

## Levels of Programming Languages

Programming is the act of developing a piece of software. And software contains instructions that tell a computer what to do. Therefore, programming is the process of developing instructions that tell a computer what to do.

And you can tell the computer what to do with with a with a programming language. Each programming language was invented for a specific purpose. Each succeeding programming language builds on the strength of it's predecessors. For example, Machine Language is succeeded by Assembly Language, which is succeeded by high-level languages. In other words, High-level programming languages are build on Assembly Language, which is built on Machine Language. The idea is that higher level languages calls functions of a lower level languages with a single-simple short code.

### What is Machine Language?

A computer's native language is called *Machine Language*. Machine language is the most primitive or basic programming language that starts or takes instructions in the form of raw binary code. So that if you wanted to give a computer an instruction in its native or *Machine* language, you have to manually enter the instructions as binary code.

For example, adding two numbers together in machine language would look like this:

```
1101101010011010
```

### What is Assembly Language?

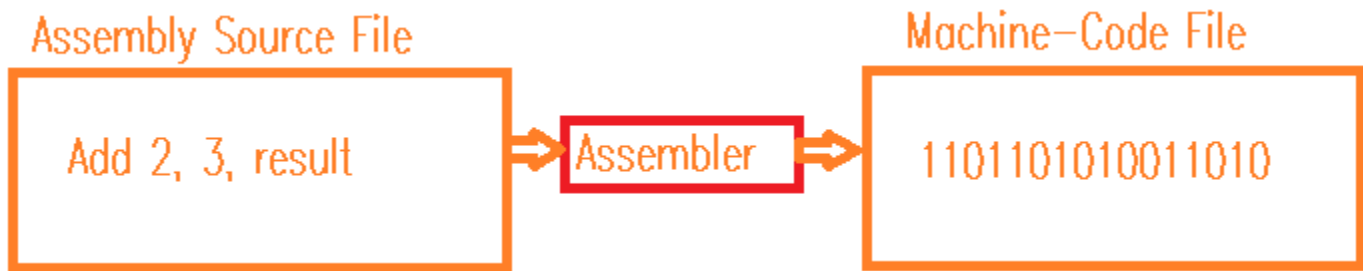
Programming in Machine language is tedious (you have to program every command from scratch) and hard to read & modify (the 1s and 0s are kind of hard to work with...). For these reasons, Assembly language was developed as an alternative to Machine language.

Assembly Language uses short descriptive words (mnemonic) to represent each of the Machine Language instructions.

For example the mnemonic **add** means to add numbers together, and **sub** means to subtract the numbers. So if you want to add the numbers 2 and 3 in assembly language, it would look like this:

```
add 2, 3, result
```

So Assembly Languages were developed to make programming easier. However, the computer cannot directly execute the assembly language. First another program called the **assembler** is used to translate the Assembly Language into machine code.



### What is a High-Level Language?

High-Level languages are platform independent, meaning that you can write & run High-Level Languages on different types of machines. High-Level Languages are English like and therefore easier to learn and use. Note that instructions in a High-Level Language are called **statements**.

Note that a program written in a high-level language is called the **source code**. Note that the Source Code must be translated into machine code before the computer can execute the source code. And the translations are done by programming tools called an **interpreter** or **compiler**.

Here's an example of a High-Level Language statement that calculates the area of a circle with a radius of 5:

```
area = 5 * 5 * 3.14159;
```

Examples of High-Level Programming Languages include Ada, BASIC, C, C++, C#, COBOL, FORTRAN, Java, Pascal, Python, and Visual Basic.

### Types of Programming Languages

*[Under Construction]*

- **Data-oriented Language:** These programming languages are designed for searching and manipulating relation that have been described as entity relationship tables which map one set of things into other sets. Example: SQL
- Imperative Language: ?
- Object-oriented programming (OOP) support objects defined by their class. ... Focuses on objects over action, data over logic.

## MACHINE LANGUAGE

## ASSEMBLY LANGUAGE

Machine language is only understood by the computers.

Assembly language is only understood by human beings not by the computers.

In machine language data is only represented with the help of binary format (0s and 1s), hexadecimal and octadecimal.

In assembly language data can be represented with the help of mnemonics such as Mov, Add, Sub, End etc.

Machine language is very difficult to understand by the human beings.

Assembly language is easy to understand by the human being as compared to machine language.

Modifications and error fixing cannot be done in machine language.

Modifications and error fixing can be done in assembly language.

Machine language is very difficult to memorize so it is not possible to learn the machine language.

Easy to memorize the assembly language because some alphabets and mnemonics are used.

Execution is fast in machine language because all data is already present in binary format.

Execution is slow as compared to machine language.

There is no need of translator.The machine understandable form is the machine language.	Assembler is used as translator to convert mnemonics into machine understandable form.
Machine language is hardware dependent.	Assembly language is the machine dependent and it is not portable.

Generation of Programming Languages

There are five generation of Programming languages.They are:

**First Generation Languages :**

These are low-level languages like machine language.

**Second Generation Languages :**

These are low-level assembly languages used in kernels and hardware drives.

**Third Generation Languages :**

These are high-level languages like C, C++, Java, Visual Basic and JavaScript.

**Fourth Generation Languages :**

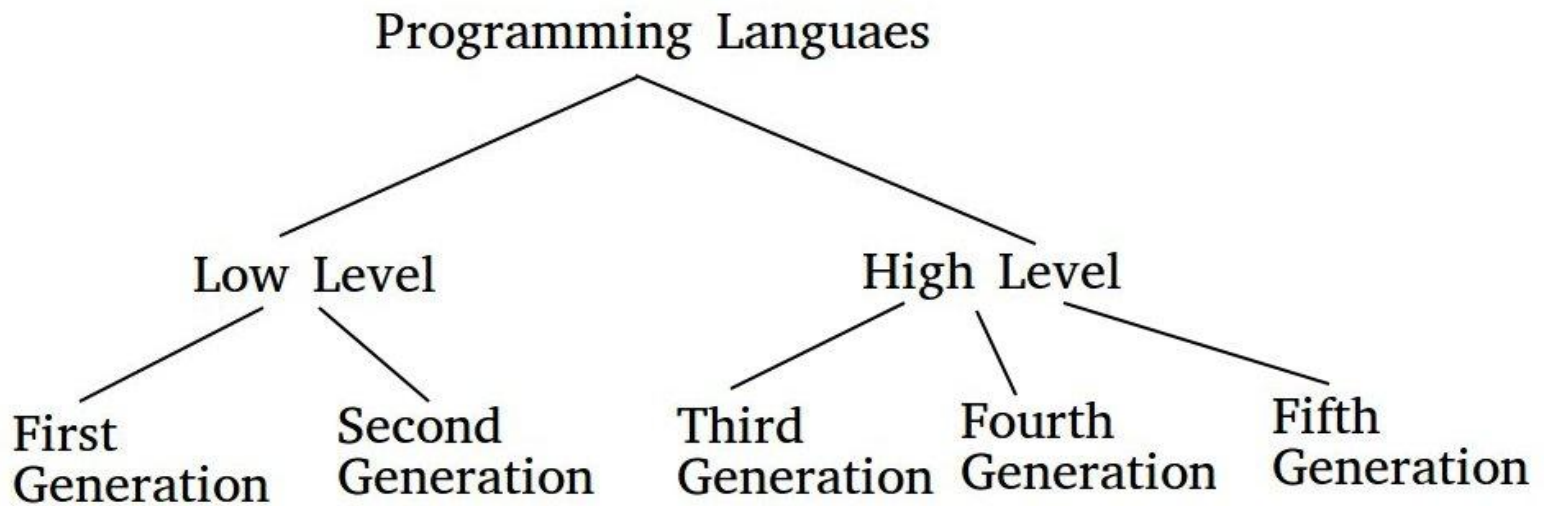
These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting. Example of these languages include Perl, Python, Ruby, SQL, MatLab(MatrixLaboratory).

**Fifth Generation Languages :**

These are the programming languages that have visual tools to develop a program.

Examples of fifth generation language include Mercury, OPS5, and Prolog.

The first two generations are called low level languages. The next three generations are called high level languages.



## What is OOPS?

**OBJECT ORIENTED PROGRAMMING (OOP)** is a programming concept that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create the objects that they want and then, create methods to handle those objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

Object Oriented Programming popularly known as OOP, is used in a modern programming language like Java

## Core OOPS concepts are

### 1) Class

The class is a group of similar entities. It is only a logical component and not the physical entity. For example, if you had a class called “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc. Its properties(data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.

### 2) Object

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object contains both the data and the function, which operates on the data. For example - chair, bike, marker, pen, table, car, etc.

### 3) Inheritance

Inheritance is an OOPS concept in which one object acquires the properties and behaviors of the parent object. It's creating a parent-child relationship between two classes. It offers a robust and natural mechanism for organizing and structure of any software.

#### 4) Polymorphism

Polymorphism refers to the ability of a variable, object or function to take on multiple forms. For example, in English, the verb *run* has a different meaning if you use it with *a laptop*, *a foot race*, and *business*. Here, we understand the meaning of *run* based on the other words used along with it. The same also applied to Polymorphism.

#### 5) Abstraction

An abstraction is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

#### 6) Encapsulation

Encapsulation is an OOP technique of wrapping the data and code. In this OOPS concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example - in school, a student cannot exist without a class.

#### 7) Association

Association is a relationship between two objects. It defines the diversity between objects. In this OOP concept, all object have their separate lifecycle, and there is no owner. For example, many students can associate with one teacher while one student can also associate with multiple teachers.

#### 8) Aggregation

In this technique, all objects have their separate lifecycle. However, there is ownership such that child object can't belong to another parent object. For example consider class/objects department and teacher. Here, a single teacher can't belong to multiple departments, but even if we delete the department, the teacher object will never be destroyed.

#### 9) Composition

A composition is a specialized form of Aggregation. It is also called "death" relationship. Child objects do not have their lifecycle so when parent object deletes all child object will also delete automatically. For that, let's take an example of House and rooms. Any house can have several rooms. One room can't become part of two different houses. So, if you delete the house room will also be deleted.

#### Advantages of OOPS:

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development cost.

- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.

## Comparison of OOPS with other programming styles with help of an Example

Let's understand with example how OOPs is different than other programming approaches.

Programming languages can be classified into 3 primary types

1. **Unstructured Programming Languages:** The most primitive of all programming languages having sequentially flow of control. Code is repeated through out the program
2. **Structured Programming Languages:** Has non-sequentially flow of control. Use of functions allows for re-use of code.
3. **Object Oriented Programming:** Combines Data & Action Together.

Let's understand these 3 types with an example.

Suppose you want to create a Banking Software with functions like

1. Deposit
2. Withdraw
3. Show Balance

### Unstructured Programming Languages

The earliest of all programming language were unstructured programming language. A very elementary code of banking application in unstructured Programming language will have two variables of one account number and another for account balance

```
int account_number=20;  
int account_balance=100;
```

Suppose deposit of 100 dollars is made.

```
account_balance=account_balance+100
```

Next you need to display account balance.

```
printf("Account Number=%d,account_number)  
printf("Account Balance=%d,account_balance)
```

Now the amount of 50 dollars is withdrawn.

```
account_balance=account_balance-50
```

Again, you need to display the account balance.

```
printf("Account Number=%d,account_number)
```

```
printf("Account Balance=%d,account_balance)
```

```
int account_number = 20;  
int account_balance = 100;  
  
account_balance = account_balance+100  
printf("Account Number = %d",account_number)  
printf("Account Balance = %d",account_balance)  
account_balance = account_balance-50  
printf("Account Number = %d",account_number)  
printf("Account Balance = %d",account_balance)  
account_balance = account_balance-10  
printf("Account Number = %d",account_number)  
printf("Account Balance = %d",account_balance)
```

A diagram illustrating unstructured programming. It shows a sequence of code blocks. A red box at the top right contains the text "unstructured programming same code is repeated". Three green dashed arrows point from this box to three identical blocks of code, each consisting of a printf statement for account number and a printf statement for account balance, demonstrating the repetition of code in unstructured programming.

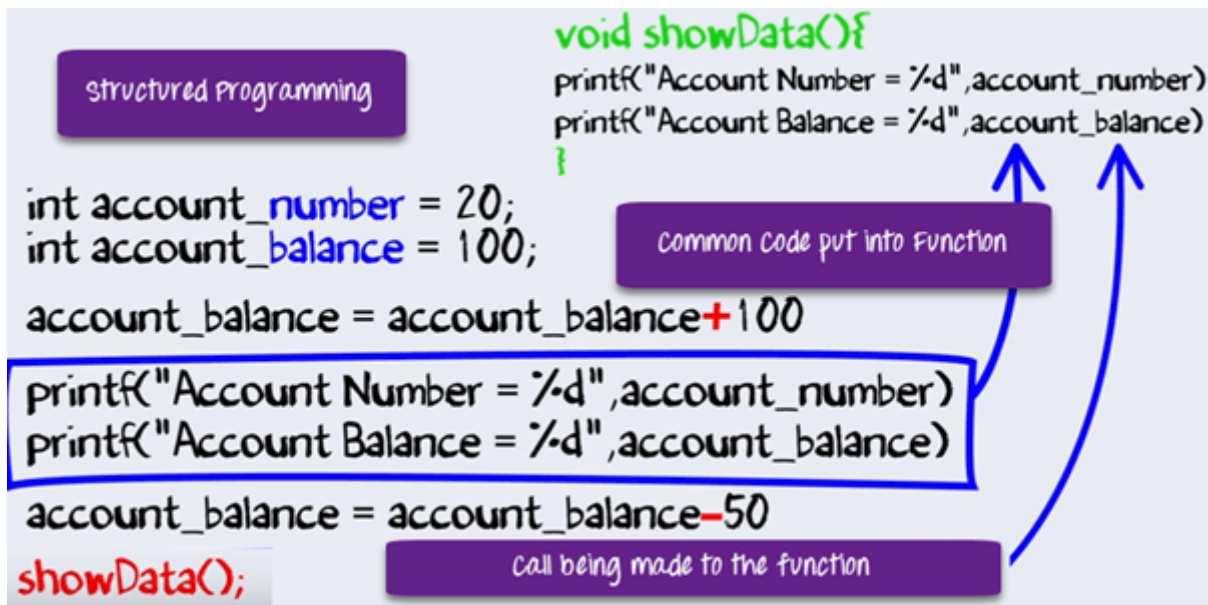
unstructured programming  
same code is repeated

For any further deposit or withdrawal operation – you will code repeat the same lines again and again.

## Structured Programming

With the arrival of Structured programming repeated lines on the code were put into structures such as functions or methods. Whenever needed, a simple call to the function is made.





## Object-Oriented Programming

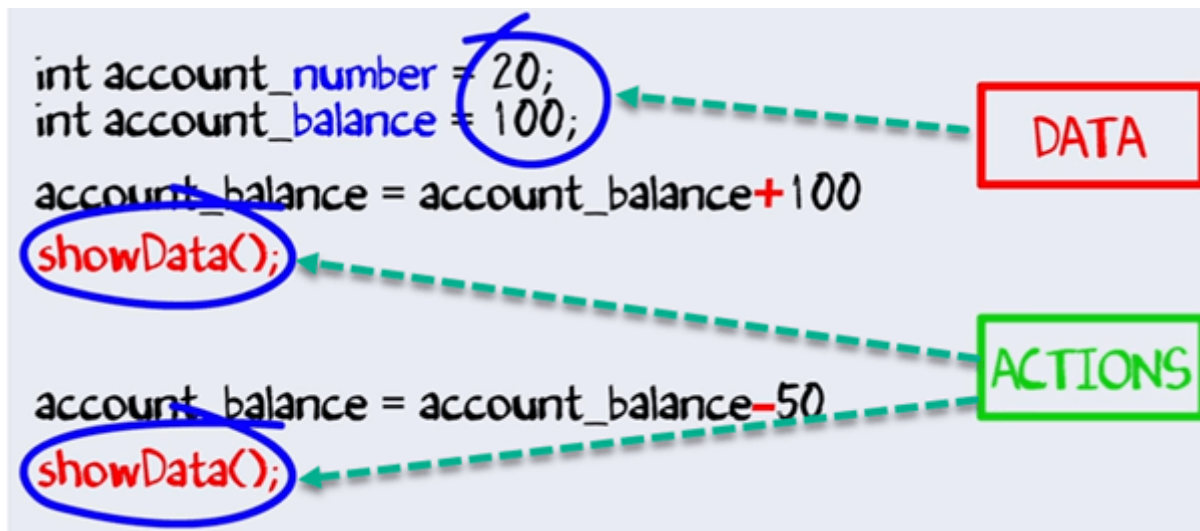
In our program, we are dealing with data or performing specific operations on the data.

In fact, having data and performing certain operation on that data is very basic characteristic in any software program.

Experts in Software Programming thought of combining the Data and Operations. Therefore, the birth of Object Oriented Programming which is commonly called OOPS.

The same code in OOPS will have same data and some action performed on that data.

```
Class Account{
    int account_number;
    int account_balance;
public void showdata(){
    system.out.println("Account Number"+account_number)
    system.outprintln("Account Balance"+ account_balance)
}
}
```



By combining data and action, we will get many advantages over structural programming viz,

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Basic Concepts of SQL:

- 1. **SQL stands for Structured Query Language. SQL is a programming language that lets you communicate with databases** to store, manipulate, and retrieve data, and also modify the structure of the database. **A database can be considered as a container for tables** and a table is a grid with rows and columns to hold data. Individual statements in SQL are called queries.
- 2. **SQL is case-INsensitive.** However it is become standard in SQL community to use all capital letters for SQL keywords.
- 3. In SQL, **extra white spaces** like spaces, tabs and carriage returns **are stripped out before the SQL statement is processed.**
- 4. SQL commands can be divided into **DDL, DML, DCL and TCL:**
  - a. **DDL stands for Data Definition Language**, and contain commands for creating, modifying and deleting databases, schemas, tables, columns etc. such as **CREATE, ALTER, RENAME, DROP etc.**
  - b. **DML stands for Data Manipulation Language**, and these queries are used to create or manipulate the actual data. **SQL implements CRUD operations through INSERT, SELECT, UPDATE and DELETE** commands that belong to DML. *CRUD (create, read, update and delete) operations are the four basic functions of any persistent storage mechanisms.*
  - c. **DCL stands for Data Control Language** and contain commands such as **GRANT, REVOKE etc.**
  - d. **TCL stands for Transaction Control** and contain commands such as **COMMIT, SAVEPOINT, ROLLBACK etc.**

- 5. **Common data types** that we can use in SQL are **INT, VARCHAR, FLOAT, DATE and TIME**. We can also have **user defined data types** based on predefined SQL types.
- 
- 6. Different **constraints** can be used while creating tables in SQL such as **UNIQUE, PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK, DEFAULT** etc.
- a. A **PRIMARY KEY** is a column or group of columns that represents a **unique identifier for each row** in a table. PRIMARY key cannot have NULL values.
- b. A **UNIQUE** constraint is similar to PRIMARY key, but you can have **more than one UNIQUE constraint per table**. Also, UNIQUE constraints **can accept NULL**, but just once. If the constraint is defined in a combination of fields, then every field can accept NULL and can have some values on them, as long as the combination values is unique.
- c. A **FOREIGN KEY** in one table **points to a PRIMARY KEY in another table**; and is used to specify relationship between tables. The FOREIGN KEY constraint **prevents invalid data from being inserted into the foreign key column**, because it has to be one of the PK values in the table it points to.
- d. The **CHECK constraint** lets you **constrain values** stored in your table.
- e. **DEFAULT constraint** gives all fields in a **default value**.
- 
- 7. You can use **INSERT** command to insert data into a table.
- INSERT INTO employee values ('Heartin',1);
- 
- 8. We can **use INSERT and SELECT together to insert rows from a query**. The column names of the columns returned by the SELECT statement are ignored by SQL in an INSERT SELECT statement. So the names of the returned columns do not need to match the column names of the columns data is going into.
- INSERT INTO employee SELECT \* from emp WHERE id >3;
- 
- 9. You can use SELECT to retrieve data. You can **specify which columns** to retrieve **in an SQL SELECT statement or use \*** for all columns. You can also specify an SQL expression, a UNIQUE or DISTINCT, or a combination in a SELECT statement.
- a. SELECT \* FROM employee;
- b. SELECT empName, id FROM employee;
- c. SELECT empName, MAX(id) FROM employee;
- d. SELECT empName, CURRENT\_DATE FROM employee;
- e. SELECT UNIQUE empName FROM Employees;
- f. SELECT COUNT(DISTINCT lastname) FROM Employees;
- 
- 10. You can use the UNIQUE and DISTINCT keywords as a field constraint, in the WHERE clause or in the SELECT statement. The difference is that **UNIQUE treats two NULL rows as still unique, while DISTINCT treats two NULL rows as NOT UNIQUE**. It is **better to use DISTINCT** as SELECT UNIQUE is considered "old" SQL and your DBMS might not support it.
- 
- 11. We can use **ORDER BY** to sort our data and it comes at the end of SELECT statements. The COLUMN you sort on does not need to be returned by SELECT query.

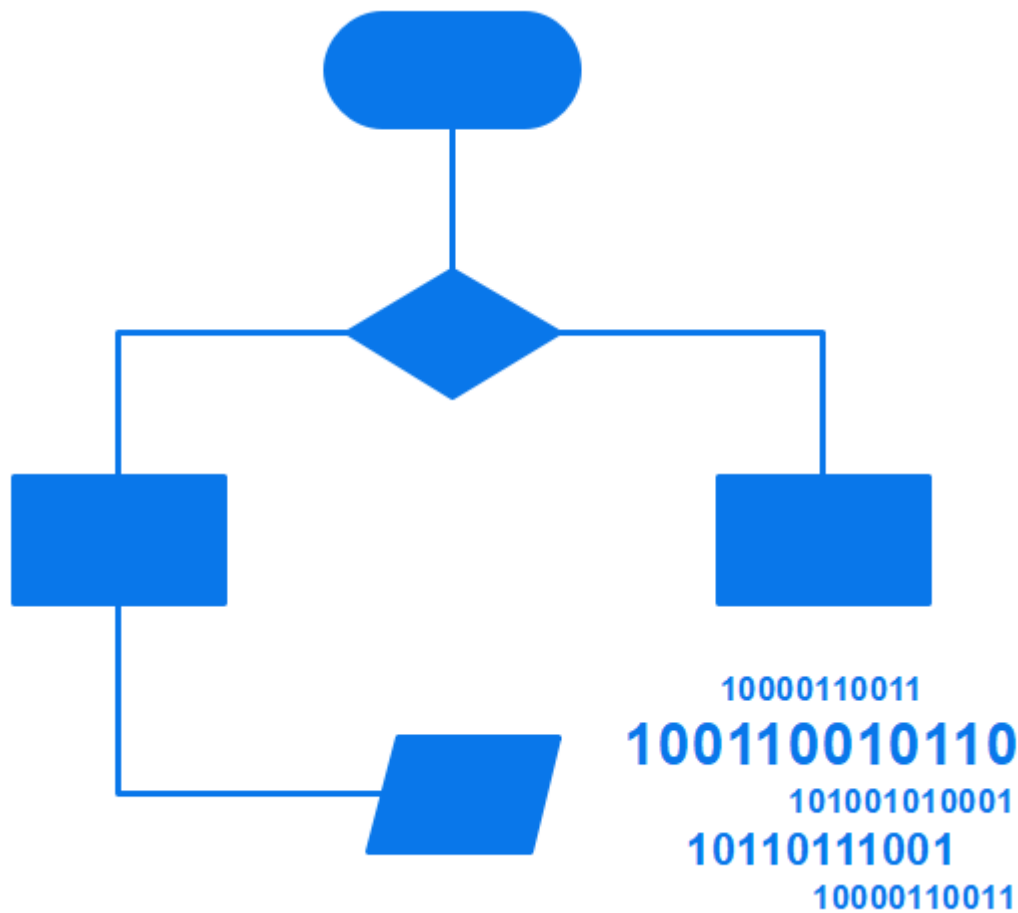
We can sort on the value returned by SQL expressions, not just columns. To do a multiple column sort, you specify the columns to sort on, separated by commas. SQL will sort on the columns in the order you give them. Default sort order is ascending. We can make it descending using DESC keyword.

- SELECT columns FROM table\_name **ORDER BY** specifiers DESC;
- The keyword for ascending sort is ASC. However since it is default you don't have to use it.
- SELECT columns FROM table\_name **ORDER BY** specifiers;
- We can use **CASE along with ORDER BY** to specify custom sort order.
- SELECT \* FROM Employees ORDER BY CASE id
- WHEN id<1600 THEN 1
- WHEN id<1700 THEN 2
- END, lastname;
- 
- 11. An **SQL Expression** is a term that SQL can evaluate to return a value. Expressions include:
  - a. Function calls – These are calls to built-in function.
  - b. System values – These include the time, the database user etc.
  - c. Numeric or string operators – These include + and – for numeric values and so on.
- 
- 12. Few **built-in functions in SQL** are:
  - a. SORT() – taking square roots.
  - b. LOWER() – returns the lower case.
  - c. ABS() – returns the absolute value.
  - d. YEAR() – extract year from date.
  - e. CONCAT() – joins or concatenates text strings.
  - f. MAX() – return a column with largest value.
  - g. AVG()
  - h. COUNT()
  - i. SUM()
- 
- 13. AVG(), COUNT(), MAX(), MIN(), SUM() etc. are called **aggregate functions**. **NULL values are ignored** by these functions.
- 
- 14. **System values** are legal to use in place of column names or with function calls in SELECT statements. Few built in system values are:
  - a. USER (Current SQL username.)
  - b. SESSION\_USER (Current SQL session user.)
  - c. SYSTEM\_USER (Current operating system user.)
  - d. CURRENT\_DATE
  - e. CURRENT\_TIME
  - f. CURRENT\_TIMESTAMP
- 
- 15. You can use **operators** to concatenate text strings, but this varies by DBMS. For Example oracle uses a '||' operator to concatenate text strings while SQL server uses '+'.
  -

- 16. We can create groups with the **GROUP BY** statements. GROUP BY can be used with aggregate functions. For example we can find the number of employees in each department. Many DBMS do not allow grouping based on variable-length fields. You can specify multiple columns in which to group. SQL groups by the first column, then by the second column, then third etc.
- SELECT department, COUNT(\*) FROM Employees GROUP BY department;
- 
- 17. You use the **UPDATE** statement to change data after it is stored in the table, optionally using WHERE clause. You can also update data using fetched data from a query.
- a. UPDATE employee set id=id+1000;
- b. UPDATE employee set id=1001 WHERE id=1;
- 
- 18. You can use **DELETE** to delete rows from a table.
- DELETE from employee WHERE id=5;
- 
- *The **WHERE clause** lets you add further conditions on your query.*

Algorithm and flowchart are two types of tools to explain the process of a program. This page extends the differences between an algorithm and a flowchart, and how to create a flowchart to explain an algorithm in a visual way.

Algorithms and flowcharts are two different tools used for creating new programs, especially in computer programming. An algorithm is a step-by-step analysis of the process, while a flowchart explains the steps of a program in a graphical way.



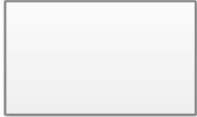


### **Definition of Algorithm**

To write a logical step-by-step method to solve the problem is called algorithm, in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step of the procedure. An algorithm includes calculations, reasoning and data processing. Algorithms can be presented by natural languages, pseudo code and flowcharts, etc.

### **Definition of Flowchart**

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes and arrows in order to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of a flowchart is to analyze different processes. Several standard symbols are applied in a flowchart:

Terminal Box - Start / End	
Input / Output	

Process / Instruction	
Decision	
Connector / Arrow	

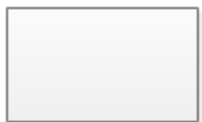
- Terminal Box - Start / End



- Input / Output



- Process / Instruction



- Decision



- Connector / Arrow



The symbols above represent different part of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight a certain element and the relationships between each part.

## Difference between Algorithm and Flowchart

If you compare a flowchart to a movie, then an algorithm is the story of that movie. In other words, an algorithm is the core of a flowchart. Actually, in the field of computer programming, there are many differences between algorithm and flowchart regarding various aspects, such as the accuracy, the way they display, and the way people feel about them. Below is a table illustrating the differences between them in detail.

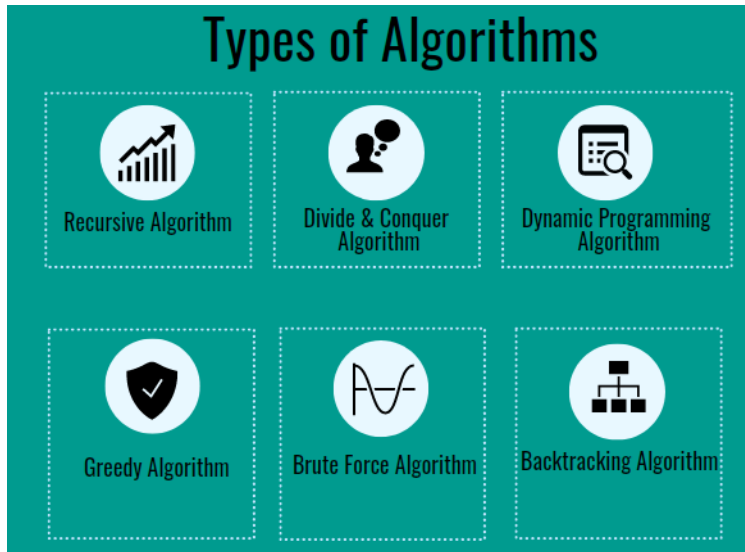
Algorithm	Flowchart
It is a procedure for solving problems.	It is a graphic representation of a process.
The process is shown in step-by-step instruction.	The process is shown in block-by-block information diagram.
It is complex and difficult to understand.	It is intuitive and easy to understand.
It is convenient to debug errors.	It is hard to debug errors.
The solution is showcased in natural language.	The solution is showcased in pictorial format.
It is somewhat easier to solve complex problem.	It is hard to solve complex problem.
It costs more time to create an algorithm.	It costs less time to create a flowchart.

## Types of Algorithm

It is not surprising that algorithms are widely used in computer programming. However, it can be applied to solving mathematical problems and even in everyday life. Here come a question: how many types of algorithms? According to Dr. Christoph Koutschan, a computer



scientist working at the Research Institute for Symbolic Computation (RISC) in Austria has surveyed about voting for the important types of algorithms. As a result, he has listed 32 important algorithms in computer science. Despite the complexity of algorithms, we can generally divide algorithms into 6 fundamental types based on their function.



## 1. Recursive Algorithm

It refers to a way to solve problems by repeatedly breaking down the problem into sub-problems of the same kind. The classic example of using recursive algorithm to solve problems is the Tower of Hanoi.

## 2. Divide and Conquer Algorithm

Traditionally, the divide and conquer algorithm consists of two parts: 1. breaking down a problem into some smaller independent sub-problems of the same type; 2. finding the final solution of the original problems after solving these smaller problems separately.

The key points of the divide and conquer algorithm are:

- If you can find the repeated sub-problems and the loop substructure of the original problem, you may easily turn the original problem into a small simple problem.
- Try to break down the whole solution into various steps (different steps need different solutions) to make the process easier.
- Are sub-problems easy to solve? If not, the original problem may cost lots of time.

## 3. Dynamic Programming Algorithm

Developed by Richard Bellman in the 1950s, the dynamic programming algorithm is generally used for optimization problems. In this type of algorithm, past results are collected for future use. Similar to the divide and conquer algorithm, a dynamic programming

algorithm simplifies a complex problem by breaking it down into some simple sub-problems. However, the biggest difference between them is that the latter requires overlapping sub-problems, while the former doesn't need to.

#### **4. Greedy Algorithm**

This is another way of solving optimization problems – greedy algorithm. It refers to always finding the best solution in every step instead of considering the overall optimality. That is to say, what he has done is just at a local optimum. Due to the limitations of the greedy algorithm, it has to be noted that the key to choosing a greedy algorithm is whether to consider any consequences in the future.

#### **5. Brute Force Algorithm**

The brute force algorithm is a simple and straightforward solution to the problem, normally based on the description of the problem and the definition of the concept involved. You can also use "just do it!" to describe the strategy of brute force. In short, a brute force algorithm is considered as one of the simplest algorithms, which iterates all possibilities and ends up with a satisfactory solution.

#### **6. Backtracking Algorithm**

Based on a depth-first recursive search, the backtracking algorithm focusing on finding the solution to the problem during the enumeration-like searching process. When it cannot satisfy the condition, it will return "backtracking" and tries another path. It is suitable for solving large and complicated problems, which gains the reputation of the "general solution method". One of the most famous backtracking algorithm example is the eight queens puzzle.

### **How to Use Flowcharts to Represent Algorithms**

Now that we have the definitions of algorithm and flowchart, how do we use a flowchart to represent an algorithm? In order to create an algorithm, we need to download a software.

Algorithms are mainly used for mathematical and computer programs, whilst flowcharts can be used to describe all sorts of processes: business, educational, personal and of course algorithms. So flowcharts are often used as a program planning tool to visually organize the step-by-step process of a program. Here are some examples:

#### ***Example 1: Print 1 to 20:***

**Algorithm:**

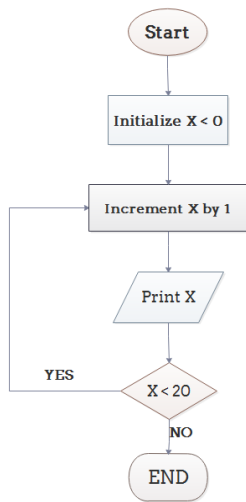
Step 1: Initialize X as 0,

Step 2: Increment X by 1,

Step 3: Print X,

Step 4: If X is less than 20 then go back to step 2.

### Flowchart:



### ***Example 2: Convert Temperature from Fahrenheit (°F) to Celsius (°C)***

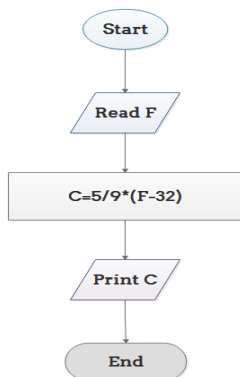
#### Algorithm:

Step 1: Read temperature in Fahrenheit,

Step 2: Calculate temperature with formula  $C = 5/9 * (F - 32)$ ,

Step 3: Print C,

### Flowchart:

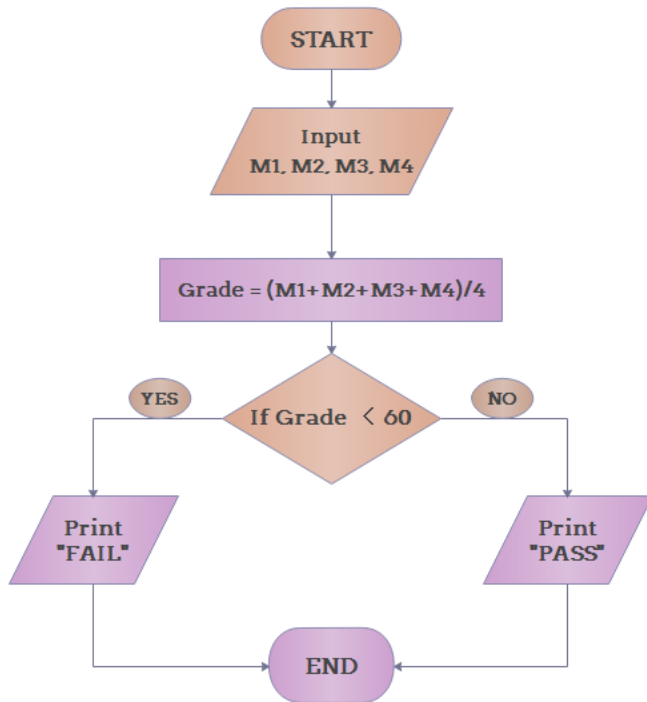


### ***Example 3: Determine Whether A Student Passed the Exam or Not:***

#### **Algorithm:**

- Step 1: Input grades of 4 courses M1, M2, M3 and M4,
- Step 2: Calculate the average grade with formula " $\text{Grade} = (\text{M1} + \text{M2} + \text{M3} + \text{M4}) / 4$ "
- Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".

#### **Flowchart:**



#### **Conclusion**

From the above we can come to a conclusion that a flowchart is pictorial representation of an algorithm, an algorithm can be expressed and analyzed through a flowchart.

An algorithm shows you every step of reaching the final solution, while a flowchart shows you how to carry out the process by connecting each step. An algorithm uses mainly words to describe the steps while a flowchart uses the help of symbols, shapes and arrows to make the process more logical.