f f f
e e e e e
d d d d
c c c
b b
a
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    for j in range(1, n+2-i):
        print(chr(96+n+1-i), end="  ")
    print()
```

Pattern-31

```
     1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print(i, end=" ")
    print()
```

Pattern-32

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print("*", end=" ")
    print()
```

Pattern-33

```
     A
    B B
   C C C
  D D D D
 E E E E E
F F F F F F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print(chr(64+i), end=" ")
    print()
```

Pattern-34

```
     1
    1 2
   1 2 3
  1 2 3 4
 1 2 3 4 5
1 2 3 4 5 6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print(j, end=" ")
    print()
```

Pattern-35

```
     a
    a b
   a b c
  a b c d
 a b c d e
a b c d e f
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print(chr(96+j), end=" ")
    print()
```

Pattern-36

```
     A
    A B
   A B C
  A B C D
 A B C D E
A B C D E F
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1, i+1):
        print(chr(64+j), end=" ")
    print()
```

Pattern-37

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),"*"*(n+1-i))
```

Pattern-38

```
666666
 55555
  4444
   333
    22
     1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),(str(n+1-i)+"")*(n+1-i))
```

Pattern-39

```
111111
 22222
  3333
   444
    55
     6
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print(i,end="")
    print()
```

Pattern-40

```
123456
 12345
  1234
   123
    12
     1
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print(j,end="")
    print()
```

Pattern-41

```
FFFFFF
 EEEEE
  DDDD
   CCC
    BB
     A
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),(str(chr(65+n-i))
+"")*(n+1-i))
```

Pattern-42

```
ABCDEF
 ABCDE
  ABCD
   ABC
    AB
     A
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(65,66+n-i):
        print(chr(j),end="")
    print()
```

Pattern-43

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

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),"*"*(2*i-1))
```

Pattern-44

```
     1
    222
   33333
  4444444
 555555555
66666666666
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),(str(i)+"")*(2*i-1))
```

Pattern-45

```
     6
    56
   456
  3456
 23456
123456
```

Program:

```
n=int(input("Enter the number"))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print((n-i+j),end="")
    print()
```

```
Enter the number: 6
          *
        * *
      * * *
    * * * *
  * * * * *
* * * * * *
Enter the number: 6
          *
        * * *
      * * * * *
    * * * * * * *
  * * * * * * * * *
* * * * * * * * * * *
Enter the number: 6
          *
        * * *
      * * * * *
    * * * * * * *
  * * * * * * * * *
* * * * * * * * * * *
  * * * * * * * * *
    * * * * * * *
      * * * * *
        * * *
          *
```

Pattern Program-46:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    print()
#.....other-1.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    for j in range(i-1,0,-1):
        print("*",end=" ")
    print()
#.....other-2.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    for j in range(i-1,0,-1):
        print("*",end=" ")
    print()
for i in range(n-1,0,-1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    for j in range(i-1,0,-1):
        print("*",end=" ")
    print()
```

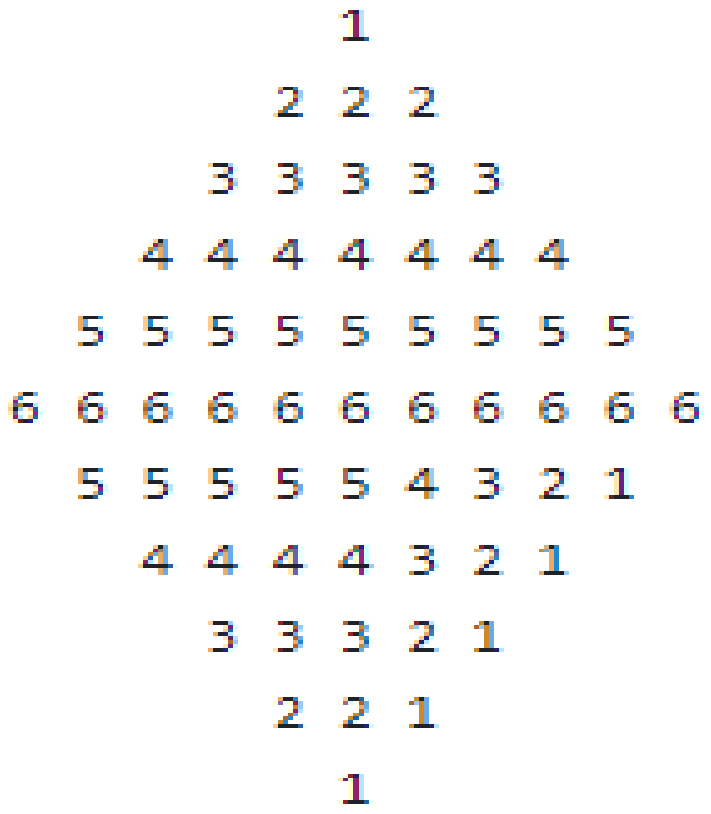
Pattern Program-47:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    print()
#.....other-1.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    for j in range(i-1,0,-1):
        print(j,end=" ")
    print()
#.....other-2.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    for j in range(i-1,0,-1):
        print(j,end=" ")
    print()
for i in range(n-1,0,-1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    for j in range(i-1,0,-1):
        print(j,end=" ")
    print()
```

```
Enter the number: 6
          1
        1 2 1
      1 2 3 2 1
    1 2 3 4 3 2 1
  1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
  1 2 3 4 5 4 3 2 1
    1 2 3 4 3 2 1
      1 2 3 2 1
        1 2 1
          1
```

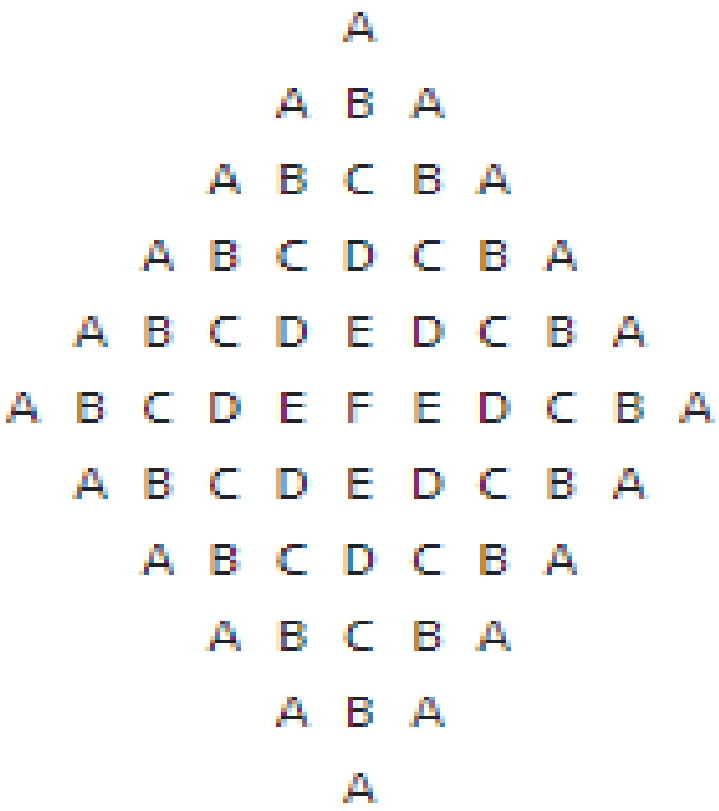
```
Enter the number: 6
          1
        1 2 1
      1 2 3 2 1
    1 2 3 4 3 2 1
  1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
```

```
Enter the number: 6
          1
        1 2
      1 2 3
    1 2 3 4
  1 2 3 4 5
1 2 3 4 5 6
```



Pattern Program-48:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      print(i,end=" ")
   for j in range(i-1,0,-1):
      print(i,end=" ")
   print()
for i in range(n-1,0,-1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      print(i,end=" ")
   for j in range(i-1,0,-1):
     •print(j,end=" ")
   print()
```



Pattern Program-49:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      print(chr(64+j),end=" ")
   for j in range(i-1,0,-1):
      print(chr(64+j),end=" ")
   print()
for i in range(n-1,0,-1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      print(chr(64+j),end=" ")
   for j in range(i-1,0,-1):
      print(chr(64+j),end=" ")
   print()
```

Pattern program-50

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,i+1):
        if j==1 or j==i or i==n:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    print()
#.............other-2...........
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,i+1):
        if j==1 or j==i or i==n:
            print(i,end=" ")
        else:
            print(" ",end=" ")
    print()
#.............other-3...........
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,i+1):
        if j==1 or j==i or i==n:
            print(chr(64+i),end=" ")
        else:
            print(" ",end=" ")
    print()
#.............other-4...........
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,i+1):
        if j==1 or j==i or i==n:
            print(chr(96+j),end=" ")
        else:
            print(" ",end=" ")
    print()
```

```
Enter the number: 6
1
1 2
1   3
1     4
1       5
1 2 3 4 5 6

Enter the number: 6
1
2 2
3   3
4     4
5       5
6 6 6 6 6 6

Enter the number: 6
A
B B
C   C
D     D
E       E
F F F F F F

Enter the number: 6
a
a b
a   c
a     d
a       e
a b c d e f
```

Pattern Program-51

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(min(i,j),end=" ")
    print()
#.............other-2...........
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(min(i,j),end=" ")
    for j in range(n-1,0,-1):
        print(min(i,j),end=" ")
    print()
#.............other-2...........
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(min(i,j),end=" ")
    for j in range(n-1,0,-1):
        print(min(i,j),end=" ")
print()
for i in range(n-1,0,-1):
    for j in range(1,n+1):
        print(min(i,j),end=" ")
    for j in range(n-1,0,-1):
        print(min(i,j),end=" ")
    print()
#.............other.......
n=int(input("Enter the number: "))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(min(chr(64+j),chr(64+i)),end=" ")
    for j in range(n-1,0,-1):
        print(min(chr(64+j),chr(64+i)),end=" ")
    print()
for i in range(n-1,0,-1):
    for j in range(1,n+1):
        print(min(chr(64+j),chr(64+i)),end=" ")
    for j in range(n-1,0,-1):
        print(min(chr(64+j),chr(64+i)),end=" ")
    print()
```

```
Enter the number: 6
1 1 1 1 1 1
1 2 2 2 2 2
1 2 3 3 3 3
1 2 3 4 4 4
1 2 3 4 5 5
1 2 3 4 5 6
Enter the number: 6
1 1 1 1 1 1 1 1 1 1 1
1 2 2 2 2 2 2 2 2 2 1
1 2 3 3 3 3 3 3 3 2 1
1 2 3 4 4 4 4 4 3 2 1
1 2 3 4 5 5 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1

Enter the number: 6
1 1 1 1 1 1 1 1 1 1 1
1 2 2 2 2 2 2 2 2 2 1
1 2 3 3 3 3 3 3 3 2 1
1 2 3 4 4 4 4 4 3 2 1
1 2 3 4 5 5 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
1 2 3 4 5 5 5 4 3 2 1
1 2 3 4 4 4 4 4 3 2 1
1 2 3 3 3 3 3 3 3 2 1
1 2 2 2 2 2 2 2 2 2 1
1 1 1 1 1 1 1 1 1 1 1

Enter the number: 6
A A A A A A A A A A A
A B B B B B B B B B A
A B C C C C C C C B A
A B C D D D D D C B A
A B C D E E E D C B A
A B C D E F E D C B A
A B C D E E E D C B A
A B C D D D D D C B A
A B C C C C C C C B A
A B B B B B B B B B A
A A A A A A A A A A A
```

Pattern Program-52:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()#.....other-2.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print() #.....other-3.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
for i in range(n-1,0,-1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

```
Enter the number: 6
     *
    *
   *
  *
 *
*
Enter the number: 6
     *
    * *
   *   *
  *     *
 *       *
*         *
Enter the number: 6
     *
    * *
   *   *
  *     *
 *       *
*         *
 *       *
  *     *
   *   *
    * *
     *
Enter the number: 6
     1
    1
   1
  1
 1
1
     1
    1 1
   1   1
  1     1
 1       1
1         1
 Enter the number: 6
     1
    1 1
   1   1
  1     1
 1       1
1         1
 1       1
  1     1
   1   1
    1 1
     1
```

Pattern Program-53:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    print() #.....other-5.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    print()  #.....other-6.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    print()
for i in range(n-1,0,-1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    for j in range(i-1,0,-1):
        if j==1:
            print(j,end=" ")
        else:
            print(" ",end=" ")
    print()
```

Pattern Program-54:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   print() #.....other-8.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   print()
#.....other-9.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   print()
for i in range(n-1,0,-1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+j),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(j,end=" ")
      else:
         print(" ",end=" ")
   print()
```

```
Enter the number: 6
      A
     A
    A
   A
  A
 A
Enter the number: 6
      A
     A A
    A   A
   A     A
  A       A
 A         A
Enter the number: 6
      A
     A A
    A   A
   A     A
  A       A
 A         A
A           A
Enter the number: 6
      a
     a a
    a   a
   a     a
  a       a
 a         a
a           a
Enter the number: 6
      A
     B B
    C   C
   D     D
  E       E
 F         F
```

Pattern Program-55:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+i),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(64+i),end=" ")
      else:
         print(" ",end=" ")
   print()
for i in range(n-1,0,-1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(64+i),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(64+i),end=" ")
      else:
         print(" ",end=" ")
   print()
#.....other-11.....
n=int(input("Enter the number: "))
for i in range(1,n+1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(96+i),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(96+i),end=" ")
      else:
         print(" ",end=" ")
   print()
for i in range(n-1,0,-1):
   print("  "*(n-i),end="")
   for j in range(1,i+1):
      if j==1:
         print(chr(96+i),end=" ")
      else:
         print(" ",end=" ")
   for j in range(i-1,0,-1):
      if j==1:
         print(chr(96+i),end=" ")
      else:
         print(" ",end=" ")
```

Pattern Program-56:

```
n=int(input("Enter the number: "))
c=1
for i in range(1,n+1):
    for j in range(1,i+1):
        print("{:2d}".format(c),end=" ")
    c+=1
    print()
n=int(input("Enter the number: "))
c=1
for i in range(1,n+1):
    for j in range(1,i+1):
        print("{:02d}".format(c),end=" ")
        c+=1
    print()

n=int(input("Enter the number: "))
c=1
for i in range(1,n+1):
    print("  "*(n-i),end="")
    for j in range(1,i+1):
        print("{:02d}".format(c),end=" ")
        c+=1
    print()
```

.....o/p.....

```
Enter the number: 6
 1
 2  2
 3  3  3
 4  4  4  4
 5  5  5  5  5
 6  6  6  6  6  6
Enter the number: 6
01
02 03
04 05 06
07 08 09 10
11 12 13 14 15
16 17 18 19 20 21
Enter the number: 6
        01
       02 03
     04 05 06
   07 08 09 10
  11 12 13 14 15
16 17 18 19 20 21
```

Pattern Program-57:

```
n=int(input("Enter the number: "))
for i in range(1,n+1):
    c=i
    for j in range(1,n+1):
        print("{:3d}".format(c),end=" ")
        c=c+n
    print()
```

.....o/p.....

```
Enter the number: 6
  1   7  13  19  25  31
  2   8  14  20  26  32
  3   9  15  21  27  33
  4  10  16  22  28  34
  5  11  17  23  29  35
  6  12  18  24  30  36
```

*****Our requirement is*****

```
1
2
3
```

*****program-1*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(i)
```

*****Our requirement is*****

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

*****program-2*****

```
n=int(input("n="))
for i in range(1,n+1):
    print("* "*n)
```

*****Our requirement is*****

```
1
2
3
1
2
3
```

*****Our requirement is*****

```
1 1 1
2 2 2
3 3 3
```

*****program-4*****

```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(i,end=" ")
    print()
```

Our requirement is
```
1 2 3
1 2 3
1 2 3
```
program-5
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(j,end=" ")
    print()
```
Our requirement is
```
1 1 2 3
2 1 2 3
3 1 2 3
```
program-6
```
n=int(input("n="))
for i in range(1,n+1):
    print(i,end=" ")
    for j in range(1,n+1):
        print(j,end=" ")
    print()
```
Our requirement is
```
A A A A A A
B B B B B B
C C C C C C
D D D D D D
E E E E E E
F F F F F F
```
program-7
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(64+i),end=" ")
      #print(chr(96+i),end=" ")
    print()
```
Our requirement is
```
a b c d e f
a b c d e f
a b c d e f
a b c d e f
a b c d e f
a b c d e f
```
program-8
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(96+j),end=" ")
      #print(chr(64+j),end=" ")
    print()
```

Our requirement is
```
6 6 6 6 6 6
5 5 5 5 5 5
4 4 4 4 4 4
3 3 3 3 3 3
2 2 2 2 2 2
1 1 1 1 1 1
```
program-8
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(n+1-i,end=" ")
    print()
```
Our requirement is
```
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
```
program-8
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(n+1-j,end=" ")
    print()
```
Our requirement is
```
F F F F F F
E E E E E E
D D D D D D
C C C C C C
B B B B B B
A A A A A A
```
program-9
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(65+n-i),end=" ")
    print()
```
Our requirement is
```
F E D C B A
F E D C B A
F E D C B A
F E D C B A
F E D C B A
F E D C B A
```
program-10
```
n=int(input("n="))
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(65+n-j),end=" ")
    print()
```

Our requirement is
```
     1
    222
   33333
  4444444
 555555555
66666666666
```
program-36
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),(str(i)+"")*(2*i-1))
```

Our requirement is
```
     A
    BBB
   CCCCC
  DDDDDDD
 EEEEEEEEE
FFFFFFFFFFF
```
program-37
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),(str(chr(64+i)+""))*(2*i-1))
```

Our requirement is
```
    A
   CCC
  EEEEE
 GGGGGGG
IIIIIIIII
KKKKKKKKKKK
```
program-38
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),(str(chr(64+2*i-1)+""))*(2*i-1))
```
Our requirement is
```
     1
   1 2 3
  1 2 3 4 5
 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10 11
```
program-39
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,2*i):
        print(j,end=" ")
    print()
```

Our requirement is
```
    1
  3 2 1
 5 4 3 2 1
 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
11 10 9 8 7 6 5 4 3 2 1
```
program-40
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(2*i-1,0,-1):
        print(j,end=" ")
    print()
```
Our requirement is
```
    A
   A B C
  A B C D E
 A B C D E F G
A B C D E F G H I
A B C D E F G H I J K
```
program-41
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(65,65+2*i-1):
        print(chr(j),end=" ")
    print()
```
Our requirement is
```
    A
   C B A
  E D C B A
 G F E D C B A
I H G F E D C B A
K J I H G F E D C B A
```
program-42
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(65+2*i-2,64,-1):
        print(chr(j),end=" ")
    print()
```

Our requirement is
```
    0
   1 0 1
  2 1 0 1 2
 3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
```
Our requirement is
```
    0
   1 0 1
  2 1 0 1 2
 3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
```
program-43
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i):
        print(i-j,end=" ")
    for k in range(0,i):
        print(k,end=" ")
    print()
```
Our requirement is
```
    A
   B A B
  C B A B C
 D C B A B C D
E D C B A B C D E
F E D C B A B C D E F
```
program-44
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i):
        print(chr(i-j+65),end=" ")
    for k in range(0,i):
        print(chr(k+65),end=" ")
    print()
```
Our requirement is
```
    1
   1 2 1
  1 2 3 2 1
 1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
```
program-45
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    for k in range(i-1,0,-1):
        print(k,end=" ")
    print()
```

Our requirement is
```
    0
   1 0 1
  2 1 0 1 2
 3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
```
Our requirement is
```
    0
   1 0 1
  2 1 0 1 2
 3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
```
program-43
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i):
        print(i-j,end=" ")
    for k in range(0,i):
        print(k,end=" ")
    print()
```
Our requirement is
```
    A
   B A B
  C B A B C
 D C B A B C D
E D C B A B C D E
F E D C B A B C D E F
```
program-44
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i):
        print(chr(i-j+65),end=" ")
    for k in range(0,i):
        print(chr(k+65),end=" ")
    print()
```
Our requirement is
```
    1
   1 2 1
  1 2 3 2 1
 1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
```
program-45
```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    for k in range(i-1,0,-1):
        print(k,end=" ")
    print()
```

*****Our requirement is*****

```
    A
   A B A
  A B C B A
 A B C D C B A
A B C D E D C B A
A B C D E F E D C B A
```

*****program-46*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(chr(64+j),end=" ")
    for k in range(i-1,0,-1):
        print(chr(64+k),end=" ")
    print()
```

*****Our requirement is*****

```
    6
   6 5
  6 5 4
 6 5 4 3
6 5 4 3 2
6 5 4 3 2 1
```

*****program-47*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(n+1-j,end=" ")
    print()
```

*****Our requirement is*****

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

*****program-48*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print("*",end=" ")
    for k in range(1,n+1-i):
        print("*",end=" ")
    print()
```

*****Our requirement is*****

```
6 6 6 6 6 6 6 6 6 6 6
 5 5 5 5 5 5 5 5 5
  4 4 4 4 4 4 4
   3 3 3 3 3
    2 2 2
     1
```

*****program-49*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(0,n+1-i):
        print(n+1-i,end=" ")
    for k in range(1,n+1-i):
        print(n+1-i,end=" ")
    print()
```

*****Our requirement is*****

```
9 9 9 9 9 9 9 9 9
 7 7 7 7 7 7 7
  5 5 5 5 5
   3 3 3
    1
```

*****program-50*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(0,n+1-i):
        print(2*n+1-2*i,end=" ")
    for k in range(1,n+1-i):
        print(2*n+1-2*i,end=" ")
    print()
```

*****Our requirement is*****

```
1 2 3 4 5 6 7 8 9 10 11
 1 2 3 4 5 6 7 8 9
  1 2 3 4 5 6 7
   1 2 3 4 5
    1 2 3
     1
```

*****program-51*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print(j,end=" ")
    for k in range(2,n+2-i):
        print(n+k-i,end=" ")
    print()
```

*****Our requirement is*****

```
F F F F F F F F F F F
 E E E E E E E E E
  D D D D D D D
   C C C C C
    B B B
     A
```

*****program-52*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print(chr(65+n-i),end=" ")
    for k in range(2,n+2-i):
        print(chr(65+n-i),end=" ")
    print()
```

*****Our requirement is*****

```
K K K K K K K K K K K
 I I I I I I I I I
  G G G G G G G
   E E E E E
    C C C
     A
```

*****program-53*****

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(i-1),end="")
    for j in range(1,n+2-i):
        print(chr(65+2*n-2*i),end=" ")
    for k in range(2,n+2-i):
        print(chr(65+2*n-2*i),end=" ")
    print()
```

*****Our requirement is*****

```
    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
 1 2 3 4 5
  1 2 3 4
   1 2 3
    1 2
     1
```

program-55

```
n=int(input("n="))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(l,end=" ")
    print()
```

Our Requirements

```
    *
   **
  ***
 ****
*****
******
*******
********
*******
******
*****
****
***
**
*
1
22
333
4444
55555
666666
7777777
88888888
7777777
666666
55555
4444
333
22
1
1
12
123
1234
12345
123456
1234567
12345678
2345678
345678
45678
5678
678
78
8
1
12
123
1234
12345
123456
1234567
12345678
1234567
123456
12345
1234
123
12
1
A
BB
CCC
DDDD
EEEEE
FFFFFF
GGGGGGG
HHHHHHHH
GGGGGGG
FFFFFF
EEEEE
DDDD
CCC
BB
A
A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
BCDEFGH
CDEFGH
DEFGH
EFGH
FGH
GH
H
```

.....program.....

```
n=8
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print("*",end=" ")
    print()
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(i,end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(n-k,end=" ")
    print()
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(k+l,end=" ")
    print()
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(j,end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(l,end=" ")
    print()
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(chr(64+i),end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(chr(64+n-k),end=" ")
    print()
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print(chr(64+j),end=" ")
    print()
for k in range(1,n):
    print(" "*k,end="")
    for l in range(1,n+1-k):
        print(chr(64+k+l),end=" ")
    print()
```

*****Our requirement is*****

```
   * * * * * *
enter a number: 6
    *       *
   * *     * *
  * * *   * * *
 * * * *  * * * *
* * * * * * * * * *
* * * * * * * * * * *
    ***program-87***
n=int(input("enter a number: "))
for i in range(1,n+1):
    print(" "*(n-i),end="")
    for j in range(1,i+1):
        print("*",end=" ")
    print("  "*(n-i),end="")
    for k in range(1,i+1):
        print("*",end=" ")
    print()
...our requirements....
enter a number: 6
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
0 1 0 1 0 1
    ***program-88***
n=int(input("enter a number: "))
for i in range(1,n+1):
    for j in range(1,i+1):
        if (i%2!=0 and j%2!=0)or(i%2==0 and j%2==0):
            print("1",end=" ")
        else:
            print("0",end=" ")
    print()
...our requirements....
enter a number: 6
     *
    * *
   * * *
  * * * *
 * * * * *
* * * * * *
   * * *
  * * * *
 * * * * *
* * * * * *
 * * * * * * *
* * * * * * * *
  * * * * *
  * * * * * *
 * * * * * * *
* * * * * * * *
 * * * * * * * * *
* * * * * * * * * *
   ***
   ***
   ***
   ***
   ***
```

********program-89*****

```
n=int(input("enter a number: "))
for a in range(1,n+1,2):
    for i in range(1,n+1):
        print(" "*(2*n-i-a),end="")
        for j in range(1,i+a):
            print("*",end=" ")
        print()
for b in range(1,n+1):
    print("  "*(n-2),end="")
    print("*"*3)
```

LIST

- it is a collection of elements. it may be homogenous or heterogenous.
- it is mutable, meaning you can modify their elements.
- it is Versatility: means Lists can store elements of different types.

➢ why list data type is need in python:

1. For ordered collections of elements.
2. For grow or shrink Dynamically elements.
3. For store many elements in a single variable

List:

- To store multiple value under a single variable name list are useful. A list is a collection of elements. Lists are mutable.
- the list is created by putting comma-separated value in square brackets.
- A list can have any number of items and they may be of different types.
- if we want to represent a group of individual objects as a single entity where insertion order preserved and duplicates are allowed, then we should go for list
- list is dynamic because based on our requirements we can increase size and decrease the size.
- in list the elements will be placed within square brackets and with comma separator.
- we can differentiate duplicate elements by using the index and we can preserve insertion order but using index. hence index will play very important role.
- python support both positive and negative indexes. +ve index means from left to right where as negative index means right to left.\

CREATION OF LIST OBJECT

❖ How to create a list:

- Declare a variable and assign it a list of values enclosed in square brackets [].

Example:

- l=[2,3.1,'skillscenter', 3+6j,[1,2,3,4],(3,4,5,6),{2,3,4},{'name': 'raj', 'age': 30},True]

i). you can create empty list object as follows.

```
List=[]
print(List)
print(type(List))
```

Output: []

```
<class 'list'>
```

ii). if you know elements already then you can create list as follows.

```
l= [4,56,7,78,8,98,5,434]
```

iii). With Dynamic input:

```
l=eval (input ("Enter list: "))
print(l)
print(type(l))
```

Output: Enter list: [6,7,8,9]

```
[6,7,8,9]
<class 'list'>
```

How to create list dynamically

- by using this all function we can create dynamically list.

i). append()
ii). extend()
iii).index()
iv). insert()
v). list ()
vi). split()

i). By using append():

- It is adding the element in end position of a list.
- It is used to add a single element to the end of an existing list. It modifies the list in place by adding the specified element as the last item, increasing the list's length by one.

- **Syntax: list_name.append(element)**

Example:

```
l=[]
n=int(input("enter the number "))
for i in range(1,n+1):
    l.append(int(input("enter the element's: ")))
print("l=",l)
```

Output:
```
enter the number 3
enter the element's: 5
enter the element's: 9
enter the element's: 3
l= [5, 9, 3]
```

ii). extend() function

- it is used to add multiple elements to the end of an existing list. It takes an iterable (such as another list, tuple, or string) as an argument and appends each element of that iterable to the list, one by one.

Syntax: lsit_name.extend(l2)

- all items present in l2 will be added to l1

Example:
```
l1=["t3","skill", "center"]
l2=["Led","based","skill",'center']
l1.extend(l2)
print(l1)
```

Output: ['t3', 'skill', 'center', 'Led', 'based', 'skill', 'center']

Write the difference between the append() and extend()

1. Based on Argument Type:
- append(): Adds the argument as a single element (e.g., a list remains nested).
- extend(): Adds each element of an iterable individually.

2. Based on List Length:
- append(): Increases length by one.
- extend(): Increases length by the number of items in the iterable.

3. Based on Use Case:
- append(): Used when you want to add a single item (even if it's a list or other iterable) as one element at end.
- extend(): Used when you want to merge elements from another iterable into the list individually.

Ex: list1.append([4, 5])
output: [1, 2, 3, [4, 5]]
Ex: list2.extend([4, 5])
output: [1, 2, 3, 4, 5]

iii). By using the index() :

- you can update a value at a specific index in a Python list by using the index() method.
- Finds the position of the first occurrence of a specified element in the list.
- It will replace the old value with new value
- If range will not found in list then it will through the index error

❖ **Updating a Value at a Specific Index**

1. **Find Position with index():** Use index() to get the index of the element you want to update.
2. **Assign the New Value:** Use the index to directly update the element in the list.

❖ **Syntax: List_name[index_number]=new_value**

❖ **Example-1:**

```
List1 = [10, 20, 30, 40,20]
List1[0]=100
print(List1) # [100, 20, 30, 40, 20]
List1[9]=100
print(List1) # IndexError: list assignment index out of range
```

❖ **Example-2**

```
List1 = [10, 20, 30, 40] # Find the index of the element 20
index_number = List1.index(20) # Update the element at this index
List1 [index_number] = 25
print(l)  # Output: [10, 25, 30, 40]
```

Note: in empty list you cannot update the element by using the index().

- in case of insert function provided index is out of list it will consider as last index position.

Example:

```
n=int(input("enter the number: "))
l=[None]*n
print(l)
for i in range(n):
   l[i]=int(input("enter the elements: "))
print("l=",l)
```

Output:
```
enter the number: 3
[None, None, None]
enter the elements: 1
enter the elements: 2
enter the elements: 5
 l= [1, 2, 5]
```

❖ index()

- **Purpose:** Finds the position of the first occurrence of a specified element in the list.
- **Syntax**: list.index(element[, start[, end]])

❖ **Example:**
```
list1 = [10, 20, 30, 40,30,30]
pos = list1.index(30)
print(pos)  # Output: 2
```

❖ **Note**: If the element isn't found, it raises a ValueError.

Example:
```
list1 = [10, 20, 30, 40,30,30]
pos = list1.index(60)
print(pos)
```

Output: **ValueError**: 60 is not in list

iv). By using insert():

- Adds an element at a specific index in a list.
- Shifts existing elements to the right.
- Unlike append(), it doesn't just add to the end.
- If index is out of range, it behaves like append() or inserts at start.
- you **cannot** insert multiple values at once using insert() directly — it only inserts **one item** at a time.

Syntax: list.insert(index, element)

- **index:** The position where you want to insert the element.
- **element:** The value to insert at that position.

Example-1:

```
l=[3,5,7,4,3,4,3]
l.insert(0,9)
print(l)
l.insert(7,39)
print(*l)
```

Output: [9, 3, 5, 7, 4, 3, 4, 3, 9 3 5 7 4 3 4 39 3

Example-2:

```
my_list = [10, 20, 30, 40]
my_list.insert(2, 25)  # Inserts 25 at index 2
print(my_list)
```

Output: [10, 20, 25, 30, 40]

Example 2: Trying to insert multiple values (won't work as expected)

```
my_list = [1, 2, 5]
my_list.insert(2, [3, 4])
print(my_list)  # Output: [1, 2, [3, 4], 5]  ← inserts as a single list element
```

Correct way to insert multiple values:

```
my_list = [1, 2, 5]
my_list[2:2] = [3, 4]
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

iv). with list () function: By using type casting concept we can create dynamically list

Example-1:

```
l=list(range(0,6,1))
print(l)
print(type(l))
```

Output: [0, 1, 2, 3, 4, 5]
<class 'list'>

Example-2:

```
s='t3skillcenter'
print(type(s))
l=list(s)
print(l)
print(type(l))
```

Output: <class 'str'>
['t', '3', 's', 'k', 'i', 'l', 'l', 'c', 'e', 'n', 't', 'e', 'r']
<class 'list'>

v). with split() Function:

- It is used to split a string into a list of substrings, based on a specified delimiter (default is any whitespace). This allows you to create a dynamic list from a string.

Key Points:

- **Functionality**: split() takes a string and divides it into a list based on a delimiter.
- **Applicable**: It can only be used on iterable data types like strings, tuples, or sets.

Syntax: **string.split(delimiter, maxsplit)**

- **delimiter**: The character or substring that separates the elements (optional, default is whitespace).
- **maxsplit**: The maximum number of splits to do (optional).

Example-1:
```
my_string = "apple orange banana"
my_list = my_string.split()  # Splits by whitespace
print(my_list)  # Output: ['apple', 'orange', 'banana']
```

Example-2:
```
s='t3 skill center is a Led based skill center'
print(type(s))
l=s.split()
print(l)
print(type(l))
```

Output:
```
<class 'str'>
['t3', 'skill', 'center', 'is', 'a', 'Led', 'based', 'skill', 'center']
<class 'list'>
```

❖ How to know number of element's are parents in list:

- by using the len() function

Example:
```
l=[2,3.1,'t3skillscenter', 3+6j,[1,2,3,4],(3,4,5,6),{2,3,4},{'name': 'raj', 'age': 30},True]
print(len(l))
```

Output: 9

How to accessing elements from list

- We can access elements of the list either by using index or by using slice operator(:)

➢ **How to access element from a list:**

1). By using the indexing.
2). By using slice operator.
3). By using for loop
4). By using while loop

1). By using the indexing slicing operation.

- list follows zero based index. i.e index of first element is zero.

Syntax: list_name[index]

Example:
```
l=[3,5,7,39,54,3.5]
print(l[3])
```
Output: 39

2). By using slice operator:

syntax: list=list1[start:stop:step]

- **start:** it indicates the index where slice has to start default value is 0
- **stop:** it indicated the index where slice has to end default value is max allowed index of list i.e., length of the list
- **step:** increment value

Example:
```
l=[1,5,62,21,4,1,23,3,5]
print(l[2:6:1])
print(l[::1])
print(l[:6:1])
print(l[2:3:])
print(l[:4:1])
print(l[::-1])
```
Output:
```
[62, 21, 4, 1]
[1, 5, 62, 21, 4, 1, 23, 3, 5]
[1, 5, 62, 21, 4, 1]
[62]
[1, 5, 62, 21]
[5, 3, 23, 1, 4, 21, 62, 5, 1]
```

LIST VS MUTABILITY

- once you create a list object you can modify its content. Hence list object is mutable.

Example:
```
l=[5,8,2,122,52,6]
print(l)
l[1]=333
print(l)
```
Output:
```
[5, 8, 2, 122, 52, 6]
[5, 333, 2, 122, 52, 6]
```

TRAVERSING THE ELEMENTS OF LIST

- the sequential access of each element in the list is called traversal.
- How to access multiple elements from a list:

➔ We can access elements from a list by two ways

1). By using while loop
2). By using for loop

3. By using the while loop.

Example:
```
l=[0,5,8,2,122,52,6]
i=0
while i<len(l):
    print(l[i],end=" ")
```

```
    i=i+1
```

Output: 0 5 8 2 122 52 6

4.By using the for loop.

- in these three ways you can access multiple element

Example:
```
l=[3,5,7,39,54,3.5]
For i in range(len(l)):
    print(l[i],end=" ")
```
Output: 3 5 7 39 54 3.5

...or.....
```
l=[3,5,7,39,54,3.5]
for i in l:
   print(i,end=" ")
```
Output: 3 5 7 39 54 3.5

...or.....
```
l=[3,5,7,39,54,3.5]
print(*l)
```
Output: 3 5 7 39 54 3.5

Problem Display only even numbers:

Program:
```
l=[0,5,8,2,122,52,6]
for i in l:
   if i%2==0:
      print(i,end=" ")
```
Output: 0 8 2 122 52 6

Problem: to display elements by index wise:

Program:
```
l=["t3","skill","center"]
 x=len(l)
for i in range(x):
   print(l[i],"is available at positive index: ",i," and at negative index: ",i-x)
```
Output: t3 is available at positive index: 0 and at negative index: -3
skill is available at positive index: 1 and at negative index: -2
center is available at positive index: 2 and at negative index: -1

IMPORTANT LIST FUNCTION:

- There are following important list function.

1).len():

- return the number of elements present in the list

Example:
```
l=[0,5,8,2,122,52,6]
print(len(l))
```
Output: 7

2). count():

- it returns the number if occurrence of specified item in the list

Example:
```
l=[0,2,5,8,2,2,1,6,5]
```

```
print(l.count(2))
print(l.count(0))
print(l.count(5))
```

Output: 3
1
2

3).index():

- return the index if first occurrence of the specified item.

Example:

```
l=[0,2,5,8,2,2,1,6,5]
print(l.index(1))
print(l.index(5))
print(l.index(8))
```

Output: 6
2
3

4).min()

- to find the minimum element from a given list

Example:
```
l=[21,3,4,6,2,5]
print(min(l))
```
Output: 2

5).max()

- to find the maximum element from a given list

Example:
```
l=[21,3,4,6,2,5]
print(max(l))
```
Output: 21

6).sum()

- to perform the sum of elements in a given list

Example:
```
l=[1,3,4,6,2,5]
print(sum(l))
```
Output: 21

How to delete element from the list

- By using this following method, we can delete element from the list

1.pop()
2. remove()
3. clear()
4.del()

1.pop():

- by default pop() are delete the last element from the list
- pop() are return the deleted element.
- it deleting the element by index

Syntax: list_name.pop(index_number)

Example:
```
l=[3,4,56,7,78,True,"t3skillscenter"]
l.pop(4)
print(l)
l.pop()
print(l)
```
Output:
```
[3, 4, 56, 7, True, 't3skillscenter']
[3, 4, 56, 7, True]
```

2.remove ():

- for delete the specific element without using the indes you can use remove().
- it is deleting the elent by proving the value.

Syntax: list_name.remove(element)

Example:
```
l=[4,4,65,6,7,8]
l.remove(6)
print(l)
```
Output: [4, 4, 65, 7, 8]

3. clear () function

- you can use clear () function to remove all elements of list.

Example:
```
n=[10,20,50,43,53]
print(n)
n.clear()
print(n)
```
Output:
```
[10, 20, 50, 43, 53]
[]
```

4.del (): it used for delete any object from the memory

Example:
```
a=5
print(a)
del (a)
 print(a)
```
Output:
```
5
error: name 'a' is not defined
```

Example:
```
l=[3,5,6,73,5]
del l[2]
print(l)
```
Output: [3, 5, 73, 5]

❖ Deference between remove () and pop ():

- **remove ()**
  - you can use to remove special element from the list
  - it can't return any value
  - if special element not available then you get value error
- **pop ()**
  - you can use to remove last element from the list.
  - it returned removed element.
  - if list is empty then you get error

Note: list objects are dynamic i.e., based on our requirement you can increase and decrease the list

- append(), insert() and extend():- used for increasing the size/growable nature

remove(), pop():- for decreasing the size/shrinking nature

ORDERING ELEMENTS OF LIST

1).reverse():

- you can use reverse() order of elements of list

Example:

```
l=[1,2,3,4,5,6]
l.reverse()
print(l)
```

Output:

```
[6, 5, 4, 3, 2, 1]
```

2).sort():

- in list by default insertion order is preserved. If want to sort the elements of list according to default natural sorting order then you should go for sort() method.
- For Numbers: - Default Natural Sorting order is ascending order
- For String: - Default Natural sorting order is alphabetical order

Note:

- To use sort () function, compulsory list should contain only homogenous elements. otherwise, we will get type error

Example-1:

```
l=[3,4,5,"t3"]
l.sort()
print()
```

Output:

type error

- to sort in reverse of default natural sorting order:
- we can sort according to reverse of default natural sorting order by using reverse=True argument.

Example-2:

```
n=[9,6,7,4,5,7,1,34]
n.sort()
print(n)
n.sort(reverse=True)
print(n)
n.sort(reverse=False)
print(n)
```

Output:

```
[1, 4, 5, 6, 7, 7, 9, 34]
[34, 9, 7, 7, 6, 5, 4, 1]
[1, 4, 5, 6, 7, 7, 9, 34]
```

ALIASING AND CLONING OF LIST OBJECTS

1). Aliasing: the process of giving another reference variable to the existing list is called aliasing

Example:

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

```
y=x
print(id(x))
print(id(y))
```

Output: 1530791946304
1530791946304

- to problem in this approach is by using one reference variable if we are changing content then those changes will be reflected to the other reference variable.
- to overcome this problem, you should go for cloning
- this process of creating exactly duplicate independent object is called cloning
- you can implement cloning by using slice operator or by using copy() function.

i). By using slice operator:

Example:

```
x==[10,20,30,40]
y=x[:]
y[1]=339
print(x)
print(y)
print(id(x))
print(id(y))
```

Output: [1, 2, 3, 4, 5]
[1, 339, 3, 4, 5]
1530791946304
1530791946048

ii). By using copy() function:

Example:

```
x==[10,20,30,40]
y=x.copy()
y[1]=339
print(x)
print(y)
print(id(x))
print(id(y))
```

Output:

[1, 2, 3, 4, 5]
[1, 339, 3, 4, 5]
1530791946304
1530791919936

❖ Difference between=operator and copy() Function

- = operator means for aliasing
- copy() Function mean for cloning

USING MATHEMATICAL OPERATORS FOR LIST OBJECTS

- you can use + and * operators for list objects

1). concatenation operator (+):

- you can use + to concatenate 2 list into a single list

Example:

```
a=[1,2,34,5]
b=[10,20,30]
c=a+b
print(c)
```

Output: [1, 2, 34, 5, 10, 20, 30]

Note: To use + operator compulsory both arguments should be list objects, otherwise we will get TypeError

Example:

```
c=a+30 typeError
c=a+[30] valid
```

2). Repetition Operator (*):

- we can use repetition operator * to repeat elements of list specified number of times.

Example:

```
a=[1,2,34,5]
c=a*3
print(c)
```

Output: [1, 2, 34, 5, 1, 2, 34, 5, 1, 2, 34, 5]

COMPARING LIST OBJECTS

- you can use comparison operators for list objects.

Example:

```
a=["RAM","ROM","CPU"]
b=["RAM","ROM","CPU"]
c=["rAM","rOM","CPu"]
print(a==b)
print(b==c)
print(a!=c)
```

Output:

```
True
False
True
```

Note:

- whenever you are using comparison operators (==,!=) for list object then the following should be considered
  1). the number of elements
  2). the order of elements
  3). the content of elements (case sensitive)

note: whenever you are using relational operators (<,<=,>,>=) between list objects only 1st element comparison will be performed.

Example:

```
a=[30,40,20,4]
b=[40,56,3,4,3,345,5]
print(x>y)
print(x<y)
```

```
print(x>=y)
print(x<=y)
```

Output:
```
False
True
False
True
```

Example:
```
a=["RAM","ROM","CPU"]
b=["SAM","rid","tlr"]
c=["rAM","rOM","CPu"]
print(a>b)
print(b<c)
print(a>=c)
```

Output:
```
False
True
False
```

MEMBERSHIP OPERATORS

- we can check whether element is a member of the list or not by using membership operators

1).in operator

2). not in operator

Example:
```
n=[10,20,50,43,53]
print(10 in n)
print(10 not in n)
print(53 in n )
print(53 not in n )
```

Output:
```
True
False
True
False
```

NESTED LIST

- sometimes you can take one list inside another list. such type of lists is called nested list.

Example:

```
l=[6,35,65,76,[3,30,39],34]
print(l)
print(l[0])
print(l[4])
print(l[4][0])
```

Output:
```
[6, 35, 65, 76, [3, 30, 39], 34]
6
[3, 30, 39]
3
```

Problem: Write a python program to perform the nested list function.

Program:

```
n=int(input("row= "))
m=int(input("Colum= "))
l=[]
for i in range(n):
   x=[]
   for j in range(m):
      x.append(int(input("enter elements ")))
   l.append(x)
print("nested list=",l)
```

Output:
```
row= 2
Colum= 3
enter elements 5
enter elements 6
enter elements 4
enter elements 8
enter elements 5
enter elements 3
nested list= [[5, 6, 4], [8, 5, 3]]
```

#2nd method of nested list:

```
n=int(input("Colum="))
m=int(input("row="))
l=[[int(input("enter elements")) for j in range(m)] for i in range(n)]
print(l)
```

Output:
```
Colum=2
row=3
enter elments5
enter elments6
enter elments3
enter elments2
enter elments2
enter elments4
[[5, 6, 3], [2, 2, 4]]
```

Example:
```
n=int(input("Colum="))
m=int(input("row="))
l=[[None]*m for i in range(n)]
 for i in range(n):
    for j in range(m):
        l[i][j]=int(input())
  print(l)
```
Output:
```
Colum=2
 row=3
8
6
4
6
3
2
[[8, 6, 4], [6, 3, 2]]
```
Note:

- you can access nested list elements by using index just like accessing multi-dimensional array elements.

❖ if input is different data types.

- [[102, 'raj', 'cse', 98.0], [103, 'anj', 'it', 99.0], [104, 'anjraj', 'cse', 99.9]]
- how to access data from a list by giving priority like name marks etc
- if you want to sort nested list based on the specific Colum may be use either sort or sorted function by default it will sort by first Colum if in case if we want to sort based on specific Colum you need to provide key lambda function by specific which want to search

Example: key=lambda x:x[2]

meaning we need to sort based on the third Coolum

key=lambda x:(x[2],x[4])

meaning need to sort based on the 3rd Colum of in case more than one record having same value for the third Colum it my sort based on the fifth Coolum

Example:
```
l=[]
n=int(input("n="))
for i in range(n):
   l.append([int(input("roll No")),input("Name:"), input("Branch"),float(input("mark="))])
print(l)
l=sorted(l,key=lambda X:(x[3],x[1]))
print(l)
```
Output:
```
n=3
roll No103
Name:raj
Branchcse
mark=100
roll No105
Name:anj
Branchcse
mark=99.9
roll No106
Name:anjraj
Branchit
mark=98
[[103, 'raj', 'cse', 100.0], [105, 'anj', 'cse', 99.9], [106, 'anjraj', 'it', 98.0]]
```

NESTED LIST AS MATRIX

Program:

```
m = int(input("Enter the row: "))
n = int(input("Enter the column: "))
A = []
B = []

print("Enter the elements into Matrix A:")
for i in range(m):
    e1 = []
    for j in range(n):
        e = int(input(f"Enter element A[{i}][{j}]: "))
        e1.append(e)
    A.append(e1)

print("Enter the elements into Matrix B:")
for i in range(m):
    e2 = []
    for j in range(n):
        e = int(input(f"Enter element B[{i}][{j}]: "))
        e2.append(e)
    B.append(e2)

R = [[None] * n for _ in range(m)]
for i in range(m):
    for j in range(n):
        R[i][j] = A[i][j] + B[i][j]
print("Matrix A:", A)
print("Matrix B:", B)
print("Resultant Matrix R:", R)
```

Output:

Output:

```
Enter the row: 2
Enter the column: 2
Enter the elements into Matrix A:
1
2
3
4
Enter the elements into Matrix B:
5
6
7
8
Matrix A: [[1, 2], [3, 4]]
Matrix B: [[5, 6], [7, 8]]
Resultant Matrix R: [[6, 8], [10, 12]]
```

MATRIX MULTIPLICATION

Example:

```
m = int(input("Enter the number of rows for Matrix A: "))
n = int(input("Enter the number of columns for Matrix A (and rows for Matrix B): "))
p = int(input("Enter the number of columns for Matrix B: "))
# Initialize matrices
A = [], B = []
print("Enter the elements of Matrix A:")
for i in range(m):
    row = []
    for j in range(n):
        row.append(int(input(f"Enter element A[{i}][{j}]: ")))
    A.append(row)
print("Enter the elements of Matrix B:")
for i in range(n):
    row = []
    for j in range(p):
        row.append(int(input(f"Enter element B[{i}][{j}]: ")))
    B.append(row)
# Initialize result matrix
R = [[0 for _ in range(p)] for _ in range(m)]
# Perform matrix multiplication
for i in range(m):
    for j in range(p):
        for k in range(n):
            R[i][j] += A[i][k] * B[k][j]
# Print matrices
print("Matrix A:", A), print("Matrix B:", B)
print("Resultant Matrix R (A x B):", R)
```

Output:

```
Enter the number of rows for Matrix A:  2
Enter the number of columns for Matrix A (and rows for Matrix B):  2
Enter the number of columns for Matrix B:  2
Enter the elements of Matrix A:
Enter element A[0][0]:  1
Enter element A[0][1]:  2
Enter element A[1][0]:  3
Enter element A[1][1]:  5
Enter the elements of Matrix B:
Enter element B[0][0]:  2
Enter element B[0][1]:  3
Enter element B[1][0]:  5
Enter element B[1][1]:  6
Matrix A: [[1, 2], [3, 5]]
Matrix B: [[2, 3], [5, 6]]
Resultant Matrix R (A x B): [[12, 15], [31, 39]
```

2nd method of matrix multiplication

Example:

```
m = int(input("Enter rows for Matrix A: "))
n = int(input("Enter columns for Matrix A / rows for Matrix B: "))
p = int(input("Enter columns for Matrix B: "))
# Input matrices A and B
A = [[int(input(f"A[{i}][{j}]: ")) for j in range(n)] for i in range(m)]
B = [[int(input(f"B[{i}][{j}]: ")) for j in range(p)] for i in range(n)]
# Initialize and calculate result matrix R
R = [[sum(A[i][k] * B[k][j] for k in range(n)) for j in range(p)] for i in range(m)]
# Print results
print("Matrix A:", A)
print("Matrix B:", B)
print("Resultant Matrix R:", R)
```

Output:

```
Enter rows for Matrix A: 2
Enter columns for Matrix A / rows for Matrix B: 3
Enter columns for Matrix B: 2
A[0][0]: 1
A[0][1]: 2
A[0][2]: 3
A[1][0]: 4
A[1][1]: 5
A[1][2]: 6
B[0][0]: 7
B[0][1]: 8
B[1][0]: 9
B[1][1]: 10
B[2][0]: 11
B[2][1]: 12

Matrix A: [[1, 2, 3], [4, 5, 6]]
Matrix B: [[7, 8], [9, 10], [11, 12]]
Resultant Matrix R: [[58, 64], [139, 154]]
```

LIST COMPREHENSIONS

- it is very easy and compact way of creating list objects from any iterable objects (like list, tuple, dictionary, range etc|) based on some condition.

syntax:

l=[expression for lv in sequence data type)
where lv= list variable
sequence data type: range, list, tuple, set etc.

Example:

```
n=int(input("n="))
l=[int(input("enter the element")) for i in range(n)]
print(*l)
```

Output:

```
n=3
enter the element5
enter the element5
enter the element2
5 5 2
```

Example:

```
l=[a*a for a in range(1,10)]
print(l)
p=[2**i for i in range(1,6)]
print(p)
m=[x for x in l if x%2==0]
print(m)
```

Output:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
[2, 4, 8, 16, 32]
[4, 16, 36, 64]
```

- if all the input in the same line map() are used

Example:

```
n=int(input())
l=list(map(int,input("enter the element").split()))
print(*l)
```

Output:

```
3
enter the element8 6 9
8 6 9
```

❖ without using map

```
n=int(input())
l=[int(i) for i in input("enter the element").split()]
print(*l)
```

Example:

```
w=["T3","Skill","center", "RID","WIT"]
l=[i[0] for i in w ]
print(l)
```

Output: ['T', 'S', 'c', 'R', 'W']

Example:

```
w="t3 skill center is a Led based skills
center".split()
print(w)
l=[[i.upper(),len(i)] for i in w ]
print(l)
```

Output:

```
['t3', 'skills', 'center', 'is', 'a', 'Led', 'based', 'skills', 'center']
[['T3', 2], ['SKILLS', 6], ['CENTER', 6], ['IS', 2], ['A', 1], ['LED',
3], ['BASED', 5], ['SKILLS', 6], ['CENTER', 6]]
```

S.No	Function	Description	Syntax	Example
How to Create the List dynamically				
1	append()	Adds an element to end of the list.	list_name.append(element)	lst = [1, 2] lst.append(3) # lst = [1, 2, 3]
2	extend()	Extends the list by appending elements from another iterable (e.g., list, tuple).	list_name.extend(iterable)	lst = [1, 2] lst.extend([3, 4]) # lst = [1, 2, 3, 4]
3	insert()	Inserts an element at a specified position.	list_name.insert(index, ele)	lst = [1, 3] lst.insert(1, 2) # lst = [1, 2, 3]
4	list()	Creates a new list from an iterable or empty.	list_name = list(iterable)	lst = list("abc") print(lst) # ['a', 'b', 'c']
5	split()	Splits a string into a list of substrings based on a delimiter (default: space).	string.split(delimiter)	text = "hello world" lst = text.split() print(lst) # ['hello', 'world']
How to access the element from a list				
6	Indexing	Access an element using its index (starts from 0).	list[index]	lst = [10, 20, 30] print(lst[0]) # 10 print(lst[-1]) # 30 (Last element)
7	Slicing	Access a range of elements using start, stop, and optional step.	list[start:stop:step]	lst = [10, 20, 30, 40, 50] print(lst[1:4]) # [20, 30, 40] print(lst[::2]) # [10, 30, 50]
8.	For loop	Iterate over each element in a list or other iterable objects.	for item in iterable:	lst = [10, 20, 30, 40] for num in lst: print(num)
9.	While	Executes a block of code as long as the condition is true.	while condition:	lst = [10, 20, 30, 40] i = 0 while i < len(lst): print(lst[i]) i += 1
How to delete element from the list				
10	pop()	Removes and returns the element at a specified index (default is the last element).	list_name.pop(index)	lst = [1, 2, 3] elem = lst.pop() # elem = 3, lst = [1, 2]
11	remove()	Removes the first occurrence of a specified value.	list_name.remove(element)	lst = [1, 2, 3, 2] lst.remove(2) # lst = [1, 3, 2]
12.	del ()	To delete the entire object or to delete an element from the specified index position it doesn't return anything	del list_name del list_name[index]	List1=[10,20,30,40] del List1[20] #it will 20 ele index del List1
13	clear()	To remove all elements from list	List_name.clear()	List=[1,5,4,20,3,5] List.clear() Print(List)
Important List Built Function				
14	count()	Returns the number of occurrences of a specified value in the list.	list.count(value)	lst = [1, 2, 2, 3] cnt = lst.count(2) # cnt = 2

15	sort()	Sorts the elements of the list in ascending order (or descending with reverse=True).	list.sort(reverse=False)	lst = [3, 1, 2] lst.sort() # lst = [1, 2, 3]
16	sorted()	Returns a new sorted list from the elements of the original list.	sorted(list, reverse=False)	lst = [3, 1, 2] sorted_lst = sorted(lst) # sorted_lst = [1, 2, 3]
17	len()	Returns number of elements in list.	len(list)	lst = [1, 2, 3] size = len(lst) # size = 3
18	max()	Returns the largest element in the list.	max(list)	lst = [1, 2, 3,8,9,1] max_val = max(lst) # max_val = 9
19	min()	Returns smallest element in the list.	min(list)	lst = [9,1, 2, 3,8,9,10] min_val = min(lst) # min_val = 1
20	sum()	Returns sum of all elements in the list.	sum(list)	lst = [1, 2, 3] total = sum(lst) # total = 6
21	any()	Returns True if at least one element in list evaluates to True. Otherwise, false	any(list)	lst = [0, 1, 2] or l=[] print(any(l)) result = any(lst) # result = True
22	all()	Returns True if all elements in the list evaluate to True.	all(list)	lst = [1, 2, 3] result = all(lst) # result = True
23	enumerate()	Returns an iterator producing pairs of index and value for each element.	enumerate(iterable, start=0)	list1=[10,20,30] for i,v in enumerate(list1): print(i,":",v,end= " ") # 0 : 10 1 : 20 2 : 30
24	filter()	Returns an iterator with elements matching a condition.	filter(function, iterable)	lst = [1, 2, 3] filtered = list(filter(lambda x: x > 1, lst)) # filtered = [2, 3]
25	map()	Applies a function to each element and returns an iterator.	map(function, iterable)	lst = [1, 2, 3] mapped = list(map(lambda x: x * 2, lst)) # mapped = [2, 4, 6]
26	zip()	Combines multiple iterables into tuples of paired elements.	zip(iterable1, iterable2, ...)	lst1 = [1, 2] lst2 = ['a', 'b'] zipped = list(zip(lst1, lst2)) # zipped = [(1, 'a'), (2, 'b')]

LIST-BASED PROBLEM

1) write a python program to find the min and max value from a given list.
2) write a python program to rotate a shift k element from left to right from the given list.
3) write a python program to display unique vowels present in the given word
4) write a python program to search the particular elements without using pre-defined function.
5) write a python program to remove the duplicate elements from a list.
6) write a python program to display the how many times given elements is repeated in a given list.
7) write a python program to find the maximum number of repeated elements is given list
8) write a python program to perform the slicing operation given list.
9) write a python program to shift 1st k element from left to write
10) sort the first k element ascending order remaining n-k elements in descending order
11) write a python program to shift the all zero in end of the list.
12) write a python program to find the element which is repeating exactly one time which is given list within the given list all the elements will repeated more than one times except one element
13) write a python program to perform Bauble sort.
14) write a python program to perform the selection sort.
15) write a python program to take all input are 2-d list in same row are separated will Coolum are separated with space.
16) write a python program to sort the first k-element ascending order and reaming n-k element in descending order.
17) write a python program to shift all the zeros to end of the list.
18) Write a python program to find the element which is repeating exactly one time which is given list within the given list all the elements will repeated more than one times except one element
19) Write a python program to find the maximum number of 2-d list:
20) write a python program to find the maximum number list inside list.

Answer

1.write a python program to find the min and max value from a given list

Program:

```
l=[2,6,5,2,5,5,5,2,5,6,6,2,5,6,2,9,5,52,8,45,2,6,6,5,2]
a=l
min=max=l[0]
for i in l:
   if i<min:
      min=i
   elif i>max:
      max=i
print("minimum value=",min)
print("maximum value=",max)
```

Output:

```
minimum value= 2
maximum value= 52
....or by using predefined function...
```

Program:

```
l=[2,6,5,2,5,5,5,2,5,6,6,2,5,6,2,9,5,52,8,45,2,6,6,5,2]
l1,l2=min(l),max(l)
print("min=",l1,"and\t""max=",l2)
```

Output:

```
min= 2 and max= 52
```

2.write a python program to rotate a shift k element from left to right from the given list

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
k=int(input("enter the value that you want to check how many times are repeated: "))
k=k%len(l)
for i in range(k):
   l.append(l.pop(i))
print(l)
```

Output:

```
enter the list elements:  5 6 8 63 5 2 5 4 2 5 4  2 5 4
enter the value that you want to check how many times are repeated: 3
[6, 63, 2, 5, 4, 2, 5, 4, 2, 5, 4, 5, 8, 5]
```

3.write a python program to display unique vowels present in the given word

Program:

```
w=input("enter the word to search for vowels: ")
vowels=['a','e','i','o','u']
v=[]
for i in w:
   if i in vowels:
      if i not in v:
         v.append(i)
print("vowels present in given word=",v)
```

Output:

```
enter the word to search for vowels: t3 skill center
vowels present in given word= ['i', 'e']
```

4.write a pyhton program to search the particular elements without using pre-defind function.

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
k=int(input("enter the element that you want to search"))
for i in range(len(l)):
   if l[i]==k:
      print(i)
      break
else:
   print('-1')
```

Output:

```
enter the list elements: 5 6 2 68 2 5 2 2 8 4 5 5 2 2
enter the element that you want to search6
```

```
1
2nd method:-
l=list(map(int,input("enter the list elements: ").split()))
k=int(input("enter the element that you want to search"))
for i in range(len(l)):
   if l[i]==k:
      print(i)
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2
enter the element that you want to search6
1
```

5.write a python program to remove the duplicate elements from a list.

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
print(list(set(l)))
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2
[2, 3, 5, 6, 54, 87, 24]
```

2nd method:-

delete the duplicate element maintain the insertion order

```
l=list(map(int, input("enter the list elements: ").split()))
r=[]
for i in l:
   if i not in r:
      r.append(i)
print(r)
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2
[5, 6, 3, 87, 24, 2, 54]
```

6.write a python program to display the how many times given elements is repeated in a given list.

without predefined function.....

```
l=list(map(int,input("enter the list elements: ").split()))
k=int(input("enter the element that you want to search: "))
print(l.count(k))
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2 5 2
enter the element that you want to search: 2
3
```

without predefined function....

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
k=int(input())
c=0
for i in range(len(l)):
   if l[i]==k:
```

```
        c+=1
print(c)
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2 5 2
2
3
```

7.write a python program to find the maximum number of repeated elements is given list

Program:

```
l=list(map(int, input("enter the list elements: ").split()))
e=0
m=0
for i in set(l):
   c=l.count(i)
   if c>m:
      m=c
      e=i
print("element=",e)
print("maximum coount=",m)
```

Output:

```
enter the list elements: 5 6 3 87 5 24 5 2 54 5 2 5 2
element= 5
maximum count= 5
```

8.write a python program to perform the slicing operation for given list.

Program:

```
l=[2,4,54,5,67,7,8,99,9]
print(l[::])
print(l[::1])
print(l[::-1])
print(l[:5])
print(l[::2])
print(l[5:2])
print(l[2:5:-1])
print(l[-4::])
print(l[-1])
print(l[-5:0:1])
print(l[-6:-1:1])
```

Output:

```
[2, 4, 54, 5, 67, 7, 8, 99, 9]
[2, 4, 54, 5, 67, 7, 8, 99, 9]
[9, 99, 8, 7, 67, 5, 54, 4, 2]
[2, 4, 54, 5, 67]
[2, 54, 67, 8, 9]
[]
[]
[7, 8, 99, 9]
9
```

```
[]
[5, 67, 7, 8, 99]
```

9. write a python program to shift 1st k element from left to write

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
k=int(input())
k=k%len(l)
l=l[k:]+l[:k]
print(l)
```

Output:

```
enter the list elements: 6 3 5 8 9 5 4 2 5 4 52 4
3
[8, 9, 5, 4, 2, 5, 4, 52, 4, 6, 3, 5]
```

10. sort the first k element ascending order remaining n-k elements in descending order

Program:

```
l=list(map(int, input("enter the list elements: ").split()))
k=int(input())
l=sorted(l[:k])+sorted(l[k:],reverse=True)
print(l)
```

Output:

```
enter the list elements: 5 8 7 2 3 1 4 96 45 78 5 45 9 86
3
[5, 7, 8, 96, 86, 78, 45, 45, 9, 5, 4, 3, 2, 1]
```

11. write a python program to shift the all zero in end of the list.

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
a=[]
for i in l:
    if i is 0:
        l.remove(i)
        a.append(i)
print(*l+a)
```

Output:

```
enter the list elements: 0 2 0 1 0 1 0 1 20 1
2 1 1 1 20 1 0 0 0 0
```

12. write a python program to find the element which is repeating exactly one time which is given list within the given list all the elements will repeated more than one times except one element

Program:

```
l=list(map(int, input("enter the list elements: ").split()))
for i in  set(l):
    if l.count(i)==1:
        print(i)
        break
```

Output:

```
enter the list elements: 0 2 0 1 0 1 0 1 20 1
2
```

13. write a python program to perform Bauble sort.

Program:

```
l=list(map(int,input("enter the list elements: ").split()))
for i in range(len(l)-1):
    for j in range(i+1,len(l)):
        if l[j]<l[i]:
            l[i],l[j]=l[j],l[i]
print(l)
```

Output:

```
enter the list elements: 5 8 6 5 2 3 1 4 5
[1, 2, 3, 4, 5, 5, 5, 6, 8]
```

14. write a python program to perform the selection sort.

Program:

```
l=list(map(int, input("enter the list elements: ").split()))
for i in range(len(l)-1):
    t=i
    for j in range(i+1,len(l)):
        if l[j]<l[t]:
            t=j
    l[i],l[t]=l[t],l[i]
print(l)
```

Output:

```
enter the list elements: 5 8 6 5 2 3 1 4 5
[1, 2, 3, 4, 5, 5, 5, 6, 8]
```

15.write a python program to take all input are 2-d list in same row are separated will Coolum are separated with space.

#-if all input are 2-d list in same row, row are separated will Coolum are separated with space

Program:

```
l=[]
x=input("enter the element").split(':')
print(x)
for i in x:
    l.append(list(map(int,i.split())))
print(l)
```

Output:

```
enter the element5 6 8 5 8 52  74 1 5
['5 6 8 5 8 52  74 1 5 ']
[[5, 6, 8, 5, 8, 52, 74, 1, 5]]
```

16. write a python program to sort the first k-element ascending order and remaining n-k element in descending order.

Program:

```
l=list(map(int, input("enter elements: ").split()))
k=int(input("enter k elements for shift the position"))
l=sorted(l[:k])+sorted(l[k:],reverse=True)
print(l)
```

Output:

```
enter elements: 2 8 3 4 6 1 0 5 7 9
enter k elements for shift the position3
[2, 3, 8, 9, 7, 6, 5, 4, 1, 0]
```

17.write a python program to to shift all the zeros to end of the list.

Program:

```
l=[1,0,5,0,0,6,6,0,4,7]
a=[]
for i in l:
   if i==0:
      l.remove(i)
      a.append(i)
print(a)
```

Output:

#17.write a python program to to shift all the zeros to end of the list.

Program:

```
L = [0,0,3, 1, 0, 3,0, 12, 0,0, 7]
c = L.count(0)
print(c)
L = [i for i in L if i!= 0]
print(L)
L.extend([0] * c)
print(L)
```

Output:

```
6
[3, 1, 3, 12, 7]
[3, 1, 3, 12, 7, 0, 0, 0, 0, 0, 0]
```

18. Write a python program to find the element which is repeating exactly one time which is given list within the given list all the elements will repeated more than one times except one element

Program:

```
l=list(map(int, input("enter elements: ").split()))
for i in set(l):
   if l.count(i)==1:
      print(i)
```

Output:

```
7
9
```

#19. Write a python program to find the maximum number of 2-d list:

Program:

```
l=[]
n=int(input("enter n value:"))
for i in range(n):
   l.append(list(map(int,input("enter element: ").split())))
   m=l[0][0]
   for row in l:
      for element in row:
         if element>m:
            m=element
print("lis=",l)
print("max=",m)
```

Output:

```
enter n value:6
enter element: 5
enter element: 39
enter element: 5
enter element: 6
enter element: 23
enter element: 4
lis= [[5], [39], [5], [6], [23], [4]]
max= 39
```

.....2nd method.....

Program:

```
l=[]
n=int(input("enter n value:"))
for i in range(n):
   l.append(list(map(int,input("enter element: ").split())))
   m=l[0][0]
   for i in range(len(l)):
      for j in range(len(l[i])):
         if l[i][j]>m:
            m=l[i][j]
print("lis=",l)
print("max=",m)
```

Output:

```
enter n value:6
enter element: 5
enter element: 39
enter element: 5
enter element: 6
enter element: 23
enter element: 2
lis= [[5], [39], [5], [6], [23], [2]]
max= 39
```

...By using the predefined function...

Program:

```
l=[]
n=int(input("enter n value:"))
for i in range(n):
   l.append(list(map(int,input("enter element: ").split())))
   m=l[0][0]
   for row in l:
      x=max(row)
      if x>m:
         m=x
print("lis=",l)
print("max=",m)
```

Output:

```
enter n value:6
enter element: 5
enter element: 39
enter element: 5
enter element: 6
enter element: 23
enter element: 2
lis= [[5], [39], [5], [6], [23], [2]]
max= 39
```

20. write a python program to find the maximum number list inside list.

Program:

```
l=[2,3,5,6,[3,5,6,7,39,],[9,6,5],7,5]
m=0
for i in l:
   if type(i)==type([]):
      for j in i:
         if j>m:
            m=j
   else:
      if i>m:
         m=i
print("list=",l)
print("max=",m)
```

Output:

```
list= [2, 3, 5, 6, [3, 5, 6, 7, 39], [9, 6, 5], 7, 5]
max= 39
```

TUPLE

- Tuple is exactly same as list except that it is immutable. (read-only)
- Tuple items are indexed
- Negative indexing is also allowed
- if your data is fixed and never change then we should go for Tuple.
- insertion order is preserved
- Duplicates are allowed
- Heterogenous objects are allowed
- we can represent tuple elements within parenthesis and with comma separator.
- Parenthesis are optional but recommended to use.

Example:

```
t=1,2.4,3,"T3",54,56
print(t)
print(type(t))
t=()
print(type(t))
```

Output:

```
(1, 2.4, 3, 'T3', 54, 56)
<class 'tuple'>
<class 'tuple'>
```

Note: we have to take special care about single valued tuple. compulsory the value should ends with comma, otherwise it is not treats as tuple

Example:

```
t1=(6)
print(type(t1))
t2=(6,)
print(type(t2))
```

Output:

```
<class 'int'>
<class 'tuple'>
```

❖ How to Create tuple

1).t=()

- creation empty Tuple

2). t=(6,)

- t=6,

➢ creation of single valued tuple, parenthesis are optional, should ends with comma

3).t=3,30,39

- t=(15,20,25)

➢ creation of multi values tuples & parenthesis are optional

4).by using tuple() function

Example:

```
l=[3,5,6]
t=tuple(l)
print(t,type(t))
```

```
t=tuple(range(1,10,2))
print(t,type(t))
```

Output:

```
(3, 5, 6) <class 'tuple'>
(1, 3, 5, 7, 9) <class 'tuple'>
```

Example:

```
l=tuple(map(int, input().split()))
print(l)
```

Output:

```
5 6 98 855 2 2 41 2 5 4 1
(5, 6, 98, 855, 2, 2, 41, 2, 5, 4, 1)
```

❖ ACCESSING ELEMENTS FROM THE TUPLE

- we can access elements in following way
  - **1). By using Index.**
  - **2). By using slice operator.**
  - **3). By using for loop**
  - **4). By using the while loop**

1). By using Index.

- Use indexing to access elements in a tuple.
- Index starts from 0. Negative index starts from -1 (last element).

Syntax: tuple_name[index]

Example:

```
t=(10,30,40,20)
print(t[0])
print(t[-1])
```

Output:

```
10
20
```

2). By using slice operator.

- Tuples support both **positive** and **negative** indexing.
- You can use the **slice operator** : to access a **range of elements**, just like in lists.

Syntax: tuple_name[start : end : step]

- **start → starting index (inclusive)**
- **end → ending index (exclusive)**
- **step → (optional) interval between elements**

Example:

```
t=(5,6,7,2,4,8,9,6)
print(t[::-1])
print(t[::2])
print(t[-1:-4:-1])
```

Output:

```
(6, 9, 8, 4, 2, 7, 6, 5)
(5, 7, 4, 9)
(6, 9, 8)
```

3) Accessing Tuple Elements Using for Loop

- Use a for loop to iterate over each element in the tuple directly.

Syntax:

```
for item in tuple_name:
    print(item)
```

Example:

```
t = (10, 20, 30)
for i in t:
    print(i)
```

Output:

```
10
20
30
```

4) Accessing Tuple Elements Using while Loop

- Use indexing to access elements with a while loop.

Syntax:

```
i = 0
while i < len(tuple_name):
    print(tuple_name[i])
    i += 1
```

Example:

```
t = (10, 20, 30)
i = 0
while i < len(t):
    print(t[i])
    i += 1
```

Output:

```
10
20
30
```

1st method

```
t=(5,6,7,2,4,8,9,6)
for i in t:
    print(i, end=" ")
```

Output: 5 6 7 2 4 8 9 6

2nd method

```
t=(5,6,7,2,4,8,9,6)
for i in range(len(t)):
    print(t[i],end=" ")
```

Output: 5 6 7 2 4 8 9 6

3rd method:

```
t=(5,6,7,2,4,8,9,6)
print(*t)
```

Output: 5 6 7 2 4 8 9 6

❖ TUPLE VS IMMUTABILITY

- -once we create tuple, we cannot change its content.
- hence tuple objects are immutable

Example:

```
t=(3,6,39)
t[1]=70
```

Output: typeError

❖ MATHEMATICAL OPERATORS FOR TUPLE

- we can apply + and * operators for tuple

1). concatenation operator (+)

Example:

```
a=(10,30,40,20)
b=(1,3,4,2)
c=a+b
print(c)
```

Output:

(10, 30, 40, 20, 1, 3, 4, 2)

2). Multiplication operator or repetition operator (*)

Example:

```
a=(10,30,40,20)
b=a*3
print(b)
```

Output: (10, 30, 40, 20, 10, 30, 40, 20, 10, 30, 40, 20)

❖ IMPORTANT FUNCTION OF TUPLE

1).len()

- to return number of elements present in the tuple.

Example:

```
a=(10,30,40,20)
print("length=",len(t))
```

Output: length= 4

2).count()

- to return number of occurrences of given element in the tuple

Example:

```
t=(10,3,40,3,3,6)
print("conut=",t.count(3))
```

Output: conut= 3

3).index():

- return index of first occurrence of the given element
- if the specified element is not available then we will get valueError.

Example:

```
t=(5,6,7,2,4,8,9,6)
print(t.index(7))
```

Output: 2

4).sorted():

- to sort elements based on default natural sorting order
- -sort():-sort() we cannot used in tuple
- only sorted () are used

Example:

```
t=(5,6,7,2,4,8,9,6)
print(sorted(t))
```

Output:

```
[2, 4, 5, 6, 6, 7, 8, 9]
```

5). Min() and max() Functions:

- these function return min and max values according to default natural sorting order.

Example:

```
t=(5,6,7,2,4,8,9,6)
print(min(t))
print(max(t))
print(sum(t))
```

Output:

```
2
9
47
```

6). reverse():

- reverse () is not used in tuple
- in the case of nested tuple, we can use any object as element if the object is mutable, we can modify those objects within the tuple
- if the object is type is immutable, we cannot change

Example:

```
t=((1,2),[1,2,3,4,5456,6,],(3,4,56),6,'dfg')
print(t)
t[1][2]=30
print(t)
t[1].append(100)
print(t)
```

Output:

```
((1, 2), [1, 2, 3, 4, 5456, 6], (3, 4, 56), 6, 'dfg')
((1, 2), [1, 2, 30, 4, 5456, 6], (3, 4, 56), 6, 'dfg')
((1, 2), [1, 2, 30, 4, 5456, 6, 100], (3, 4, 56), 6, 'dfg')
```

❖ Deleting element from tuple

➢ delete from begging:

Example:

```
t=(2,3,4,6,7,78,8,898)
t=t[1:]
print(t)
```

Output:

```
(3, 4, 6, 7, 78, 8, 898)
t=(2,3,4,6,7,78,8,898)
```

```
t=t[:-1:1]
print(t)
```

Output:

```
(2, 3, 4, 6, 7, 78, 8)
t=(2,3,4,6,7,78,8,898)
t=t[:len(t)-1]
print(t)
```

Output:

```
(2, 3, 4, 6, 7, 78, 8)
```

❖ Deleting element from specific index position

Example:

```
t=(2,3,4,6,7,78,8,898)
ip=3
t=t[:ip]+t[ip+1:]
print(t)
```

Output:

```
(2, 3, 4, 7, 78, 8, 898)
```

❖ How to represent single element in tuple

Example:

```
t=(3,)
t=(3)
print(type(t),t)
t=(3,)
print(type(t),t)
```

Output:

```
<class 'int'> 3
<class 'tuple'> (3,)
```

❖ Adding element at begging

Example:

```
t=(34,5,56,7,676,88)
t=(3,)+t
print(t)
```

Output:

```
(3, 34, 5, 56, 7, 676, 88)
```

❖ Adding element at ending

Example:

```
t=(34,5,56,7,676,88)
t=t+(3,)
print(t)
```

Output:

```
(34, 5, 56, 7, 676, 88, 3)
```

❖ Adding element in the specific indexing

Example:

```
t=(34,5,56,7,676,88)
ip=3
t=t[:ip]+(10,)+t[ip:]
print(t)
```

Output: (34, 5, 56, 10, 7, 676, 88)

❖ TUPLE PACKING AND UNPACKING

➢ we can create a tuple by packing a group of variables

Example:

```
a=1
b=2
c=8
d=5
e=5
f=6
g=3
t=a,b,c,d,f,g
print(t)
```

Output:

```
(1, 2, 8, 5, 6,3)
```

- -here a,b,c,d,f,g are packed into a tuple t. this is nothing but tuple packing
- -tuple unpacking is the reverse process of tuple packing
- -we can unpack a tuple and assign its value to different variables.

Example:

```
t=(1, 2, 8, 5, 6,3)
a,b,c,d,f,g=t
print("a=",a,"b=",b,"c=",c,"d=",d,"e=",e,"f=",f)
```

Output: a= 1 b= 2 c= 8 d= 5 e= 5 f= 6

Note: - at the time of tuple unpacking the number of variable and number of values should be same, otherwise we will get ValueError

❖ TUPLE COMPREHENSION:

➢ tuple comprehension is not supported by python

Program:

```
t=(a**3 for a in range(1,6))
print(type(t))
for i in t:
   print(i)
```

Output:

```
<class 'generator'>
1
8
27
64
125
```

#Here we are not getting tuple object and we are getting generator object

SET

- if we want to represent a group of unique values as a single entity then we should go for set.
- Duplicate are not allowed.
- insertion order is not preserved but we can sort the elements
- Heterogenous elements are allowed.
- set element are immutable
- set objects are mutable i.e., once we create set object, we can perform any changes in what object based on our requirement.
- we can represent set elements within curly braces and with comma separation
- we can apply mathematical operations like unio, intersection difference etc on set objects.
- it is collection of element's sets does not allow duplicate elements set is mutable
- we cannot used list set dictionary as a set element
- set does not support insertion order
- set does not support subscription or indexing operation
- if it not supporting indexing slicing also not supporting

❖ CREATION OF SET

➢ we can create set objects by following way

1. **Using Curly Braces {}**
2. **Using the set() Constructor**
3. **Creating an empty set**
4. **Dynamically adding elements**
   - **Using .add()** — to add a single element
   - **Using .update()** — to add multiple elements from list, tuple, etc.

1.Using Curly Braces {}

Example-1:

```
s={2,3,4,5}
print(s)
print(type(s))
```

Output: {2, 3, 4, 5}
<class 'set'>

Example-2:

```
s={2,3,4,5,6,7,7,5}
print(s)
```

Output: {2, 3, 4, 5, 6, 7}

Example-3:

```
s={(1,2,3,'fsp',4,4+5j)}
print(s)
```

Output: {(1, 2, 3, 'fsp', 4, (4+5j))}

Note:

- while creating empty set we have to take special care.
- compulsory we should use set() function.
- s={} it is treated as dictionary but not empty set.

2. Using the set() Constructor (Typecasting)

Example-1:

```
s={}
print(s)
print(type(s))
.....o/p.....
{}
<class 'dict'>
```

You can also pass:

- Lists → set([1, 2, 2, 3])
- Tuples → set((1, 2, 3))
- Strings → set("hello") → Output: {'h', 'e', 'l', 'o'}

Example-2:

```
s=set(s)
print()
print(type(s))
```

Output:

```
set()
<class 'set'>
```

Example-3

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

3.Creating an empty set

```
empty_set = set()  # Correct way
```

4.Dynamically adding elements

- **Using .add()** — to add a single element
- **Using .update()** — to add multiple elements from list, tuple, etc.

1). add():

- adds item a to the set. It adds random data
- it adds single value at a time
- add(a)
- Adds item a to the set.
- Adds only one element at a time.
- No duplicates are allowed in sets — if the element already exists, it won't be added again.
- The item is added in no particular order (sets are unordered).
- Works only with hashable (immutable) data types like numbers, strings, tuples.

Example-1:

```
s={10, 30, 20}
s.add(40)
print(s)
```

Output:

```
{40, 10, 20, 30}
```

Example-2:

```
colors = {"red", "green", "blue"}
colors.add("yellow")
print(colors)
```

Output:

```
{'yellow', 'green', 'red', 'blue'}
```

Example-3:

```
numbers = {1, 2, 3}
numbers.add(4, 5)  # Trying to add two values at once
print(numbers)
```

Output: TypeError: set.add() takes exactly one argument (2 given)

Example 1: Add a number

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

Example 2: Add a string

```
s = {"apple", "banana"}
s.add("cherry")
print(s)  # Output: {'banana', 'apple', 'cherry'}
```

Example 3: Try adding a duplicate

```
s = {1, 2, 3}
s.add(2)
print(s)  # Output: {1, 2, 3}  → 2 is not added again
```

2). Update(x,y,z)

- Used to add multiple items to a set at once.
- The arguments x, y, z, etc. must be iterable objects like lists, tuples, strings, sets, or ranges.
- All elements from the provided iterable(s) are added to the set.
- Duplicates are automatically removed (sets only store unique values).
- Adds items in no specific order (sets are unordered).
- You can pass one or more iterable arguments.

Example-1:

```
s={2,3,5,6,7}
l=[10,20,30,40]
s.update(l,range(6))
print(s)
```

Output: {0, 1, 2, 3, 4, 5, 6, 7, 10, 20, 30, 40}

Example-2: Example 1: Using a list and a tuple

```
s = {"apple", "banana"}
fruits = ["cherry", "mango"]
more_fruits = ("orange", "grape")
s.update(fruits, more_fruits)
print(s)
```

Output: {'grape', 'banana', 'apple', 'orange', 'mango', 'cherry'}

Example-3: Using a string and a range

```
s = {1, 2}
s.update("34", range(5, 7))
print(s)
```

Output: {'6', '5', 1, 2, '3', '4'}

Example: -4: Using a set and a list

```
s = {100, 200}
extra = [300, 400]
others = {500, 600}
s.update(extra, others)
print(s)
```

Output: {100, 200, 300, 400, 500, 600}

Example-5: Add elements from a tuple and a set

```
s = {10}
s.update((20, 30), {40, 50})
print(s)  # Output: {10, 20, 30, 40, 50}
```

Example 4: Add elements from a range

```
s = set()
s.update(range(3))
print(s)  # Output: {0, 1, 2}
```

Question. What is difference between add() and update()

- Functions in set ?
- we can use add() to add individual item to the set where as we can use update() function to add multiple items to set.
- add() function can take only one argument whereas update() function can take any number of arguments but all arguments should be iterable objects.

How to delete data from the set

- we can delete the data from set in following way

1. **pop()**
2. **remove(element)**
3. **discard(element)**
4. **clear()**
5. **del (keyword, not a method)**

1).pop():

- pop() method removes and returns a random element from the set.
- Since sets are unordered, you cannot predict which element will be removed.
- If the set is empty, calling pop() will raise a KeyError.

Example-1:

```
s={2,3,5,6,7}
print(s)
s.pop()
print(s)
```

Output:

```
{2, 3, 5, 6, 7}
{3, 5, 6, 7}
```

Example 3:

```
fruits = {"apple", "banana", "cherry", "mango"}
print("Before pop:", fruits)
removed_item = fruits.pop()
print("After pop:", fruits)
print("Removed item:", removed_item)
{3, 5, 6, 7}
```

Example-4:

```
numbers = {10, 20, 30, 40, 50}
print("Original set:", numbers)
numbers.pop()
print("Updated set:", numbers)
```

Example 3:

```
colors = {"red", "green", "blue"}
removed = colors.pop()
print("Remaining colors:", colors)
print("Removed color:", removed)
```

2). remove(x):

- The .remove(x) method is used to **remove a specific element** x from the set.
- If the **element exists**, it is removed successfully.
- If the **element does not exist**, it raises a **KeyError**.
- This method is useful when you're sure the element is present in the set.

Example:

```
s={2,3,5,6,7}
print(s)
s.remove(6)
print(s)
```

Output:

```
{2, 3, 5, 6, 7}
{2, 3, 5, 7}
```

Example 3:

```
numbers = {10, 20, 30, 40}
numbers.remove(30)
print(numbers)
```

Output: {40, 10, 20}

Example-2:

```
fruits = {"apple", "banana", "cherry"}
fruits.remove("banana")
print(fruits)
```

Output: {'apple', 'cherry'}

3).discard(x):

- .discard(x) method removes the specified element x from the set, if it is present.
- If the element is not present, it does nothing — no error is raised.
- This makes it safer than .remove() when you're not sure if the element exists in the set.

Example-1:

```
s={2,3,5,6,7}
s.discard(6)
s.discard(39)
print(s)
```

Output:

```
{2, 3, 5, 7}
```

Example 2:

```
colors = {"red", "blue", "green"}
colors.discard("blue")
colors.discard("yellow")  # Not in set, no error
print(colors)
```

Output: {'green', 'red'}

Example-3:

```
numbers = {10, 20, 30}
numbers.discard(20)
numbers.discard(40)  # No error even though 40 is not in the set
print(numbers)
```

4). clear():

- .clear() method is used to remove all elements from the set.
- After calling this method, the set becomes empty (set()), but the set still exists.
- It is useful when you want to reuse the set variable without its previous data.

Example-1:

```
s={2,3,5,6,7}
print(s)
s.clear()
print(s)
```

Output:

```
{2, 3, 5, 6, 7}
set()
```

Example 2:

```
fruits = {"apple", "banana", "mango"}
fruits.clear()
print(fruits)
```

Output: set()

Example-2:

```
numbers = {10, 20, 30, 40}
print("Before clear:", numbers)
numbers.clear()
print("After clear:", numbers)
```

Output:

```
Before clear: {40, 10, 20, 30}
After clear: set()
```

How to access data from the set

- we can access the data from the list in following
  1.for loop
  2. While loop

- Sets are unordered collections, so you can't access elements by index (like in lists or tuples).

1. **Using a for loop (most common way)**

Example-1:

```
s={3,4,6,7,8,9,954,54,6}
for i in s:
   print(i, end=" ")
```

Output: 3 4 6 7 8 9 54 954

Example-2:

```
s={3,4,6,7,8,9,954,54,6}
for i in s:
   print(i, end=", ")
print(*s)
```

Output: 3, 4, 6, 7, 8, 9, 54, 954, 3 4 6 7 8 9 54 954

2. **Since sets are unordered and don't support indexing, we can't directly use a while loop like we do with lists (because sets don't have indices)**

Example: Using while loop to access set elements

```
s = {"apple", "banana", "cherry"}
lst = list(s)  # Convert set to list
i = 0
while i < len(lst):
   print(lst[i])
   i += 1
```

Output: apple
banana
cherry

3. Using in to check if an element exists

- This doesn't return the element directly but checks for its presence.

Example-1:

```
s = {5, 15, 25}
print(15 in s)   # True
print(10 in s)   # False
```

4. Converting set to list or tuple

- This allows index-based access.

```
s = {"a", "b", "c"}
lst = list(s)
print(lst[0])   # Accessing first element
```

Note:

for loop: Best for reading all elements
in operator : Check if a specific element exists
list(set) : Convert to list for index access

❖ How to know number of elements in the set{}

Example:

```
s={1,3,5,7,8,994,3,5,6,True}
print(len(s))
print(s)
```

Output:

```
7
{1, 994, 3, 5, 6, 7, 8}
```

Example:

```
s={3,5,7,8,994,3,5,6,True}
print(len(s))
print(s)
```

Output:

```
7
{True, 994, 3, 5, 6, 7, 8}
```

4.searching

- set does not support indexing so we cannot search set element based on indexing we can use only membership operator where is given element is available or not

Example:

```
s={3,5,7,8,994,3,5,6}
print(7 in s)
```

Output: True

❖ Import function in copy()

1). copy().

- returns copy of the set.
- it is cloned object.

Example:

```
s={2,3,5,6,7}
s1=s.copy()
print(s1)
```

Output:

```
{2, 3, 5, 6, 7}
```

Example 2: Copying a set of colors

```
colors = {"red", "green", "blue"}
copied_colors = colors.copy()
print(copied_colors)
```

Output: {'blue', 'green', 'red'}

Example 2: Copying a set of numbers

```
numbers = {10, 20, 30}
new_numbers = numbers.copy()
print(new_numbers)
```

Output: {10, 20, 30}

MATHEMATICAL OPERATIONS ON THE SET

1).union():

- **Union of Sets: a.union(b) or a | b**
- The union of two sets returns a new set containing all unique elements that are in either set a or set b (or both).
- The union operation combines all elements from both sets while removing duplicates.
- You can perform the union operation using the .union() method or the | (pipe) operator.

Example:

```
a={2,3,5,6,7}
b={20,30,5,60,70}
print(a.union(b))
print(a|b)
```

Output: {2, 3, 5, 6, 7, 70, 20, 60, 30}
{2, 3, 5, 6, 7, 70, 20, 60, 30}

Example 2: Example 1: Union of two sets of fruits

```
fruits_a = {"apple", "banana", "cherry"}
fruits_b = {"orange", "banana", "kiwi"}
# Union using .union()
print(fruits_a.union(fruits_b))
# Union using | operator
print(fruits_a | fruits_b)
```

Output:
{'cherry', 'banana', 'orange', 'kiwi', 'apple'}
{'cherry', 'banana', 'orange', 'kiwi', 'apple'}

Example-3

```
numbers_a = {1, 2, 3, 4}
numbers_b = {3, 4, 5, 6}
# Union using .union()
print(numbers_a.union(numbers_b))
# Union using | operator
print(numbers_a | numbers_b)
```

Output:

{1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5, 6}

2). intersection ()

- intersection() method (or & operator) returns the common elementspresent in both sets.
- a.intersection(b) or a & b will return a set containing only the elements that are present in both ` a and b.

Example-1

```
a = {2, 3, 50, 6, 7}
b = {20, 3, 5, 6, 70}
print(a.intersection(b))
print(a & b)
```

Output

{3, 6}
{3, 6}

Example-3:

```
numbers_a = {1, 2, 3, 4, 5}
numbers_b = {3, 4, 5, 6, 7}
print(numbers_a.intersection(numbers_b))
print(numbers_a & numbers_b)
```

Output:

{3, 4, 5}
{3, 4, 5}

Example-2

```
fruits_a = {"apple", "banana", "cherry"}
fruits_b = {"banana", "kiwi", "cherry"}
print(fruits_a.intersection(fruits_b))
print(fruits_a & fruits_b)
```

Output:

{'banana', 'cherry'}
{'banana', 'cherry'}

3). difference ()

- difference() method (or `-` operator) returns the elements that are in set a but not in set b.
- a.difference(b) or a - b will return the elements that are only in a and not in b.
- b.difference(a) or b - a returns the elements in b but not in a.

Example-1.

```
a = {2, 3, 50, 6, 7}
b = {20, 3, 5, 6, 70}
print(a.difference(b))
print(a - b)
print(b - a)
```

Output:

```
{2, 50, 7}
{2, 50, 7}
{20, 5, 70}
```

Example 2: Difference with an empty set

```
set_a = {1, 2, 3}
set_b = set()
print(set_a.difference(set_b))
print(set_a - set_b)
```

Output:

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

Example-2

```
numbers_a = {1, 2, 3, 4}
numbers_b = {3, 4, 5, 6}
print(numbers_a.difference(numbers_b))
print(numbers_a - numbers_b)
```

Output:

```
{1, 2}
{1, 2}
```

Example 3: Difference between two sets of fruits

```
fruits_a = {"apple", "banana", "cherry"}
fruits_b = {"banana", "kiwi", "cherry"}
print(fruits_a.difference(fruits_b))
print(fruits_a - fruits_b)
```

Output:

```
{'apple'}
{'apple'}
```

4). symmetric_difference()

- symmetric_difference() method (or ^ operator) returns the elements that are in either set a or set b but not in both.
- a.symmetric_difference(b) or a ^ b will return a set of elements that are in a or b, but not in both.

Example:-1 for Symmetric Difference:

```
a = {2, 3, 50, 6, 7}
b = {20, 3, 5, 6, 70}
print(a.symmetric_difference(b))
print(a ^ b)
```

Output

```
{2, 20, 5, 70, 50, 7}
{2, 20, 5, 70, 50, 7}
```

Example:-2: Symmetric difference of two sets of fruits

```
fruits_a = {"apple", "banana", "cherry"}
fruits_b = {"banana", "kiwi", "cherry"}
print(fruits_a.symmetric_difference(fruits_b))
print(fruits_a ^ fruits_b)
```

Output:

```
{'kiwi', 'apple'}
{'kiwi', 'apple'}
```

Example: -3: Symmetric difference with two sets of numbers

```
numbers_a = {1, 2, 3, 4}
numbers_b = {3, 4, 5, 6}
print(numbers_a.symmetric_difference(numbers_b))
print(numbers_a ^ numbers_b)
```

Output:

```
{1, 2, 5, 6}
{1, 2, 5, 6}
```

Example 3: Symmetric difference with an empty set

```
set_a = {1, 2, 3}
set_b = set()
print(set_a.symmetric_difference(set_b))
print(set_a ^ set_b)
```

Output:

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

MEMBERSHIP OPERATORS (IN, NOT IN)

Example:

```
s=set("t3 skill center")
print(s)
print('t' in s)
print('z' in s)
```

Output:

```
{'3', 'i', 'l', ' ', 'k', 's', 'e', 'r', 't', 'n', 'c'}
True
False
```

❖ set comprehension

- **Set comprehension** is a concise way to create sets in Python using a single line of code.
- **Syntax: {expression for item in iterable if condition}**

Example-1:

```
s={a*a for a in range(6)}
print(s)
s={2**a for a in range(2,10,1)}
print(s)
```

Output:

```
{0, 1, 4, 9, 16, 25}
{32, 64, 128, 256, 4, 512, 8, 16}
```

Example-2:

```
s = {x for x in range(10) if x % 2 == 0}
print(s)
```

Output: {0, 2, 4, 6, 8}

Example-3:

```
s = {char for char in "banana"}
print(s)
```

Output: {'b', 'a', 'n'}

INDEXING AND SLICING

- set objects won't support both indexing and slicing.

Q. write a python program to combines the two sets by using the union function.

- **union:** -combines two sets used union set()

Example:

```
s={3,5,7,8,9,3,5,6}
t={4,6,8,906,54}
r=s.union(t)
print(r)
r=r|t
print(r)
```

Output:

```
{3, 4, 5, 6, 7, 8, 9, 906, 54}
{3, 4, 5, 6, 7, 8, 9, 906, 54}
{3, 4, 5, 7, 9, 906, 54}
```

❖ subset()

- subset() concept in Python refers to checking whether all elements of one set are present in another set. This is typically done using the issubset() method or the comparison operators < and <=.

Explanation:

- s.issubset(t) checks if **every element of s is in t**.
- s < t checks if s is a **proper subset** of t (i.e., all elements of s are in t and s ≠ t).
- s <= t checks if s is a **subset** of t (including the case where s == t).

Example-1:

```
s={3,5,7,8,9,3,5,6}
t={4,6,8,906,54}
print(s.issubset(t))
print(s<t)
print(s<=t)
```

Output:

```
False
False
False
```

Example-3:

```
x = {10, 20, 30}
y = {10, 20, 30}
print(x.issubset(y))
print(x < y)
print(x <= y)
```

Output:

```
True
False
True
```

Example-2

```
a = {1, 2, 3}
b = {1, 2, 3, 4, 5}
print(a.issubset(b))
print(a < b)
print(a <= b)
```

Output:

```
True
True
True
```

❖ superset()

Example:

```
s={1,2,3}
t={1,2,3,4,6,8,906,54}
print(t.issuperset(s))
print(s>t)
print(s>=t)
```

Output:

```
True
False
False
```

❖ dijoint set()

Example:

```
s={1,2,3}
t={4,6,8,906,54}
print(s.isdisjoint(t))
s={1,2,3}
t={2,3,5,6}
print(s.isdisjoint(t))
```

Output:

```
True
False
```

DICTIONARY

- we can use list tuple and set to represent a group of individual objects as a single entity.
- if you want to represent a group of objects as key and pairs then we should go for dictionary.

Example:

- **roll:** name, phone: address
- duplicate keys are not allowed but value can be duplicated
- Heterogeneous objects are allowed for both key and value
- insertion order is not preserved
- it is mutable, dictionary is dynamic
- indexing and slicing are not applicable

Note: in c++ and java dictionaries are known as "map" where as in Perl and ruby it is known "hash"

❖ HOW TO CREATE DICTIONARY:

1. **{} (curly braces)**
2. **dict()**
3. **dict() with iterable**
4. **dict comprehension**
5. **zip()**
6. **dict() (copying from another dict)**
7. **dict.fromkeys()**

1. Using Curly Braces ({})

- Curly Braces is used for create the dictionary .

Syntax: dictionary = {"key1": value1, "key2": value2, ...}

Example

```
my_dict = {"name": "Sangam", "age": 25, "city": "Patna"}
print(my_dict)
```

Output: {'name': 'samga', 'age': 25, 'city': 'patan'}

2. Creating an Empty Dictionary

- You can create an empty dictionary to populate later.

Example: dictionary = {}

Example:

```
empty_dict = {}
print(empty_dict)
```

Output: {}

Or using dict()

```
empty_dict = dict()
print(empty_dict)  # Output: {}
```

3.Using dict() Constructor

- You can create a dictionary using the dict() function with key-value pairs as arguments.
- **Syntax: dictionary = dict(key1=value1, key2=value2, ...)**
- **Example:**

```
my_dict = dict(name="Sangam Kumar", age=25, city="Patna Bihar")
print(my_dict)
```

- **Output:** {'name': 'Sangam Kumar', 'age': 25, 'city': 'Patna Bihar'}

4. Using dict() with Tuples or Lists

- You can pass a list or tuple of key-value pairs to dict().

❖ **Using a list of tuples**

- **Syntax:** dictionary = dict([(key1, value1), (key2, value2), ...])
- **Example:**

```
my_dict = dict([("name", "Sangam Kumar"), ("age", 25), ("city", "Patna Bihar")])
print(my_dict)
```

- **Output:** {'name': 'Sangam Kumar', 'age': 25, 'city': 'Patna Bihar'}

5. Using Dictionary Comprehension

- Create a dictionary dynamically based on an expression or condition.
- **Syntax:** **dictionary = {key_expression: value_expression for item in iterable}**
- **Example:**

```
squared = {x: x ** 2 for x in range(5)}
print(squared)
```

- **Output:** {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

6. Using dict.fromkeys()

- This method creates a dictionary from a sequence of keys, all having the same value.
- **Syntax:** **dictionary = dict.fromkeys(keys, value)**
- **Example:**

```
keys = ["a", "b", "c"]
my_dict = dict.fromkeys(keys, 0)  # All values initialized to 0
print(my_dict)
```

- **Output:** {'a': 0, 'b': 0, 'c': 0}

7. Using a Nested Dictionary

- You can define a dictionary containing other dictionaries.
- **Syntax:** **nested_dict = {"key1": {"subkey1": value1, ...}, ...}**
- **Example:**

```
nested_dict = {
    "person1": {"name": "Sangam Kumar", "age": 25},
     person2": {"name": "Bob", "age": 30},  }
    print(nested_dict)
```

- **Output:** {'person1': {'name': 'Sangam Kumar', 'age': 25}, 'person2': {'name': 'Bob', 'age': 30}}

8. Using zip() to Combine Two Lists

- You can use zip() to combine two iterables into a dictionary.
- **Syntax:** dictionary = dict(zip(keys, values))
- **Example:**

```
keys = ["name", "age", "city"]
values = ["Sangam Kumar", 25, "Patna Bihar"]
my_dict = dict(zip(keys, values))
print(my_dict)
```

- **Output:** {'name': 'Sangam Kumar', 'age': 25, 'city': 'Patna Bihar'}

9. Directly Adding Key-Value Pairs

- You can dynamically create a dictionary by assigning key-value pairs.

Example:

```
my_dict = {}
my_dict["name"] = "Sangam Kumar"
my_dict["age"] = 25
print(my_dict)   }
```

❖ How to access data from the dictionary

1. [] (square brackets)
2. get()
3. keys()
4. values()
5. items()
6. for loop (direct iteration)

1. Accessing Values by Key Using Square Brackets ([])

- Directly access the value using its key.

Syntax: value = dictionary[key]

Example: my_dict = {"name": "Sangam", "age": 25, "city": "Patna Bihar"}
print(my_dict["name"])

Output: Sangam

2. Using the get() Method

- Access the value by key, with an optional default value if the key is missing.

Syntax: value = dictionary.get(key, default_value)

Example: my_dict = {"name": "Sangam", "age": 25}
print(my_dict.get("name"))

Output: Sangam

print(my_dict.get("city", "NA"))

Output: NA

3. Accessing Keys Using keys() Method

- Retrieve all keys in the dictionary.

Syntax: keys = dictionary.keys()

Example: my_dict = {"name": "Sangam", "age": 25}
print(list(my_dict.keys())) # Output: ['name', 'age']

4. Accessing Values Using values() Method

- Retrieve all values in the dictionary.

Syntax: values = dictionary.values()

Example: my_dict = {"name": "Alice", "age": 25}
print(list(my_dict.values())) # Output: ['Alice', 25]

5. Accessing Key-Value Pairs Using items() Method

- Retrieve all key-value pairs as tuples.

Syntax: items = dictionary.items()

Example: my_dict = {"name": "Sangam", "age": 25}
print(list(my_dict.items()))

Output: [('name', 'Sangam'), ('age', 25)]

6. Iterating Through a Dictionary

❖ **Iterating Over Keys**

my_dict = {"name": "Sangam ", "age": 25}
for key in my_dict:
print(key)

Output: name, age

❖ **Iterating Over Values**

for value in my_dict.values():
print(value)

Output: Sangam, 25

❖ **Iterating Over Key-Value Pairs**

for key, value in my_dict.items():
print(f"{key}: {value}")

Output: name: Sangam
age: 25

7. Using in Operator to Check Existence

- Check if a key exists in the dictionary.

Syntax: exists = key in dictionary

Example:

my_dict = {"name": "Sangam ", "age": 25}
print("name" in my_dict)

Output: True

print("city" in my_dict) # Output: False

Example:

```
d={1:20,  2:20, 3:30, 4:40, 5:50}
for i in d:
  print(i, ":", d[i])
```

Output:

1 : 20
2 : 20
3 : 30
4 : 40
5 : 50

❖ Update data in a dictionary

- There are several ways to update the data in a dictionary. Here are the main methods with syntax and examples.

1. Using Square Brackets ([])

- Directly assign a new value to an existing key or add a new key-value pair.

Syntax: dictionary[key] = new_value

Example:

```
my_dict = {"name": "Sangam Kumar", "age": 25}
my_dict["age"] = 30  # Update existing key
my_dict["city"] = "New York"  # Add a new key-value pair
print(my_dict)
```

Output: {'name': 'Sangam', 'age': 30, 'city': 'New York'}

2. Using the update() Method

- Update the dictionary with another dictionary or key-value pairs.

Syntax: dictionary.update(other_dict_or_pairs)

Example 1 (with another dictionary):

```
my_dict = {"name": "Sangam Kumar", "age": 25}
updates = {"age": 30, "city": "New York"}
my_dict.update(updates)
print(my_dict)
```

Output: {'name': 'Sangam Kumar', 'age': 30, 'city': 'New York'}

Example 2 (with key-value pairs):

```
my_dict.update(city="Patna", country="India")
print(my_dict)
```

Output: {'name': 'Sangam Kumar', 'age': 30, 'city': 'Patna', 'country': 'India'}

3. Using setdefault() Method

- Update a dictionary by adding a key with a default value only if the key does not exist.

Syntax: dictionary.setdefault(key, default_value)

Example:

```
my_dict = {"name": "Sangam Kumar", "age": 25}
my_dict.setdefault("city", "Unknown")  # Adds 'city' only if it doesn't exist
my_dict.setdefault("age", 30)  # Does nothing as 'age' already exists
print(my_dict)
```

Output: {'name': 'Sangam Kumar', 'age': 25, 'city': 'Unknown'}

4. Using a Loop

- Update multiple values in a dictionary dynamically.

Syntax:

```
for key in dictionary:
    dictionary[key] = new_value_based_on_condition
```

Example:

```
my_dict = {"a": 1, "b": 2, "c": 3}
for key in my_dict:
    my_dict[key] += 10
print(my_dict)
```

Output: {'a': 11, 'b': 12, 'c': 13}

5. Merging Two Dictionaries (Python 3.9 and Later)

- Use the | operator to merge two dictionaries, creating a new one, or the |= operator to update in place.

Syntax:

```
new_dict = dict1 | dict2  # Merge into a new dictionary
dict1 |= dict2            # Update `dict1` in place
```

Example:

```
dict1 = {"name": "Sangam ", "age": 25}
dict2 = {"age": 30, "city": "Patna"}
```

- Merge into a new dictionary

```
merged_dict = dict1 | dict2
print(merged_dict)
```

Output: {'name': 'Sangam', 'age': 30, 'city': 'Patna'}

Update dict1 in place

```
dict1 |= dict2
print(dict1)
```

Output: {'name': 'Sangam', 'age': 30, 'city': 'Patna'}

6. Updating Nested Dictionaries

- Update values in a nested dictionary using square brackets or loops.

Example:

```
nested_dict = {"person": {"name": "Sangam", "age": 25}}
nested_dict["person"]["age"] = 30  # Update nested key
nested_dict["person"]["city"] = "Patna"  # Add new key in nested dictionary
print(nested_dict)
```

Output: {'person': {'name': 'Sangam', 'age': 30, 'city': 'Patna'}}

❖Delete data from the dictionary

1. **del**
2. **pop()**
3. **popitem()**
4. **clear()**
5. **del (entire dictionary)**
6. **Dictionary comprehension**

1. Using del Statement

- Removes a specific key-value pair or deletes the entire dictionary.

Syntax:

```
del dictionary[key]  # Removes a specific key-value pair
del dictionary       # Deletes the entire dictionary
```

Example:

```
my_dict = {"name": "Sangam Kumar", "age": 25, "city": "Patna "}
del my_dict["age"]  # Deletes the 'age' key
print(my_dict)
```

Output: {'name': 'Sangam', 'city': 'Patna'}

```
del my_dict  # Deletes the entire dictionary
```

2. Using pop() Method

- Removes a specific key and returns its value. Raises KeyError if the key doesn't exist (unless a default value is provided).

Syntax: value = dictionary.pop(key, default)

Example:

```
my_dict = {"name": "Sangam", "age": 25, "city": "Patna"}
age = my_dict.pop("age")  # Removes 'age' and returns its value
print(my_dict)
```

Output: {'name': 'Sangam', 'city': 'Patna'}

```
print(age)
```

Output: 25

3. Using popitem() Method

- Removes and returns the last inserted key-value pair as a tuple. Raises KeyError if the dictionary is empty.

Syntax: key, value = dictionary.popitem()

Example:

```
my_dict = {"name": "Sangam", "age": 25, "city": "Patna"}
last_item = my_dict.popitem()  # Removes the last item
print(my_dict)
```

Output: {'name': 'Sangam', 'age': 25}

```
print(last_item)
```

Output: ('city', 'Patna')

4. Using clear() Method

- Removes all key-value pairs, leaving the dictionary empty.

Syntax: dictionary.clear()

Example:

```
my_dict = {"name": "Alice", "age": 25, "city": "New York"}
my_dict.clear()
print(my_dict)
```

Output: {}

5. Using del with a Loop

- Removes multiple keys dynamically based on conditions.

Syntax:

```
for key in keys_to_remove:
   del dictionary[key]
```

Example:

```
my_dict = {"a": 1, "b": 2, "c": 3, "d": 4}
keys_to_remove = ["b", "d"]
for key in keys_to_remove:
   del my_dict[key]
print(my_dict)
```

Output: {'a': 1, 'c': 3}

6. Deleting Using Dictionary Comprehension

- Create a new dictionary excluding certain keys.

Syntax: new_dict = {key: value for key, value in dictionary.items() if condition}

Example:

```
my_dict = {"a": 1, "b": 2, "c": 3}
new_dict = {k: v for k, v in my_dict.items() if k != "b"}
print(new_dict)
```

Output: {'a': 1, 'c': 3}

7. Using dict.pop() in a Loop

- Dynamically remove multiple keys using pop() in a loop.

Syntax:

```
for key in keys_to_remove:
    dictionary.pop(key, None)  # Avoid KeyError if the key doesn't exist
```

Example:

```
my_dict = {"a": 1, "b": 2, "c": 3}
keys_to_remove = ["b", "d"]  # 'd' doesn't exist
for key in keys_to_remove:
    my_dict.pop(key, None)
print(my_dict)
```

Output: {'a': 1, 'c': 3}

Function Name	Syntax	Example
len()	len(dictionary)	my_dict = {"a": 1, "b": 2}; print(len(my_dict)) # Output: 2
get()	dictionary.get(key, default_value)	my_dict = {"a": 1}; print(my_dict.get("b", "Not Found")) # Output: Not Found
keys()	dictionary.keys()	my_dict = {"a": 1}; print(my_dict.keys()) # Output: dict_keys(['a'])
values()	dictionary.values()	my_dict = {"a": 1}; print(my_dict.values()) # Output: dict_values([1])
items()	dictionary.items()	my_dict = {"a": 1}; print(my_dict.items()) # Output: dict_items([('a', 1)])
pop()	dictionary.pop(key, default_value)	my_dict = {"a": 1}; print(my_dict.pop("a")) # Output: 1
popitem()	dictionary.popitem()	my_dict = {"a": 1}; print(my_dict.popitem()) # Output: ('a', 1)
clear()	dictionary.clear()	my_dict = {"a": 1}; my_dict.clear(); print(my_dict) # Output: {}
update()	dictionary.update(other_dict_or_pairs)	my_dict = {"a": 1}; my_dict.update({"b": 2}); print(my_dict) # Output: {'a': 1, 'b': 2}
setdefault()	dictionary.setdefault(key, default_value)	my_dict = {"a": 1}; my_dict.setdefault("b", 2); print(my_dict) # Output: {'a': 1, 'b': 2}
fromkeys()	dict.fromkeys(keys, value)	keys = ["a", "b"]; print(dict.fromkeys(keys, 0)) # Output: {'a': 0, 'b': 0}

❖ Zip()

- it used for combine the key and value or two list
- convert list into dictionary
- if the key and value element are not same count then convert dietary what value or element are same up to that value they will display

Example: write a python program to convert the list inti dictionary.

Program:

```
k=[1,2,3,4,5,6]
v=[10,20,30,40,50,60]
d=dict(zip(k,v))
print(d)
```

Output: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Example:

```
k=[1,2,3,4,5,6]
v=[10,20,30,40,50,60,70]
d=dict(zip(k,v))
print(d)
```

Output: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

❖ without using zip()

Program:

```
k=[1,2,3,4,5,6]
v=[10,20,30,40,50,60]
d={}
for i in range(min(len(k),len(v))):
   d[k[i]]=v[i]
print(d)
```

Output: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

❖ combine the two dictionaries

Q. write a python program to combine the two dictionaries.

Program:

```
d={1:10, 2:20, 3:30}
t={4:10, 5:20, 6:30}
d.update(t)
print(d)
```

Output: {1: 10, 2: 20, 3: 30, 4: 10, 5: 20, 6: 30}

❖ nested dictionary

- **nested dictionary is a dictionary inside another dictionary. It allows you to store complex, hierarchical data in a structured way, like storing information about multiple items where each item has its own dictionary of details.**

Example:

```
d={101:{'Name':'Anjraj','age':'15', 'dept':'cse','marks':'90','id':'333'},
  102:{'Name':'Anj','age':'16', 'dept':'cse','marks':'80','id':'303'},
  103:{'Name':'raj','age':'25', 'dept':'cse','marks':'90','id':'313'}}
print(d[102]['marks'])
```

Output: 80

Example:

```
d={101:{'Name':'Anjraj','age':'15', 'dept':'cse','marks':'90','id':'333'},
  102:{'Name':'Anj','age':'16', 'dept':'cse','marks':'80','id':'303'},
  103:{'Name':'raj','age':'25', 'dept':'cse','marks':'90','id':'313'}}
for k in d.keys():
   if d[k]['Name']=='raj':
      print(d[k] ['marks'])
print(d[102]['marks'])
```

❖ adjacent Node

➢ list as a dictionary key value

Example:

```
d={1:[2,3,5,6],2:[1,2,3,4],3:[3,4,6,7,8],4:[]}
print(d[3])
```

Output: [3, 4, 6, 7, 8]

❖ aliasing

Example:

```
d={1:10,2:20,3:30,}
t=d
t[1]=100
print(t)
print(l)
```

Output:

```
{1: 100, 2: 20, 3: 30}
[3, 16, 6, 5, 2, 8, 31]
```

❖ cloning

Example:

```
d={1:10,2:20,3:30}
t=d.copy()
t[-1]=100
print(t)
print(d)
```

Output:

```
{1: 10, 2: 20, 3: 30, -1: 100}
{1: 10, 2: 20, 3: 30}
```

❖ IMPORTANT FUNCTION OF DICTIONARY:

1).dict():

- to create a dictionary
  - d=dict(): it creates empty dictionary
  - d=dict({100:"raj",200:"anj"}) it creates dictionary with specified elements
  - d=dict([(100:"raj",(200:"anj")]) it creates dictionary with the given list of tuple elements

2).len() return the number of items in the dictionary

3).clear(): to remove the elements from the dictionary

4).get(): to get the value associated with the key

5). d.get() if the key is available then returns the corresponding value otherwise returns None. it won't raise any error

6). d.get(key, default value) if the key is available then returns the corresponding value otherwise returns default value

Example:

```
d={100:"raj",200:"ravi",300:"anj"}
print(d[100])
print(d.get(100))
print(d.get(400))
print(d.get(100,"guest"))
print(d.get(400,"guest"))
print(d[400])
```

Output:

```
raj
raj
None
raj
guest
KeyError: 400
```

pop()

- d.pop(key)
- it removes the entry associated with the specified key and returns the corresponding value.
- if the specified key is not available then we will get keyError

Example:

```
d={100:"raj",200:"ravi",300:"anj"}
print(d.pop(100))
print(d)
print(d.pop(400))
```

Output:

```
raj
{200: 'ravi', 300: 'anj'}
KeyError: 400
```

popitem():

- it removes an arbitrary item(key-value) from the dictionary and returns it.

Example:

```
d={100:"raj",200:"ravi",300:"anj"}
```

```
print(d)
print(d.popitem())
print(d)
```

Output:

```
{100: 'raj', 200: 'ravi', 300: 'anj'}
(300, 'anj')
{100: 'raj', 200: 'ravi'}
```

Note: if the dictionary is empty then we will get keyError

```
d={}
print(d.popitem())  keyError
```

keys()

- it returns all keys associated either dictionary

```
d={100:"raj",200:"ravi",300:"anj"}
print(d.keys())
for k in d.keys():
   print(k)
```

Output:

```
dict_keys([100, 200, 300])
100
200
300
```

values():

- it returns all values associated with the dictionary

Example:

```
d={100:"raj",200:"ravi",300:"anj"}
print(d.values())
for v in d.values():
   print(v)
```

Output:

```
dict_values(['raj', 'ravi', 'anj'])
raj
ravi
anj
```

items(): it returns list of tuples representing key-value pairs

[(k,v),(k,v),(k,v)]

Program:

```
d={100:"raj",200:"ravi",300:"anj"}
for k,v in d.items():
   print(k,"---",v)
```

Output:

```
100 --- raj
200 --- ravi
    300--- anj
```

copy() : to create exactly duplicate dictionary (cloned copy)

- d1=d.copy()

setdefault():

d.setdefault(k,v)

- if the key is already available then this function returns the corresponding value.
- if the key is not available then the specified key-value will be added as new item to the dictionary

Example:

```
d={100:"raj",200:"ravi",300:"anj"}
print(d.setdefault(400,"ram"))
print(d)
print(d.setdefault(100,"sachin"))
print(d)
```

Output:

```
ram
{100: 'raj', 200: 'ravi', 300: 'anj', 400: 'ram'}
raj
{100: 'raj', 200: 'ravi', 300: 'anj', 400: 'ram'}
```

12).update():

- d.update(x)
- all items present in the dictionary x will be added to dictionary d

❖ DICTIONARY COMPREHENSION

- comprehension concept applicable for dictionaries also.

Example:

```
squares={x:x*x for x in range(1,6)}
print(squares)
doubles={x:2*x for x in range(1,6)}
print(doubles)
```

Output:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
```

Q. write a program to take dictionary from the keyboard and print the sum of values?

Program:

```
d=eval(input("enter dictionary:"))
s=sum(d.values())
print("sum=",s)
```

Output: enter dictionary:{"A":3,"B":33,"C":333}
sum= 369

Q. write a program to find number of occurrences of each letter present in the given string?

Program:

```
w=input("Enter any word:")
d={}
for x in w:
    d[x]=d.get(x,0)+1
for k,v in d.items():
    print(k,"occurred",v,"times")
```

Output:

```
Enter any word:t3skillcenter
t occurred 2 times
3 occurred 1 times
s occurred 1 times
k occurred 1 times
i occurred 1 times
l occurred 2 times
c occurred 1 times
e occurred 2 times
n occurred 1 times
r occurred 1 times
```

Q. write a program to find number of occurrences of each vowel present in the given string?

Program:

```
w=input("Enter any word:")
vowels={'a','e','i','o','u'}
d={}
for x in w:
    if x in vowels:
        d[x]=d.get(x,0)+1
for k,v in sorted(d.items()):
    print(k,"occurred",v,"times")
```

Output:

```
Enter any word:t3skillcenter
e occurred 2 times
i occurred 1 times
```

Q. write a program to accept student name and marks from the keyboard and creates a dictionary. also display student marks by taking student name as input?

Program:

```
n=int(input("enter the number of students:"))
d={}
for i in range(n):
    name=input("enter student name:")
    marks=input("enter student marks:")
    d[name]=marks
while True:
    name=input("enter student name to get marks:")
    marks=d.get(name,-1)
    if marks==-1:
        print("student not foound")
    else:
        print("the marks of",name,"are",marks)
    option=input("do you want to find another student marks[yes|no]")
    if option=="no":
        break
print("thanks for using our application:")
```

Output:

```
enter the number of students:3
enter student name:raj
enter student marks:96
enter student name:raju
enter student marks:93
enter student name:ramu
enter student marks:91
enter student name to get marks:raju
the marks of raju are 93
do you want to find another student marks[yes|no]no
thanks for using our application:
```

Question-1

1. Calculate the sum of all numerical values in a dictionary.
2. Find the key that has the highest numerical value in a dictionary.
3. Determine the average of all values in a numerical dictionary.
4. Count how many values in a dictionary are greater than a given number (e.g., 50).
5. Sort a dictionary based on its values in descending order.
6. Create a new dictionary with the square of each numerical value.
7. Compute the product of all values in a dictionary.
8. Find all keys in a dictionary that have a specific value (e.g., 100).
9. Increase each value in a dictionary by a fixed number (e.g., add 10 to all values).
10. Filter out and create a new dictionary containing only the key-value pairs where the value is an even number.

Question-2

1. **What is a dictionary in Python?**
   → A dictionary is a built-in data type that stores data in **key-value pairs**.
2. **Are dictionaries ordered in Python?**
   → Yes, since Python 3.7, dictionaries maintain **insertion order**.
3. **Can dictionary keys be duplicated?**
   → No, dictionary keys must be **unique**.
4. **What types can be used as dictionary keys?**
   → Only **immutable types** (like strings, numbers, tuples) can be used as keys.
5. **Can dictionary values be duplicated?**
   → Yes, values in a dictionary can be **duplicated**.
6. **How do you access a value in a dictionary?**
   → By using the **key inside square brackets** (e.g., dict[key]).
7. **What happens if you access a non-existent key?**
   → It raises a **KeyError** unless you use .get().
8. **How can you remove a key-value pair from a dictionary?**
   → By using the **del statement** or the .pop() method.
9. **How can you get all the keys of a dictionary?**
   → Use the .keys() method.
10. **Is a dictionary mutable or immutable?**
    → A dictionary is a **mutable** data structure

STRING

- Most commonly used object in any project and in any programming, language is string only.

❖ What is string?

- sequence of characters within either single quotes or double quotes is considered as a string.

Syntax:

```
S= 'TWKSAA'
S= "skills"
S= "RID bharat center"
```

Note: in most of the other languages like C, C++, java, a single character with in single quotes is treated as char data types value. But in python we are having char data type. Hence it is treated as string only.

Example:

```
s='a'
print(type(s))
s='skills'
print(type(s))
s="center"
print(type(s))
```

Output

```
<class 'str'>
<class 'str'>
<class 'str'>
```

❖ How to define multi-line string literals?

- We can define multi-line literals by using triple or double quotes.

Example:

```
s=''' twksaa skill center
   twksaa wit center
twksaa rid center'''
print(s)
```

Output

```
twksaa skill center
   twksaa wit center
twksaa rid center
```

Note:

- we can also use triple quotes to use single quotes or double quotes as symbol inside string literal.

Example:

```
s2='this is \'twksaa skills center'
s3='this is \"twksaa skills center'
s3="this is 'twksaa wit center"
s4='this is "twksaa skills & \"wit \'rid center'
s5='''This is Twksaa "skills"  'wit' "rid" center'''
print(s2)
print(s3)
print(s4)
print(s5)
```

Output

```
this is 'twksaa skills center
this is 'twksaa wit center
this is "twksaa skills & "wit 'rid center
This is Twksaa "skills"  'wit' "rid" center
```

❖ How to access characters of a string?

- We can access characters of a string by using the following ways.
  1). By using index.
  2). By using slice operator
  3). for loop
  4). while loop

1) By using index:

- Python supports both +Ve and -ve index.
- +Ve index means Left to Right(Forward Direction)
- -Ve Index means Right to Left (Backward Direction)

Example:

```
s='Twksaa skills center'      Output
print(s[0])                   T
print(s[2])                   k
print(s[10])                  l
print(s[-1])                  r
print(s[-10])                 l
print(s[-12])                 k
```

Note: if we are trying to access characters of a string with out of range index then we will get error saying : IndexError

Question: Write a program to accept some string from the keyboard and display its characters by index wise (both positive and negative index)

Program:

```
s=input("Enter a sentance")
i=0
for j in s:
    print("The character at +Ve Index= {} and -Ve Index= {} is: {}".format(i,i-len(s),j))
    i+=1
```

Output

```
Enter a sentance TWKSAA
The character at +Ve Index= 0 and -Ve Index= -6 is: T
The character at +Ve Index= 1 and -Ve Index= -5 is: W
The character at +Ve Index= 2 and -Ve Index= -4 is: K
The character at +Ve Index= 3 and -Ve Index= -3 is: S
The character at +Ve Index= 4 and -Ve Index= -2 is: A
The character at +Ve Index= 5 and -Ve Index= -1 is: A
```

2) Accessing characters by using slice operator:

- **Syntax:** s[Begging index : ending index : step]
  - **Begin Index:** from where we have to consider slice(substring)
  - **End index:** we have to terminate the slice (substring) at endindex-1
  - **Step:** incremented value.

Note:

- If we are not specifying begin index then it will consider from beginning of the string
- If we are not specifying end index then it will consider up to end of the string.
- The default value for step is 1.

Example:

```
s='Twksaa skills center'
print(s[2:8:3])
print(s[::4])
print(s[0::2])
print(s[1:9:])
print(s[6::1])
print(s[12:0:-1])
print(s[1:-9:2])
print(s[-9::2])
```

Output

```
ka
Taksn
Tka klscne
wksaa sk
 skills center
slliks aaskw
wsasi
l etr
```

❖ Behaviour of slice operator:

1) S[begging: end : step]
2) Step value can be either +Ve or -ve
3) If +Ve then it should be forward direction (left to right) and we have to consider begging to end -1
4) If -Ve then it should be backward direction (right to left) and we have to consider begging to end +1

Note: In the backward direction if end direction if end value is -1 then result is always empty.
In the forward direction if the end value is 0 then result is always empty.

❖ In forward direction:

- Default value for begging: 0
- Default value for end: length of string
- Default value for step: +1

❖ In Backward direction:

- Default value for begging: -1
- Default value for end: -(length of string +1)

Note: Either forward or backward direction, we can take both +ve and -ve for begging and end index. slice operator never raises indexError

3.By Using for loop:

1) Character-by-character access

```
text = "hello"
for ch in text:
   print(ch)
```

Output:

```
h
e
l
l
o
```

3) Print only vowels

```
text = "python"
for ch in text:
   if ch in "aeiou":
      print(ch)
```

Output:

```
o
```

2) Access by index

```
text = "world"
for i in range(len(text)):
   print(text[i])
```

Output:

```
w
o
r
l
d
```

4. By Using while loop.

1) Character-by-character access

```
text = "hello"
i = 0
while i < len(text):
    print(text[i])
    i += 1
```

2) Reverse the string

```
text = "world"
i = len(text) - 1
while i >= 0:
    print(text[i])
    i -= 1
```

3) Print characters until vowel appears

```
text = "bcrane"
i = 0
while i < len(text):
    if text[i] in "aeiou":
        break
    print(text[i])
    i += 1
```

Output:

```
b
c
r
```

Task-1: Count the number of vowels and consonants in a string.

```
text = "Programming is fun"
vowels = "aeiouAEIOU"
vowel_count = 0
consonant_count = 0

for ch in text:
    if ch.isalpha():
        if ch in vowels:
            vowel_count += 1
        else:
            consonant_count += 1

print("Vowels:", vowel_count)
print("Consonants:", consonant_count)
```

Output: Vowels: 5
Consonants: 11

Task-2: Find the first repeating character in a string.

```
text = "abcadefbg"
seen = ""
i = 0
found = False
while i < len(text):
    if text[i] in seen:
        print("First repeating character:", text[i])
        found = True
        break
    else:
        seen += text[i]
    i += 1
if not found:
    print("No repeating characters found.")
```

Output: First repeating character: a

❖ Mathematical operators for string:

- We can apply the following mathematical operators for string.
  1) + operator for concatenation
  2) * Operator for repetition

Example:

```
s1='Twksaa'
s2='skills center'
print(s1)
print(s2)
print(s1+s2)
s3="twksaa"
print(s3*3)
print(s3)
print(s3*1)
```

Output

```
Twksaa
skills center
Twksaaskills center
twksaatwksaatwksaa
twksaa
twksaa
```

Note:

1. To use + operator for string, compulsory both arguments should be str type.
2. To use * operator for string, compulsory one arguments should be str and other argument should be int.

❖ len() in-built function:

- we can use len() function to find the number of characters present in the string.

Example:

```
s='skills'
print(len(s)) # 6
```

Question: write a program to access each character of string in forward and backward direction by using while loop?

Program:

```
s='twksaa skills center'
n=len(s)
i=0
print('Forward direction')
while i<n:
    print(s[i], end='')
    i+=1
print()
print('Backward direction')
i=-1
while i>=-n:
    print(s[i],end='')
    i=i-1
```

Output

```
Forward direction
twksaa skills center
Backward direction
retnec slliks aaskwt
```

Alternative ways:

Program:

```
s= "LED skills center"
print('Forward direction')
for i in s:
```

Output

```
Forward direction
LED skills center
Forward direction
LED skills center
Backward direction
retnec slliks DEL
```

```
    print(i,end="")
print()
print('Forward direction')
for i in s[::]:
    print(i, end='')
print()
print('Backward direction')
for i in s[::-1]:
    print(i,end='')
```

❖ Checking membership:

we can check whether the character or string is the member of another string or not by using in and not in operators.

Example:

```
s=input("enter the string: ")
if 'askill' in s:
    print("yes")
else:
    print("No")
```

Output

```
enter the string:  twksaa skills center
No
enter the string:  twksaa askill center
yes
```

Example:

```
s='TWKSAA'
print ('t' not in s)
print ('T' not in s)
print ('AA' not in s)
print ('D' not in s)
print('TS' not in s)
print('KS' not in s)
print('Ta' not in s)
```

Output

```
True
False
False
True
True
False
True
```

Example:

```
main_string=input("Enter the string")
sub_string=input("Enter the substring")
if sub_string in s:
    print(sub_string, "is found in main string")
else:
    print(sub_string, "is not found in main string")
```

Output

```
Enter the string twksaa skills center
Enter the substring skills
skills is found in main string
```

❖ Comparison of string:

- we can use comparison operators (<, <=, >, >=) and equality operators (==, !=) for strings.
- Comparison will be performed based on alphabetical order.

Example:

```
s1=input("Enter first string:")
s2=input("Enter first string:")
if s1==s2:
    print("Both strings are equal")
elif s1<s2:
    print("first string is less than second string")
else:
    print("first string is greater than second string")
```

Output:

```
Enter first string: skills
Enter first string: skills
Both strings are equal
Enter first string: twksaa
Enter first string: dksra
first string is greater than second string
Enter first string: ram
Enter first string: shyam
first string is less than second string
```

❖ Removing spaces from the string:

- We can use the following 3 methods:
  1) strip → To remove spaces both sides
  2) rstrip() → To remove space at right hand side
  3) lstrip() → To remove spaces at left hand side

1) strip() : strip is used to remove either prefix or suffix character from a given string.

Example:

```
s= '  skills'
print(s)
print(s.strip())
```

Output

```
  skills
skills
```

-by default, strip will remove leading trading spaces of a given string

Example:

```
s='abcskillsc'
print(s.strip('abc'))
```

Output

```
skills
```

Note:

- if the given character is available prefix or suffix it will remove all those characters. In the above example abc remove from the prefix c remove from the suffix.

2) rstrip() : it will remove only from the prefix.

3) lstrip(): it is used for remove only from the prefix.

Example:

```
s='  skills '
print(s.rstrip())
print(s.lstrip())
```

Output

```
  Skills
skills
```

❖ Searching or finding substring:

- For finding the substring or searching the substring we will this following 4 method:
  1) find()
  2) rfind()
  3) index()
  4) rindex()

➢ **For forward direction:**
  1) find()
  2) index()

➢ **For backward direction:**
  1) rfind()
  2) rindex()

- If we want to search sub string to given main string:

1) index() will return first occurrence of the sub string within the main string if sub string. If sub string will not available through error.

- Syntax:
- s.index(sub string, starting index, end index)

2) rindex() will return last occurrence of a substring, if substring available if substring is not available it will through Error

- Syntax:
- s.rindex(sub string, starting index, end index)

3) find(): will return 1st occurrence of the substring within the main string if sub string is available if sub string is not available return -1.

- Syntax:
- s.find(sub string, starting index, end index)

4) rfind(): will return last occurrence of the sub string within the main string if substring is available in main string if substring is not available return -1.

- Syntax:
- s.rfind(sub string, starting index, end index)

Example:

```
s='welcome to all hai to all'   Output
print(s.index('to'))            8
print(s.rindex('to'))           19
print(s.find('to'))             8
print(s.rfind('to'))            19
print(s.find('too'))            -1
print(s.index('too'))           ValueError: substring not found
```

Question: without using predefined function. How to searching operation will perform.

Program:

```
s=input("enter the string")
sub=input("enter sub string")
for i in range(len(s)-len(sub)+1):
    for j in range(len(sub)):
        if s[i+j]!=sub[j]:
            break
        else:
            print(i)
            break
    else:
        print(-1)
```

Output

```
enter the string skills
enter sub string s
0
5
```

Question: Program to display all positions of substring in a given main string.

Program:

```
s=input("Enter the string")
sub=input("Enter sub string")
flag=False
pos=-1
n=len(s)
while True:
    pos=s.find(sub,pos+1,n)
    if pos==-1:
        break
    print("Found at position", pos)
    flag=True
if flag==False:

    print("Not Found")
```

Output

```
Enter the string ababdasffbasaba
Enter sub string a
Found at position 0
Found at position 2
Found at position 5
Found at position 10
Found at position 12
Found at position 14

Enter the string absdgabababgdagba
Enter sub string ab
Found at position 0
Found at position 5
Found at position 7
Found at position 9
```

❖ Counting substring in the given string:

- We can find number of occurrence of substring present in given string by using **count()** method.
  - ➢ S.count(substring) --> it will search throughout the string.
  - ➢ S.count(substring, begging, end)--> it will search from begging index to end-1 to index.

Example:

```
s='abcabacbcbacbabcabc'
print(s.count('a'))
print(s.count('ab'))
print(s.count('a',3,6))
```

Output

```
6
4
2
```

❖ Replacing a string with another string:

- **Syntax:** S.replace(oldsring,newstring)

Example:

```
s='TWKSAA skills center'
print("old string=", s)
new_string=s.replace("skills","RID")
new_string=s.replace("center","CENTER")
print("new string=",new_string)
```

Output

```
old string= TWKSAA skills center
new string= TWKSAA skills CENTER
```

Example:

```
s='abababababababababa'
print("main string=",s)
s1=s.replace("b","a")
print("after replace=",s1)
```

Output

```
main string= abababababababababa
after replace= aaaaaaaaaaaaaaaaaaa
```

❖ Splitting of strings:

- We can split the given string according to specified separator by split() method.
- By default, delimiter for split is space default number of splits are all possible split.
- The return type of split() method is list.
- **Syntax:** s.split(delimiter, no of split)

Example:

```
s='twksaa skills center'
l=s.split()
for i in l:
    print(i)
```

Output

```
twksaa
skills
center
```

Example:

```
print("foundation day")
s='30-09-2023'
l=s.split('-')
for i in l:
    print(i)
```

Output

```
foundation day
30
09
2023
```

Example:

```
s='tw ab sk rp tr pm cm dm'
l=s.split()
print(l)
```

Output

```
['tw', 'ab', 'sk', 'rp', 'tr', 'pm', 'cm', 'dm']
```

Example:

```
s='tw ab sk rp tr pm cm dm'
l=s.split(' ',3)
print(l)
```

Output

```
['tw', 'ab', 'sk', 'rp tr pm cm dm']
```

- In the above example we specified number of splits are 3 splits will divide token be always from left to right if divide only three possibilities.
- if the specified no of splits is greater than possible number of splits it will consider only possible number of splits.

Example:

```
s='tw ab sk rp tr pm cm dm'
l=s.split(' ',12)
print(l)
```

Output

['tw', 'ab', 'sk', 'rp', 'tr', 'pm', 'cm', 'dm']

Example:

```
s='30/09/2023'
l=s.split('/')
print(l)
```

Output

['30', '09', '2023']

- rsplit(): split the given string in token from right side onwards. If you do all possible split, there is no difference in result split and rsplit.

Example:

```
s='30/09/2023'
l=s.split('/',1)
print(l)
r=s.rsplit('/',1)
print(r)
```

Output

['30', '09/2023']
['30/09', '2023']

Example:

```
s='30/09/2023'
l=s.split('/')
print(l)
r=s.rsplit('/',)
print(r)
```

Output

['30', '09', '2023']
['30', '09', '2023']

❖ joining of strings:

- To combine list of tokens into a single string join () will used based on the delimiter(like - ,:,/,#,@, “ “, & , any symbol, word , etc) join () function will be used.
- we can join a group of strings (List or tuple) with the given separator.
- Syntax: S=separator.join(group of strings)

Example:

```
l=['30', '09', '2023']
s='-'.join(l)
print(s)
```

Output

30-09-2023

Example:

```
a=['1','2','3', '4', '5','6']
print("".join(a))
```

Output

123456

Example:

```
l=['twksaa', 'skills', 'center']
s="".join(l)
print(s)
```

Output

twksaaskillscenter

❖ Changing case of the string:

- We can change the case of string by using this following 4 method.

1. **upper()** → To convert all characters to upper case alphabet.
2. **lower()** → To convert all characters to lower case alphabet.
3. **swapcase()** → convert all lower case characters to upper case and all upper case characters to lower case.
4. **title()**→ to convert all character to title case i.e., first character in every word should be upper case and all remaining characters should be in lower case.
5. **Capitalize()**→only first character will be converted to upper case all remaining characters can be converted to lower case.

Example:

```
s="twksaa skills center"
s1="TWKSAA RID CENRTER"
print(s.upper())
print(s1.lower())
print(s.swapcase())
print(s.title())
print(s.capitalize())
```

Output

```
TWKSAA SKILLS CENTER
twksaa rid cenrter
TWKSAA SKILLS CENTER
Twksaa Skills Center
Twksaa skills center
```

❖ Checking starting and ending part of the string:

- We will used following method for checking starting and ending part of the string.
    1. s. startwith(substring)
    2. s.endswith(substring)

Example:

```
s="skills workshop internship training research innovation discovery"
print(s.startswith('skills'))
print(s.endswith("discovery"))
print(s.endswith("skills"))
```

Output

```
True
True
False
```

❖ To check type of characters, present in a string:

- **isalnum()**→ Return True if characters are alphanumeric means (a to z, o to 9)
- **isalpha()** → Return True if all characters are only alphabet symbols (a to z, A to Z)
- **islower()**→ Return True if all characters are lower case alphabet symbols
- **isdigit()** → Return True if all characters are digit only (0 to 9)
- **isupper()**→ Return True if all characters are upper case alphabet symbols
- **istitle()** → Return True if string is in title case
- **isspace()** → Return True if string contains only spaces.

Example:

	Output
s='skills30923'	
print(s.isalpha())	False
print('rid'.isalpha())	True
print('twksaa'.isdigit())	False
print('30092023'.isdigit())	True
print('RPT'.islower())	False
print('abt'.isupper())	False
print('abt'.islower())	True
print('skills30923'.islower())	True
print('Twksaa skills Center'.istitle())	False
print('Twksaa Skills Center'.istitle())	True
print(' '.isspace())	True

❖ Formatting the strings:

- We can format the string with variable values by using replacement operator() and format() method.

Example:

```
name='Twksaa skills center'
place='India'
fd='30-09-2023'
print("{} is a led based skills center located in {} Foundation day is: {}".format(name,place,fd))
print("{0} is a led based skills center located in {1} Foundation day is: {2}".format(name,place,fd))
print("{x} is a led based skills center located in {y} Foundation day is:
{z}".format(x=name,y=place,z=fd))
```

Output:

```
Twksaa skills center is a led based skills center located in India Foundation Day is: 30-09-2023
Twksaa skills center is a led based skills center located in India Foundation Day is: 30-09-2023
Twksaa skills center is a led based skills center located in India Foundation Day is: 30-09-2023
```

Important Method in python

Method	Syntax	Example
1. capitalize()	str.capitalize()	"hello".capitalize() → 'Hello'
2. casefold()	str.casefold()	"HELLO".casefold() → 'hello'
3. center()	str.center(width, fillchar)	"hi".center(5, '*') → '**hi*'
4. count()	str.count(substring)	"banana".count('a') → 3
5. encode()	str.encode()	"hello".encode() → b'hello'
6. endswith()	str.endswith(suffix)	"hello.py".endswith('.py') → True
7. expandtabs()	str.expandtabs(tabsize)	"a\tb".expandtabs(4) → 'a b'
8. find()	str.find(substring)	"hello".find('e') → 1
9. format()	"{}".format(value)	"My name is {}".format("Tanmay") → 'My name is Tanmay'
10. format_map()	str.format_map(mapping)	"Hello {name}".format_map({'name':'Raj'}) → 'Hello Raj'
11. index()	str.index(substring)	"hello".index('l') → 2
12. isalnum()	str.isalnum()	"abc123".isalnum() → True
13. isalpha()	str.isalpha()	"abc".isalpha() → True
14. isascii()	str.isascii()	"abc".isascii() → True
15. isdecimal()	str.isdecimal()	"123".isdecimal() → True
16. isdigit()	str.isdigit()	"123".isdigit() → True
17. isidentifier()	str.isidentifier()	"var1".isidentifier() → True
18. islower()	str.islower()	"hello".islower() → True
19. isnumeric()	str.isnumeric()	"123".isnumeric() → True
20. isprintable()	str.isprintable()	"abc!".isprintable() → True
21. isspace()	str.isspace()	" ".isspace() → True
22. istitle()	str.istitle()	"Hello World".istitle() → True
23. isupper()	str.isupper()	"HELLO".isupper() → True
24. join()	"sep".join(iterable)	"-".join(["a","b"]) → 'a-b'

Method	Syntax	Example
25. ljust()	str.ljust(width, fillchar)	"hi".ljust(4, '.') → 'hi..'
26. lower()	str.lower()	"HELLO".lower() → 'hello'
27. lstrip()	str.lstrip(chars)	" hello".lstrip() → 'hello'
28. maketrans()	str.maketrans(x, y)	str.maketrans("a", "b")
29. translate()	str.translate(table)	"apple".translate(str.maketrans("a", "b")) → 'bpple'
30. partition()	str.partition(sep)	"a-b-c".partition("-") → ('a', '-', 'b-c')
31. replace()	str.replace(old, new)	"hello".replace("l", "x") → 'hexxo'
32. rfind()	str.rfind(substring)	"hello".rfind("l") → 3
33. rindex()	str.rindex(substring)	"hello".rindex("l") → 3
34. rjust()	str.rjust(width, fillchar)	"hi".rjust(5, '*') → '***hi'
35. rstrip()	str.rstrip(chars)	"hi ".rstrip() → 'hi'
36. rsplit()	str.rsplit(sep)	"a,b,c".rsplit(",", 1) → ['a,b', 'c']
37. split()	str.split(sep)	"a b c".split() → ['a', 'b', 'c']
38. splitlines()	str.splitlines()	"a\nb\nc".splitlines() → ['a', 'b', 'c']
39. startswith()	str.startswith(prefix)	"hello.py".startswith("h") → True
40. strip()	str.strip(chars)	" hi ".strip() → 'hi'
41. swapcase()	str.swapcase()	"HeLLo".swapcase() → 'hEllO'
42. title()	str.title()	"hello world".title() → 'Hello World'
43. upper()	str.upper()	"hello".upper() → 'HELLO'
44. zfill()	str.zfill(width)	"42".zfill(5) → '00042'

STRING BASED PROBLEM

1. Write a program to reverse the given string.
2. Write a program to reverse the order of the words
3. Write a program to check the given string is palindrome or not
4. Write a program to reverse the internal content of each word
5. Write a program to reverse the string without reversing the special symbols
6. Write a program to print characters at odd position and even position for the given string
7. Write a program to extract only numeric character from a largest possible even number first that given string are contains alpha number characters.
8. Write a program to merge characters of 2 string into a single string by taking characters alternatively.
9. Write a program to return the super reduced string.
10. Write a program to sort the characters of the string and first alphabets symbols followed by numeric values.
11. Write a program to check where given string is pangram or not
12. Write a program to check where given string is anagram or not
13. Write a program for the following requirement
    Input: a3b6c2
    Output: aaabbbbbbcc
14. Write a program to perform the following activity
    Input: a4k3b2
    Output: aeknbd
15. Write a program to Remove Duplicate characters from the given input string?
    Input: ABCDABACBEEDBCFBDBDCFCFDFC
    Output: ABCDEF
16. Write a program to find the number of occurrences of each character present in the given string
17. Write a program to perform the following task?
    Input: 'one two three four five six seven'
    Output: 'one owt three ruof five xis seven'
18. Write a program to find the character weight of alphabet of given string s "TWKSAA"
19. Write a program to perform the following task
    Input: aaabbcddcde
    Output: [('a',4), ('b',2),('c',1),('d', 3),('e',1)}
    a4b2c1d3e1
20. Write a program to get opposite of previous (above) question.
21. Write a program to check the how many times 'rama' can be form a given string.
    Input:rrmmaaarwxyz
22. Write a program to display the edit distance to make given string as a palindrome. Edit distance between a toc is 2 i.e, a-b then b-c , a to z is 25

Program-1

```
s=input("Enter the string")
#1st method
print(s[::-1])
#2nd method
print("".join(reversed(s)))
#3rd method
i=len(s)-1
t=""
while i>0:
   t=t+s[i]
   i=i-1
print(t)
```

Output:

```
Enter the string skill
lliks
lliks
llik
```

Program-2

```
s=input("Enter the string: ")
l=s.split()
l1=[]
i=len(l)-1
while i>=0:
   l1.append(l[i])
   i=i-1
r=" ".join(l1)
print(r)
```

Output:

```
Enter the string:  TWKSAA SKILLS CNETER
CNETER SKILLS TWKSAA
```

Program-3

```
s=input("Enter the string: ")
if s==s[::-1]:
   print("palindrome")
else:
   print("not palindrome")
#2nd method
s=input("Enter the string: ")
i=0
j=len(s)-1
while i<j:
   if s[i]!=s[j]:
      print("not palindrome")
      break
   i=i+1
   j=j-1
else:
   print("palindrome")
```

output:

```
Enter the string: madam
palindrome
Enter the string: skills
not palindrome
```

Program-4

```
s=input("Enter the sring: ")
l=s.split()
l1=[]
i=0
while i<len(l):
   l1.append(l[i][::-1])
   i=i+1
r=" ".join(l1)
print(r)
```

Output:

```
Enter the sring: twksaa skills center
aaskwt slliks retnec
```

Program-5 write a python program to reverse the string without reversing the special symbol.

```
l=list(input())
i=0
j=len(l)-1
while i<j:
   if not l[i].isalnum():
      i=i+1
   elif not l[j].isalnum():
      j=j-1
   else:
      l[i],l[j]=l[j],l[i]
      i=i+1
      j=j-1
print("".join(l))
```

output:

```
a@g#c%dgh@#ghg@
g@h#g%hgd@#cga@
```

6. Write a program to print characters at odd position and even position for the given string

Program-6:

```
s=input("Enter the sring: ")
print("charcter at even position: ", s[0::2])
print("charcter at odd position: ", s[1::2])
#2nd method
s=input("Enter the sring: ")
i=0
print("charcter at even position:")
while i<len(s):
```

```
        print(s[i],end=',')
        i=i+2
    print()
    print("charcter at odd position:")
    i=1
    while i<len(s):
        print(s[i],end=',')
        i=i+2
```

Output:

```
Enter the sring: skills
charcter at even position:  sil
charcter at odd position:  kls
Enter the sring: twksaa
charcter at even position:
t,k,a,
charcter at odd position:
w,s,a,
```

7. write a python program to take input string alphanumeric and extract only numeric character from a largest possible even number.

Program-7:

```
s=input()
l=[]
for i in s:
    if i.isdecimal():
        l.append(i)
l.sort(reverse=True)
for i in range(len(l)-1,-1,-1):
    if int(l[i])%2==0:
        l.append(l.pop(i))
        print(int(''.join(l)))
        break
else:
    print(-1)
```

Output:

```
d4f6g5h7
7654
```

8. Write a program to merge characters of 2 string into a single string by taking characters alternatively.

Program-8:

```
s1=input("Enter the sring: ")
s2=input("Enter the sring: ")
r=''
i,j=0,0
while i<len(s1) or j<len(s2):
    if i<len(s1):
        r=r+s1[i]
```

```
        i+=1
    if j<len(s2):
        r=r+s2[j]
        j+=1
print(r)
```

Output:

```
Enter the sring: raja
Enter the sring: ram
rraajma
```

9. write a python program to reduce super reduced string (means if two adjacent characters are same, we need to delete this two character)

Program-9:

```
s=list(input())
i=0
while i<len(s)-1:
    if s[i]==s[i+1]:
        del s[i]
        del[i]
        i=0
    else:
        i=i+1
if len(s)==0:
    print("empty string")
else:
    print("".join(s))
```

Output:

```
abbcbbca
abcbca
```

10. Write a program to sort the characters of the string and first alphabets symbols followed by numeric values.

Program-10:

```
s=input("Enter the sring: ")
s1=s2=r=''
for i in s:
    if i.isalpha():
        s1=s1+i
    else:
        s2=s2+i
for i in sorted(s1):
    r=r+i
for i in sorted(s2):
    r=r+i
print(r)
```

Output:

```
Enter the string: B6C3D1E2A8
ABCDE12368
```

14. write a python program to check where the given string is pangram or not(means it should contain all (A-Z) alphabet.)

Program-11:

```
s=input().lower()
for i in range(ord('a'),ord('z')+1):
   if chr(i) not in s:
      print("not")
      break
else:
   print("yes")
```

Output:

```
asdfghjklqwertyuiopzxcvbnm
yes
```

15. write a python program to check the given string is anagram or not. (Means if rearrange the one string it will need to from other (one string character present in other string (order not managed)))

Program-12:

```
s1=list(input())
s2=list(input())
s1.sort()
s2.sort()
if s1==s2:
   print("anagrams")
else:
   print("not")
```

Output:

```
abcd
bdac
anagrams
```

2nd method.....

```
s1=input()
s2=input()
for i in set(s1):
   if s1.count(i)!=s2.count(i):
      print("no")
      break
else:
   print("yes")
```

Output:

```
abcd
bdac
yes
```

3rd method.....

```
s1=input()
s2=input()
if len(s1)==len(s2):
   for i in s1:
      s2=s2.replace(i," ",1)
```

```
if len(s2)==0:
   print("anagrams")
else:
   print("not anagrams")
```

Output:

```
aabbcc
abba
not anagrams
```

16. Write a program for the following requirement

Input: a3b6c2

Output: aaabbbbbbcc

Program-13:

```
s=input("Enter the string: ")
r=''
for i in s:
   if i.isalpha():
      r=r+i
      previous=i
   else:
      r=previous*(int(i)-1)
print(r)
```

Output:

```
Enter the string: a3b6c2
aaabbbbbbcc
```

17. Write a program to perform the following activity

Input: a3k3b2c6

Output: adknbdci

Program-14:

```
s=input("Enter the sring: ")
r=''
for i in s:
   if i.isalpha():
      r=r+i
      previous=i
   else:
      r=r+chr(ord(previous)+int(i))
print(r)
```

Output:

```
Enter the sring: a3k3b2c6
adknbdci
```

18. Write a program to Remove Duplicate characters from the given input string?

Input: ABCDABACBEEDBCFBDBDCFCFDFC

Output: ABCDEF

Program-15

```
s=input("Enter the string: ")
l=[]
for i in s:
```

```
    if i not in l:
        l.append(i)
r=''.join(l)
print(r)
```

Output:

```
Enter the string: ABCDABACBEEDBCFBDBDCFCFDFC
ABCDEF
```

19. Write a program to find the number of occurrences of each character present in the given string

Program-16

```
s=input("Enter the string: ")
d={}
for i in s:
    if i in d.keys():
        d[i]=d[i]+1
    else:
        d[i]=1
for k,v in d.items():
    print("{}={} Times".format(k,v))
```

Output:

```
Enter the string: ABABSBADBDABABSADBAABSADBFHSFJSF
A=9 Times
B=9 Times
S=5 Times
D=4 Times
F=3 Times
H=1 Times
J=1 Times
```

20. Write a program to perform the following task?

Input: 'one two three four five six seven'

Output: 'one owt three ruof five xis seven'

Program-16

```
def reverse_even_chars(sentence):
    words = sentence.split()
    result = []
    for word in words:
        reversed_word = word[0] + ''.join(word[i] if i % 2 == 0 else word[i - 1] for i in
range(len(word), 0, -1))
        result.append(reversed_word)
    return ' '.join(result)
input_sentence = 'one two three four five six seven'
output_sentence = reverse_even_chars(input_sentence)
print(output_sentence)
```

Output:

```
one owt three ruof five xis seven
```

21. write a python program to find the weight of the all-alphabet characters.

Program:

```
s=input().lower()
sum=0
for i in s:
   sum=sum+(ord(i)-96)
print(sum)
```

Output:

```
aaa
3
```

23. write a python program to calculate the sum of given world (world s=d1+d2+d3).

Program:

```
s=input()
d=dict(zip([chr(i) for i in range(ord('a'),ord('z')+1)],range(1,27)))
w=0
i=0
j=len(s)-1
while i<j:
   w=w+abs(d[s[i]]-d[s[j]])
   i=i+1
   j=j-1
if i==j:
   w=w+d[s[i]]
print(w)
```

Output:

```
world
40
```

Theoretical Question and answer based on string

1. **Q:** What is a string in Python?
   A: A string is a sequence of characters enclosed in quotes.
2. **Q:** How do you create a string in Python?
   A: Use single ('), double ("), or triple (''') quotes.
3. **Q:** Are strings mutable in Python?
   A: No, strings are immutable.
4. **Q:** How can you access individual characters in a string?
   A: Using indexing like s[0].
5. **Q:** How do you get a substring in Python?
   A: Using slicing like s[1:4].
6. **Q:** What does len() do with strings?
   A: Returns the number of characters in the string.
7. **Q:** How do you convert a string to uppercase?
   A: Use str.upper().
8. **Q:** How do you convert a string to lowercase?
   A: Use str.lower().
9. **Q:** How do you remove whitespace from both ends of a string?
   A: Use str.strip().

10. **Q:** How do you replace characters in a string?
    A: Use str.replace(old, new).
11. **Q:** How can you check if a substring exists in a string?
    A: Use the in keyword, e.g., 'a' in 'apple'.
12. **Q:** How do you split a string into a list?
    A: Use str.split().
13. **Q:** How do you join a list into a string?
    A: Use separator.join(list).
14. **Q:** How do you check if a string starts with a prefix?
    A: Use str.startswith('prefix').
15. **Q:** How do you check if a string ends with a suffix?
    A: Use str.endswith('suffix').
16. **Q:** How do you count occurrences of a character in a string?
    A: Use str.count('char').
17. **Q:** How do you find the position of a character in a string?
    A: Use str.find('char').
18. **Q:** How do you capitalize the first letter of a string?
    A: Use str.capitalize().
19. **Q:** How do you check if all characters are digits?
    A: Use str.isdigit().
20. **Q:** How do you reverse a string in Python?
    A: Use slicing str[::-1].

DATA STRUCTURE

- A data structure is a fundamental concept in computer science and programming that refers to a way of organizing and storing data in a computer's memory or storage media.

❖ Types of Data Structure:

➢ There are two types of data structures

1. Linear Data Structures
2. Non-Linear Data Structures

❖ Linear Data Structures:

- linear data structures are a type of data structure in computer science that organize and store data elements in a sequential manner, where each element is connected to its predecessor and successor, forming a linear order.

➢ **Example:** stack, Queue

❖ Non-Linear Data Structures:

- Non-linear data structures, also known as hierarchical or tree-like data structures, are a type of data structure in computer science where data elements are organized and stored in a way that does not follow a strict linear order.

➢ **Example:** Tree and Graph

Stack

- A stack is a fundamental linear data structure in computer science that follows the Last-In-First-Out (LIFO) principle. In a stack, the last element added to the structure is the first one to be removed.

❖ Basic Operations:

- **Push:** This operation is used to add an element to the top of the stack.
- **Pop:** This operation is used to remove and return the top element from the stack.
- **Peek (or Top):** This operation retrieves the top element from the stack without removing it.
- **isEmpty:** Checks whether the stack is empty or not.

Use:

1. To reverse the given string
2. In the syntax analysis to check balancing parenthesis
3. In the recursion to store intermediated function called stack will be used
4. To convert even infixed expression into prefix or post fixed to expression

Note: to evaluate the post fixed expression stack will be used push and pop

Example:

```
stack = []
while True:
    print("\nOptions:")
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Exit")
    choice = input("Enter your choice: ")
    if choice == "1":
```

```
            item = input("Enter item to push: ")
            stack.append(item)
            print(f"Pushed: {item}")
        elif choice == "2":
            if stack:
                popped_item = stack.pop()
                print(f"Popped: {popped_item}")
            else:
                print("Stack is empty. Cannot pop.")
        elif choice == "3":
            if stack:
                print(f"Peeked: {stack[-1]}")
            else:
                print("Stack is empty. Cannot peek.")
        elif choice == "4":
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please choose a valid option.")
```

Output:

```
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter item to push: A
Pushed: A
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 1
Enter item to push: B
Pushed: B
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 3
Peeked: B
Options:
1. Push
2. Pop
3. Peek
4. Exit
```

```
Enter your choice: 2
Popped: B
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Popped: A
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 2
Stack is empty. Cannot pop.

Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 3
Stack is empty. Cannot peek.
Options:
1. Push
2. Pop
3. Peek
4. Exit
Enter your choice: 4
Exiting the program.
```

Example: write a python program to reverse the string by using the stack

```
stack = []
# Input the string
input_string = input("Enter a string: ")
# Push each character of the input string onto the stack
for char in input_string:
    stack.append(char)
# Pop characters from the stack to reverse the string
reversed_string = ""
while stack:
    reversed_string += stack.pop()
# Output the reversed string
print("Reversed string:", reversed_string)
```

Output: twksaa
aaskwt

Example:

```
stack = []
# Input the expression with parentheses
expression = input("Enter an expression with parentheses: ")
# Define a dictionary to match opening and closing parentheses
parentheses_map = {")": "(", "]": "[", "}": "{"}
# Iterate through each character in the expression
for char in expression:
   # If it's an opening parenthesis, push it onto the stack
   if char in "([{":
      stack.append(char)
   # If it's a closing parenthesis
   elif char in ")]}":
      # Check if the stack is empty or if the top of the stack doesn't match the corresponding
opening parenthesis
      if not stack or stack.pop() != parentheses_map[char]:
         print("Unbalanced parentheses")
         break
else:
   # If the loop completes without breaking, the parentheses are balanced
   if not stack:
      print("Balanced parentheses")
   else:
      print("Unbalanced parentheses")
```

Output:

```
Enter an expression with parentheses: (a + b) * [c - d]
Balanced parentheses
Enter an expression with parentheses: (a + b) * [c - d
Unbalanced parentheses
```

Note: Detail Expiation about data structure you can see TWSAA Data structure and algorithm Book

FUNCTION

❖ **Why function need?**

- if a group of statements is repeatedly required then it is not recommended to write these statements every times separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but function.

Function: function is a block of code that run when we call.

- The main advantage of function is code reusability.
- **Note:** in other languages are known as method as methods, procedures, subroutines etc.
- Pythion supports 2 types of functions.
    1) **Built in function**
    2) **User defined function**

1) Built in function:

- The functions which are coming along with python software automatically, are called built in function or pre-defined function.

Example: id()
type()
input()
eval() etc.

2) User defined function.

- The function which are developed by programmer explicitly according to business requirements are called user defined functions.

Syntax: to create user defined function.

```
def function_name(parameters):
        """ doc string """
                        ..............
                        ..............
        return value
function_name(arguments) #function call
```

- **def**:- Keyword to define a function.
- **function_name**:- Name of the function.
- **parameters**:- Inputs (optional) passed to the function.
- **"""doc string"""**:- (Optional) Description of what the function does.
- **return**:- Sends back the result when the function is called.
- **function_name()**:- Calls or runs the function.
- **Arguments:-** are the **actual values** you pass to a function **when calling it**.
- **: (colon)** is used for create the function block

Note: While creating functions we can use 2 keywords.

1) **def (mandatory)**
2) **return(optional)**

Example-1:

write a function to print twksaa skills center.

```
def tech():
   print("TWKSAA SKILLS CNETER")
tech()
```

Output: TWKSAA SKILLS CNETER

Example-2:

```
def greet(name):    # 'name' is a parameter
   print("Hello", name)
greet("Sangam Kumar")    # "Sangam Kumar" is an argument
```

Output: Hello Sangam Kumar

❖ What is return?

The return statement is used **inside a function** to:

1. **Send a result (output)** back to the caller.
2. **End the function's execution**.

🔸 Syntax:

```
def function_name(parameters):
    # code
    return value
```

🔸 **Example:**

```
def add(a, b):
    return a + b

result = add(2, 3)
print(result)  # Output: 5
```

🔹 Key Points:

- A function can return **any type**: number, string, list, tuple, object, etc.
- You can return **multiple values** as a tuple:
- **Example: -**

```
def stats(a, b):
    return a + b, a * b
s, m = stats(2, 3)
print(s, m)  # Output: 5 6
```

- If there is **no return**, the function returns None by default.

🔻 **Example Without return:**

```
def greet(name):
    print("Hello", name)

result = greet("Tanmay")
print(result)   # Output: Hello Tanmay \n None
```

❖ What is Parameter?

- A **parameter** is a **placeholder** or **variable** that you write **inside the function definition** to receive values.
- **Parameter = Input name** (used in function definition)
- **Argument = Actual value** (given during function call)

Example:

```
def add(a, b):  # 'a' and 'b' are parameters
    print(a + b)
add(3, 4)      # 3 and 4 are arguments → Output: 7
```

❖ What is argument?

- **argument is the actual value you give to a function when you call it**
- **Example:**

```
def greet(name="Guest"):
    print("Hello", name)

greet()          # Output: Hello Guest
greet("Ankit")   # Output: Hello Ankit
```

Types of arguments

1. Default arguments

Definition: Arguments that take default values if not provided during the function call.

Example:

```
def fun(a=0, b=0):
    print(f"{a}+{b}={a+b}")

fun()      # Output: 0+0=0
fun(1)     # Output: 1+0=1
fun(2, 3)  # Output: 2+3=5
```

2. Positional arguments:

Definition: Values passed to a function in the same order as the parameters are defined.

```
def fun(a, b):
    print(f"{a}+{b}={a+b}")
fun(10, 20)          # Output: 10+20=30)
```

3. Keyword arguments:

Definition: Arguments passed using parameter names, regardless of order.

```
def fun(a, b):
    print(f"{a}+{b}={a+b}")
fun(b=10, a=20)      # Output: 20+10=30
```

Example of combining positional as well as keyword arguments

You can mix positional and keyword arguments, but positional must come first.

```
def fun(a, b, c):
    print(f"{a}+{b}+{c}={a+b+c}")
fun(10, c=20, b=30)  # Output: 10+30+20=60
```

4. Variable length positional arguments:

Definition: Allows passing any number of positional arguments, stored as a tuple.

```
def fun(*a):
    print(f"Length of arguments: {len(a)} and the sum is - {sum(a)}")

fun(1)          # Output: Length of arguments: 1 and the sum is - 1
fun(2, 3)       # Output: Length of arguments: 2 and the sum is - 5
fun()           # Output: Length of arguments: 0 and the sum is - 0
fun(1, 2, 3, 4)  # Output: Length of arguments: 4 and the sum is – 10
```

Question. Program to find length of given arguments as well as sum

```
def fun(*a):
```

```
    s=0
    for i in a:
        s+=i
    print(s)
    print("Length of given arguments:",len(a))
fun(10,20,30,50)
```

5. Variable length keyword arguments:

Definition: Accepts any number of keyword arguments, stored as a dictionary.

```
def fun(**a):
    print(a)

fun(a=1, b=20)                  # Output: {'a': 1, 'b': 20}
fun(f="blue", s="red", t="green")   # Output: {'f': 'blue', 's': 'red', 't': 'green'}
```

Type	Syntax	Collected As	Purpose
Positional	`def fun(a)`	Variable	Normal fixed input
Default	`def fun(a=10)`	Variable	Uses default if no value is passed
Keyword	`fun(a=5)`	—	Argument by name
Variable Positional (`*`)	`def fun(*a)`	Tuple	Accepts multiple positional arguments
Variable Keyword (`**`)	`def fun(**a)`	Dictionary	Accepts multiple keyword arguments

Combination of variable length args

- You can combine *args and **kwargs in a function to accept.
  - ***args** → any number of **positional arguments** (stored as a tuple)
  - ****kwargs** → any number of **keyword arguments** (stored as a dictionary)

Example:

```
def fun(*a, **arg):
    print(a, arg)

fun(1, 2, 3, f="hi", g="two", h="three")
```

Explanation:

- *a captures: (1, 2, 3) → as a tuple
- **arg captures: {'f': 'hi', 'g': 'two', 'h': 'three'} → as a dictionary

Important Rule: - In function definition, the correct order is:

```
def fun(positional, *args, default=0, **kwargs):
    ...
```

Note: Always place *args before **kwargs.

➔ Combination of default, positional and keyword

```
"""
Default Arguments
```

Non-Default Arguments- Positional & Keyword Arguments

Rule- Positional & Keyword Arguments, then followed by default args

```
"""
def fun(a,b,c,d,e=0,f=0):
    print(a,b,c,d,e,f)
fun(10,20,d=40,c=30)
```

Example-1: of combination of all five types of args

```
"""
1.Positional, 2. Default Argument 3. Keyword Args 4. Variable-length positional
5.Variable length keyword Args
"""
def fun(a,b,c=0,*arg,**args):
    print(f"a={a},b={b},c={c},var-pos:{arg},var-key-{args}")
fun(10,11,1,2,3,f=100,s=200)
```

- While calling the functions, the args should be given in order of
- positional first and then keyword args

```
def fun(a,b=0,**arg):
    print(a,b,arg)
fun(10,f=22,g=55,b=20)
fun(10,b=20,f=22,g=55)
```

➔ **use of / restricts the arguments declared before the / to be positional**

```
def fun(a,b,/,c,d):
    print(f"a={a},b={b},c={c},d={d}")
fun(10,20,d=50,c=40)
fun(10,20,30,40)
```

➔ **use of *- It restricts the arguments declared after * as keyword only**

```
def fun(a,b,*,c,d):
    print(f"a={a},b={b},c={c},d={d}")
fun(1,2,d=3,c=4)
fun(b=1,a=2,d=3,c=4)
```

➔ Use of return keyword- The result of the function(or the value returned **by the function) can be stored**

```
def exp():
    a=10
    b=20
    return a+b
a=exp()
print("Value stored of fun",a)
```

Numerical Question Based on Function

Simple Questions (1–10)

1. Write a function to add two numbers.
2. Write a function that returns the square of a number.
3. Create a function that checks if a number is even or odd.
4. Write a function to find the maximum of three numbers.
5. Define a function to calculate the factorial of a number.
6. Create a function that prints the multiplication table of a number.
7. Write a function to calculate the area of a circle (use $\pi = 3.14$).
8. Define a function that returns the sum of all numbers from 1 to n.
9. Write a function that returns the average of three numbers.
10. Create a function that returns the cube of a number.

Medium-Level Questions (11–20)

11. Write a function to check if a number is prime.
12. Create a function that finds the sum of digits of a number.
13. Write a function that returns the reverse of a number.
14. Define a function that checks if a number is a palindrome.
15. Write a function to find the greatest common divisor (GCD) of two numbers.
16. Create a function that counts the number of digits in a number.
17. Write a function that returns the sum of even numbers in a given range.
18. Define a function to print all prime numbers between 1 and n.
19. Create a function that converts Celsius to Fahrenheit.
20. Write a function that accepts marks of 5 subjects and returns the percentage.

High-Level Function-Based Questions

21. Write a function to find the nth Fibonacci number using recursion.
22. Create a function to check if a number is an Armstrong number (e.g., $153 = 1^3 + 5^3 + 3^3$).
23. Write a function that returns all factors of a given number.
24. Create a function that takes a number and returns True if it is a perfect number.
    (A perfect number is equal to the sum of its proper divisors, e.g., 28 = 1 + 2 + 4 + 7 + 14)
25. Define a function that finds the least common multiple (LCM) of two numbers.
26. Write a function that accepts a list of numbers and returns the second largest number.
27. Create a function that returns the sum of all prime numbers between 1 and n.
28. Write a function that returns True if the given number is a Harshad number.
    (A number divisible by the sum of its digits)
29. Create a function that calculates and returns the digital root of a number.
    (Repeat summing digits until a single digit is left. Example: 987 → 9+8+7=24 → 2+4=6)
30. Write a function that returns the number of trailing zeros in the factorial of a number.

Answer

1. Add Two Numbers

```
def add(a, b):
    return a + b
print(add(5, 3)) # Output: 8
```

2. Square of a Number

```
def square(x):
    return x ** 2
print(square(4)) # Output: 16
```

3. Even or Odd

```
def check_even_odd(n):
    return "Even" if n % 2 == 0 else "Odd"
print(check_even_odd(7))
# Output: Odd
```

4. Maximum of Three Numbers

```
def max_of_three(a, b, c):
    return max(a, b, c)
print(max_of_three(10, 25, 15))
# Output: 25
```

5. Factorial of a Number

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
print(factorial(5)) # Output: 120
```

6. Multiplication Table

```
def table(n):
    for i in range(1, 11):
        print(f"{n} x {i} = {n*i}")
table(3)
```

7. Area of a Circle

```
def area_circle(r):
    return 3.14 * r * r
print(area_circle(7)) # Output: 153.86
```

8. Sum from 1 to n

```
def sum_n(n):
    return n * (n + 1) // 2
print(sum_n(10)) # Output: 55
```

9. Average of Three Numbers

```
def average(a, b, c):
    return (a + b + c) / 3
print(average(10, 20, 30)) # Output: 20.0
```

10. Cube of a Number

```
def cube(x):
    return x ** 3
print(cube(3))  # Output: 27
```

11. Check if a Number is Prime

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
print(is_prime(7))  # Output: True
```

12. Sum of Digits of a Number

```
def sum_of_digits(n):
    return sum(int(digit) for digit in str(n))
print(sum_of_digits(123))  # Output: 6
```

13. Reverse of a Number

```
def reverse_number(n):
    return int(str(n)[::-1])
print(reverse_number(12345))  # Output: 54321
```

14. Check if a Number is a Palindrome

```
def is_palindrome(n):
    return str(n) == str(n)[::-1]
print(is_palindrome(121))  # Output: True
```

15. Find the Greatest Common Divisor (GCD)

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
print(gcd(56, 98))  # Output: 14
```

16. Count the Number of Digits in a Number

```
def count_digits(n):
    return len(str(n))
print(count_digits(12345))  # Output: 5
```

17. Sum of Even Numbers in a Given Range

```
def sum_even_numbers(start, end):
    return sum(i for i in range(start, end + 1) if i % 2 == 0)
print(sum_even_numbers(1, 10))  # Output: 30
```

18. Print All Prime Numbers Between 1 and n

```
def print_primes(n):
    for i in range(2, n + 1):
        if is_prime(i):
            print(i, end=" ")
print_primes(10)  # Output: 2 3 5 7
```

19. Convert Celsius to Fahrenheit

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32
print(celsius_to_fahrenheit(25))  # Output: 77.0
```

20. Calculate Percentage from Marks of 5 Subjects

```
def calculate_percentage(marks):
    return sum(marks) / len(marks)
print(calculate_percentage([80, 90, 85, 70, 95]))
```

Output: 84.0

21. Nth Fibonacci Number using Recursion

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
print(fibonacci(6))  # Output: 8
```

22. Check Armstrong Number

```
def is_armstrong(n):
    digits = [int(i) for i in str(n)]
    return sum(i ** len(digits) for i in digits) == n
print(is_armstrong(153))  # Output: True
```

23. Return All Factors of a Number

```
def factors(n):
    return [i for i in range(1, n+1) if n % i == 0]
print(factors(12))  # Output: [1, 2, 3, 4, 6, 12]
```

24. Check Perfect Number

```
def is_perfect(n):
    return sum(i for i in range(1, n) if n % i == 0) == n
print(is_perfect(28))  # Output: True
```

25. Find LCM of Two Numbers

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def lcm(a, b):
    return (a * b) // gcd(a, b)
print(lcm(12, 18))  # Output: 36
```

26. Second Largest in a List

```
def second_largest(lst):
    unique = list(set(lst))
    unique.sort()
    return unique[-2] if len(unique) >= 2 else None
print(second_largest([1, 4, 7, 4, 9, 7]))  # Output: 7
```

27. Sum of All Prime Numbers Between 1 and n

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
def sum_primes(n):
    return sum(i for i in range(2, n+1) if is_prime(i))
print(sum_primes(10))  # Output: 17 (2+3+5+7)
```

28. Check Harshad Number

```
def is_harshad(n):
    digit_sum = sum(int(d) for d in str(n))
    return n % digit_sum == 0
print(is_harshad(18))  # Output: True
```

29. Digital Root

```
def digital_root(n):
    while n > 9:
        n = sum(int(d) for d in str(n))
    return n
print(digital_root(987))  # Output: 6
```

30. Trailing Zeros in Factorial

```
def trailing_zeros(n):
    count = 0
    while n >= 5:
        n //= 5
        count += n
    return count
print(trailing_zeros(100))  # Output: 24
```

Problem-1: Create a function isPrime which accepts a number and returns True if it's

Program:

```
prime else False.
def isPrime(num):
    if num<=1:
        return False
    else:
        for i in range(2,int(num**(0.5)+1)):
            if num%i==0:
                return False
        return True
isPrime(1)
```

Problem-2: Create a function count Factors which returns the count of factors. Use this value returned in order to check prime or not.

```
def countFactors(n):
    c=0
    for i in range(1,n+1):
        if n%i==0:
            c+=1
    return c
num = int(input())
print(countFactors(num))
if countFactors(num)==2:
    print("Prime")
else:
    print("Not Prime")
```

Problem-3: create a function which accepts variable length positional args and returns the count of args as well as print the sum of prime numbers

Program:

```
def var_length(*n):
    s=0
    for i in n:
        if i>1:
            for j in range(2,i//2+1):
                if i%j==0:
                    break
            else:
                s+=i
    print("Sum of prime factors:",s)
    return len(n)

var_length(1,2,3,4)
```

Problem-4: Create a function which will accept *args and return the max & min element.

Program:

```
def var_length(*n):
    return max(n),min(n)
```

```
var_length(1,4,5,7,2,3,-1,0)
```

Problem-5: Create a function which will accept *args and return the GCD of it.

Program:

```
def gcd(a,b):
    while b!=0:
        a,b=b,a%b
    return a
gcd(2,4)
def multiple_gcd(*args):
    res=args[0]
    for i in range(1,len(args)):
        res=gcd(res,args[i])
    return res
multiple_gcd(2,4,8,16)
```

Problem-6: Step-by-step iterations of gcd of n numbers.

Program:

```
def gcd(a,b):
    while b!=0:
        a,b=b,a%b
    return a
t=(2,4,8,16)
r=t[0]
for i in range(1,len(t)):
    r=gcd(r,t[i])
    print(f"gcd of {r}&{t[i]} is {r}")
print(r)
```

Problem-7: nth prime number using isPrime function which returns True if prime else False

Program:

```
def isPrime(num):
    if num<=1:
        return False
    else:
        for i in range(2,int(num**(0.5)+1)):
            if num%i==0:
                return False
        return True
num=2#first prime number is 2
ct=0
n=int(input())#nth value taken from user
while ct<n:
    if isPrime(num):
        ct+=1
        if ct==n:
            print(f"{n} prime num is- {num}")
    num+=1
```

lambda function

- A **lambda function** is a small, **anonymous function** in Python.
- It is used when you need a **short function** for a **quick task**.
- It has **no name** (unless you assign it).
- It can have **any number of arguments**, but only **one expression**.
- It is often used with **map()**, **filter()**, **reduce()**, or inside other functions.

Syntax: lambda arguments: expression

Example 1: Add two numbers

```
add = lambda x, y: x + y
print(add(5, 3)) # 8
```

Example 2: Square of a number

```
square = lambda x: x * x
print(square(4))  #16
```

Example 3: Check even number

```
is_even = lambda x: x % 2 == 0
print(is_even(6))  # True
print(is_even(7))  # False
```

Example 4: Lambda with if-else expression

```
check = lambda x: "Even" if x % 2 == 0 else "Odd"
print(check(10))  # Even
print(check(7))   # Odd
```

Example 5: Use lambda inside another function

```
def apply_function(f, value):
   return f(value)
# Using lambda to define the function on the spot
result = apply_function(lambda x: x**3, 5)
print(result)
```

Example 1: Sort a list of tuples by the second value

```
data = [(1, 3), (2, 1), (4, 2)]
# Sort by second item in each tuple
sorted_data = sorted(data, key=lambda x: x[1])
print(sorted_data)
```

map vs reduce vs filter

map() function

- map() function is used to apply a function to every item in an iterable (like a list or tuple) and returns a map object, which can be converted to a list.
- **Syntax:** map(function, iterable)
  1. **function:** A function to apply.
  2. **iterable:** A sequence like list, tuple, etc.
- **Exanple-1:**

```
l=list(map(int,input().split(",")))
    def fun(l):
       return l+2
    l=[1,2,3,4]
    list(map(fun,l))
            num=[1,2,3]
            for i in num:
               print(i)
```

Example-2

```
# Function to square a number
def square(x):
   return x * x
numbers = [1, 2, 3, 4]
squared_numbers = list(map(square, numbers))
print(squared_numbers)
```

Example-3

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

Example-4:

```
mylist1=[10,20,30,40,50,60]
mylist2=[1,2,3,4,5,6]
a=list(map(lambda x,y: x+y, mylist1, mylist2))
print(a)
```

or

```
def add(x,y):
   return x+y
b=list(map(add, mylist1, mylist2))
print("new list=",b)
```

Output:

```
[11, 22, 33, 44, 55, 66]
new list= [11, 22, 33, 44, 55, 66]
```

Example-5

```
list1 = ["Hello", "Good", "Nice"]
list2 = ["World", "Morning", "Day"]
result = list(map(lambda x, y: x + " " + y, list1, list2))
print(result)
# or
def combine(x, y):
   return x + " " + y
result2 = list(map(combine, list1, list2))
print("Combined list =", result2)
```

Output

```
['Hello World', 'Good Morning', 'Nice Day']
Combined list = ['Hello World', 'Good Morning', 'Nice Day']
```

Example-6

```
floats1 = [1.5, 2.5, 3.5]
floats2 = [2.0, 4.0, 6.0]
product = list(map(lambda x, y: x * y, floats1,
floats2))
print(product)
# or
def multiply(x, y):
   return x * y
product2 = list(map(multiply, floats1, floats2))
print("Product list =", product2)
Output:
[3.0, 10.0, 21.0]
Product list = [3.0, 10.0, 21.0]
```

filter() Function

- Applies the **function** to each item in the **iterable**.
- Returns only the items for which the function returns **True**.
- The result is a **filter object**, so we usually convert it to a list using list().

Syntax: filter(function, iterable)

Example 1: Filter even numbers

```
def is_even(n):
    return n % 2 == 0
l = [2, 3, 4, 5, 6]
even_numbers = list(filter(is_even, l))
print(even_numbers)
```

Output: [2, 4, 6]

Example 2: Filter strings that start with 'A'

```
def starts_with_a(s):
    return s.startswith('A')
names = ["Ankit", "Ravi", "Amit", "Sonal", "Aryan"]
a_names = list(filter(starts_with_a, names))
print(a_names)
```

Output: ['Ankit', 'Amit', 'Aryan'])

Example 3: Filter positive numbers

```
def is_positive(x):
    return x > 0

numbers = [-3, -1, 0, 2, 5, -7]
positive_numbers = list(filter(is_positive, numbers))
print(positive_numbers)
```

Output: [2, 5]

Example 4: Filter vowels from a list of characters

```
def is_vowel(ch):
    return ch.lower() in 'aeiou'

chars = ['a', 'b', 'c', 'e', 'i', 'o', 'u', 'x', 'y']
vowels = list(filter(is_vowel, chars))
print(vowels)
```

Output: ['a', 'e', 'i', 'o', 'u']

Example 5: Filter words with length greater than 4

```
def long_word(word):
    return len(word) > 4
words = ["cat", "elephant", "dog", "tiger", "lion", "monkey"]
long_words = list(filter(long_word, words))
print(long_words)
```

Output: ['elephant', 'tiger', 'monkey']

Example 6: Filter numbers that are divisible by both 3 and 5

```
def divisible_by_3_and_5(n):
    return n % 3 == 0 and n % 5 == 0
nums = [10, 15, 20, 30, 45, 50, 60]
result = list(filter(divisible_by_3_and_5, nums))
print(result)
```

Output: [15, 30, 45, 60]

Feature	`map()`	`filter()`
Purpose	Transform all items	Select specific items
Returns	All items after applying function	Only items where condition is True
Function Type	Returns new values	Returns original values

reduce()

- reduce() is used to **apply a function to elements of a sequence cumulatively**, so as to **reduce the iterable to a single value**.
- It is available in the functools module.

Syntax:

```
from functools import reduce
reduce(function, iterable)
```

Example 1: Find the maximum number

```
from functools import reduce
def max_fun(i, j):
    return i if i > j else j
l = [1, 4, 5, 2, 3]
result = reduce(max_fun, l)
print("Maximum number =", result)
```

Output: Maximum number = 5

Example 2: Find the sum of all numbers

```
from functools import reduce
def add(x, y):
    return x + y
numbers = [10, 20, 30, 40]
total = reduce(add, numbers)
print("Sum =", total)
```

Output: Sum = 100

Example 3: Multiply all numbers in a list

```
from functools import reduce
def multiply(x, y):
    return x * y
nums = [2, 3, 4]
product = reduce(multiply, nums)
print("Product =", product)
```

Output: Product = 24

Example 4: Find the longest word in a list

```
from functools import reduce
def longest_word(x, y):
    return x if len(x) > len(y) else y
words = ["apple", "banana", "grape", "watermelon", "kiwi"]
longest = reduce(longest_word, words)
print("Longest word =", longest)
```

Output: Longest word = watermelon

Difference between map(), filter() and reduce()

Function	Purpose	Returns	Works On Each Element?	Output Type	Needs `functools`?
`map()`	Transforms each item	All transformed items	✅ Yes	List (usually)	❌ No
`filter()`	Selects items that match a condition	Only items where function is `True`	✅ Yes	List (usually)	❌ No
`reduce()`	Combines all items into one value	A single final result	✅ Yes, cumulatively	Single value	✅ Yes

Note:

- Use map() when you want to **change all values**.
- Use filter() when you want to **select some values**.
- Use reduce() when you want to **combine all values into one**.

Problem: find the sum of given list using reduce method:

Program:

```
from functools import reduce
def fun(a,b):
    return a+b
l=[1,4,5,2,3]
reduce(fun,l)
```

Example:

```
import random
#randint(lower,upper)-- it generates a random number b/w given range
a=random.randint(100000,999999)#it generates random 6 digit otp
print(a)
```

Example:

```
import random
#choice(iterable)--it generates a random value from given iterable
l=[10,20,30,40,50]
a=random.choice(l)
print(a)
```

Problem: **Create a dice rolling game in which user has to guess the correct dice roll**

Program:

```
from random import *
l=[1,2,3,4,5,6]
play="yes"
while play=="yes":
   guess=int(input("Enter your guess for dice roll:"))
   if 1<=guess<=6:
      comp=choice(l)
      print("Computer's dice roll:",comp)
      if guess==comp:
         print("You win!!!")
      else:
         print("You lost..")
   else:
      print("Invalid choice")
   play=input("Do you wanna play again(yes/no)? ")
```

Problem: Find the hypotenuse of a right triangle using pythagoras theorem provided

Program:

```
#inputs of base and perpendicular using math module
#Pythagoras Theorem- h^h=b^b+p^p
    from math import *
    b,p=map(int,input().split())
    h=sqrt(pow(b,2)+pow(p,2))
    print("Hypotenuse is:",h)
```

Problem: creating a mini calculator using user-defined module custom

Program:

```
from custom import *
a=int(input('enter num1: '))
b=int(input('enter num2: '))
print("Choose any one operation:\n1 for +\n2 for -\n3 for *\n4 for /")
ch=int(input())
if ch==1:
   print("Addition:",add(a,b))
elif ch==2:
   print("Subtraction:",abs(sub(a,b)))
elif ch==3:
   print("Product:",mul(a,b))
elif ch==4:
   print("Division:",div(a,b))
else:
   print("Invalid input!!!")
```

Decorator

- It is a special type of function which accepts another function as an argument and returns the enhanced functionality of it without modifying its original source code.

How it works:

- A decorator takes a function as an argument.
- It wraps that function in another function (usually called wrapper or inner).
- It returns the new function, which adds extra functionality.

❖ Why use decorators?

- To reuse code and avoid rewriting logic.
- To follow the DRY (Don't Repeat Yourself) principle.
- To add features like:
  - ➢ Logging
  - ➢ Access control
  - ➢ Input validation
  - ➢ Execution timing

Key Points:

- Decorators are used to **wrap another function** to add more behavior.
- Commonly used in frameworks like **Flask**, **Django**, etc.
- Can be applied to **functions or classes**.

Syntax:

```
@decorator_name
def original_function():
    ...
```

This is the same as:

```
original_function = decorator_name(original_function)
```

Example-1:

```
def dec(fun):#It accepts the function as argument
    def mfun(num):#It enhances the functionality of the function
        return num*fun(num)
    return mfun
@dec
def sqr(n):#original function
    return n*n
sqr(10)
```

Example-2:

```
def greet_decorator(func):
    def new_func():
        print("Hello!")
        func()
    return new_func

@greet_decorator
def say_name():
    print("My name is Tanmay.")
say_name()
```

Output:

```
Hello!
My name is Tanmay.
```

Example 3: Double the result

```
def double_result(func):
    def wrapper(x):
        return 2 * func(x)
    return wrapper

@double_result
def increment(n):
    return n + 1

print(increment(5))
```

Output: 12

Example-4: Create a function which prints the text:"hello world".Create a decorator which #will print "python" as well as "hello world"

Program:

```
def dec_text(fun):
    def mfun():
        return "python\n"+fun()

    return mfun
@dec_text
def text():
    return "hello world"
print(text())
```

Example-5

```
def check_positive(func):
    def wrapper(n):
        if n > 0:
            return func(n)
        else:
            return "Only positive numbers allowed"
    return wrapper

@check_positive
def square(n):
    return n * n

print(square(4))
print(square(-3))
```

Output: 16
Only positive numbers allowed

Generator

- A **generator** is a special type of function that:
- **Returns a sequence of values** (like a list),
- But **does not store them in memory**,
- Uses **yield** instead of return,
- Produces **one value at a time** (on demand).

❖ **Why use Generators?**

- **Saves memory** — good for large data.
- **Faster for iteration**.
- Useful when you don't need all values at once.

Syntax:

```
def my_generator():
    yield value1
    yield value2
```

Example-1:

```
def is_even(n):
    for i in range(1,n+1):
        if i%2==0:
            yield i
a=is_even(10)
for i in a:
    print(i)
```

Example 2: A generator to yield numbers

```
def numbers():
    yield 1
    yield 2
    yield 3
gen = numbers()
for num in gen:
    print(num)
```

Output:

1
2
3

Example 3: Using next() with a generator

```
def greet():
    yield "Hello"
    yield "Hi"
    yield "Welcome"

g = greet()
print(next(g))  # Hello
print(next(g))  # Hi
print(next(g))  # Welcome
```

Example 4: Even numbers using generator

```
def even_numbers(n):
    for i in range(n + 1):
        if i % 2 == 0:
            yield i

for num in even_numbers(10):
    print(num,end= " ")
```

Output: 0 2 4 6 8 10

Difference between `return` and `yield`:

Feature	`return`	`yield`
Stops	Ends the function	Pauses and resumes the function
Memory use	Stores all data	Generates one item at a time
Use case	One-time result	Multiple values, one-by-one use

Example-5 Odd Numbers Generator

```
def odd_numbers(n):
    for i in range(n + 1):
        if i % 2 != 0:
            yield i
for num in odd_numbers(10):
    print(num, end= " ")
```

Output: 1,3,5,7,9

Example 2: Multiples of 5

```
def multiples_of_five(n):
    for i in range(1, n + 1):
        if i % 5 == 0:
            yield i
for num in multiples_of_five(30):
    print(num, end= " ")
```

Output: 5 10 15 20 25 30

Problem: Create a generator which return the sequence of values in reverse order

Program:

```
def sequence(n):
    for i in range(n,0,-1):
        yield i
a=sequence(5)
for i in a:
    print(i)
#Method-2
a=[i for i in range(5,0,-1)]
for i in a:
    print(i)
```

Recursion

- Recursion is a programming technique where a function calls itself to solve a problem.
- A recursive function keeps calling itself until it reaches a base condition (stopping point).
- It is commonly used for solving problems that can be broken into smaller sub-problems.

Structure of a Recursive Function

Example:

```
def function_name(parameters):
    if base_condition:
        return result
    else:
        return function_name(smaller_input)
```

Why use Recursion?

- Simpler code for problems like:
  - Factorial
  - Fibonacci
  - Sum of numbers
  - Tree or graph traversal
  - Reverse string

Example 1- implementing factorial of a number

```
def fact(n):
    if n==1:#base step
        return n
    else:
        return n*fact(n-1)
fact(5)
```

Example 2- implementing gcd of a number

```
def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
gcd(12,16)
gcd(2,4)
```

Example-3- implementing fibonacci series(find nth term)

```
#0,1,1,2,3....
def fib(n):
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return fib(n-1)+fib(n-2)
fib(5)
```

Example 1: Factorial of a number

```
def factorial(n):
    if n == 0 or n == 1:
        return 1  # base case
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Example 2: Sum of first N natural numbers

```
def sum_n(n):
    if n == 1:
        return 1
    else:
        return n + sum_n(n - 1)

print(sum_n(5))  # Output: 15
```

Example 3: Fibonacci Series (Nth term)

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(6))  # Output: 8
```

Example 4: Countdown using Recursion

```
def countdown(n):
    if n == 0:
        print("Go!")
    else:
        print(n)
        countdown(n – 1,end= “ “)
```

Output: 5 4 3 2 1 Go !

Example 5: Reverse a String

```
python
CopyEdit
def reverse_string(s):
    if len(s) == 0:
        return ""
    else:
        return s[-1] + reverse_string(s[:-1])

print(reverse_string("HELLO"))  # Output: OLLEH
```

MODULES

- A group of function, variables and classes saved to a file, which is nothing but module.
- Every python file(.py) acts as a module.
- **Example-1** **Skillscal.py**

```
s= “rid Bharat”
def add(a,b):
   print("The sum:", a+b)
def sub(x,y):
   print("The Sub:",x-y)
def product(a,b):
   print("The Product:",a*b)
```

Demo.py

```
import skillscal
print(skillscal.s)
skillscal.add(10,20)
skillscal.sub(60,30)
skillscal.product(10,20)
```

- Skillcal module contains one variable and 3 functions.
- If we want to use members of module in our program then we should import that module.
  Syntax: Import modulename
- **We can access members by using module name.**

  Modulename.variable
  Modulename.function()

```
project_folder/
│
├── skillscal.py      # This is module file containing  variable and functions
└── demo.py           # This is the script that imports and uses skillscal
```

Output:

```
rid Bharat
The sum: 30
The Sub: 30
The Product: 200
```

Note: whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.

Example-2 Step 1: Create the module with variable method and class file (e.g., m1.py)

```
# Variables
greeting = "Hello"
number = 42

# Functions
def add(a, b):
   return a + b

def greet(name):
   return f"{greeting}, {name}!"

# Class
class Person:
   def __init__(self, name):
      self.name = name

   def introduce(self):
      return f"My name is {self.name}."
```

Step 2: Create the main file (e.g., main.py) to use the module

```
import m1
# Access variables
print(mymodule.greeting)
print(mymodule.number)

# Use functions
print(mymodule.add(5, 10))
print(mymodule.greet("Tanmay"))

# Use class
person = mymodule.Person("Tanmay")
print(person.introduce())
```

Output:

```
Hello
42
15
Hello, Sangam!
My name is Sangam.
```

How to rename a module at the time of import

- You can **rename a module using the as keyword** during import to give it a shorter or more convenient alias.

Syntax: import module_name as new_module_ name

- **Example:** import skillscal as m
- Here skillscal is original module name and m is alias name.
- We can access members by using alias name m

Example:

Demo.py

```
import skillscal as m
print(m.s)
m.add(10,20)
m.sub(60,30)
m.product(10,20)
```

Output:

```
TWKSAA SKILLS CENTER
The sum: 30
The Sub: 30
The Product: 200
```

❖ Create Three Modules

```
my_project/
│
├── main.py
├── mathmodule.py
├── stringmodule.py
└── convertmodule.py
```

mathmodule.py

```
def add(a, b):
    return a + b
```

stringmodule.py

```
def greet(name):
    return f"Hello, {name}!"
```

convertmodule.py

```
def km_to_miles(km):
    return km * 0.621371
```

❖ Create main.py and Rename Modules on Import

main.py:

```
import mathmodule as mth
import stringmodule as strmod
import convertmodule as conv

# Using mathmodule (renamed as mth)
print("Addition:", mth.add(10, 20))

# Using stringmodule (renamed as strmod)
print("Greeting:", strmod.greet("Sangam kumar"))

# Using convertmodule (renamed as conv)
print("10 km in miles:", conv.km_to_miles(10))
```

Output:

```
Addition: 30
Greeting: Hello, Sangam Kumar!
10 km in miles: 6.21371
```

from … import

- The from … import statement in Python allows you to **import specific members (like variables, functions, or classes)** from a module, rather than importing the whole module.
- we can import particular members of module by using from …import.
- **Advantage:**
    - **Selective Import** – You import **only what you need**.
    - **No need to use module name** – You can **use the members directly** without prefixing them with the module name.

Syntax: from module_name import member_name

Note: You can also import **multiple members** like this:

Syntax: from module_name import member1, member2

Example-1: Importing a Function

mathmodule.py

```
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
```

main.py

```
from mathmodule import add
print(add(5, 3))   #8
```

Note: No need to write mathmodule.add()

Example-2: Importing Multiple Members

stringmodule.py

```
def greet(name):
    return f"Hello, {name}!"

def farewell(name):
    return f"Goodbye, {name}!"
```

main.py

```
from stringmodule import greet, farewell
print(greet("Sangam Kumar"))
print(farewell("Sangam Kumar"))
```

Example 3: Importing a Class

personmodule.py

```
class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        return f"My name is {self.name}."
```

main.py

```
from personmodule import Person
p = Person("Sangam Kumar")
print(p.introduce())
```

Output:

```
8
Hello, Sangam Kumar!
Goodbye, Sangam Kumar!
My name is Sangam Kumar.
```

Example-3:

1. Importing **specific members**.
2. Importing **all members using ***.

skillscal.py

```
s = "TWKSAA SKILLS CENTER"
def add(a, b):
   print("The sum:", a + b)
def sub(a, b):
   print("The Sub:", a - b)
def product(a, b):
   print("The Product:", a * b)
```

Example 1: Import specific members

Demo.py

```
from skillscal import s, add
print(s)
add(25, 50)
sub(30, 10)
# ❌ NameError: sub is not defined
```

Output:

```
TWKSAA SKILLS CENTER
The sum: 75
NameError: name 'sub' is not defined
```

Example 2: Import all members

Demo.py

```
from skillscal import *
print(s)
add(3, 6)
sub(20, 10)
product(3, 6)
```

Output:

```
TWKSAA SKILLS CENTER
The sum: 9
The Sub: 10
The Product: 18
```

❖ Various Possibilities of import:

	Syntax	Use Case
1.	**import modulename**	Import whole module
2.	**import module1, module2**	Import multiple modules
3.	**import modulename as m**	Import with alias
4.	**import module1 as m1, module2 as m2**	Multiple imports with aliases
5.	**from modulename import member**	Import specific function/class/var
6.	**from modulename import member1, ...**	Import multiple specific members
7.	**from modulename import member as x**	Import specific member with alias
8.	**from modulename import ***	Import everything from a module

Example for all

Folder Structure

```
my_project/
|
├── main.py
├── mathops.py
├── stringops.py
```

mathops.py

```
x = 100
def add(a, b):
    return a + b
def sub(a, b):
    return a - b
def mul(a, b):
    return a * b
```

stringops.py

```
msg = "Hello from StringOps!"

def greet(name):
    return f"Hello, {name}!"

def farewell(name):
    return f"Goodbye, {name}!"
```

main.py

```
# 1. import modulename
import mathops
print("1:", mathops.add(5, 3))  # Output: 8
# 2. import module1, module2
import mathops, stringops
print("2:", stringops.greet("Tanmay"))  # Output: Hello, Tanmay!

# 3. import modulename as m
import mathops as mo
print("3:", mo.sub(10, 4))  # Output: 6

# 4. import module1 as m1, module2 as m2
import mathops as m1, stringops as m2
print("4:", m2.farewell("Tanmay"))  # Output: Goodbye, Tanmay!

# 5. from modulename import member
from mathops import mul
print("5:", mul(2, 5))  # Output: 10
# 6. from modulename import member1, ...
from stringops import greet, farewell
print("6:", greet("Student"), farewell("Student"))
# Output: Hello, Student! Goodbye, Student!
# 7. from modulename import member as x
from mathops import add as addition
print("7:", addition(20, 30))  # Output: 50

# 8. from modulename import *
from stringops import *
print("8:", msg)  # Output: Hello from StringOps!
```

Reloading a module

- By default, a module is loaded only once during a Python program's execution, no matter how many times you import it.
- This means if you import the same module multiple times, Python uses the already loaded module from memory — it does not reload or re-execute the module code again..

Example: module1.py

```
s='twksaa skills center'
s1='twksaa rid center'
print(s)
print(s1)
```

demo.py

```
import module1
import module1
import module1
import module1
import module1
print("this is led skills center")
```

Output:

```
twksaa skills center
twksaa rid center
this is led skills center
```

Note: Notice that the messages from module1.py are printed **only once** even though it is imported multiple times.

Problem with this behavior:

- If you **update the module file (module1.py) after it has already been imported**, those changes will **not be reflected** in the running program because the module is not reloaded automatically.
- In the above demo module will be loaded only even though we are importing multiple times.
- The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.

Solution: Reload the module explicitly

- You can reload a module using the **reload()** function from importlib module (in Python 3.x).

Why Reload?

- By default, a module is loaded **only once**, even if imported multiple times.
- If the module is **updated after being loaded**, those changes **won't reflect** automatically.
- To get the latest version, we use **explicit reloading** with reload().

Example: module1.py

```
s = 'twksaa skills center'
s1 = 'twksaa rid center'
print(s)
print(s1)
```

demo.py

```
import module1          # Loaded automatically (1st time)
from importlib import reload

reload(module1)         # Reloaded manually (2nd time)
reload(module1)         # Reloaded manually (3rd time)
reload(module1)         # Reloaded manually (4th time)
print("this is test module1")
```

Output

```
twksaa skills center
twksaa rid center
twksaa skills center
twksaa rid center
twksaa skills center
twksaa rid center
twksaa skills center
twksaa rid center
this is test module1
```

➔ Finding Members of module by using dir() function:

- Python provides inbuilt function dir() to list out all members of current module or a specified module.
- **dir()** ➔ to list out all members of current module
- **dir(moduleName)** ➔to list out all members of specified module.

Example:

```
x = 20
y = 30
def f1():
   print("skills")
   print(dir())
 #to print all members of current module
```

Output:

```
skills

['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'f1', 'x', 'y']
```

Example: to display members of particular module

Raj.py

```
x=393
def add(a,b):
   print("The sum of a and b=", a+b)
def product(a,b):
   print("The Product=", a*b)
```

Output:

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'add', 'product', 'x']
```

Note: for every module at the time of execution python interpreter will add some special properties automatically for internal use.

Example:

```
'__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__'
```

- based on our requirement we can access these properties also in our program

➔ The Special variable ___name___:

- For every python program, a special variable ___name___ will be added internally.
- This variable store information regarding whether the program is executed as an individual program or as a module.
- If the program executed as an individual program, then the value of this variable is __main__
- If the program executed as a module from some other program, then the value of this variable is the name of module where it is defined.
- Hence by using this __name__ variable we can identify whether the program executed directly or as a module.

Example:

```
def f1():
    if __name__=='__main__':
        print("The code executed as a program")
    else:
        print("The code executed as a module from some other program")
f1()
```

Output:

```
The code executed as a module from some other program
The code executed as a module from some other program
```

➔ Working with math module:

- Python provided inbuilt module math
- This module defines several function which can be used for mathematical operations.
- The main important functions are

1) sqrt(x)
2) ceil(x)
3) floor(x)
4) fabs(x)
5) log(x)
6) sin(x)
7) tan(x)
   ……..
   …….

Example:

```
from math import*
print(sqrt(9))
print(ceil(12.3))
print(floor(12.3))
print(fabs(-9.3))
print(fabs(9.3))
```

Output:

```
3.0
13
12
9.3
9.3
```

Note: we can find help for any module by using help() function

Example:

```
import math
Help(math)
```

➔ working with random module:

- this module defines several functions to generate random numbers.
- We can use these functions while developing games, in cryptography and to generate random numbers on fly for authentication.

1) random() function:

- this function always generates some float value between 0 and 1 (not inclusive)
- 0<a<1

Example:

```
from random import*
for i in range(6):
    print(random())
```

Output:

0.0602312327616481
0.5840871889511808
0.8177632631088152
0.6285178285758605
0.3348565819577446
0.28528602489553256

2) randint () function:

- To generate random integer between two given numbers (inclusive)

Example:

```
from random import*
for i in range(6):
    print(randint(1,50)) # generate random int value between 1 and 50 (inclusive)
```

Output:

5
38
20
10
9
44

Note: your output can come different also because it is generating random values.

3) uniform() function:

- if returns random float values between 2 given numbers (not inclusive)

Example:

```
from random import*
for i in range(6):
    print(uniform(1,10))
```

output:

8.932640802901647
7.922143314827758
5.53234767706302
1.4958613157116858
2.0526314565757464
2.2653597842057027

Note: your output can come different also because it is generating random values.

- random() → in between 0 and 1 (not inclusive)
- randint(x, y) → in between x and y (inclusive)
- uniform(x, y) → in between x and y (not inclusive)

4) randrange([start], stop, [step]) :

- Returns a random number from range
- Start <=x<stop
- Start argument is optional and default value is 0
- Step argument is optional and default values is 1

Example:

- randrange(10) → generates a number from 0 to 9
- randrange(1,11) → generates a number from 1 to 10
- randrange(1,11,2) → generates a number from 1,3,5,7,9

Example-1:

```
from random import*
for i in range(6):
   print(randrange(10))
```

Output:

```
7
5
5
0
4
9
```

Example-2:

```
from random import*
for i in range(6):
 print(randrange(1,11))
```

Output:

```
3
10
10
4
4
3
```

Example-3:

```
from random import*
 for i in range(6):
print(randrange(1,11,2))
```

Output:

```
7
3
3
3
7
9
```

Note: your output can come different also because it is generating random values.

5) choice() function:

- it won't return random number.
- It will return a random object from the given list or tuple.

Example:

```
from random import*
l=['ram','shyam','mohan', 'sohan', 'ravi', 'ramu']
for i in range(6):
   print(choice(l,end=" ))
```

Output:

```
Shyam ravi ram shyam mohan ramu
```

How to import the module that are create folder inside folder

Folder Structure Example

```
my_project/
├── main.py
└── f1/
   └── f2/
      ├── module1.py
      └── module2.py
```

❖ Create Folders and Modules

- **Using terminal:** mkdir -p my_project/f1/f2
- **Using Python:**

```
import os
os.makedirs('my_project/f1/f2', exist_ok=True)
```

Create modules: module1.py (inside f1/f2)

```
def greet():
   print("Hello from Module 1!")
```

module2.py (inside f1/f2)

```
def welcome():
   print("Welcome from Module 2!"
```

Create main.py in root folder

```
from f1.f2 import module1, module2

module1.greet()
module2.welcome()
```

Output:

```
Hello from Module 1!
Welcome from Module 2!
```

#How to know only how many function are present in any module

What This Script Does:

- Imports everything from a module.
- Uses dir() to list all names in the current namespace.
- Uses globals().get() to fetch objects by name.
- Uses callable() to check if they are functions.
- Collects and prints only the **functions**.

Example:

```
my_project/
├── f1/
│   └── f2/
│       └── demo.py
└── main.py
```

demo.py

```
def add(a, b):
    return a + b
def sub(a, b):
    return a - b
def mul(a, b):
    return a * b
x = 10  # Variable (not a function)
```

main.py

```
from f1.f2.f3demo import *  # Import all members from module
items = dir()              # Get all names in the current scope
function_list = []         # Empty list to store function names
for name in items:
    if callable(globals().get(name)):
        function_list.append(name)

print("Total functions:", len(function_list))
print("Function names:", function_list)
```

Output:

```
Total functions: 3
Function names: ['add', 'sub', 'mul']
```

Theoretical questions and answers

1. **Q:** What is a module in Python?
   A: A module is a file containing Python code like functions, classes, or variables.

2. **Q:** How do you import an entire module?
   A: Using import module_name.

3. **Q:** How do you import specific members from a module?
   A: Using from module_name import member.

4. **Q:** What is the advantage of from module import member?
   A: It allows direct access to the member without the module prefix.

5. **Q:** How can you give an alias to a module while importing?
   A: Using import module_name as alias.
6. **Q:** How do you import multiple modules in a single line?
   A: Using import module1, module2.
7. **Q:** What does from module import * do?
   A: It imports all public members of the module into the current namespace.
8. **Q:** Which function lists all members of a module?
   A: dir(module_name)
9. **Q:** How do you reload a module after it has been modified?
   A: Using reload(module_name) from importlib.
10. **Q:** What keyword is used to prevent importing private members using *?
    A: A leading underscore _ before the name.

Commonly Used Python Modules

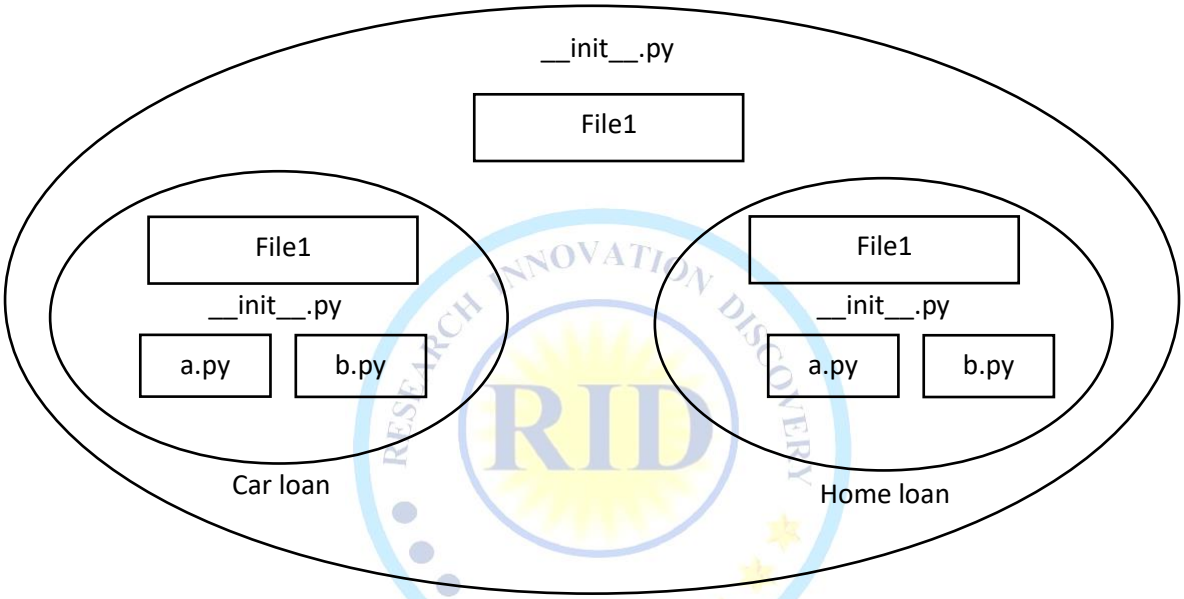
Module Name	Purpose
`math`	Mathematical functions like sqrt, sin, cos, factorial, etc.
`random`	Generate random numbers, shuffle, randint, etc.
`datetime`	Work with dates and times
`time`	Time-related functions like sleep, time, etc.
`os`	Interact with the operating system (files, directories)
`sys`	Access system-specific parameters and functions
`json`	Parse and write JSON data
`re`	Regular expressions for pattern matching
`statistics`	Perform statistical operations like mean, median
`csv`	Read and write CSV files
`math`	Advanced math operations (sqrt, pi, log, etc.)
`collections`	Provides specialized container datatypes like Counter, deque
`itertools`	Functions for efficient looping
`functools`	Higher-order functions like reduce, lru_cache
`typing`	Support for type hints
`pathlib`	Object-oriented filesystem paths
`shutil`	File operations like copy, move, delete
`urllib`	Work with URLs (fetching data from the web)
`tkinter`	Create GUI applications
`logging`	Log events in your application
`threading`	Run code concurrently in threads

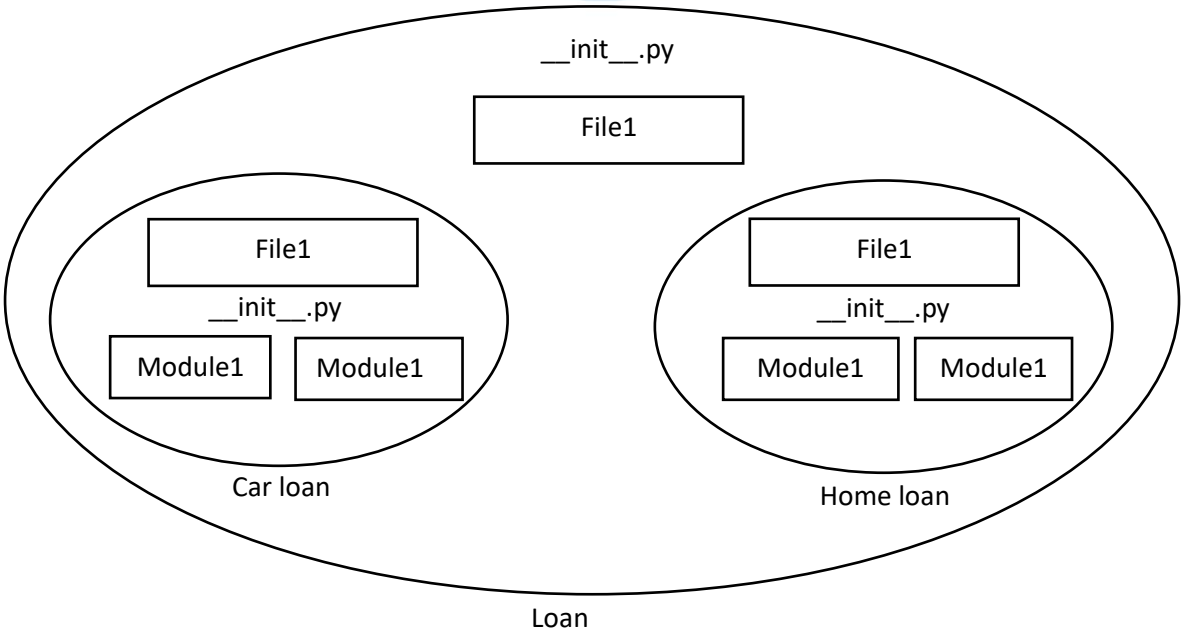
PACKAGE

- A package is a way to group related modules (Python .py files) into a single folder (or directory). It helps organize and reuse code efficiently.

❖ **Key Points:**

- A **package** is just a **folder** that contains Python files (modules).
- To be recognized as a **package**, the folder **must contain an __init__.py file**.
- The __init__.py file can be **empty**, or it can execute initialization code for the package.
- A package can also have **sub-packages** (folders with their own __init__.py files).





Example-1: Folder Structure

```
my_project/
├── main.py
└── mypackage/
    ├── __init__.py
    ├── module1.py
    └── module2.py
```

File Contents:

mypackage/module1.py

```
def add(a, b):
    return a + b
```

mypackage/module2.py

```
def multiply(a, b):
    return a * b
```

mypackage/__init__.py

```
# Optional: Can be left empty or used for setup
print("mypackage initialized")
```

main.py

```
from mypackage import module1, module2
print("Sum:", module1.add(10, 5))
print("Product:", module2.multiply(3, 4))
```

Output:

```
mypackage initialized
Sum: 15
Product: 12
```

Example-2: Real-World Problem: BMI Calculator

- BMI (Body Mass Index) is used to estimate body fat based on height and weight.

Folder Structure:

```
bmi_project/
├── bmi_user.py
├── bmi_doctor.py
└── bmi_calculator/
    ├── __init__.py
    ├── calculate.py
    └── analysis.py
```

1. bmi_calculator/calculate.py

```
def calculate_bmi(weight_kg, height_m):
    """Returns BMI as weight (kg) / (height (m))^2"""
    bmi = weight_kg / (height_m ** 2)
    return round(bmi, 2)
```

2. bmi_calculator/analysis.py

```
def interpret_bmi(bmi):
    """Returns the BMI category based on the value."""
    if bmi < 18.5:
```

```
        return "Underweight"
    elif 18.5 <= bmi < 25:
        return "Normal weight"
    elif 25 <= bmi < 30:
        return "Overweight"
    else:
        return "Obese"
```

3. bmi_calculator/__init__.py

```
# Optional initialization
print("BMI Calculator Package Loaded")
```

4. bmi_user.py — used by a fitness app user

```
from bmi_calculator import calculate, analysis
weight = 68  # in kg
height = 1.75  # in meters
bmi = calculate.calculate_bmi(weight, height)
status = analysis.interpret_bmi(bmi)
print("Your BMI:", bmi)
print("Health Status:", status)
```

5. bmi_doctor.py — used by a doctor to analyze multiple patients

```
from bmi_calculator import calculate, analysis

patients = [
    ("Sangam", 72, 1.80),
    ("Sushil", 50, 1.55),
    ("Sujeet", 95, 1.70)
]

for name, weight, height in patients:
    bmi = calculate.calculate_bmi(weight, height)
    status = analysis.interpret_bmi(bmi)
    print(f"{name} - BMI: {bmi}, Status: {status}")
```

Sample Output:

When running bmi_user.py:

```
BMI Calculator Package Loaded
Your BMI: 22.2
Health Status: Normal weight
```

When running bmi_doctor.py:

```
BMI Calculator Package Loaded
Sangam - BMI: 22.22, Status: Normal weight
Sujeet - BMI: 20.81, Status: Normal weight
Bob - BMI: 32.87, Status: Obese
```

Example-3: Real-World Problem: Temperature Conversion

- We need to convert temperatures between Celsius, Fahrenheit, and Kelvin in different contexts.

Folder Structure:

```
temp_project/
├── weather_app.py
├── science_lab.py
└── temp_converter/
    ├── __init__.py
    ├── celsius.py
    └── fahrenheit.py
```

1. temp_converter/celsius.py

```
def to_fahrenheit(c):
  return round((c * 9/5) + 32, 2)
def to_kelvin(c):
  return round(c + 273.15, 2)
```

2. temp_converter/fahrenheit.py

```
def to_celsius(f):
  return round((f - 32) * 5/9, 2)
def to_kelvin(f):
  return round(((f - 32) * 5/9) + 273.15, 2)
```

3. temp_converter/__init__.py

```
# Optional: acts as the initializer for the package
print("Temperature Converter Package Ready")
```

4. weather_app.py — Used by a weather app

```
from temp_converter import celsius
temp_c = 25
print("Temperature in Fahrenheit:", celsius.to_fahrenheit(temp_c))
print("Temperature in Kelvin:", celsius.to_kelvin(temp_c))
```

5. science_lab.py — Used by a science lab system

```
from temp_converter import fahrenheit
temp_f = 98.6
print("Temperature in Celsius:", fahrenheit.to_celsius(temp_f))
print("Temperature in Kelvin:", fahrenheit.to_kelvin(temp_f))
```

Sample Output:

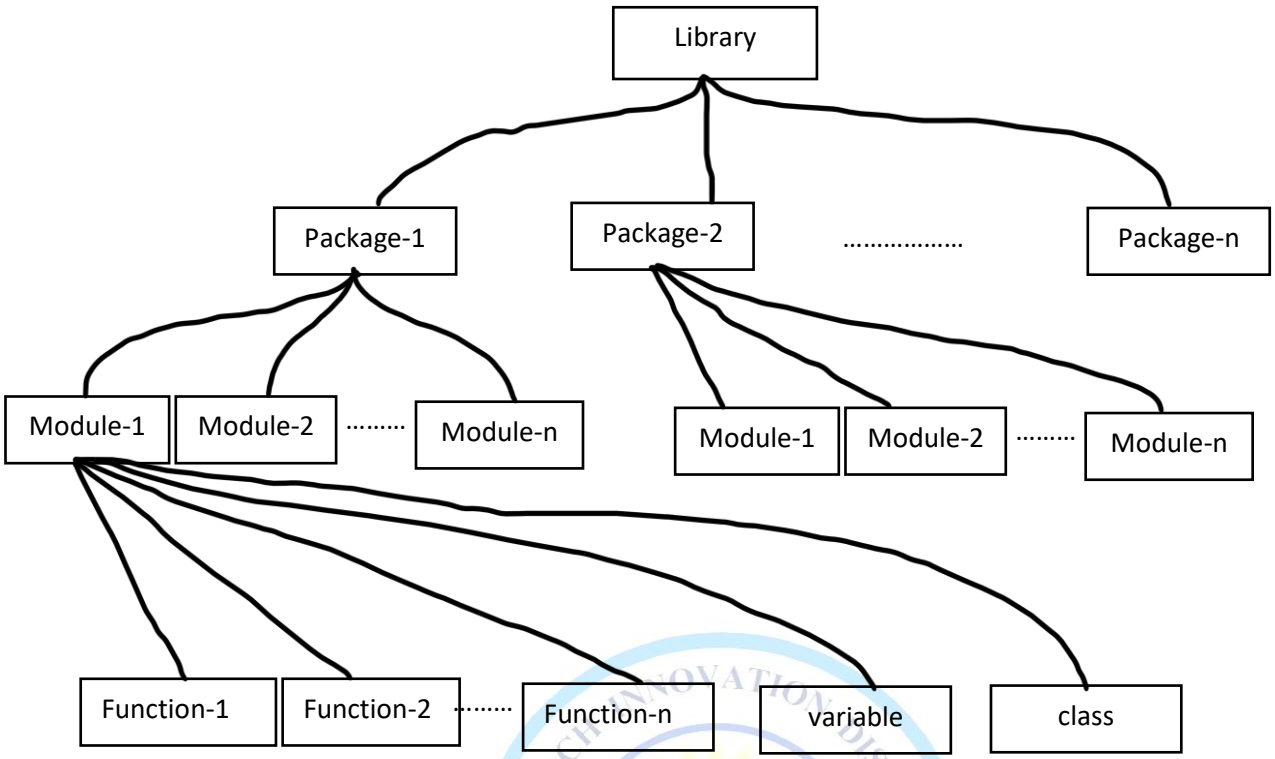
Running weather_app.py:

```
Temperature Converter Package Ready
Temperature in Fahrenheit: 77.0
Temperature in Kelvin: 298.15
```

Running science_lab.py:

```
Temperature Converter Package Ready
Temperature in Celsius: 37.0
Temperature in Kelvin: 310.15
```

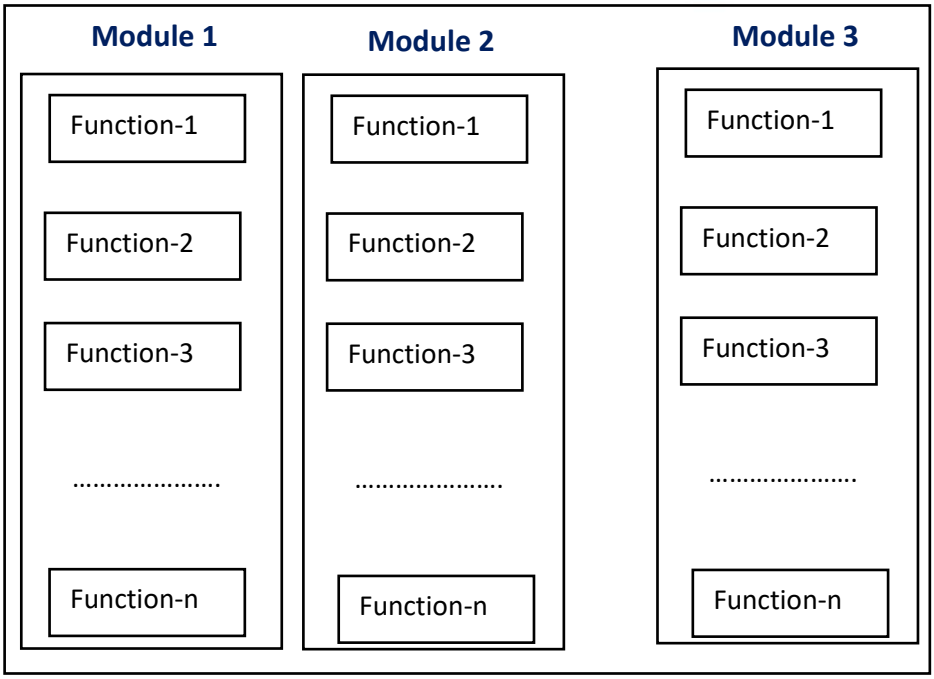
➔ **Library, packages, modules which contains function, classes and variables.**



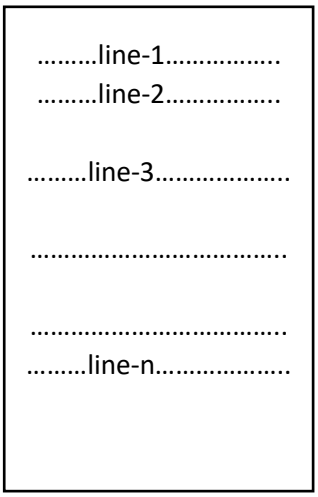
➔ Function vs Module vs Library

1. **Function:** a group of lines with some name is called a function
2. **Module:** a group of function saved to a file, is called module
3. **Library:** a group of modules is called Library.

Library



Function



Variable, Operator, Control Statement, Function, Module, Package, Library

1. Variable

- A **variable** is a name used to store data in memory.
- x = 10 # 'x' is a variable storing value 10

2. Operator

- An **operator** is a symbol that performs operations on variables or values.
- a + b # '+' is an addition operator

3. Control Statement

Control statements are used to control the flow of the program (decision making and loops).

- if, else, elif → for decisions
- for, while → for loops
- break, continue, pass → loop control

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

4. Function

A **function** is a block of reusable code that performs a specific task.

```
def greet(name):
  print("Hello", name)
```

5. Module

A **module** is a single .py file that can contain variables, functions, or classes.

```
# mymodule.py
def add(a, b):
    return a + b
```

6. Package

A **package** is a folder containing multiple related Python modules and an __init__.py file.

```
mypackage/
├── __init__.py
├── module1.py
└── module2.py
```

7. Library

A **library** is a collection of **modules or packages** that provide ready-to-use tools for different tasks.

- Examples: math, pandas, numpy, matplotlib

```
import math
print(math.sqrt(16))  # Using a function from math library
```

FILE HANDLING

- To store data permanently or to give large inputs to the program files will be used.
- **Types:**
    1. Text file
    2. Binary file

- **To open the file, we will used:**
    - f=open ("filename", "mode")
      filename like: abt.text, skills.text, x.text etc.
      mode: read, write, append, create

➔ Text Mode: read, write, append, create

➔ Binary Mode: rb, wb, ab, xb, rb+wb+

❖ **read mode:**
- if you open any file in the read mode if file is not available. It will through error if file is available. It will open that file and file reference refer at beginning of the file, you can able to do only read operation.

❖ **write mode:**
- If you will open write, if file is not available it will create new file, if file is available, it overwrites old content will overwrite to new content, file pointer reference beginning of the file, you can do only write operation.

❖ **append mode:**
- If you will append file in append mode if file is not exit it will create a new file if file is available, it will open existing file, always file pointer will refers at end of the file you can add content to the file.

❖ **create mode(x):**
- if you will open any file in create mode, if file is not available it will create new file
- always file pointer references beginning of the file here we can do only wrote operation.

❖ **r+ mode: replace mode:**
- if file is not available it will through error if files is available, it will open file, file pointer reference as beginning of the file it will give permission fir read and write.

❖ **W+ mode:**
- If file is not available creates new file if file is available, it overwrites old content to new content.
- File pointer refer beginning you do write and read both operations.

❖ **a+ mode append +:**
- if file is not available create new file if file is available, it will open that file, file pointer reference at end of the file you can do write and read operation.

❖ **X+ mode (create +mode):**
- If file is not available create new file, file is available through error file pointer reference or beginning of the file. We can do both write and read operation.

1. Read Mode (r)

- **Purpose**: Opens an existing file for **reading only**.
- **Error**: Throws error if file **does not exist**.
- **Pointer**: Starts at the **beginning** of the file.
- **Use Case**: When you only want to **read** file content without modifying it.

Syntax: f = open("file.txt", "r")

Example 1:

```
f = open("file.txt", "r")
print(f.read())
f.close()
```

Example 2:

```
f = open("file.txt", "r")
print(f.readline())
f.close()
```

Example 3:

```
f = open("file.txt", "r")
for line in f:
   print(line)
      f.close()
```

2. Write Mode (w)

- **Purpose**: Opens a file for **writing only**.
- **Error**: **Creates** file if not exists; **overwrites** if exists.
- **Pointer**: Starts at **beginning**; clears old content.
- **Use Case**: When you want to **start fresh** and write new data.

Syntax: f = open("file.txt", "w")

Example 1

```
f = open("file.txt", "w")
f.write("Hello, Python!")
f.close()
```

Example 2

```
f = open("file.txt", "w")
f.write("Overwritten content.\n")
f.write("Line 2")
f.close()
```

Example 3

```
f = open("file.txt", "w")
data = ["One\n", "Two\n",
"Three\n"]
f.writelines(data)
f.close()
```

3. Append Mode (a)

- **Purpose**: Opens file to **append** data.
- **Error**: **Creates** file if it does not exist.
- **Pointer**: Starts at **end** of file; preserves existing data.
- **Use Case**: When you want to **add new content** without removing the old.

Syntax: f = open("file.txt", "a")

Example 1

```
f = open("file.txt", "a")
f.write("\nAppended line.")
f.close()
```

Example 2

```
f = open("file.txt", "a")
f.write("\nAdding another entry.")
f.close()
```

Example 3

```
f = open("file.txt", "a")
f.write("\nNew data added again.")
                f.close()
```

4. Create Mode (x)

- **Purpose**: **Creates** a new file for writing.
- **Error**: Throws error if file **already exists**.
- **Pointer**: Starts at the **beginning**.
- **Use Case**: When you want to ensure a **new file** is created.

Syntax: f = open("newfile.txt", "x")

Example 1

```
f = open("created1.txt", "x")
f.write("Created using x mode.")
f.close()
```

Example 2

```
f = open("created2.txt", "x")
f.write("New file generated.")
f.close()
```

Example 3

```
f = open("created3.txt", "x")
f.write("Secure creation of file.")
                    f.close()
```

5. Read & Write Mode (r+)

- **Purpose**: Opens existing file for **reading and writing**.
- **Error**: Throws error if file **does not exist**.
- **Pointer**: Starts at **beginning**.
- **Use Case**: When you want to **read and modify** the content.

Syntax: f = open("file.txt", "r+")

Example 1:

```
f = open("file.txt", "r+")
print(f.read())
f.write("\nExtra line in r+ mode.")
f.close()
```

Example 2:

```
f = open("file.txt", "r+")
f.write("Start edit.")
f.close()
```

Example 3:

```
f = open("file.txt", "r+")
lines = f.readlines()
f.seek(0)
f.write("Modified line 1\n" +
"".join(lines[1:]))
f.close()
```

6. Write & Read Mode (w+)

- **Purpose**: Opens file for **writing and reading**.
- **Error**: **Creates** file if not exists, **overwrites** if it does.
- **Pointer**: Starts at **beginning**.
- **Use Case**: When you want to **reset** a file and also read what you write.

Syntax: f = open("file.txt", "w+")

Example 1

```
f = open("file.txt", "w+")
f.write("Testing w+ mode.")
f.seek(0)
print(f.read())
f.close()
```

Example 2

```
f = open("file.txt", "w+")
f.write("Overwritten data.\n")
f.seek(0)
print(f.read())
f.close()
```

 # Move the file pointer to the beginning of the file
 # Because after writing, the pointer is at the end, and reading from there would return nothing.

Example 3

```
f = open("file.txt", "w+")
f.writelines(["Alpha\n", "Beta\n"])
f.seek(0)
print(f.read())
f.close()
```

7. Append & Read Mode (a+)

- **Purpose**: Opens file for **reading and appending**.
- **Error**: **Creates** file if not exists.
- **Pointer**: Starts at **end**, use seek(0) to read from start.
- **Use Case**: When you want to **read old data** and **add new**.

Syntax: f = open("file.txt", "a+")

Example 1

```
f = open("file.txt", "a+")
f.write("\nData via a+ mode.")
f.seek(0)
print(f.read())
f.close()
```

Example 2

```
f = open("file.txt", "a+")
f.seek(0)
print(f.readlines())
f.write("\nNext line.")
f.close()
```

Example 3

```
f = open("file.txt", "a+")
f.write("\nAdded again.")
f.seek(0)
print(f.read())
f.close()
```

8. Create + Read & Write Mode (x+)

- **Purpose**: **Creates** new file for **reading and writing**.
- **Error**: Throws error if file **already exists**.
- **Pointer**: Starts at the **beginning**.
- **Use Case**: When you want to ensure **safe creation** and read/write access.

Syntax: f = open("file.txt", "x+")

Example 1

```
f = open("secure1.txt", "x+")
f.write("New x+ file.")
f.seek(0)
print(f.read())
f.close()
```

Example 2

```
f = open("secure2.txt", "x+")
f.write("Created with x+ mode.")
f.seek(0)
print(f.read())
f.close()
```

Example 3

```
f = open("secure3.txt", "x+")
f.write("Initial content for new file.")
f.seek(0)
print(f.read())
f.close()
```

How to create file inside folder and Directory

- Create the empty file by using the double backword slash

Syntax: f=open("path", "mode")

1st method

F=open("D:\\folder_name\\file_name.txt", "x")

2nd Method

F=open("D:\\folder_name\\file_name.txt", "X") as f1

Example-1:

```
import os
folder_path = "D:\\newfolder"
file_path = os.path.join(folder_path, "f1.txt")
# Create folder if it doesn't exist
if not os.path.exists(folder_path):
   os.makedirs(folder_path)
# Now safely create the file
f = open(file_path, "x")
f.write("File created successfully.")
f.close()
```

#Example-2

```
import os
fp="D:\\python"
file_p=os.path.join(fp, "f2.txt")
f=open(fp, "x")
f.write("file created successfully !")
f.close()
```

Example:

```
f=open('skills.text','r')
print(f.read())
 f.close()
output
welcome to T3 skills center
 This is Led based skill center
we are providng python full stack course
--> tell() is used to know the current file position
```

skills.text

welcome to T3 skills center
This is Led based skill center
we are providng python full stack course

Example:

```
f=open('skills.text','r')
print(f.tell())
print(f.read())
print(f.tell())
print(f.close())
```

Output: 0

```
welcome to T3 skills center
This is Led based skill center
we are providng python full stack course
syllabus:
1.python
```

skills.text

welcome to T3 skills center
This is Led based skill center
we are providng python full stack course
syllabus:
1.python
2.HTML |CSS | JS
3.React | Node js | Express JS
4. my sql | mongo DB

2.HTML |CSS | JS
3.React | Node js | Express JS
4. my sql | mongo DB
188
None

- **Code:** f = open('skills.text', 'r')
  print(f.tell()) # Shows current pointer position (starts at 0)
  print(f.read()) # Reads the whole file
  print(f.tell()) # Now pointer is at the end (total characters read)
  print(f.close()) # Closes the file, returns None
- **Example Output (if file has Python, HTML, CSS):**
  0
  Python, HTML, CSS
  20
  None
- **Summary of Functions:**
  - read() – Reads file content as a string.
  - tell() – Shows file pointer position.
  - close() – Closes the file, returns None.

read() Function in Python

Definition:

- The read() function is used to **read data from a file**. When you open a file in read mode, you can use .read() to fetch its contents.

Syntax: file_object.read(size)

- file_object: The file variable (after opening a file using open()).
- size *(optional)*: Number of bytes (or characters) to read. If not provided, it reads **entire file**.

- **Key Points:**
  - Returns the data in the form of a **string** (for text files).
  - If size is not given, it reads the **entire content** of the file at once.
  - If size is given, it reads **only that many characters**.
  - Reading large files with read() can consume **a lot of memory**.

Example 1: Basic Use of read()

```
file = open("sample.txt", "r")
content = file.read()
print(content)
file.close()
```

Explanation:

- This reads the entire file into the content variable.
- If sample.txt has:
- Hello, world!
- This is a file.

The output will be:
Hello, world!
This is a file.

Example 2: Using read(size)

```
file = open("sample.txt", "r")
```

```
content = file.read(5)
print(content)
file.close()
```

Output: Hello

Explanation: Only first 5 characters are read: "Hello"

read() vs readline() vs readlines()

❖ **readline() — Reads one line at a time**

Syntax: file.readline()

Example:

```
file = open("sample.txt", "r")
line1 = file.readline()
line2 = file.readline()
print(line1)
print(line2)
file.close()
```

If file contains:

```
Line 1
Line 2
Line 3
```

Output: Line 1

Line 2

Note: The newline character \n is preserved at the end of each line.

🔹 **readlines() — Reads all lines and returns them as a list**

Syntax: file.readlines()

Example:

```
file = open("sample.txt", "r")
lines = file.readlines()
print(lines)
file.close()
```

Output: ['Line 1\n', 'Line 2\n', 'Line 3']

Important Notes:

1. After calling read() once, the **file pointer moves to the end**. If you try to read again, it will return an **empty string**.
2. file = open("sample.txt", "r")
3. print(file.read())
4. print(file.read()) # Will print nothing
5. file.close()
6. Always remember to **close the file** using file.close() or use a with block:
7. with open("sample.txt", "r") as file:
8. data = file.read()
9. print(data)
10. If the file is **very large**, avoid using read() without a size. It can **crash your program** due to memory overflow.

Best Practices:

- Use read(size) for large files, read in chunks.
- Use readline() when processing line-by-line (e.g., logs).
- Use with open(...) as file: for better memory and safety.

❖ write (): write mode

```
f=open('skills.text','w')
f.write("interted student can join our course") #copy new data in old content
#f.close()
f=open('skills.text','r')
print(f.read())  #interted student can join our course
```

Example:

```
f1=open("skills.text",'r') #interted student can join our course
f2=open("skills1.text",'w') # new file skills1.text will create
f2.write(f1.read()) # skills.text content will copy in kills.text file interted student can join our course
f1.close()
f2.close()
```

Problem: replace the java with python

Problem:

```
f=open("skills3.text", 'w') ) #new file skills3.text file will create
f.write("Java full stack course")
f=open('skills3.text','r')
s=f.read()
print(s) # Java full stack course
s1=s.replace('Java','python')
print(s1) #python full stack course
f=open('skills3.text','r')
 s=f.read()
print(s)
f.close()
f=open("skills4.text","w")
 f.write(s)
f.close()
f=open('skills3.text','r')
s=f.read()
print(s)
```

problem: write a python program to reverse the string that contains with file

Problem:

```
f=open("sangam.text",'w') ) #new file sangam.text file will create
f.write("python full stack course")
f=open('sangam.text','r')
s=f.read()
print(s)
s=s[::-1]
print(s)
f.close()
f=open("sangam.text",'w')
f.write(s)
f.close()
output:
```

python full stack course
esruoc kcats lluf nohtyp

Adding content in the existing file:

```
f=open("sangam.text",'a') #new file sangam.text file will create
f.write("-sangam is a very inteligent student")
f=open('sangam.text','r')
s=f.read()
print(s)
output:
esruoc kcats lluf nohtypsangam is a very inteligent student
```

Seek ()

❖ **seek():** used to change the file pointer position from the begging and specified upset position
syntax: f.seek(ofset, 0) # from beginng (0,1,2)

🔹 **seek() in Python**

- **Definition** seek() is used to **move the file pointer** to a specific position in a file.
- **Syntax:**
- file.seek(offset, whence)
  - offset: number of bytes to move.
  - whence: optional, default is 0 (beginning).
    - 0 = from beginning (default)
    - 1 = from current position
    - 2 = from end

Example:

```
f = open("sample.txt", "w+")
f.write("Hello World")
f.seek(0)  # Move to beginning of the file
print(f.read())  # Output: Hello World
f.close()
```

🔹 **Use Case:**

- When you want to **read from a specific position** after writing.
- Helps reset pointer to re-read or overwrite specific parts.

#adding the new content to a file and read the content

Example:

```
f=open("skills3.text", 'a+') # New skills3.text file will create
print(f.tell())
f.write("tech class")
print(f.tell())
f.seek(0,0)
print(f.tell())
print(f.read())
f.close()
output:
```

22
32
0
Java full stack coursetech class

Problem: readline is going to be used to read only one line text at a time

- readline used to read all the lines in a list of string formates

Example:

```
f=open("skills.text","r")
print(f.readline())
print(f.readline())
f.close()
output:
interted student can join our course
```

Example:

```
f=open("skills.text",'r')
print(f.readlines())
f.close()
```

Output: ['interted student can join our course']

- to read the specific number of bits from a file you will used

f.read(not bytes)
example:

```
f=open("skills3.text",'r')
print(f.read(6))
f.close()
output:
Java f
l=['\nhello'," how"," are"]
f=open("skills3.text", 'a')
f.writelines(l)
f.close()
f=open("skills3.text",'r')
print(f.read())
```

Output:

hellohigoodbadfungunpen
hello how are
hello how are

Example:

demo.text
101 raja 20 70
102 sushil 18 99
103 rakesh 30 83
104 sangam 21 100
105 satyam 23 98
106 sujeet 25 97

Program:

```
f=open('demo.text','r')
l=f.readlines()
```

```
for s in l:
    sid,name,age,marks=s.split()
    if int (marks.strip('\n'))>90:
        print(sid,name,age,marks)
    f.close()
```

Output:

```
102 sushil 18 99
104 sangam 21 100
105 satyam 23 98
106 sujeet 25 97
```

➔ replace sangam with golu

Problem:

```
f=open("demo.txt",'r+')
l=f.readlines()
for i in range(len(l)):
    l[i]=l[i].replace('sangam','golu')
f.seek(0,0)
f.writelines(l)
# print(f.read())
res=f.readlines()
print(res)
f.close()
```

Output:

```
['101 raja  20 70\n', '102 sushil 18 99\n', '103 rakesh 30 83\n', '104 golu 21 100\n',
'105 satyam 23 98\n', '106 sujeet 25 97\n',]
```

❖ modify the marks is less than 90 add some marks to add marks 90

demo.txt

```
['101', 'raja', '20', '70']
['102', 'sushil', '18', '99']
['103', 'rakesh', '30', '83']
['104', 'sangam', '21', '100']
['105', 'satyam', '23', '98']
['106', 'sujeet', '25', '97']
f=open("demo.txt", 'r+')
l=f.readlines()
for i in range(len(l)):
    x=l[i].split()
    #print(x)
    m=int(x[3])
    if m<90:
        m=m+20
        x[3]=str(m)+'\n'
        l[i]=' '.join(x)
f.seek(0,0)
f.writelines(l)
```

```
f.close()
f=open("demo.txt",'r')
print(f.read())
```

output:

```
 01 raja 20 90
102  sushil  18  99
103 rakesh 30 105
104  sangam  21  100
105  satyam  23  98
106  sujeet  25  97 97
```

Important concept in file handline

6. Using with Statement (Context Manager)

Definition: Automatically handles file closing; safer and cleaner.

Syntax: with open("filename.txt", "mode") as f:

Example:

```
with open("data.txt", "r") as f:
   print(f.read())
```

🔹 **7. Checking File Existence (Using os module)**

Definition: Used to check whether a file exists before performing operations.

Syntax:

```
import os
os.path.exists("filename.txt")
```

Example:

```
import os
if os.path.exists("data.txt"):
   print("File found!")
else:
   print("File not found!")
```

🔹 **8. Deleting a File**

Definition: Permanently removes a file using os.remove().

Syntax:

```
import os
os.remove("filename.txt")
```

Example:

```
import os
os.remove("data.txt")
```

🔹 **9. Renaming a File**

Definition: Changes the file name using os.rename().

Syntax: os.rename("old.txt", "new.txt")

Example:

```
import os
os.rename("data.txt", "mydata.txt")
```

🔹 **10. File Pointer Control (seek() and tell())**

Definition:

- seek() changes the position of the file pointer.
- tell() returns the current position of the file pointer.

Syntax:

```
f.seek(offset)
f.tell()
```

Example:

```
f = open("data.txt", "r")
print(f.tell())   # Shows position
f.seek(5)       # Moves pointer to 5th byte
print(f.read())
f.close()
```

Important question based on file handling

1. **Q:** What function is used to open a file in Python?
   A: open()
2. **Q:** Which mode opens a file for reading?
   A: 'r'
3. **Q:** What does the 'w' mode do?
   A: Opens a file for writing and truncates it if it exists.
4. **Q:** Which mode is used to append data to a file?
   A: 'a'
5. **Q:** What does 'b' indicate in file modes?
   A: Binary mode.
6. **Q:** How do you read the entire file content at once?
   A: Using read()
7. **Q:** Which method reads one line at a time?
   A: readline()
8. **Q:** What method returns all lines as a list?
   A: readlines()
9. **Q:** How do you close a file in Python?
   A: Using close()
10. **Q:** What does the with statement do in file handling?
    A: Automatically manages file opening and closing.
11. **Q:** What will open("file.txt", "r") do if the file doesn't exist?
    A: Raise FileNotFoundError.
12. **Q:** Which module is used to check if a file exists?
    A: os
13. **Q:** How do you write text to a file?
    A: Using write() method.
14. **Q:** How do you move the file pointer to a specific position?
    A: Using seek()
15. **Q:** How do you get the current position of the file pointer?
    A: Using tell()
16. **Q:** What is the default file mode in open()?
    A: 'r'
17. **Q:** What happens if you open a file in 'x' mode and it exists?
    A: Raises FileExistsError.
18. **Q:** Can we use for loop to read file lines?
    A: Yes.

19. **Q:** Which mode should be used to write binary data?
    A: 'wb'
20. **Q:** Is file handling part of Python's standard library?
    A: Yes.
21. **Q:** What does 'r+' mode do in file handling?
    A: Opens file for both reading and writing without truncating it.
22. **Q:** What does 'w+' mode do?
    A: Opens file for reading and writing, truncates the file first.
23. **Q:** What does 'a+' mode do?
    A: Opens file for reading and appending, creates if it doesn't exist.
24. **Q:** What is the difference between 'r' and 'r+'?
    A: 'r' is read-only; 'r+' allows both reading and writing.
25. **Q:** What will write() return?
    A: Number of characters written.
26. **Q:** What does file.closed return?
    A: True if file is closed, else False.
27. **Q:** Can you open a file in both binary and write mode?
    A: Yes, using 'wb'.
28. **Q:** What is a file pointer?
    A: It indicates the current position in the file.
29. **Q:** What is the output of tell() just after opening a file?
    A: Usually 0.
30. **Q:** How do you create a new file only if it doesn't exist?
    A: Use 'x' mode.
31. **Q:** What does truncate() do?
    A: Resizes the file to a given number of bytes.
32. **Q:** Can seek() be used in text and binary files?
    A: Yes.
33. **Q:** What is the purpose of buffering in file handling?
    A: To improve performance by reducing I/O operations.
34. **Q:** What error occurs if you write to a file opened in 'r' mode?
    A: io.UnsupportedOperation
35. **Q:** Can read() be called multiple times?
    A: Yes, but subsequent calls read from current file pointer position.
36. **Q:** How do you open a file using UTF-8 encoding?
    A: open("file.txt", "r", encoding="utf-8")
37. **Q:** Which method is used to remove a file?
    A: os.remove("filename")
38. **Q:** Can with statement handle binary files?
    A: Yes.
39. **Q:** What happens if you open a binary file in text mode?
    A: May cause decoding errors.
40. **Q:** What does flush() do in file handling?
    A: Forces the write buffer to be written to disk.

Numerical Question based on file handling

1. Write a Python program to count the number of characters in a file.

```
with open("sample.txt", "r") as f:
    data = f.read()
    print("Total characters:", len(data))
```

2. Write a Python program to count the number of words in a file.

```
with open("sample.txt", "r") as f:
    data = f.read()
    words = data.split()
    print("Total words:", len(words))
```

3. Write a Python program to count the number of lines in a file.

```
with open("sample.txt", "r") as f:
    lines = f.readlines()
    print("Total lines:", len(lines))
```

4. Write a program to count how many times the word "Python" appears in a file.

```
with open("sample.txt", "r") as f:
    data = f.read()
    print("Occurrences of 'Python':", data.count("Python"))
```

5. Write a program to print the first 5 lines of a file.

```
with open("sample.txt", "r") as f:
    for i in range(5):
        print(f.readline(), end='')
```

6. Write a program to count the number of vowels in a file.

```
with open("sample.txt", "r") as f:
    data = f.read().lower()
    count = sum(1 for char in data if char in "aeiou")
    print("Total vowels:", count)
```

7. Write a Python program to count the number of blank lines in a file.

```
with open("sample.txt", "r") as f:
    lines = f.readlines()
    blank_lines = sum(1 for line in lines if line.strip() == "")
    print("Blank lines:", blank_lines)
```

8. Write a program to count the number of digits in a file.

```
with open("sample.txt", "r") as f:
    data = f.read()
    digit_count = sum(1 for char in data if char.isdigit())
    print("Total digits:", digit_count)
```

9. Write a Python program to display the length of the longest line in a file.

```
with open("sample.txt", "r") as f:
    longest = max(len(line) for line in f)
    print("Length of the longest line:", longest)
```

10. Write a program to copy the content of one file into another.

```
with open("sample.txt", "r") as source:
    with open("copy.txt", "w") as destination:
        destination.write(source.read())
print("Content copied successfully.")
```

IMPORT QUESTION REGARDING CORE PYTHON

- What is Python?
  - Python is a high-level, interpreted programming language known for its simplicity and readability.
- How do you comment in Python?
  - Use the `#` character to create single-line comments. For multi-line comments, you can enclose the text within triple quotes (`'''` or `"""`).
- What is PEP 8?
  - PEP 8 is the Python Enhancement Proposal that provides style guidelines for writing Python code.
- What is a variable in Python?
  - A variable is a name assigned to a value or object in Python.
- What are the basic data types in Python?
  - Python has several built-in data types, including int, float, str, bool, and more.
- How do you declare a variable in Python?
  - You can declare a variable by assigning a value to it, e.g., `x = 10`.
- What is a list in Python?
  - A list is an ordered collection of items that can contain different data types.
- How do you create an empty list in Python?
  - You can create an empty list using `my_list = []` or `my_list = list()`.
- What is a tuple in Python?
  - A tuple is an ordered, immutable collection of items.
- How do you create a tuple in Python?
  - Tuples are created using parentheses, e.g., `my_tuple = (1, 2, 3)`.
- What is a dictionary in Python?
  - A dictionary is an unordered collection of key-value pairs.
- How do you create a dictionary in Python?
  - Dictionaries are created using curly braces `{}` or the `dict()` constructor.
- What is a set in Python?
  - A set is an unordered collection of unique elements.
- How do you create a set in Python?
  - Sets are created using curly braces `{}` or the `set()` constructor.
- What is the `if` statement used for in Python?
  - The `if` statement is used for conditional execution of code based on a condition.
- What is a `for` loop in Python?
  - A `for` loop is used for iterating over a sequence (e.g., a list) or other iterable objects.
- What is a `while` loop in Python?
  - A `while` loop is used to repeatedly execute a block of code as long as a condition is true.
- What is the purpose of the `range()` function?
  - The `range()` function generates a sequence of numbers, often used for iterating in loops.
- What is a function in Python?
  - A function is a reusable block of code that performs a specific task.
- How do you define a function in Python?
  - Functions are defined using the `def` keyword followed by a function name and parameters.
- What is a lambda function?
  - A lambda function is a small, anonymous function defined using the `lambda` keyword.

- What is a module in Python?
  - A module is a file containing Python code that can be reused in other Python programs.
- How do you import a module in Python?
  - You can import a module using the `import` statement, e.g., `import math`.
- What is the purpose of the `__init__` method in a class?
  - The `__init__` method is a special method used to initialize objects of a class.
- What is inheritance in Python?
  - Inheritance is a mechanism that allows a new class to inherit properties and methods from an existing class.
- What is polymorphism in Python?
  - Polymorphism allows objects of different classes to be treated as objects of a common superclass.
- What is encapsulation in Python?
  - Encapsulation is the concept of restricting access to certain parts of an object and exposing only what is necessary.
- What is a generator in Python?
  - A generator is a special type of iterable that generates values on-the-fly using the `yield` keyword.
- What is a decorator in Python?
  - A decorator is a function that can modify the behavior of other functions or methods.
- What is exception handling in Python?
  - Exception handling is a way to handle errors or exceptions that may occur during program execution.
- What is the purpose of the `try` and `except` blocks in exception handling?
  - The `try` block is used to enclose code that might raise an exception, while the `except` block handles the exception if it occurs.
- What is the purpose of the `finally` block in exception handling?
  - The `finally` block is used to execute code regardless of whether an exception was raised or not.
- What is a Python package?
  - A package is a directory that contains a collection of Python modules.
- How do you install external packages in Python?
  - You can use the `pip` command to install external packages, e.g., `pip install package_name`.
- What is a virtual environment in Python?
  - A virtual environment is a self-contained environment that allows you to install and manage Python packages separately from the system-wide installation.
- What is the purpose of the `with` statement in Python?
  - The `with` statement is used for resource management, ensuring that resources like files are properly opened and closed.
- What is list comprehension in Python?
  - List comprehension is a concise way to create lists based on existing lists or other iterable objects.
- What is a docstring in Python?
  - A docstring is a string that provides documentation for a module, function, class, or method.
- What is the difference between `==` and `is` in Python?

- `==` compares the values of two objects, while `is` checks if two objects are the same (i.e., they occupy the same memory location).

❖ What is the Global Interpreter Lock (GIL) in Python?
- The GIL is a mutex that protects access to Python objects, allowing only one thread to execute Python code at a time.

❖ What is the purpose of the `import` statement in Python?
- The `import` statement is used to bring in modules or packages for use in your code.

❖ What is the difference between a list and a tuple in Python?
- Lists are mutable, while tuples are immutable. Lists are defined using square brackets, and tuples use parentheses.

❖ What is a Python generator expression?
- A generator expression is a compact way to create a generator. It uses parentheses and is similar to list comprehension.

❖ What is a dictionary comprehension in Python?
- A dictionary comprehension is a way to create dictionaries using a concise and readable syntax.

❖ What is the purpose of the`else` clause in a `for` loop in Python?
- The `else` clause is executed when the `for` loop completes without being interrupted by a `break` statement.

❖ What is a context manager in Python?
- A context manager is an object that defines methods (`__enter__` and `__exit__`) for resource management using the `with` statement.

❖ What is the purpose of the `enumerate()` function in Python?
- `enumerate()` is used to iterate over an iterable while keeping track of the index of the current item.

❖ What is the purpose of the `zip()` function in Python?
- `zip()` is used to combine multiple iterables (e.g., lists) element-wise into tuples.

❖ What is the purpose of the `map()` function in Python?
- `map()` applies a function to each item in an iterable and returns an iterable of the results.

❖ What is the purpose of the `filter()` function in Python?
- `filter()` filters an iterable based on a given function, returning only the elements that satisfy a condition.

❖ What is a Python iterator?
- An iterator is an object that implements the `__iter__` and `__next__` methods, allowing you to iterate over a sequence of values.

❖ What is a Python generator function?
- A generator function is a special type of function that contains one or more `yield` statements, creating a generator object when called.

❖ What is a Python closure?
- A closure is a function that remembers the environment in which it was created, including the variables from that environment.

❖ What is a module-level global variable in Python?
- A module-level global variable is a variable defined at the module level and can be accessed throughout the module.

❖ How do you read data from the keyboard in Python?
- You can use the `input()` function to read user input from the keyboard.

❖ What is the purpose of the `break` statement in Python?

- The `break` statement is used to exit a loop prematurely.

❖ What is the purpose of the `continue` statement in Python?
- The `continue` statement is used to skip the current iteration of a loop and proceed to the next one.

❖ What is the difference between a shallow copy and a deep copy in Python?
- A shallow copy creates a new object with references to the same objects as the original, while a deep copy creates a completely independent copy of the original object.

❖ How do you open and read a file in Python?
- You can use the `open()` function to open a file and methods like `read()`, `readline()`, or `readlines()` to read its contents.

❖ What is the purpose of the `os` module in Python?
- The `os` module provides functions for interacting with the operating system, including file and directory manipulation.

❖ What is a Python slice?
- A slice is a way to extract a portion of a sequence (e.g., a list or string) using the `[start:stop]` notation.

❖ How do you remove an item from a list in Python?
- You can use the `remove()` method to remove an item by value, or the `pop()` method to remove an item by index.

❖ What is the purpose of the `del` statement in Python?
- The `del` statement is used to delete variables, items from a list, or attributes from objects.

❖ What is a decorator in Python?
- A decorator is a design pattern that allows you to modify the behavior of functions or methods without changing their code.

❖ What is the purpose of the `assert` statement in Python?
- The `assert` statement is used to check if a condition is `True`, and if not, it raises an `AssertionError` exception.

❖ What is the `None` object in Python?
- `None` is a special object representing the absence of a value or a null value.

❖ What is a docstring in Python?
- A docstring is a string that provides documentation within a Python function, module, class, or method.

❖ What is the purpose of the `global` keyword in Python?
- The `global` keyword is used inside a function to indicate that a variable is a global variable, rather than a local one.

❖ What is the purpose of the `nonlocal` keyword in Python?
- The `nonlocal` keyword is used inside a nested function to indicate that a variable should refer to an outer (non-global) variable.

❖ How do you sort a list in Python?
- You can use the `sort()` method to sort a list in-place or the `sorted()` function to create a new sorted list.

What is RID Organization (RID संस्था क्या है)?

- **RID Organization** यानि **Research, Innovation and Discovery Organization एक** संस्था हैं जो TWF (TWKSAA WELFARE FOUNDATION) **NGO** द्वारा RUN किया जाता है | जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही **इस RID Organization** की स्थपना 30.09.2023 किया गया है | जो TWF द्वारा संचालित किया जाता है |
- TWF (TWKSAA WELFARE FOUNDATION) NGO की स्थपना 26-10-2020 में बिहार की पावन धरती सासाराम में Er. RAJESH PRASAD एवं Er. SUNIL KUMAR द्वारा किया गया था जो की भारत सरकार द्वारा मान्यता प्राप्त संस्था हैं |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

MISSION, VISSION & MOTIVE OF "RID ORGANIZATION"	
मिशन	हर एक ONE भारत के संग
विजन	TALENT WORLD KA SHRESHTM AB AAYEGA भारत में और भारत का TALENT भारत में
मकसद	NEW (RID, PMS, TLR)

MOTIVE OF RID ORGANIZATION NEW (RID, PMS, TLR)		
NEW (RID)		
R	I	D
Research	Innovation	Discovery
NEW (TLR)		
T	L	R
Technology, Theory, Technique	Law	Rule
NEW (PMS)		
P	M	S
Product, Project, Production	Machine	Service

RID रीड संस्था की मिशन, विजन एवं मकसद को सार्थक हमें बनाना हैं |
भारत के वर्चस्व को हर कोने में फैलना हैं |
कर के नया कार्य एक बदलाव समाज में लाना हैं |
रीड संस्था की कार्य-सिध्दांतों से ही, हमें अपनी पहचान बनाना हैं |

Er. Rajesh Prasad (B.E, M.E)
Founder:
TWF & RID Organization

Core Python के इस **E-Book** में अगर मिलती त्रुटी मिलती है तो कृपया हमें सूचित करें | **WhatsApp's No: 9202707903 or**
Email Id: ridorg.in@gmail.com

|| धन्यवाद ||

RESEARCH INNOVATION DISCOVERY
RID
BHARAT