ACDS Lecture Series

Lecture - 1

CSIR

Programming Languages

G. N. Sastry & Team

ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

1. Programming Languages

Introduction

Program

- A sequence of instructions to perform some specific task using a computer.

Programming

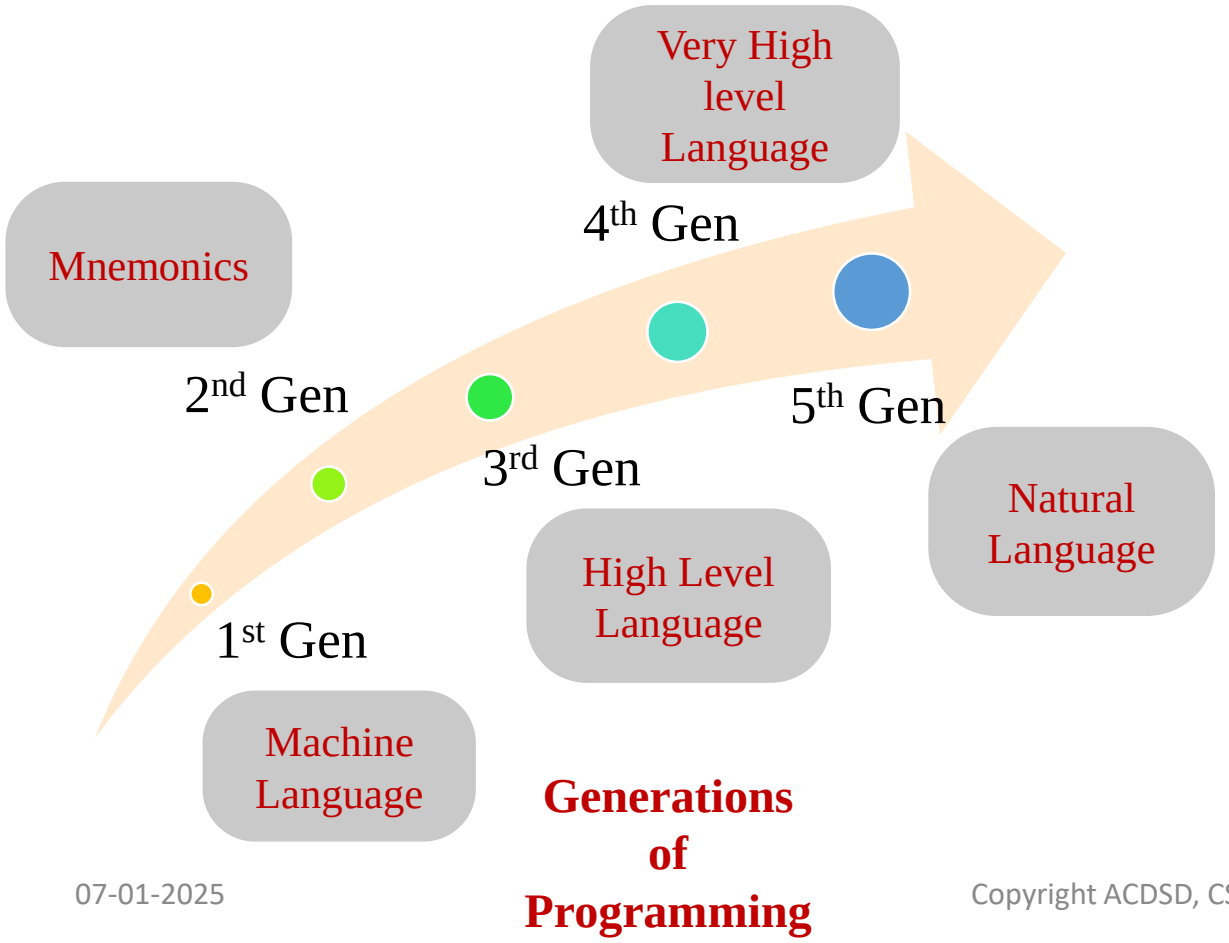
- Programming language is a notational system for describing computation in a machine-readable and human-readable form.

History

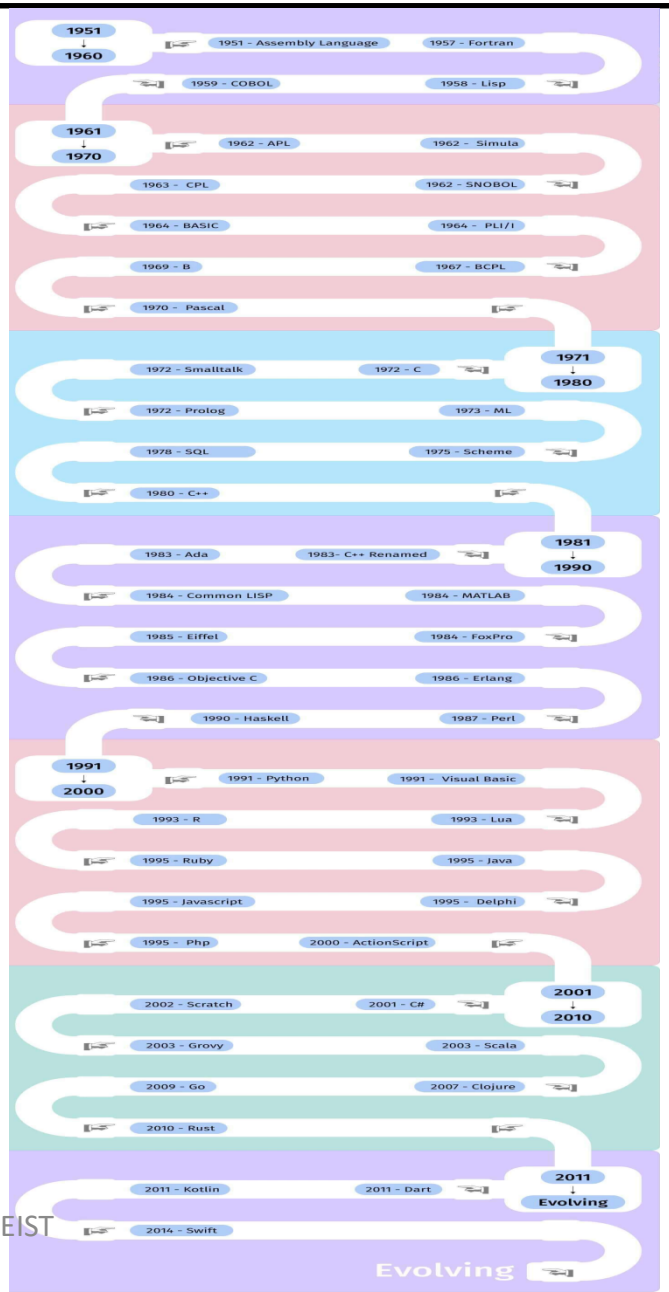
- The first computer programming language was created in **1883**, by a woman named **Ada Lovelace**.

Why Programming Languages?

Vocabulary and a set of rules for
"instructing a computer or computing device to perform some specific tasks"



Generations of Programming



Evolution of Programming Languages

Levels of Programming Languages


```
class acds
        {
            int a, b, c;
            float x, y;
                . . . .
            return(0);
         }
```

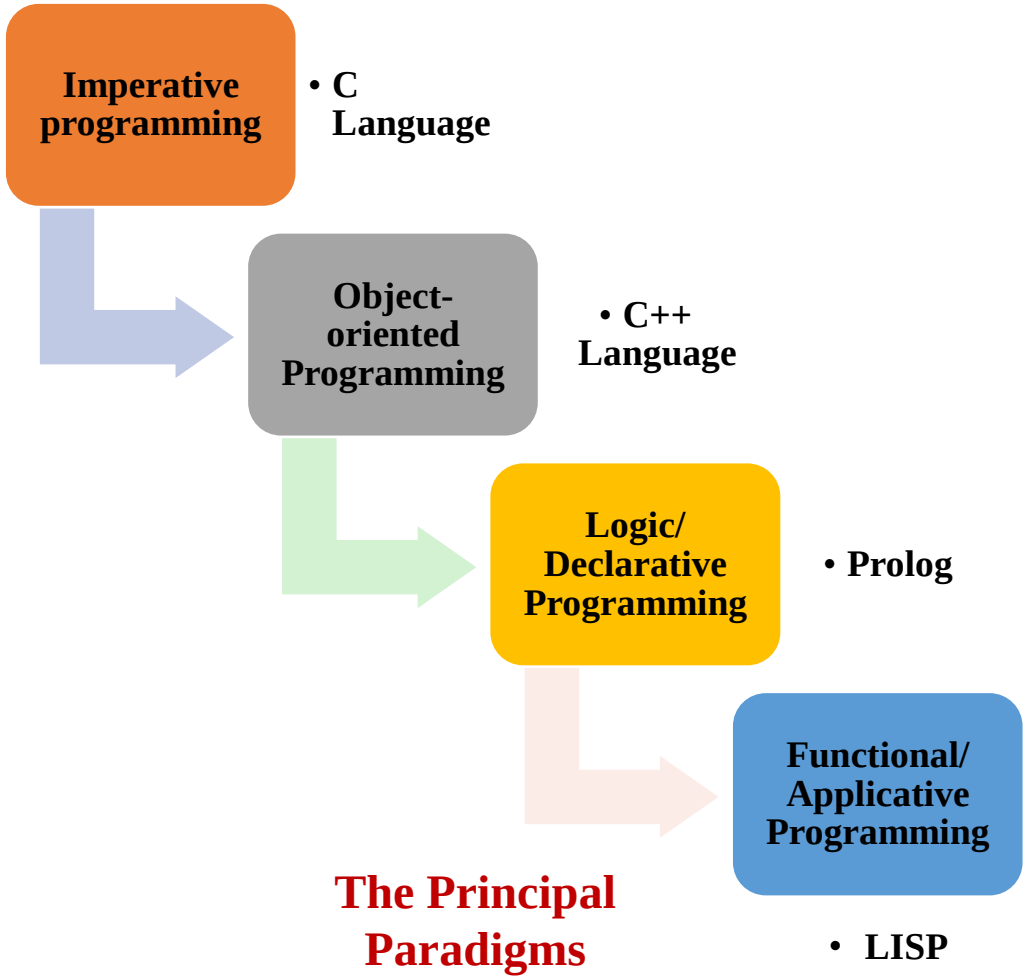
High-level programming

```
LOAD r1, b
LOAD r2, h
MUL r1, r2
ADD r1, r2
DIV r2, #2
RET
```

Low-level programming

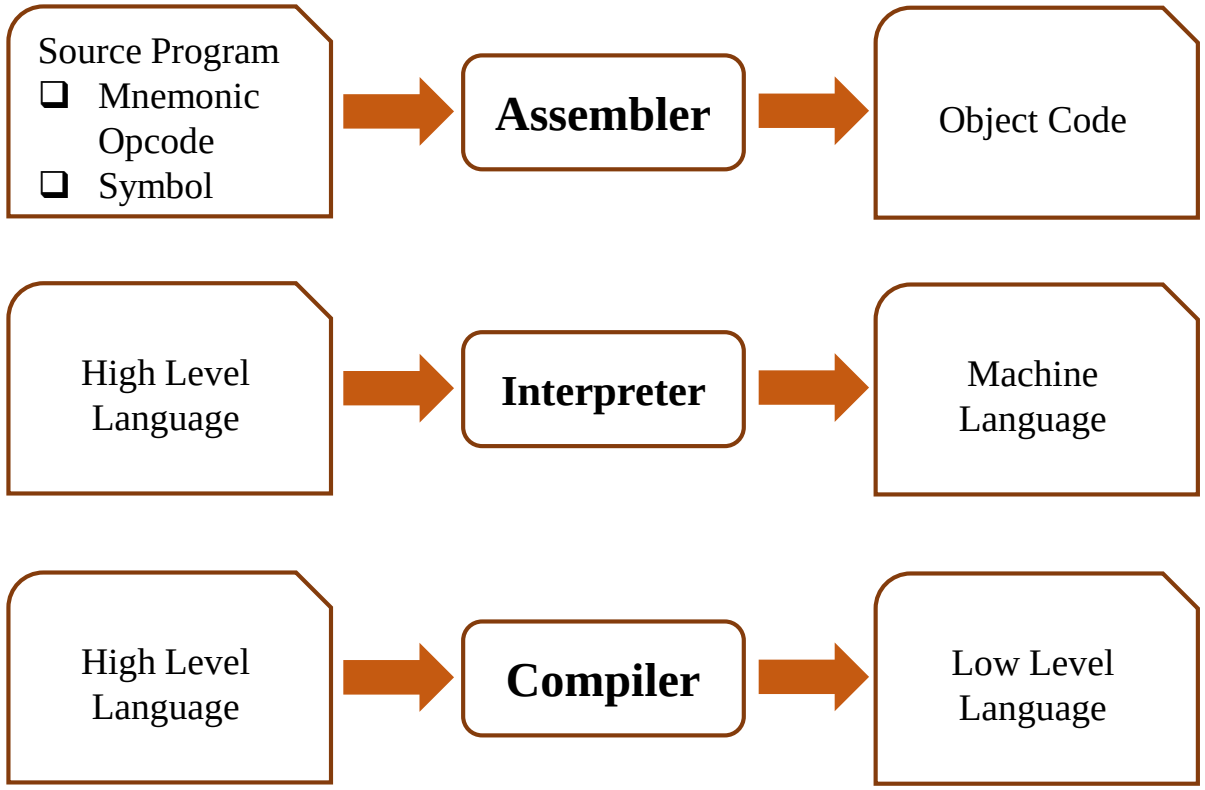
```
000101010111101010101010
101010101011101100101010
    101010101010101
……
……0101010101
010101010101010101000
```

Executable Machine Code



Language Translators

Translator itself is a program which performs translation from one computer language to another without changing the functional and logical structure of original code



Language Translators

```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS    #STACK

                        ***************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013           RESETA   EQU    %00010011
0011           CTLREG   EQU    %00010001

C003 86 13     INITA    LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR

                        ***************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH     LDA A  ACIA      GET STATUS
C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC    INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A  ACIA+1    GET CHAR
C019 84 7F              AND A  #$7F      MASK PARITY
C01B 7E C0 79           JMP    OUTCH     ECHO & RTS

                        ***************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input

C01E 8D F0     INHEX    BSR    INCH      GET A CHAR
C020 81 30              CMP A  #'0       ZERO
C022 2B 11              BMI    HEXERR    NOT HEX
C024 81 39              CMP A  #'9       NINE
C026 2F 0A              BLE    HEXRTS    GOOD HEX
C028 81 41              CMP A  #'A
C02A 2B 09              BMI    HEXERR    NOT HEX
C02C 81 46              CMP A  #'F
C02E 2E 05              BGT    HEXERR
C030 80 07              SUB A  #7        FIX A-F
C032 84 0F     HEXRTS   AND A  #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF  HEXERR   JMP    CTRL      RETURN TO CONTROL LOOP
```

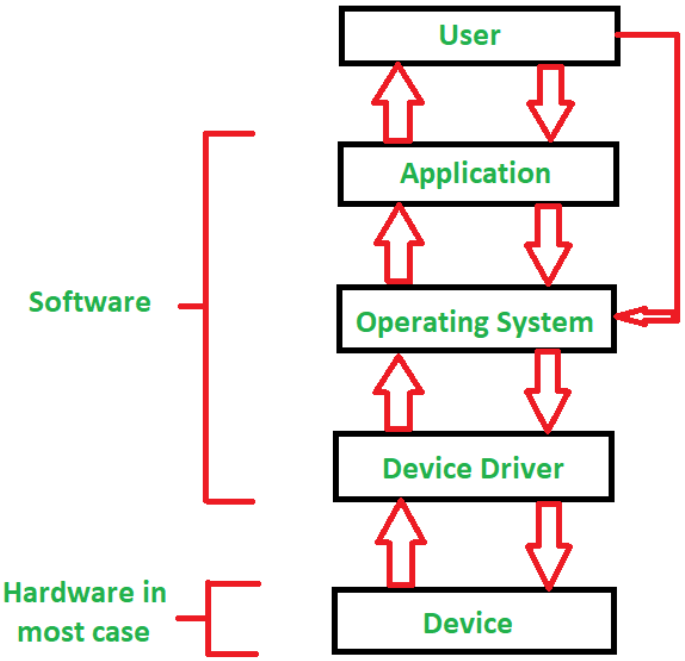
Platform for Programming

The **Operating System** provides the platform for programming using a computer.

Some of the operating systems which are widely used for programming:

- Linux
- Windows
- MacOS
- MSDos
- iOS . . . .

However most of the programming languages are platform independent meaning, they support all types of operating systems.



Linux is most preferred operating system for complex programming because:

- ❑ It is virus free, (The .exe files not accessible in Linux)
- ❑ It is reliable
- ❑ It is available as open source
- ❑ It has a strong terminal with many libraries
- ❑ Reliable and smooth for a larger period

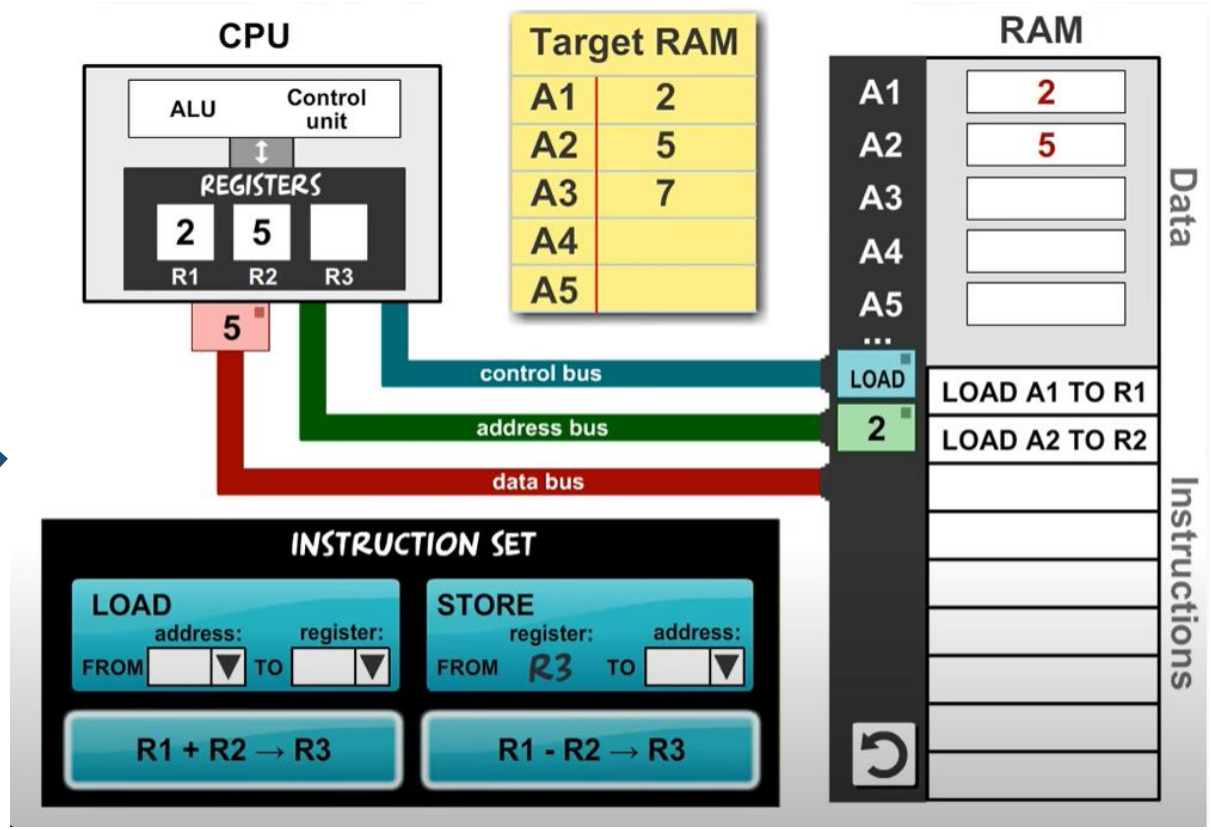
Programs to Hardware

High Level Program

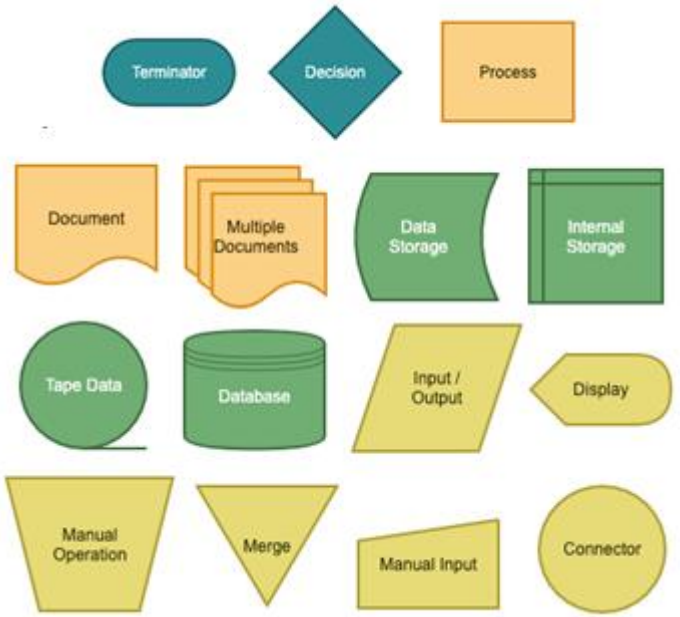
```
#include <stdio.h>
int main()
{
   int A, B, Sum;
   printf("Enter two integers: ");
   scanf("%d %d", &A, &B);
   Sum = A + B;
   printf("%d + %d = %d", A, B, Sum);
   return 0;
}
```

Low Level Program

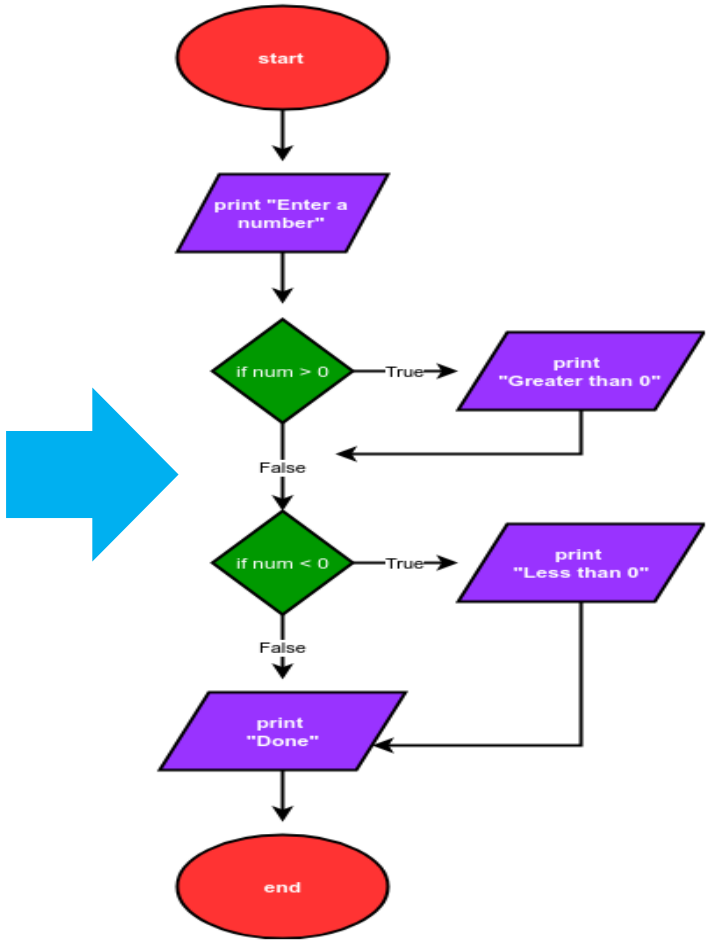
```
2000    LDA 2050
2003    MOV H, A
2004    LDA 2051
2007    ADD H
2008    MOV L, A
2009    MVI A 00
200B    ADC A
200C    MOV H, A
200D    SHLD 3050
2010    HLT
```



Flowchart and Program



Flowchart Symbols



Flowchart of the program to identify a number

```
#include <iostream>
int main()
   {
   int number;
   cout << "Enter a number";
   cin >> number;
   if (number > 0)
     {
         cout << "Greater than zero";
     }
   else if( number < 0)
     {
         cout << "Less than 0";
     }
   count<< "Done";
   return 0;
}
```

Transformation from flowchart to program

Libraries and Packages

Libraries that are loaded into an executable at run-time are called:

- Dynamic Link Libraries (Windows)
- Shared Libraries (Unix and Linux)

Libraries that are bound at compile-time are called:

- Static Libraries (both Linux and Windows)

Static Libraries

- Static libraries are created with the ar tool:
  - ar options archive object-file, …
  - ar r libdemo.a file1.o file2.o file3.o
- Naming convention: lib<name>.a
- Some options: r replace or insert, d delete, t show contents ..
- Create example:
  - g++ -c file1.cpp file2.cpp file3.cpp
  - ar r libdemo.a file1.o file2.o file3.o
  - rm file1.o file2.o file3.o
- Using library
  - g++ -c main.cpp
  - g++ -o main main.olibdemo.a
- Alternate use when library is in standard place like /usr/lib
  - g++ -o main main.o -ldemo

Shared Libraries

- Shared libraries are created with g++:
  - g++ -shared –o lib<name>.so file1.o …
- Naming convention: lib<name>.so
- Object files must be position independent code:
  - g++ -c -fPIC -Wall file1.cpp …
- Creating shared library:
  - g++ -c -fPIC -Wall file1.cpp file2.cpp file3.cpp
  - g++ -shared –o libdemshare.so file1.o file2.o file3.o
- Using a shared library:
  - g++ -Wall –o main main.cpp libdemshare.so
  - LD_LIBRARY_PATH=. (lib in current directory)
  - ./main

Libraries and Packages

A package is a hierarchical file directory structure that defines a single application environment that consists of modules and sub packages and sub-sub packages, and so on

There are **2,00,000** packages in Python alone

10,000 packages in CRAN for R

These packages are exclusively designed for Machine Learning, natural language processing, bioinformatics, cheminformatics

Popular Python packages

Numpy, Seaborn, Pandas, pyplot, Matplotlib, sklearn,

Popular R packages

tidyR, ggplot2, dplyr, shiny, plotly, mlr3, xgboost

Packages for Chemoinformatics

Rdkit, ChemoPy, PyBioMed, pubmed, pychem, Stk, Mol2vec

Packages for Text Mining

nltk, KoRpus, wordcloud, tm, RISmed, Rweka, lsa, maxent..

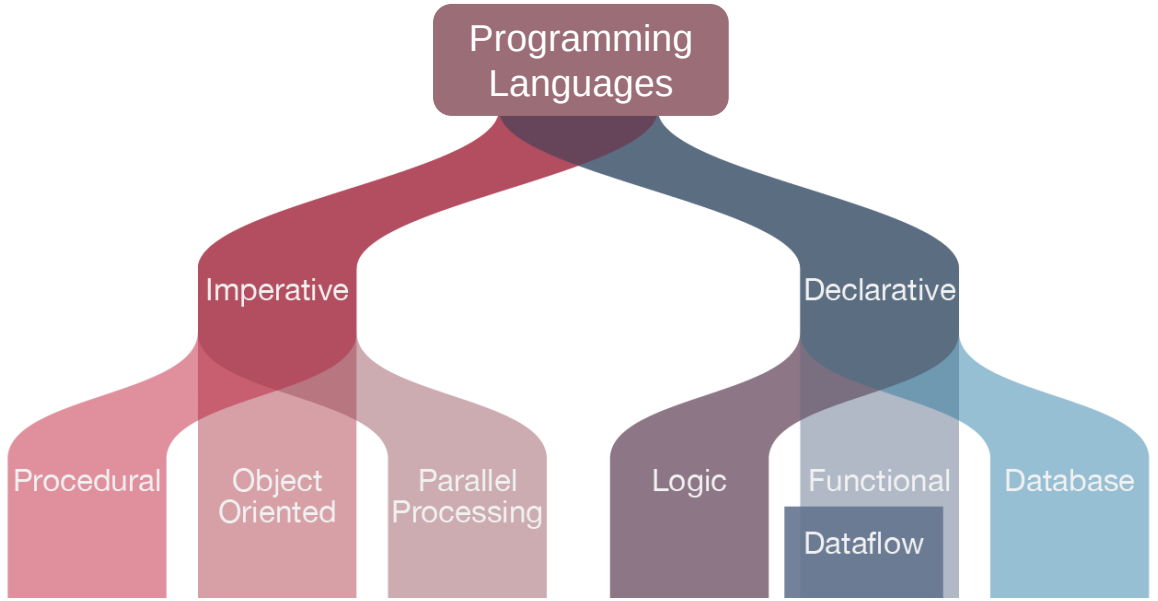
Importing packages in Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import
LogisticRegression
from sklearn.model_selection import
train_test_split
import os
```

Importing packages in R

```
install.packages('RISmed')
install.packages('tm')
library(RISmed)
library(tm)
library(naivebayes)
library(dplyr)
library(ggplot2)
library(psych)
```

Classifying Programming Languages



Procedural programming language is used to execute a sequence of statements which lead to a result

Functional programming language typically uses stored data, frequently avoiding loops in favor of recursive functions

One of the main principle of **object oriented** programming language is encapsulation that everything an object will need must be inside of the object

Database designing involves data definition (DD), data manipulation (DM) and data control (DC) languages at different stages of building

Logic languages let programmers make declarative statements and allow the machine to reason about the consequences of those statements

Scripting languages are often procedural and may comprise object-oriented language elements. They are normally not full-fledged programming languages

Procedural Programming

- A top-down approach of programming with limited code reuse.
- The programming is quite complex and global data gets more focus.
- The data is exposed through out the program making it less secure
- In procedural programming, the process is systematically broken into sub-process and so on
- Programming languages like C, Pascal, FORTRAN, BASIC are procedural in nature

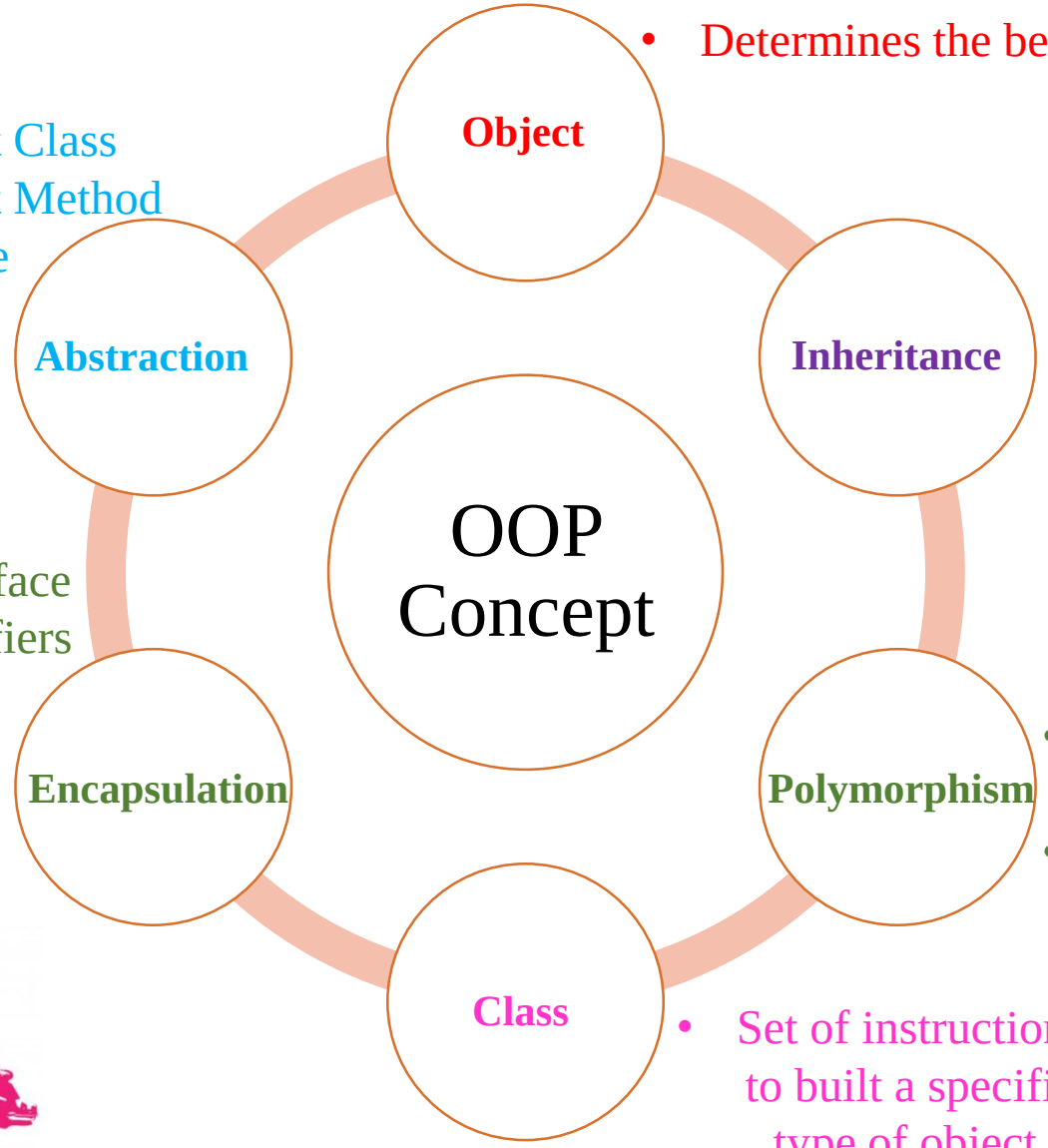
Procedural programming is difficult to relate to real world, hence object oriented programming evolved

```
#include<stdio.h>
void Fibonacci(int n)
     {
          static int n1=0,n2=1,n3;
          if(n>0)    {
              n3 = n1 + n2;
              n1 = n2;
              n2 = n3;
              printf("%d ",n3);
              Fibonacci(n-1);
    }
}
int main()     {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("Fibonacci Series: ");
    printf("%d %d ",0,1);
    Fibonacci(n-2);
 return 0;
 }
```

Function to compute Fibonacci series

Calling the function for execution

Concepts of Object Oriented Programming



- Determines the behavior of a class.

- Abstract Class
- Abstract Method
- Interface

1. Parent child concept.
2. Type of inheritance.
   - Single
   - Multi level

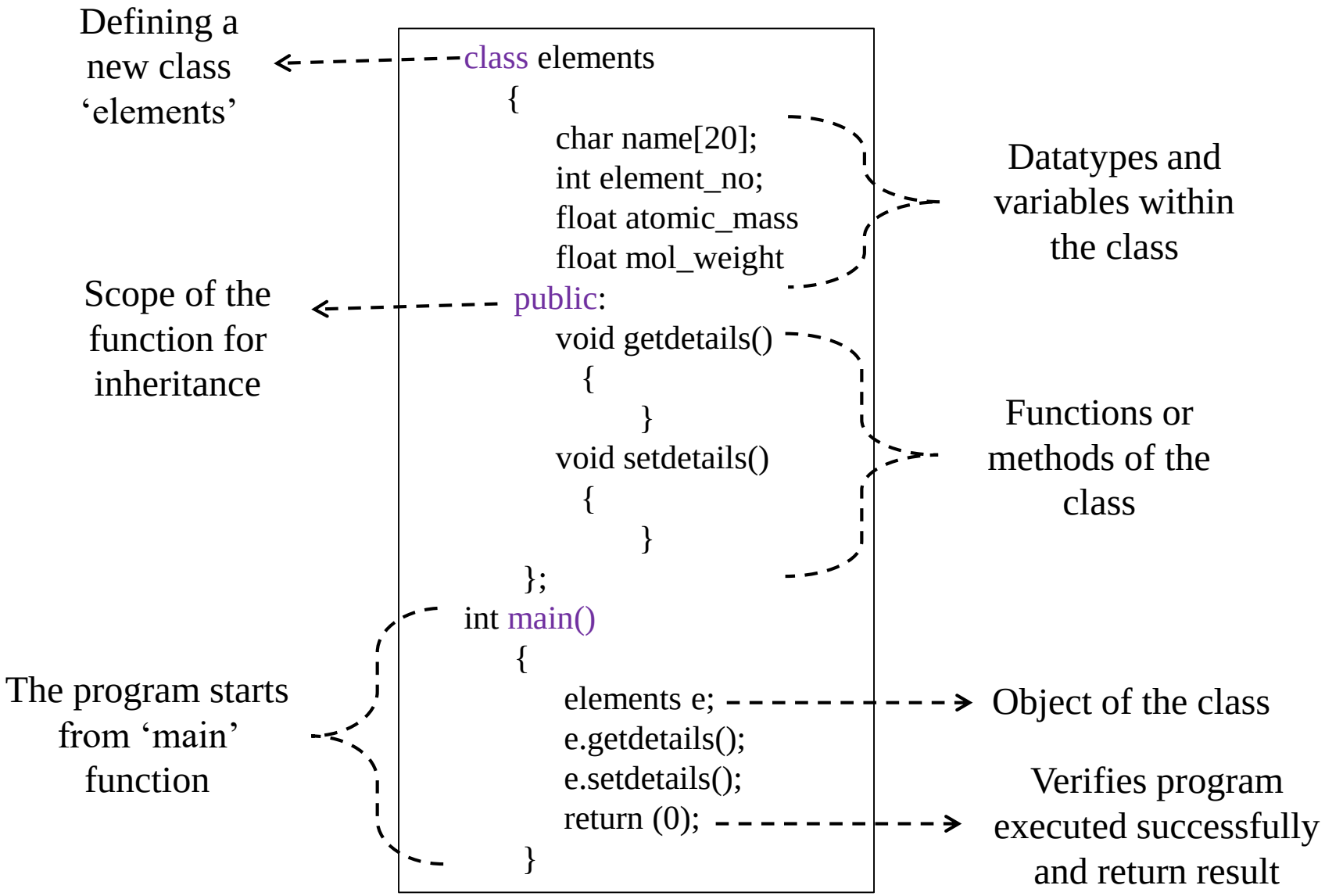
1. Class or interface
2. Access Modifiers
   - Default
   - Public
   - Private
   - Protected

- Compile time or method overloading.
- Runtime or method overriding.

- Set of instructions to built a specific type of object.

Object Oriented Programming Languages:
C++, Java, Python, R, PHP, Visual basic, Perl, Ruby . . .

Example of Object Oriented Program



Development Frameworks

A large number of development frameworks called "Integrated Development Environment (IDE)" are available for programming

- IDE provides faster setup, fast development, continual learning and standardization

Multi language IDEs

Eclipse, Netbeans, Anaconda, Komodo, Aptana, Geany, Anacoda, Code::Blocks, AWS Cloud9, JetBrains

Web Development IDEs

HomeSite, DreamWeaver FrontPage, PHP MySQL, Laravel, Django, Atom, PHPStorm, BlueFish

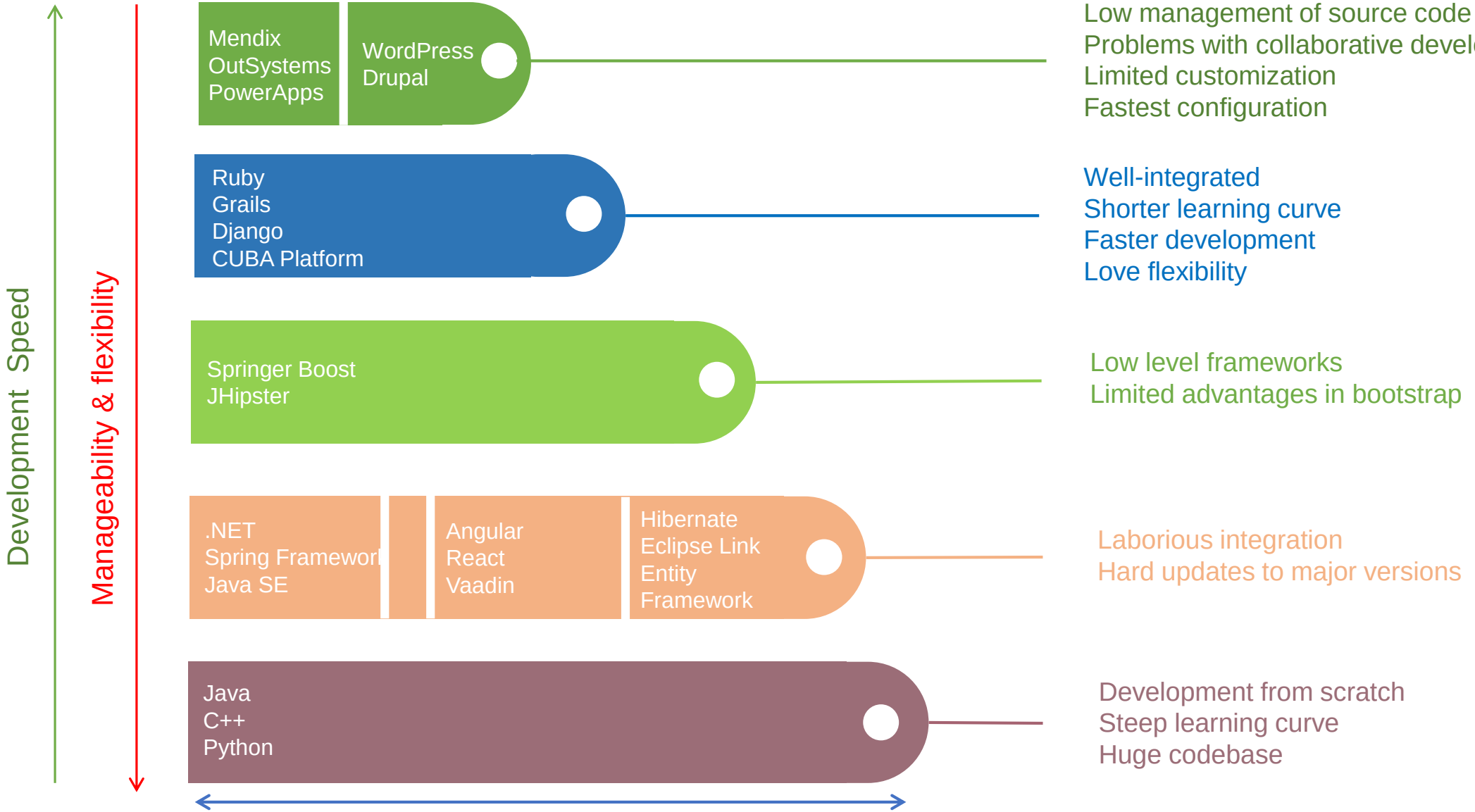
Specific IDEs

R Studio, Android Studio, Matlab, PyCharm, Spyder, Jupyter Notebook, Hadoop

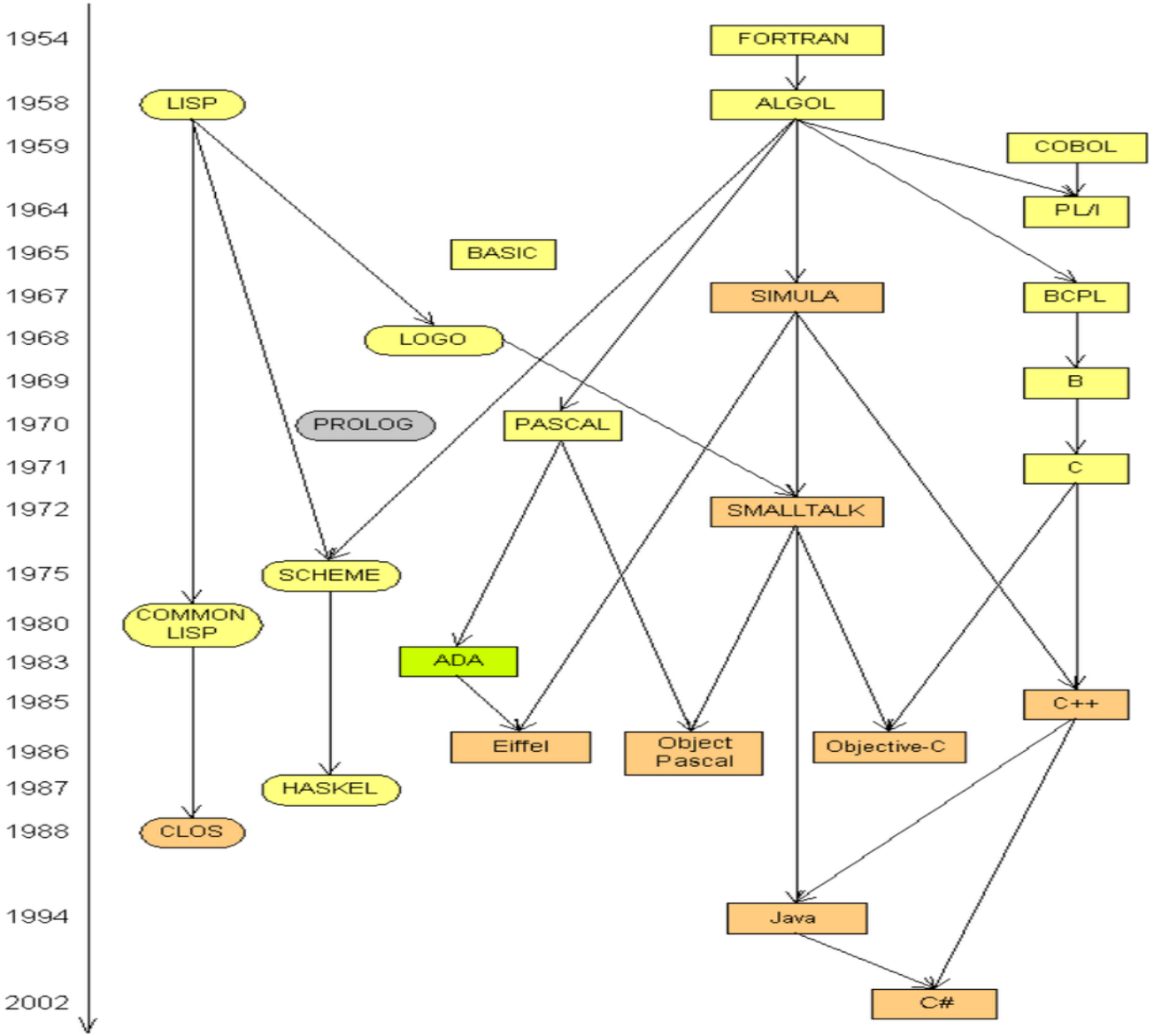
Cloud Based IDEs

XCode, Visual Studio, MonoDevelop, Coda, Expresso, CodePen , JSFiddle, Microsoft Azure Notebooks, Observable, Repl.it, Codenvy, Google Cloud Shell

Speed versus Flexibility for Frameworks



1.2 History and Evolution



https://www.researchgate.net/figure/history-of-high-level-programming-languages-evolution-form-1954-2002_fig3_267736864 [accessed 5 Dec, 2019]

1.3 Programming Language Generations

1st Generation

- Directly understood by Computer.
- Uses processor instruction.
- Processor dependent in binary form.

2nd Generation

- Processor dependent.
- Uses mnemonics to represent binary
- Easier to remember and read.

3rd Generation

- Structural high level programming language.
- Uses variables with sequences.
- Includes branches and loops.

4th Generation

- Domain-specific high level programming language.
- Computer-aided graphics.

5th Generation

- Uses AI techniques.
- Computer draws inferences from the code.

1.3 1st Generation

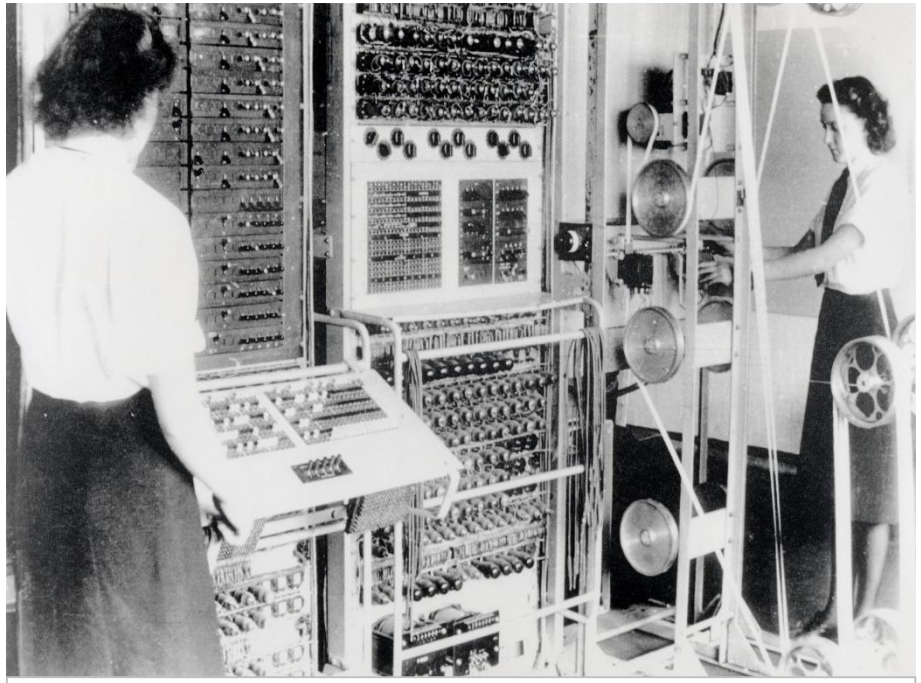
- This generation program is pure machine code, that is just ones and zeros e.g.00100101011111010110.
- Programmers have to design their code by hand then transfer it to a computer by using a punch card, punch tape or flicking switches.
- Benefits:

  Code can be fast and efficient.

  Code can make use of specific processor features such as special registers.
- Drawbacks:

  Code cannot be ported to other systems and has to be rewritten.

  Code is difficult to edit and update.
- Languages used: 1GL (Instructions based on 0s and 1s)



The Colossus Mark 2 was the world's first electronic digital programmable computer. Operators had to write the machine code directly by setting switches.

1.3 2nd Generation

- These programming languages are a way of describing Assembly code.
- The usage of these mnemonic codes such as LDA for load and STA for store means the code is easier to read and write.
- To convert an assembly code program into object code to run on a computer requires an Assembler.
- E.g.

Assembly code		Object code
LDA A		000100110100
ADD #5	-> Assembler<-	001000000101
STA A		001100110100

- Benefits:

  Codes can be fast and efficient.

  As it is closer to plain English, it is easier to read and write when compared to machine code.

- Drawback:
- Code cannot be ported to other systems and has to be rewritten.
- Languages used: COBOL (Common Business Oriented Languages)

1.3 3rd Generation(Hgh Level Language)

- Also known as imperative language- code is executed line by line , in a programmer defined sequence.
- For example:

```
1  dim x as integer
2  X = 3
3  dim Y as integer
4  Y = 5
5  X = X+Y
6  console. writeline(X)
```

Output: 8

- This generation language can be platform independent, that code written for one system will work on another.
- To convert a third generation program into object code requires a Compiler and Interpreter.
- However code produced might not make the best use of processor specific features unlike 1st and 2nd generation.
- Languages used: ALGOL, BASIC, C, JAVA, PASCAL

1.3 4th Generation

- Also known as declarative language- describe what computation should be performed and not how to perform it.
- These are designed to reduce programming effort and the time it takes to develop software, resulting in a cost of software development.
- Languages have been designed with a specific purpose in mind and this include –
  1. Languages to query databases(SQL)
  2. Languages to make reports (Oracle reports)
  3. Languages to control user interface (XUL)
- An example of a SQL to select student details from a database:

```
SELECT name, height, DoB FROM students WHERE height > 150;
```

- Languages used: C++, C#, JavaScript, Python, R, Ruby, Perl, FOCUS, SQL, PL/SQL, VHDL, Verilog

1.3 5th Generation

- 5GL is any programming language based on problem solving using constraints given to the program.
- These are designed to make the computer solve a given problem without the programmer.
- Fifth generation language are used mainly in artificial intelligence research.
- Languages used:

  AIML(Artificial Intelligence Markup Language) – used for creating natural language software agents.

  Prolog – is a logic programming language associated with artificial intelligence and computational linguistics.

  R – is a programming language and free software environments for statistical computing and graphics. It offers several paradigms of programming like vectorial computation, functional programming and object-oriented programming.

  OPS5 – Rule based or production system computer language generally used in expert systems

  Mercury – A purely declarative logic programming language related to Prolog and Haskell

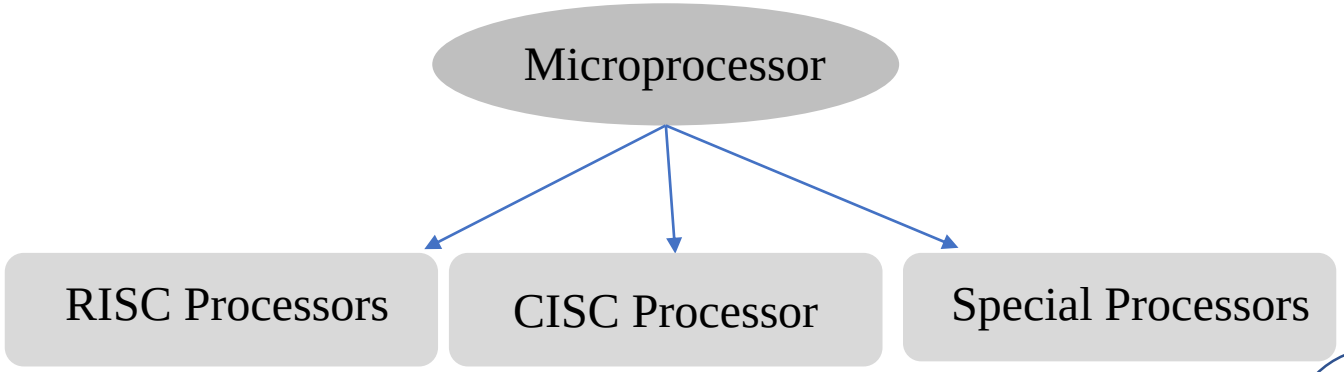
1.4 Low level Languages

- The main function of low level language is to interact with the hardware.
- Programs and applications written in a low-level level language are directly executable on the computing hardware without any interpretation or translation.

1) First Generation: Machine Language
2) Second Generation: Assembly Language

1.4.1 Microprocessor

- A microprocessor is a controlling unit of a micro-computer, produced on a small chip capable of performing Arithmetic logical Unit(ALU) operations and communicating with the other devices connected to it.
- It consists of an ALU, register array, and a control unit.
- A microprocessor can be classified into three categories-



Reduced Instruction Set Computer is designed to reduce the execution time by simplifying the instruction set of the computer.
E.g.-Power PC: 601, 604, 615, 620
DEC Alpha: 210642, 211066, 21068, 21164

Complex Instruction Set Computer is designed to minimize the no. of instructions per program.
E.g.- IBM 370/168
VAX 11/780
Intel 80486

Coprocessor: can handle its particular function many times faster than the ordinary microprocessor. e.g.- Math coprocessor.
Input/output Processor: having a local memory of its own, which is used to control I/O devices with minimum CPU involvement.
e.g. - Graphic display controller

1.4.2 Registers of 8085 Microprocessor

- A 8085 microprocessor is a second generation 8-bit and is the base for studying and using all the microprocessor available in the market.

Registers in 8085-

(a) General purpose Registers:- The 8085 has six general-purpose registers to store 8-bit data; these are identifies as- B,C,D,E,H and L.

- These can be combined as register pairs- BC, DE, and HL to perform some 16-bit operation.
- These registers are used to store or copy temporary data, by using instructions, during the execution of program.

(b) Specific Purpose Registers:-

Accumulator - It is a part of Arithmetic and Logical unit. After performing arithmatic or logical operations, the result is stored in accumulator.

Flag Register – It is completely different from other registers in microprocessor. It consists of 8 bits and only 5 of them are useful. The other three are left vacant and are used in the future Intel versions.

(c) **Memory Registers-** Program Counter, Stack Pointer

Write a program to exchange the data at 5000M & 6000M memory location.

LDA 5000M: "Getting the contents at 5000M location into accumulator"
MOV B, A : "Save the contents into B register"
LDA 6000M: "Getting the contents at 6000M location into accumulator"
STA 5000M : "Store the contents of accumulator at address 5000M"
MOV A, B : "Get the saved contents back into A register"
STA 6000M : "Store the contents of accumulator at address 6000M"

1.4.3 Machine Language

Machine Language

- Binary codes are used.
- Pros: program of machine language runs very fast.
- Cons: Machine Dependant
  Difficult to modify
  Difficult to program

Example:- A light bulb is controlled by a processor running a program in main memory. The controller can turn the light bulb fully on and fully of, can brighten or dim the bulb (but not beyond fully on or off). The machine instructions are one byte long, and corresponds to the following machine operations.

Machine Instruction	Machine Operation
00000000	Stop Program
00000001	Turn bulb fully on
00000010	Turn bulb fully off
00000100	Dim bulb by 10%
00001000	Brighten bulb by 10%
00010000	If bulb is fully on, skip over next instruction
00100000	If bulb is fully off, skip over next instruction
01000000	Go to start of program (address 0)

1.4.4 Assembly Language

Assembly Language

Symbolic/mnemonic codes are used

Pros :
- Easy to understand and use.
- Easier to locate and correct errors.
- Easy to modify.

Cons :
- Machine dependent
- Knowledge of hardware required
- Machine level coding

Example:-

Assembly code		Object code
LDA A ADD #5 STA A	-> Assembler<-	000100110100 001000000101 001100110100

1.5 Hgh Level Languages

- Easier to understand.
- These are basically symbolic languages that use English words and mathematical symbols rather than mnemonic codes.
- Needs a compiler or an interpreter to translate its code to machine language code.

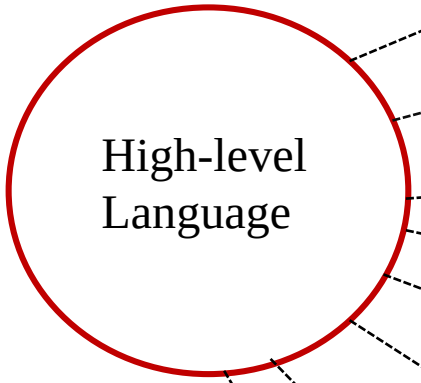
Pros

- Programmer friendly.
- Provide higher level abstraction from machine languages.
- Machine independent.
- Easy to learn.

Cons

- Takes additional translation time to translate the source to machine code.
- Comparatively slower than low level programs.
- Less memory efficient.
- Cannot communicate directly with the hardware.

1.5.1 Features



First Language developed
(1950s)Fortran

Each instructions create many machine instructions(one to many).

More portable between machines.

The binary created is called “machine code”.

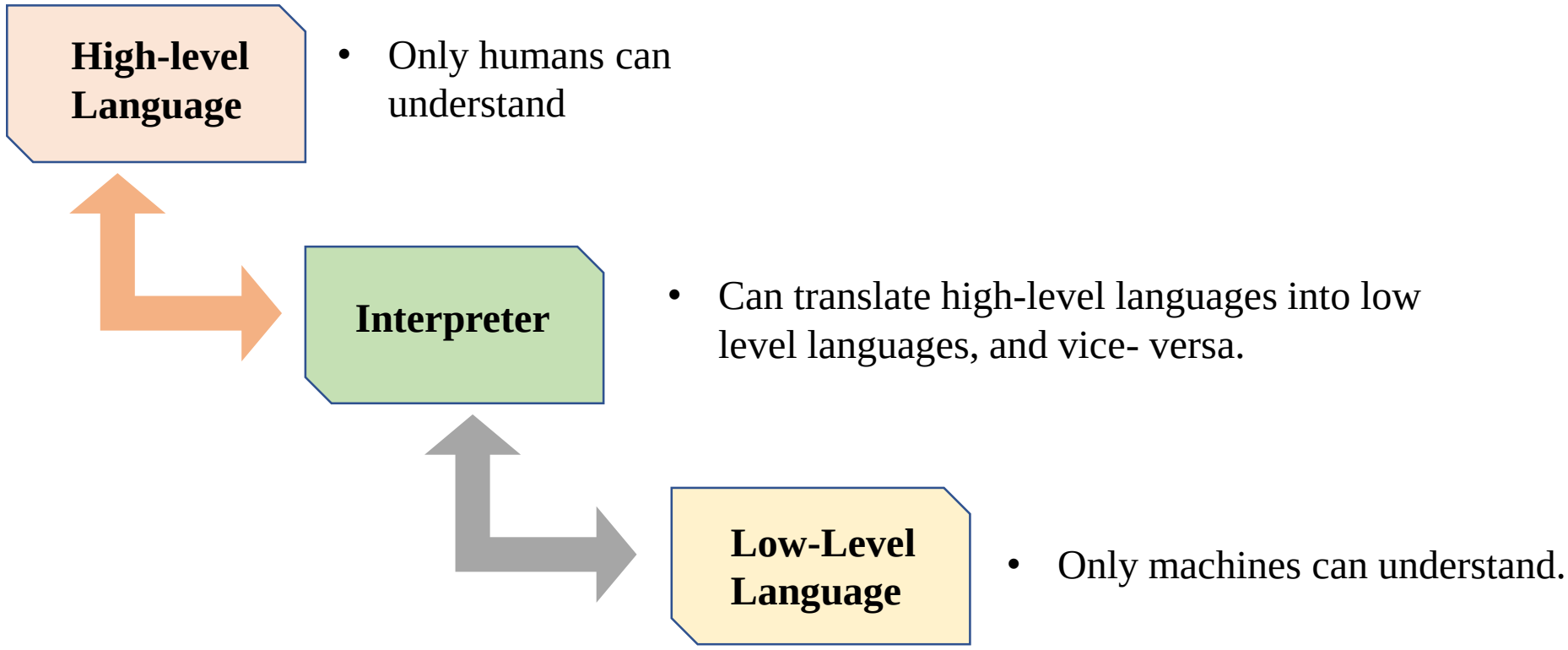
Programs are called source code.

Uses “keyword” to create a set of instructions.

Each language has its own set of keywords and syntax.

Requires a translator to execute the program

1.5.2 Flow of High Level Languages

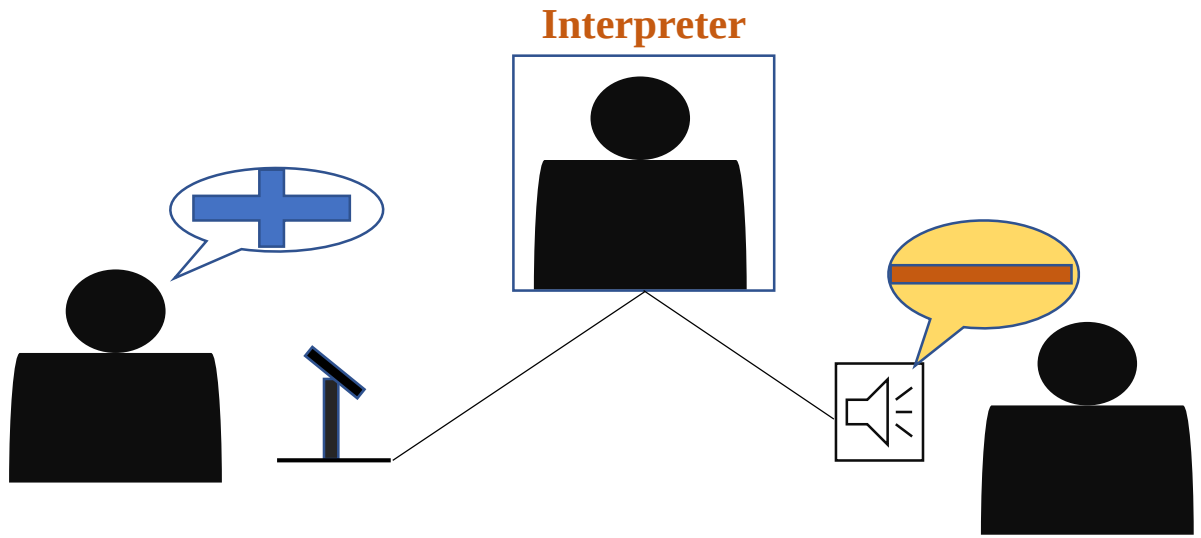


1.5.2 Assembler

- Program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.
- Translates assembly language to an object file or machine language format.
- Software that translates assembly language into machine language.

1.5.2 Interpreter

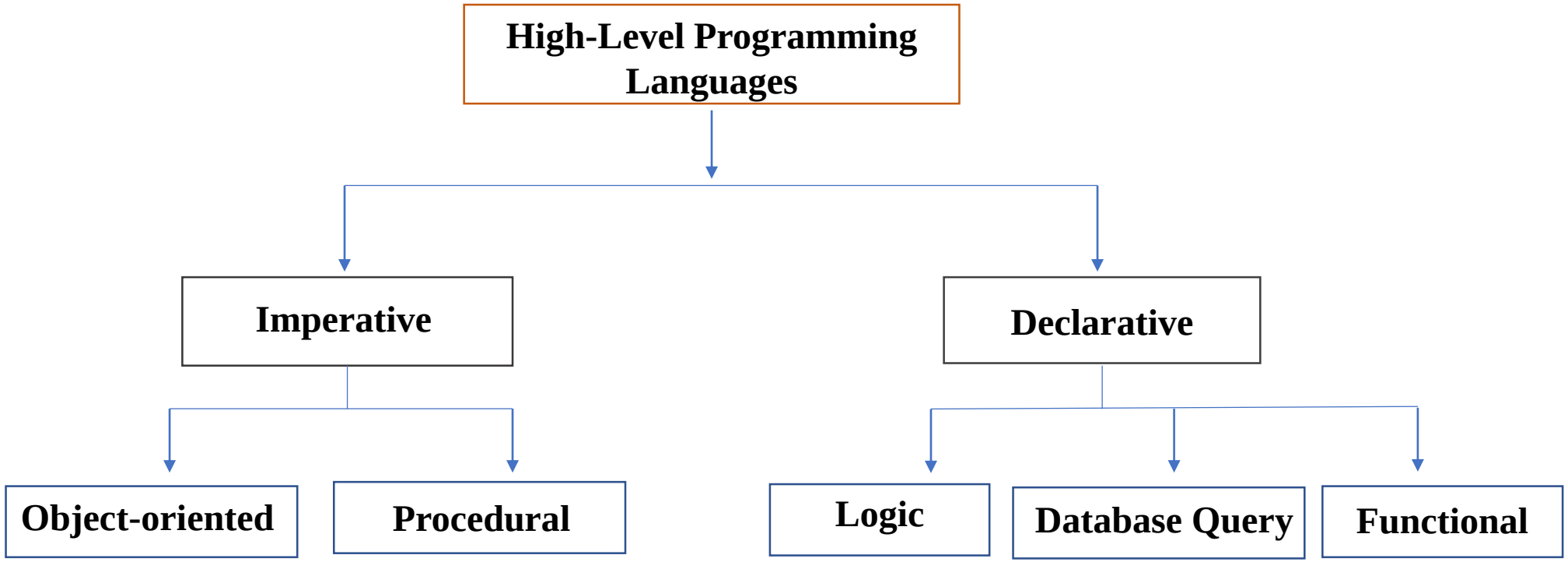
- Computer language processor that translates a program line- by- line and carries out the specified actions.
- An interpreter translates high level instructions into an intermediate form, which it then executes.
- In contrast a compiler translates high level instructions directly into machine language.



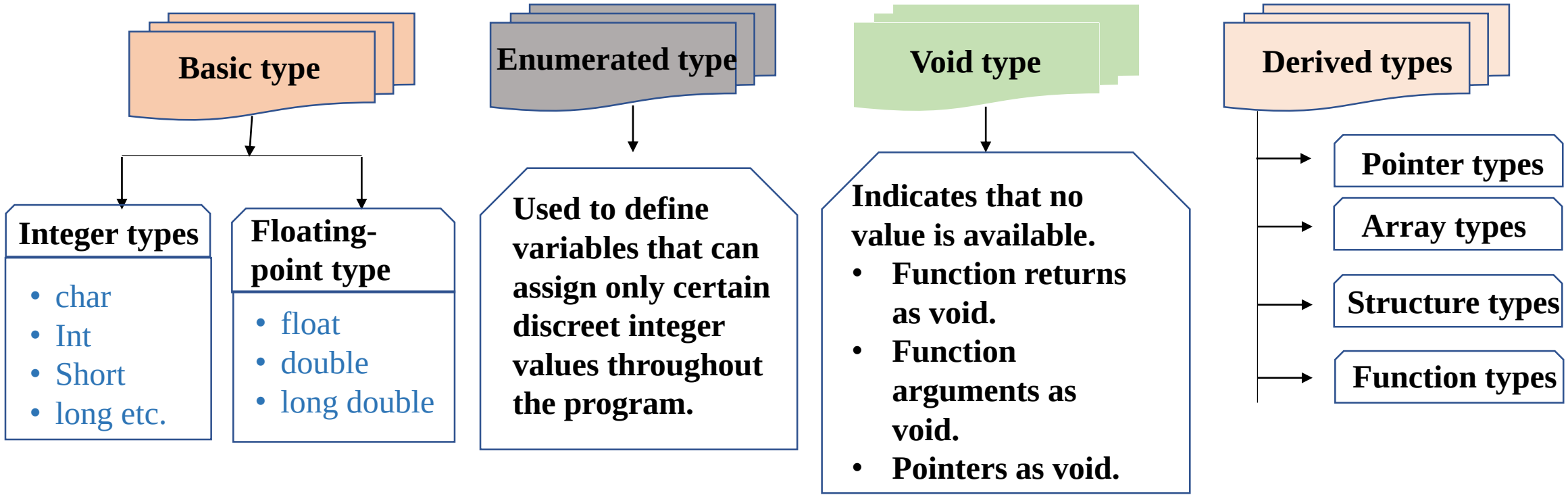
1.5.2 Compiler

- A special program that processes statements written in a particular programming language and turns them into machine language or “code” that a computer’s processor uses .
- Compiler is a program or set of programs that transforms source code written in a programming language(source language) into another computer language (target language) having a binary form known as object code.

1.5.3 Herarchy of Hgh-Level Languages



1.5.4 Data Types



1.5.5 Variables

- Variable is a name given to a storage area that programs can manipulate.
- The name of the variable can be composed of letters, digits and underscore character. It must begin with either letter or underscore.

Type	Description
char	• Stores single character and requires a single byte of memory in almost all compilers.
int	• An int variable is used to store an integer.
float	• Used to store decimal numbers(numbers with floating point value) with single precision.
double	• Used to store decimal numbers with double precision.
void	• Represents the absence of type.

1.5.6 Operators

- Operator is a symbol that tells the compiler to perform specific mathematical or logical functions.
- C language is rich in built-in operators.
  - Arithmetic Operators: +, −, *, /, % etc.
  - Relational Operators: ==, !=, <, >, >=, <=
  - Logical Operators: Logical AND(&&), Logical OR(||), Logical NOT (!)
  - Bitwise Operators: AND(&), OR(|),XOR(^),
  - Assignment Operators: =, +=, *=, /= etc.
  - Misc Operators: sizeof(), &, conditional expression(? :)

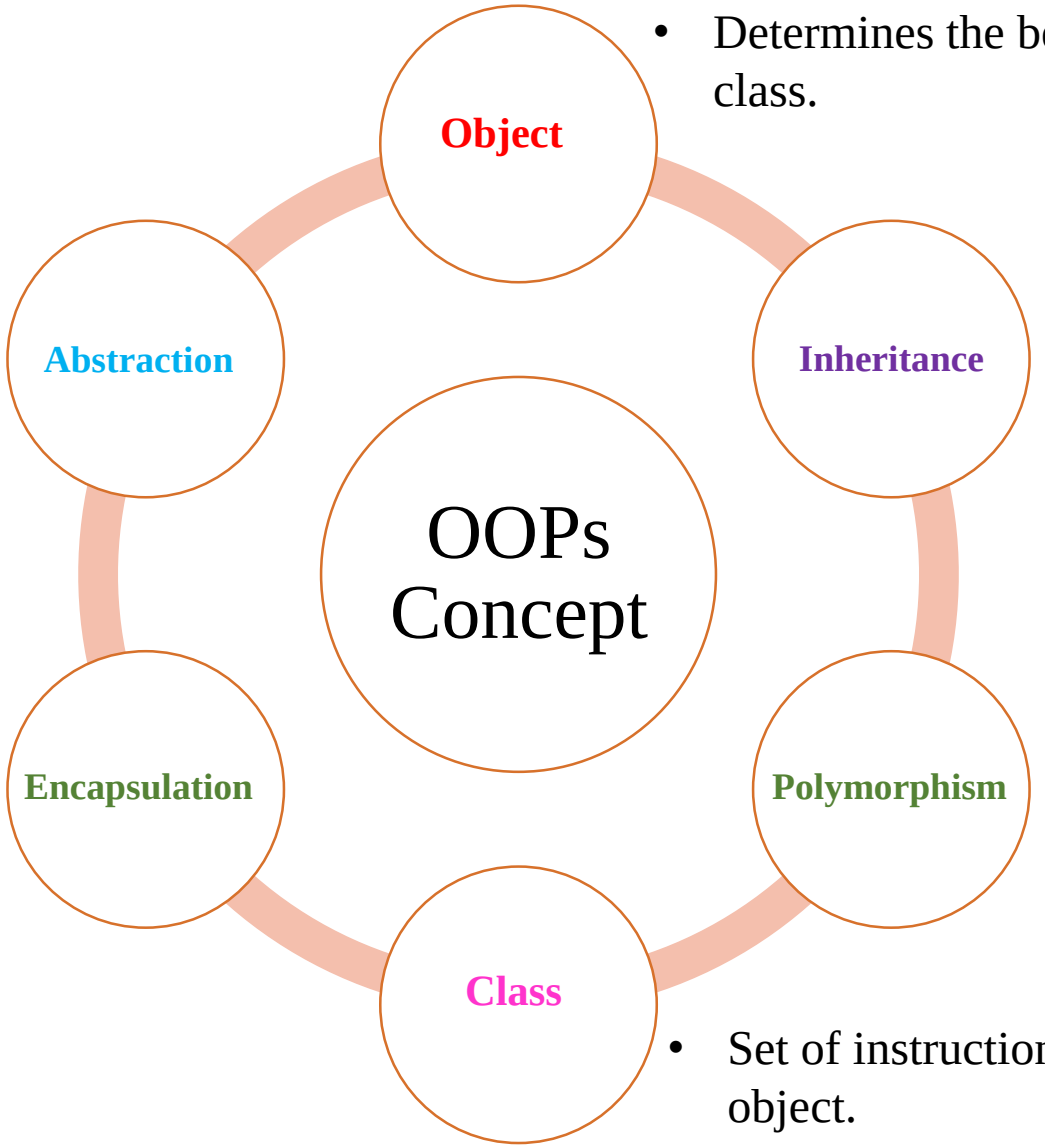
1.6 Object-Oriented Language

- A language that condenses data and instructions together into objects.
- These are organized around objects and data instead of actions and logic.
- Object-Oriented programming can be further grouped into classes.
- Example:- Java

  C++

  C#

  Python etc.



- Determines the behavior of a class.

1. Parent child concept.
2. Type of inheritance.
   - Single
   - Multi level

- Compile time or method overloading.
- Runtime or method overriding.

- Set of instructions to built a specific type of object.

1. Class or interface
2. Access Modifiers
   - Default
   - Public
   - Private
   - Protected

- Abstract Class
- Abstract Method
- Interface

1.7 C Language

Introduction

- General-purpose programming language.
- Machine independent structured programming language.
- Used to design system software.
- Used for mathematical evaluation.

History

- Originally implemented in assembly language on a PDP-7 by Dennis Ritchie and Ken Thompson.

Features

- Simple
- Rich library
- Memory management
- Fast speed

Example

❑ **Add two integer**

```
# include <stdio.h>
int main()
{
        int a, b, c ;
        print("Enter two integers:");
        scanf("%d %d, &a, &b);
        c = a + b ;
        print ("%d + %d = %d", a, b, c);
return 0;
}
```

1.7.1 C Program Structure

- Consists of the parts like:-
  - Preprocessor Commands
  - Functions
  - Variables
  - Statements and Expressions
  - Comments

Example

```
#include <stdio.h>
int main()
{
    /* My first program in C */
    printf("Dennis Ritchie \n");

    return 0;
 }
```

Output: Dennis Ritchie

1.7.2 Control Statements and Decision Making

- Sometimes, need to execute a block of code several number of times.
- Python provides the following kinds of loop to handle looping requirements -

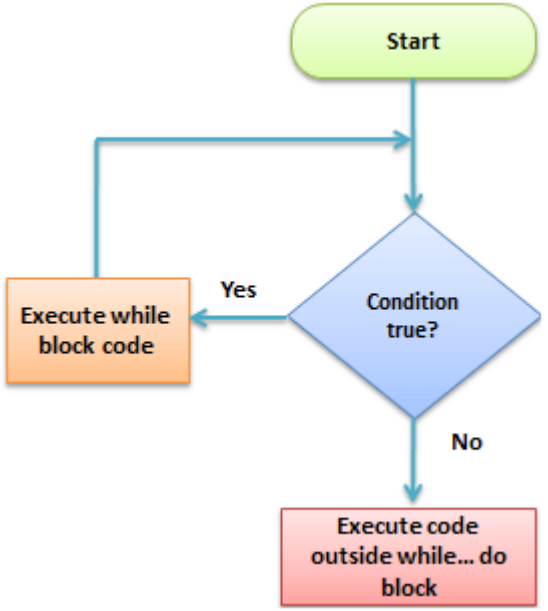
For loop: for i in range[k]:

#statements

While loop: while (test_expression)

#statements

do-while loop: do{ #statements }while (test_expression);



Loop Control Statements

Break Statement (*keyword: break****)***

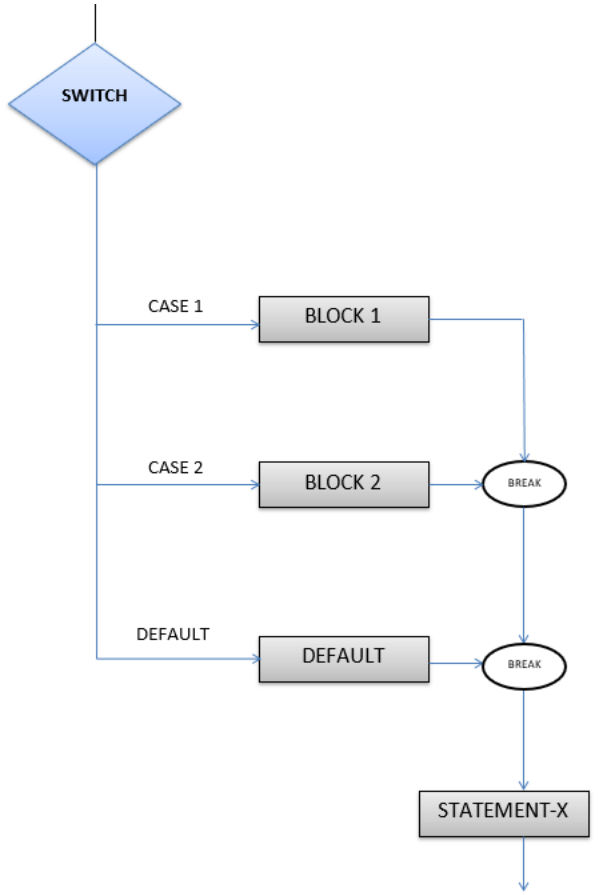
- Terminates loop statement and transfers execution to the statement immediately following the loop.

Continue Statement (*Keyword: continue****)***

- Causes loop to skip the remainder portion and immediately retest its condition prior to reiterating.

Switch statement: A switch statement is used for **multiple way selections** that will branch into different code segments based on the value of a variable or expression.

```
switch (expression)
    {
        case value1:
                code segment1;
                break;
        case value2:
                code segment2;
                break;
         .    .    .
        case valueN:
                code segmentN;
                break;
        default:
                default code segment;
    }
```



1.7.2 Control Statements and Decision Making

If statement:

```
if(boolean_expressions){
 # statement(s) will execute if the Boolean expressions are true
  }
```

If-else statement:

```
if(boolean_expressions){
        # statement(s) will execute if the boolean expression is true }
else {
        # statement(s) will execute if the boolean expression is false }
```

Nested If-else Statement:

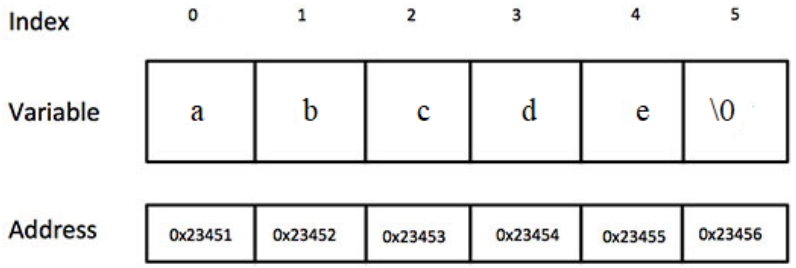
```
if(boolean_expression 1) {
        # Executes when the boolean expression 1 is true }
elseif( boolean_expression 2) {
        # Executes when the boolean expression 2 is true  }
else  {
        # executes when none of the above condition is true }
```

1.7.3 Arrays and String

- one-dimensional array of characters terminated by a **null** character '\0'
- a null-terminated string contains the characters that comprise the string followed by a **null**

Declaration of arrays

Syntax: char array_name [size];
Example: char str [5] = {'a', 'b', 'c', 'd', 'e'}



String Functions

Functions	Purpose
strcpy(s1, s2);	Copies string s2 into string s1.
strcat(s1, s2);	Concatenates string s2 onto the end of string s1.
strlen(s1);	Returns the length of string s1.
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
strchr(s1, ch);	Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1.

```
#include <stdio.h>
int main ()
  {
    char greeting[6] = {'a', 'b', 'c', 'd', 'e', '\0'};
    printf("Greeting message: %s\n", greeting );
    return 0;
  }
```

1.7.4 Structures and Unions

- **Structure** is another user defined data type available in C that allows to combine data items of different kinds. *'struct'* keyword is used to define a structure

Defining a structure

```
struct [structure tag]
{
    member definition;
    member definition;
        ...
    member definition;
 } [one or more structure variables];
```

To access any member of a ***structure***, the member access operator (.) is used.
This operator is coded as a period between the ***structure*** variable name and the ***structure*** member to access

- **Union** is a special data type available in C that allows to store different data types in the same memory location

Defining a union

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

To access any member of a ***union***, the member access operator (.) is used.
This operator is coded as a period between the ***union*** variable name and the ***union*** member to access

1.7.5 Functions

- A function is a group of statements that together perform a task
- Functions are used for reusability of codes which is an important property of OOP
- A function has three parts: *definition, declaration and calling*

Defining a function: The function definition is can be done anywhere in a program. Function definition consists of its return type, name, parameters and body.

Example:
```
int add(int a, int b)
{
    int c = a+b;
    return c;
}
```

Declaring the function: Declaration tells the compiler about a function name and how to call the function. Declaration of function needs return type, name and parameter list with a terminator (semi colon).

Calling the function: The calling of the function signifies it should start executing. There are two ways for calling a function based on the parameters passed to it. These are as follows:

Call by value – In this method the values which will be used in executing the function are directly set as parameter into the function. *Syntax:* add(5, 5);

Call by reference - In this method the values are not directly set to the function but there address is being passed as parameters with the "*" operator which is a pointer. *Syntax : add(*a, *b);* The values are accessed using "&" operator.

1.7.6 Examples

Program to generate Fibonacci Series

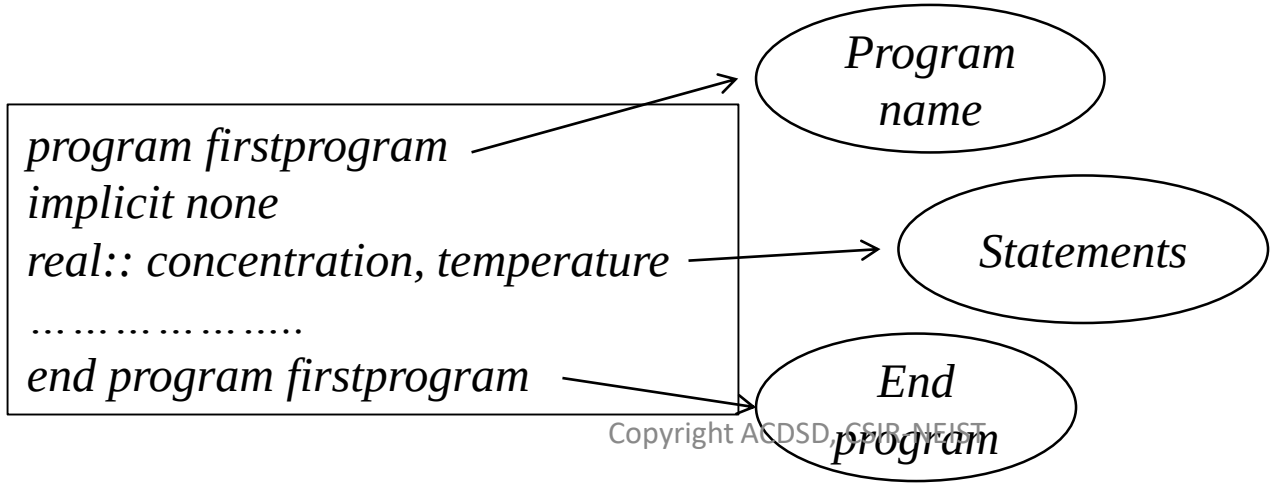
```
Include <stdio.h>
Int main ()
{
 Int count, first term = 0, second term = 1, next term, i;
        // Ask user to input number of terms
       Printf("First %d terms of Fibonacci series:\n",count);
       For ( i=0 ; i < count ; i++)
       {
         If (i <= 1)
              next term = i;
         else
      {
         next term = first term + second term;
         first term = second term;
         second term = next term;
      }
         returns 0;
}
```

1.8 FORTRAN

- Formula Translation (FORTRAN) is the first high level programming language
- Versions: FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003, Fortran 2008
- Fortran is the dominant programming language used in engineering applications

- Program organization in Fortran
  - *Program name:* Every Fortran program must begin with a program line, giving the name of the program
  - *Declarations:* Variables in Fortran are declared in different types like Integer, Real, Double precision, Character, Complex & Logical
  - *Statements:* It is recommended to start Fortran program with "implicit none" to treat every variable as integer by default.
  - *End:* Every program in Fortran ends with the "end" statement followed by program name.

Example:



Basic I/O Controls

Print: Instructs the compiler to print items to the screen
Syntax: print*, x

Read: Reads the user input
Syntax: print*,"enter the initial amount”
read*, amt

Advanced I/O Controls

Open: open(unit=2, file=‘initial.txt’, status=old)
open(unit=3,file=‘final.txt’, status=new)

Read: read(2,*) c0

Write: write(3,*) cf
or
write(3,10) cf
10 format(f10.4)
close(2,3)

1.8.2 Programming Units

A complete Fortran program is made up from a number of distinct program units. Program units are:

- main program
- function or subroutine subprogram
- module or submodule
- block data program unit

The main program and some procedure (function or subroutine) subprograms may be provided by a language other than Fortran.

These Fortran program units may be given be distinct files or within a single file.

Type of unit	Code	Explanation
Main program	PROGRAM	Essential; invokes the others.
Subroutine	SUBROUTINE	A type of sub-program.
Function, compiled	FUNCTION	A type of sub-program.
Function, intrinsic	*variable*	Supplied as standard.
Function, statement	*variable*	Similiar to a macro in C.
Common block	COMMON	For sharing declarations.
Block-data	BLOCK DATA	For sharing data.

1.8.2.1 External Procedures

An external procedure is one which is defined outside another program unit, or by a means other than Fortran. The function contained in a file like:

```
integer function f()
        implicit none
end function f
```

is an external function.

For external procedures, their existence may be declared by using an interface block (to given an explicit interface) or by a declaration statement to give an implicit interface or even the external attribute is not necessary. Some of the mentioned external features are as follows:

```
program prog
        implicit none
        interface
                integer function f()
        end interface
end program prog
```

```
program prog
        implicit none
        integer, external :: f
end program prog
```

```
program prog
        implicit none
        integer f
        external f
end program prog
```

```
program prog
        implicit none
        integer i
        integer f
        i = f() ! f is now an external function
end program prog
```

1.8.2.2 Internal Procedures

A program unit which is not an internal subprogram may contain other program units, called *internal subprograms*.

```
program prog
            implicit none
Contains
            function f()
            end function f
            subroutine g()
            end subroutine g
end program
```

Such an internal subprogram has a number of features:

- there is host association between entities in the subprogram and the outer program
- implicit typing rules are inherited (implicit none is in effect in f above)
- internal subprograms have an explicit interface available in the host

Module subprograms and external subprograms may have internal subprograms, such as

```
module mod
            implicit none
Contains
            function f()
            contains
                        subroutine s()
                        end subroutine s
            end function f
end module mod
```

1.8.2.3 Modules

A module is like a package where you can keep your functions and subroutines, in case you are writing a very big program, or your functions or subroutines can be used in more than one program. Modules provide you a way of splitting your programs between multiple files.

Modules are used for −

- Packaging subprograms, data and interface blocks.
- Defining global data that can be used by more than one routine.
- Declaring variables that can be made available within any routines you choose.
- Importing a module entirely, for use, into another program or subroutine.

```
module name
        [statement declarations]
        [contains [subroutine and function definitions] ]
end module [name]
```

1.8.2.4 Subroutines

In the main program, a subroutine is activated by using a CALL statement which include the subroutine name followed by the list of inputs to and outputs from the subroutine surrounded by parenthesis. The inputs and outputs are collectively called the arguments. A subroutine name follows the same rules as for function names and variable names: less than six letters and numbers and beginning with a letter. Because of this, subroutine names should be different than those used for variables or functions. As with functions, there are some rules for using subroutines. Keep these in mind when writing your subroutines:

- You do not need to declare the subroutine name in the main program as you do with a function name.
- They begin with a line that includes the word SUBROUTINE, the name of the subroutine, and the arguments for the subroutine.

```
PROGRAM SUBDEM
REAL A,B,C,SUM,SUMSQ
CALL INPUT( + A,B,C)
CALL CALC(A,B,C,SUM,SUMSQ)
CALL OUTPUT(SUM,SUMSQ)
END

SUBROUTINE
INPUT(X, Y, Z)
REAL X,Y,Z
PRINT *,'ENTER THREE NUMBERS => `
READ *,X,Y,Z
RETURN
END
```

1.8.2.5 Functions

The purpose of a function is to take in a number of values or arguments, do some calculations with those arguments and then return a single result. There are some functions which are written into FORTRAN and can be used without any special effort by you, the programmer. They are called intrinsic functions. There are over 40 intrinsic functions in FORTRAN and they are mainly concerned with mathematical functions. The general way to activate a function is to use the function name in an expression. The function name is followed by a list of inputs, also called arguments, enclosed in parenthesis:

```
answer = functionname (argumentl, argument2, . . .
```

Example:

PRINT*, ABS (T)	The compiler evaluates the absolute value of T and prints it out
Y = SIN (X) + 45	The compiler calculates the value of sin x, adds 45 then puts the result into the variable Y, where x is in radians.
M=MAX(a,b,c,d)	The compiler puts the maximum value of a, b, c and d into the variable M. If a=2, b=4, c=1 and d=7, then M would have a value of 7.
C=SQRT (a* * 2 +b* * 2)	The compiler evaluates the expression, a**2+b**2, sends that value to the SQRT function, and places the answer in the variable

1.8.2.6 Recursion

A completely new capability of Fortran is recursion. Note that it requires that you assign a new property RESULT to the output variable in the function declaration. This output variable is required inside the function as the "old" function name in order to store the value of the function. At the actual call of the function, both externally and internally, you use the outer or "old" function name. The user can therefore ignore the output variable. Here follows two examples: first the recursive calculation of factorials, then the recursive calculations of the Fibonacci-numbers.

```
RECURSIVE FUNCTION FACTORIAL(N) RESULT (FAC_RESULT)
IMPLICIT NONE
INTEGER, INTENT(IN) :: N
INTEGER :: FAC_RESULT
IF ( N <=1 ) THEN
        FAC_RESULT = 1
ELSE
        FAC_RESULT = N * FACTORIAL(N-1)
END IF
END FUNCTION FACTORIAL
```

```
RECURSIVE FUNCTION FACTORIAL(N) RESULT (FAC_RESULT)
IMPLICIT NONE
INTEGER, INTENT(IN) :: N
INTEGER :: FIBO_RESULT
IF ( N <=2 ) THEN
        FIBO_RESULT = 1
ELSE
        FIBO_RESULT = FIBONACCI(N-1) * FIBONACCI(N-2)
END IF
END FUNCTION FIBONACCI
```

1.8.3 Arrays and Array Operations

- Specify static dimensions in declaration:
  - real, dimension(10,3,5) :: x
  - integer, dimension(10) :: i
- Can also specify ranges of values
  - integer, dimension(3:11, -15:-2) :: ival, jval
- Access array elements using parenthesis
  - a = y(3) + y(4)
- Dynamic allocation
  - Useful when size is not known at compile time, e.g., input value
  - Need to specify no. dimensions in declaration
  - Need to specify that it's an allocatable array
  - real, dimension(:,:,:), allocatable :: x, y
  - allocate function performs allocation
  - allocate(x(ni,nj,nk), y(ldim,mdim,ndim))
  - When you're done with the variables, deallocate
  - deallocate(x, y)
    - not necessary at very end of code; Fortran will clean them up for you

1.8.4 User Defined Types and Structures

A ***user defined*** **type** is defined in ***Fortran 90*** as follows:

Example:

```
TYPE TypeName
        type1 [,attr] :: element1;
        type2 [,attr] :: element2;
        type3 [,attr] :: element3;
        ...
END TYPE [TypeName]
```

- Structures act like derived types with the SEQUENCE attribute. Otherwise they may contain no specifiers.
- Structures may contain a special field with the name %FILL. This will create an anonymous component which cannot be accessed but occupies space just as if a component of the same type was declared in its place, useful for alignment purposes. As an example, the following structure will consist of at least sixteen bytes:

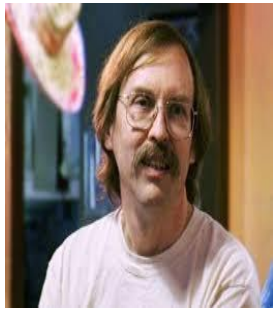
Structures may share names with other symbols. For example, the following is invalid for derived types, but valid for structures:

```
structure /header/
        ! ...
end structure
record /header/ header
```

Structure types may be declared nested within another parent structure. The syntax is:

```
structure /type-name/
        ...
        structure [/<type-name>/] <field-list>
...
```

1.9 Perl



Practical Extraction and reporting Language (Perl) was developed by Larry Wall in 1987

PERL was originally designed under Unix, but now also runs under all OS (Including Windows)

What is Perl?

Interpreted language that is optimized for string manipulation, I/O and system tasks.

Why Perl?

- Speed of development – Don't have to compile create object file and then execute.
- Power of flexibility of a high programming language.
- Easy to use, freely available and portable.
- Makes easy jobs easy, without making hard jobs impossible.

1.9.1 Overview

Basic commands in Perl:
perl – v : gives the version of the Perl the user is using.
perldoc perl : gives the list of manual pages as a part of every Perl installation.
perldoc –h: gives the brief summary of options available.
To create a Perl program, only a text editor and the perl interpreter are required. Perl file ends with *.pl* (simple.pl)

Execution Command:

- perl filename.pl, or
- ./filename.pl

The core of Perl is Perl Interpreter the engine that actually interprets, compiles, and runs Perl scripts. All Perl programs go through two phases:

- A compile phase where the syntax is checked and the source code, including any modules used, is converted into bytecode.
- A run-time phase where the bytecode is processed into machine instructions and executed.

Man commands in Perl to read documentation

Command	Description
perl	Overview (top level)
perldelta	Changes since last version
perlfaq	Frequently asked questions
perltoc	Table of contents for Perl documentation
perlsyn	Perl Syntax
perlop	Operators and precedence
perlre	Perl Regular Expression
perlfunc	Built in functions
perlsub	Subroutines
perlvar	Predefined Variables

1.9.1 Overview

Basic syntax in Perl:

- "#" is used for commenting the line.
- All statements should end with ";".
- "$_" is the special variable called default variable.
- Perl is case sensitive.
- Perl program is compiled and run in a single operation.

Example:

```
#! /usr/local/bin/perl
# Directs to perl interpreter on the system.
print "This is ACDS";
# Prints a message on the output.
```

Output
This is ACDS

Example:

```
#! /usr/local/bin/perl –c
Print "This is ACDS";
```

Output
Syntax Ok

Basic Options:

-c : Check syntax and exit without executing the script.
-v : Prints the version of perl executable.
-w : Prints warnings
-e : Used to enter and execute a line of script on the command line

1.9.2 Perl vs C/C++ Perl vs Python

Perl	Python
Perl is do not has a large collection of packages	Python has a huge library support
Per is quite messy	Python code is clear and easy to understand
Perl does not care for whitespaces	Python deals with whitespaces and can cause syntax error
Regular expression is a part of the language and makes processing easy	Need to deal with function and regex
Has extension *.pl*	Has extension *.py*
End of statement is represented by semicolon	No specific end of statement needed

Perl	C/C++
No explicit function is required	C/C++ code requires main() function to execute
It is an interpreted programming language	a general-purpose object-oriented programming (OOP) language
can use closures with unreachable private data as objects	C/C++ doesn't support closures where closures can be considered as function
Perl scripts are saved using .pl extension	The *.c* and *.cpp* file extension is used to save c and c++ code respectively
Perl uses single quotes to declare string	uses double quotes to declare a string

1.9.3 Input and Output, Control Flow

- STDIN : It is a normal input channel for the script.
- STDOUT : It is an normal output channel.
- STDERR : It is the normal output channel for errors.

Operators in Perl broadly divided into 4 types.

- Unary operator which takes one operand. Example: not operator i.e. !
- Binary operator which take two operands Example: addition operator i.e. +
- Ternary operator which take three operands. Example: conditional operator i.e. ?:
- List operator which take list operands Example: print operator

Operator	Description
+	Adds two numbers
-	Subtracts two numbers
*	Multiplies two numbers
/	Divides two numbers
++	Increments by one.(same like C)
--	Decrements by one
%	Gives the remainder (10%2 gives five)
**	Gives the power of the number. Print 2**5 ; #prints 32.

Operator	Description
<<	Left Shift Print 2 >>3 ; left shift by three positions, prints 8
>>	Right Shift Print 42 >>2; #right shift by two positions, prints 10
x	Repetition Operator. Ex print " hi " x 3; Output : hihihi Ex2: @array = (1, 2, 3) x 3; #array contains(1,2,3,1,2,3,1,2,3) Ex3 :@arr =(2)x80 #80 element array of value 2

Operator	Description
&& or AND	Return True if operands are both True
|| or OR	Return True if either operand is True
XOR	Return True if only one operand is True
! or NOT	(Unary) Return True of operand is False

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT

1.9.3 Input and Output, Control Flow

if statement : *if* keyword to execute a statement block based on the evaluation of an expression
Syntax: if ($var ==1) { }

until loop: *until* loops are used to repeat a block of statements while some condition is false.
Syntax: until($condition) { }

do-until loop: The condition is checked at the end. **Syntax:** do { } until(condition);

for loops: Runs the loop until the condition is fulfilled. **Syntax:** for(condition) { }

foreach loop: Used solely to iterate over the elements of an array. **Syntax:** foreach(@array) { }

last: The last keyword is used to exit from a statement block. **Syntax:** if (condition) { last; }

next: The next keyword use to skip the rest of the statement block and start the next iteration
Syntax: if (condition) { next; }
redo: The redo keyword causes Perl to restart the current statement block. **Syntax:** if (condition) {redo; }

1.9.4 Object Oriented Programming in Perl

An object (also called an instance), has the following: *attributes or properties*, *identity, and behavior*.

- Object is a collection of attributes usually represented with an array or hash
- Each collection of attributes is represented through with unique identity
- All functions that access or update one or more attributes of the object constitute the behavior of the object

Defining attributes

```
# Use a hash table to store student attributes
%student = ("name" => "Tony",
            "age" => 12,
            "class" => 9);
print "Name: ", $employee{name};
```

Defining behaviour

```
sub promote_employee {
    my $r_student = shift;
    $r_student->{"class"} =
lookup_next_class($r_employee->{"class"});
    }
 # To use it
promote_employee($emp1);
```

Unique identity

```
# Using an anonymous hash
sub new_student {
    my ($name, $age, $class) = @_;
    my $r_student = {       # Create a unique object
    "name" => $name,        # using an anonymous hash
    "age" => $age,
    "class" => $class
     };
    return $r_student;      # Return "object"
  }
```

1.9.4 Object Oriented Programming in Perl

Need of Polymorphism:

- Polymorphism provides ability to an object to take many forms
- Polymorphism and run-time binding (ability to call appropriate module's function) are main contribution to object oriented programming

Class methods and attributes:

- Class methods (or static methods) are functions that are relevant to a class but don't need a specific object instance to work
- Class attributes are properties that pertain to all instances of a class
- Unlike C++ or Java, Perl has no specific syntax for class attributes and methods. Class attributes are simply package global variables, and class methods are ordinary subroutines that don't work on any specific instance

Need of Inheritance:

- Perl allows a module to specify a list of other module names, in a special array called @ISA.
- When it does not find a particular class or instance method in a module, it looks to see if that module's @ISA has been initialized. If yes, it checks to see if any of those modules support the missing function, picks the first one it can find, and passes control to it.

Object Destruction:

- Perl automatically garbage collects a data structure when its reference count drops to zero.
- If a data structure turns to a module, some clean-up is performed before it destroys the object , by calling a special procedure in that module called DESTROY

1.9.5 Subroutines

The Perl model for subroutine call and return values is simple:

- all subroutine are passed as parameters one single flat list of scalars (goes to @_)
- all subroutines likewise return to their caller one scalar or a list of scalars.
- lists or hashes in these lists will collapse, losing their identities but one may always pass a reference
- The subroutine name start with a &, for simplification can be omitted.

Example:

```
sub max {
            my $mval = shift(@_);
#           my ($mval, @rest) = @_; # big copy
            foreach my $foo (@_) {
               if ( $mval < $foo ) {
                        $mval = $foo;
                         }
                     }
        return $mval;
     }
my $my_max = max(1, 9, 3, 7);
print $my_max; # prints 9
```

- The parameters are passed to the subroutine in the array @_, changing the values of the array will change the values outside the subroutine call.
- The last statement value is returned by default.

- **Smart matching:** With Perl 5.10, smart matching was introduced which is a powerful matching operator ‘ ~~ ’ can be used in place of binding operator ‘ =~ ’
  Example: *$some_text =~ /some regex/* can be replaced with *$some_text ~~ /some regex/*
  If one of its operands is a regular expression ‘~~’ does a regex match

- **Database Independent Interface (DBI):** The DBI module provides a useful and easy-to-use API that allows you to interact with many of databases including Oracle, SQL Server, MySQL, Sybase, etc.
  Example:

```
use DBI;
my $dsn = "DBI:mysql:database=$database;"."host=$hostname;port=$port";
my $dbh = DBI->connect($dsn, $user, $password);
```

- **AUTOLOAD:** When no other method is found AUTOLOAD will be called. Syntax: *use vars '$AUTOLOAD';*

- **Socket Programming**: Establishing communication between two processes like client server model

- **CGI Programming**: Perl’s approach for exchange of information between server and scripts

1.9.6.1 Socket Programming

Socket is UNIX mechanism to create connection between two processes
The **client-server** model:
Creating a client

- Create a socket with socket call.
- Connect (the socket) to the server using connect call

Creating a server

- Create a socket using socket call.
- Bind the socket to a port address using bind call.
- Listen to the socket at the port address using listen call.
- Accept client connections using accept call.

Server side sockets
socket() call – first call to establish connection
bind() call – specify the port number at which connection will be accepted
listen() call – specify the port which will wait for incoming connection
accept() call –accept all the incoming connections

Client side sockets
connect() call – connect to the server

1.9.6.1 Socket Programming

A server socket program

```
use strict;
use Socket;
my $port = shift || 7890; #default port
my $proto = getprotobyname('tcp');
my $server = "localhost";
socket(SOCKET, PF_INET, SOCK_STREAM, $proto)
   or die "Can't open socket $!\n";
setsockopt(SOCKET, SOL_SOCKET, SO_REUSEADDR, 1)
   or die "Can't set socket option to SO_REUSEADDR $!\n";

bind( SOCKET, pack_sockaddr_in($port, inet_aton($server)))
   or die "Can't bind to port $port! \n";

listen(SOCKET, 5) or die "listen: $!";
print "SERVER started on port $port\n";

my $client_addr;
while ($client_addr = accept(NEW_SOCKET, SOCKET)) {
   # send them a message, close connection
   my $name = gethostbyaddr($client_addr, AF_INET );
   print NEW_SOCKET "Smile from the server";
   print "Connection recieved from $name\n";
   close NEW_SOCKET;
```

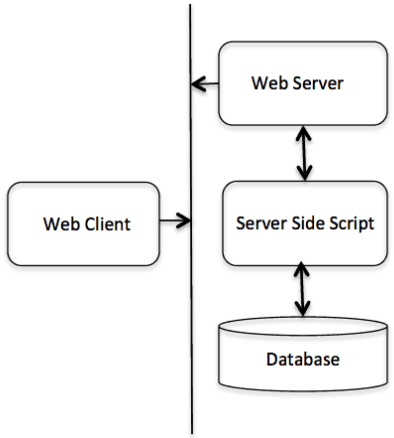
A client socket program

```
use strict;
use Socket;
my $host = shift || 'localhost';
my $port = shift || 7890;
my $server = "localhost";  # Host IP running the server

socket(SOCKET,PF_INET,SOCK_STREAM,(getprotoby
name('tcp'))[2])
   or die "Can't create a socket $!\n";
connect( SOCKET, pack_sockaddr_in($port,
inet_aton($server)))
   or die "Can't connect to port $port! \n";

my $line;
while ($line = <SOCKET>) {
   print "$line\n";
}
```

1.9.6.2 CGI Programming

- Common Gateway interface (CGI) defines how information is exchanged between server and script
- HTTP header is sent o browser to understand the content from incoming server

HTTP Header	Description
Content-type	format of the content being returned
Expires	date when the information becomes invalid
location	redirect a request to any other location
Last-modified	date of last modification of the file
Content-length	length, in bytes, of the data being returned
Set-cookie	Set the cookie passed through the *string*



HTTP Protocol

CGI Environment variable	Description
CONTENT-TYPE	Data type of content
CONTENT-LENGTH	Length of query information
HTTP-COOKIE	Returns the set cookie
PATH-INFO	Path of CGI script
REMOTE-ADDR	IP address of remote host
REMOTE-HOST	Qualified name of the host

There are many other environment variables of CGI

1.9.6.2 CGI Programming

The GET and POST methods

- GET - sends the encoded user information appended to the page URL itself
- POST - reliable method of passing information to a CGI program

hello_get.cgi

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};
```

hello_post.cgi

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};
```

The cgi files can be accessed using a simple html forms enabled with GET & POST methods

1.9.6.3 Packages and Modules

- A package statement is used to switch the current naming context to a specified namespace which is a named collection of unique variable names
- A package is a collection of code which lives in its own namespace which prevents variable name collisions between packages
- Packages enable the construction of modules which, when used, won't clobber variables and functions outside of the module's own namespace
- “: :” package qualifier is used to explicitly refer to variables within a package
- A Perl module is a reusable package defined in a library file whose name is the same as the name of the package with a .pm as extension.
- A package uses BEGIN and END blocks to create modules

Example:

```
#!/usr/bin/perl
package ACDS;
print "Begin and Block Example\n";
BEGIN
   { print "This is the BEGIN Block\n" }
END
{ print "This is the END Block\n" }
```

- The Perl module is loaded by “use” and “require”
- Both “use” and “require” call the *eval* function to process the code
- The “1;” at the end of the code causes *eval* to evaluate to TRUE

1.9.7 Examples

Renaming Files:

```
#!/usr/local/bin/perl
# rename series of frames
#
if ($#ARGV != 3) {
        print "usage: rename old new start
stop\n";
        exit;
}
$old = $ARGV[0];
$new = $ARGV[1];
$start = $ARGV[2];
$stop = $ARGV[3];

for ($i=$start; $i <= $stop; $i++) {

        $num = $i;
        if($i<10) { $num = "00$i"; }
        elsif($i<100) { $num = "0$i"; }

        $cmd = "mv $old.$num $new.$num";
        print $cmd."\n";
        if(system($cmd)) { print "rename
failed\n"; }
}
```

BioPerl is:

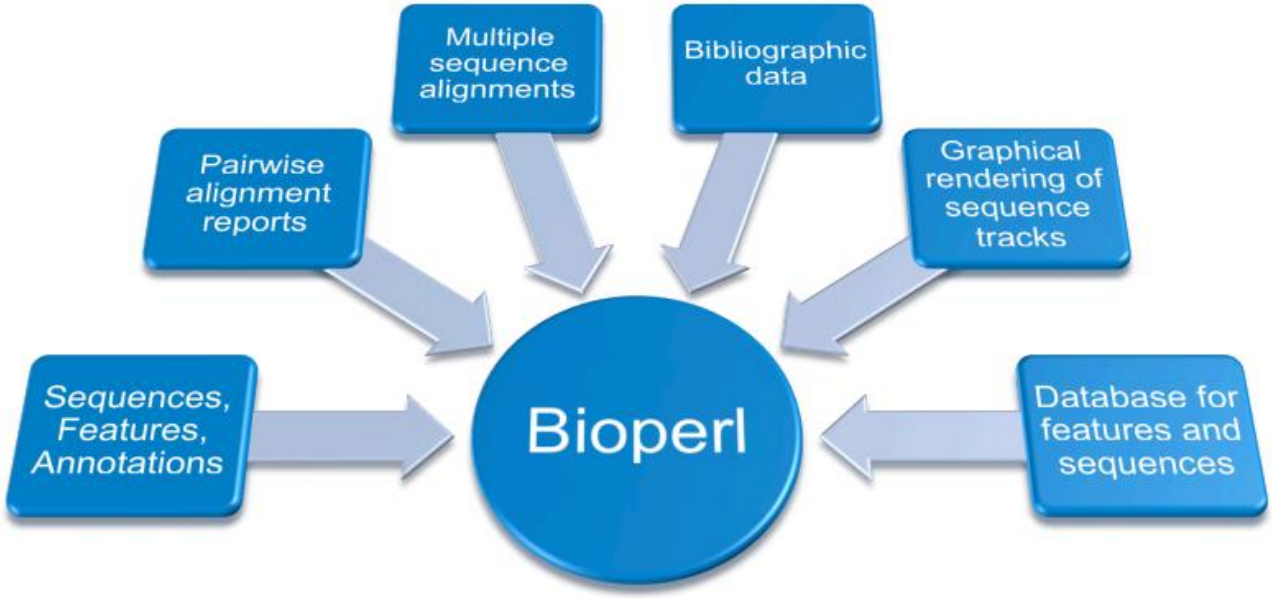
- A collection of Perl modules for biological data and analysis
- An open source toolkit with many contributors
- A flexible and extensible system for doing bioinformatics data manipulation
- Consists of >1500 modules; ~1000 considered core

Modules are interfaces to data types:

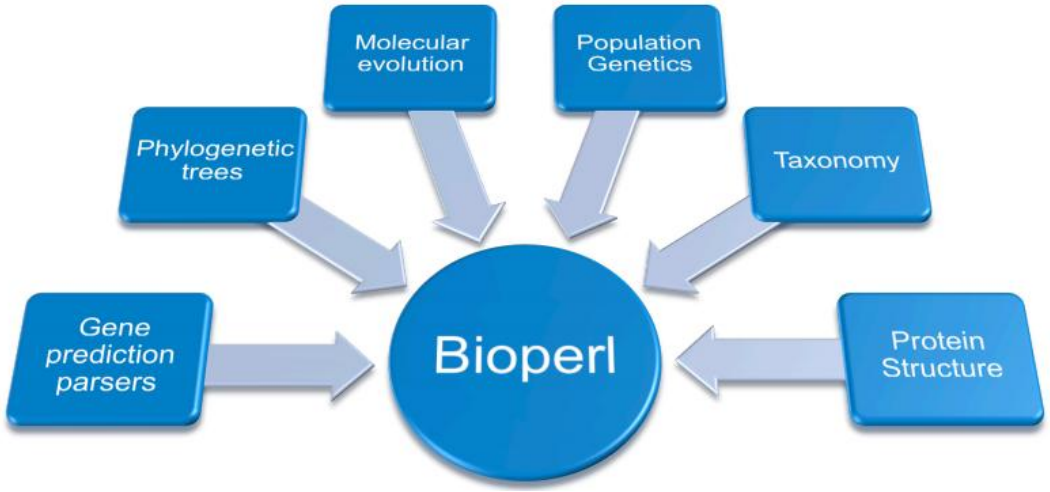
- Sequences
- Alignments
- (Sequence) Features
- Locations
- Databases

Bio Perl is used for:

- Retrieve sequence data from NCBI
- Transform sequence files from one format to another
- Parse (or search) BLAST result files
- Manipulate sequences, reverse complement, translate coding DNA sequence to protein



Major domains covered by Bio Perl



Other domains covered by Bio Perl

1.10.2 Bioperl Advances

- Collection of Perl modules for life sciences data and analysis
- Modules are interfaces to data types: Sequences, Alignments, Features, Locations, Databases
- SeqIO Module:
  - Reads a Fasta sequence file, count number of sequences
  - Reads a Fasta sequence file, count number of bases
  - Convert format and write sequences
- Bio Perl databases
  - Bio::DB::Fasta – Fast random access to Fasta seq databases
  - Bio::DB::GenBank – Query Genbank
  - Bio::DB::SeqFeature – Databases of features

1.10.3 Exception handling biopython bioinformatic

The goal of Biopython is to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and classes

Functionalities of biopython:

- Biopython is a python package which can parse a bioinformatics files (FASTA, GenBank, PubMed etc.) into Python data structures
- Files in the supported formats can be iterated over record by record or indexed and accessed via Dictionary interface.
- Code to deal with popular on-line bioinformatics destinations such as – NCBI – Blast, Entrez and PubMed services
- A standard sequence class that deals with sequences, ids on sequences, and sequence features
- Tools for performing common operations on sequences, such as translation, transcription and weight calculations.
- Code to perform classification of data using k Nearest Neighbors, Naive Bayes or Support Vector Machines.

Exception handling with biopython:

- Bio.SeqIO.to_dict() will check duplicate keys and raise exception if found
- StopIteration exception is no more records are there
- ValueException exceptions to detect invalid accession numbers
- PDBConstructionException will be generated if any problems are detected during the parse operation

Accessing sequence data from local and remote databases:

```
$seq = Bio::Seq->new('-seq'=>'actgtggcgtcaact',
                '-desc'=>'Sample Bio::Seq object',
                '-display_id' => 'something',
                '-accession_number' => 'accnum',
                '-alphabet' => 'dna' );
```

Manipulating sequence data with Seq methods:

```
$seqobj->display_id(); # the human read-able id of the sequence
$seqobj->seq(); # string of sequence
$seqobj->subseq(5,10); # part of the sequence as a string
$seqobj->accession_number(); # when there, the accession number
$seqobj->alphabet(); # one of 'dna','rna','protein'
$seqobj->primary_id(); # a unique id for this sequence irregardless
                       # of its display_id or accession number
$seqobj->desc() # a description of the sequence
```

1.11 R Programming

- R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

Evolution of R:-

R was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

Features of R:-

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.

- **Simple and Open source –** One of the simplest and open source.
- **Interpreted** − Processed at runtime by the interpreter, no need to compile before executing it.
- **Dynamic Type Checking –** Type of variable known at run time i.e. type need not be declared.
- **Object-Oriented** − Supports Object-Oriented programming that encapsulates code within objects.
- **Integrated** – Can be easily integrated with C, .Net or Python.

- **Download and Install R (**obtain at Comprehensive R Archive Network (CRAN), R maintainer**)**

- **Download and Install RStudio IDE** (for better GUI)
  - Integrated Development Environment (IDE) for R
  - Availability – Open source and Commercial
  - Editions – Desktop and Server
  - Platforms – Windows, Linux and macOS
  - https://rstudio.com/products/rstudio/download/#download

Packages

- A package is a library of prewritten code designed to accomplish some task or a collection of tasks.
- Biggest reason for R's popularity is its collection of user-contributed packages.
- If a statistical technique exists, it has been written in R and contributed to CRAN.
- As of 2013, there were 4,845 packages available on CRAN, written by an estimated 2,000 people.

Some Popular Packages:

ggplot2 - used for plotting

sp - deal with spatial data

survival - used for survival analysis

reshape2 - reshape data to transform data between wide and long formats.

plyr - split data apart, do stuff to it, and mash it back together.

Glmnet - fits a generalized linear model via penalized maximum likelihood.

Rcpp - accessing, extending or modifying R objects at the C++ level for HPC.

coefplot - plots coefficients

1.11.1 Data Preparation

- Languages like C++ requires entire section of code to be written, compiled and run to see results.
- But, R is very interactive. That is, results can be seen one command at a time.

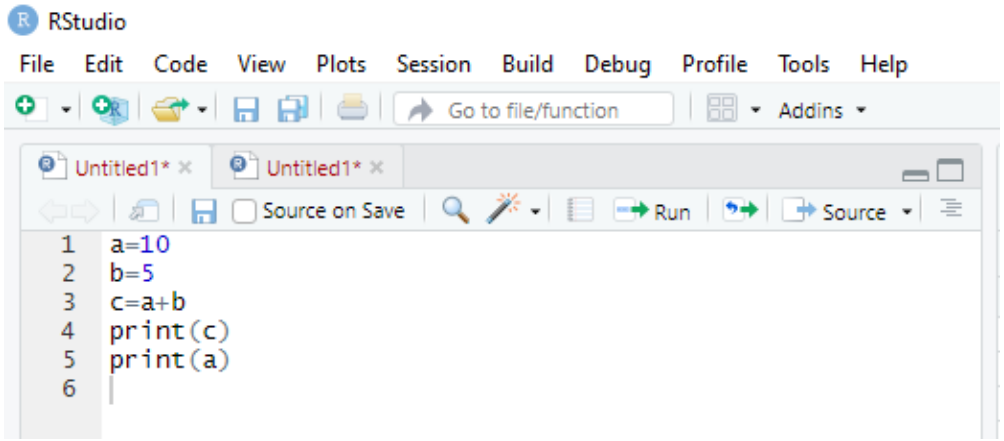
- **Try:**

```
print("Hello world")
print(5)
print(5:100)
print(1.6+3.5)
```

Comment

- Place # in front of a line
- Select lines and press Ctrl+Shift+C

```
a=10
print(a)
a

b1="Hello"
b2="How"
b3="How are you"
print(b1)
print(b2)
print(b3)

a=10
b=20
print(a==b)

a= "ranjan"
b= "qmar"
print(c==d)
```



Variables and Constants

- In any programming language, we need to use various variables to store information.
- Variables are reserved memory locations to store values.

- We store information of various data types like character, integer, floating point, Boolean etc.
- Based on the data type of a variable, OS allocates memory and decides what can be stored.

- Unlike statically typed languages such as C++, R does not require variable types to be declared.
- A variable can take any data type, at one point a number, then a character and then number again.
- R variable can store an atomic vector, group of atomic vectors or combination of many R-objects.

Predefined Constants

- pi
- letters
- LETTERS
- month.name
- month.abb

```
> pi
[1] 3.141593
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"      "August"    "September" "October"   "November"  "December"
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
>
```

Operators

- Operator tells the compiler to perform specific mathematical or logical manipulations.
- Types of Operators - Arithmetic, Relational, Logical, Assignment, Miscellaneous

Arithmetic Operators: +, -, *, /, %%, %/%, ^

Relational Operators: <, >, ==, <=, >=, !=

Logical Operators: &, |, !, &&, ||

Assignment Operators: =, <-, <<-, ->, ->>

Miscellaneous Operators: Used for specific purpose

```
V1=c( 2,5.5,6)        V2=c(8, 3, 4)

print(V1+V2)

print(V1!=V2)

print(V1&V2)

V1 <- c( 2,5.5,6)

V3 = 2:5
print(v1 %in% v2)
```

Data Types

- There are numerous data types in R that store various kinds of data.
- The four main data types mostly used are –
  - **Numeric Data**: It handles integers and decimals, both positive and negative and of course 0.
  - **Character Data:** R has two primary ways of handling character data: character and factor.

    x = "data" y =factor("data") #note: nchar() not work for factor data
  - **Dates**: R has various types of dates, useful are Date and POSIXct.
  - **Logical:** Data that can be either TRUE or FALSE. (TRUE same as 1 and FALSE same as 0)

- The type of data contained in a variable is checked with the class function.

  class(x)

  typeof(x)

Strings

- Any value written within a pair of single quote or double quotes in R is treated as a string.
- Internally R stores every string within double quotes; even we create them with single quote.

Rules Applied in String Construction

- The quotes at beginning and end of a string should be either double quotes or single quote.

Valid Strings

a = 'Start and end with single quote'

a = "Start and end with double quotes"

b= "single quote ' in between double quotes"

b= 'Double quotes " in between single quote'

1.11.2 Decision making, Loop Control and Array

- Need to control the flow of normal sequential execution of the program many times.

- There are two types of control statements –
  - **Decision Statements** – Execute certain commands only when certain conditions satisfied.

  - **Loop Statements –** Execute certain commands repeatedly and use certain logic to stop it.
  - **Loop Control Statements –** Controlling the loops by introducing some keyword.

Note: R programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Decision Making

If statement:

```
if(boolean_expressions) {
        # statement(s) will execute if the Boolean expressions are true }
```

If-else statement:

```
if(boolean_expressions) {
        # statement(s) will execute if the boolean expression is true
} else {
        # statement(s) will execute if the boolean expression is false }
```

If-else-if Statement:

```
if(boolean_expression 1) {
        # Executes when the boolean expression 1 is true
}else if( boolean_expression 2) {
        # Executes when the boolean expression 2 is true
}else {
        # executes when none of the above condition is true }
```

Switch statement:

```
switch(expression, case1, case2, case3....)
```

Loop Control

- Sometimes, need to execute a block of code several number of times.
- R programming language provides the following kinds of loop to handle looping requirements -

For: for (value in vector) {
#statements }

While: while (test_expression){
#statements }

Repeat: repeat {
#statements
if(condition) { break } }

Loop Control Statements

- Break Statement
- Next Statement

Array

- Store data in more than two dimensions while matrices are confined to two dimensions.
- For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns.

Array Creation

- Created using **array()** function which takes a dim attribute to create required dimension.

```
V1 =c(5,9,3)
V2 =c(10,11,12,13,14,15)
A = array(c(V1, V2),dim = c(3,3,2), dimnames = list(row.names,column.names,matrix.names))
column.names = c("col1","col2","col3")
row.names = c("row1","row2","row3")
matrix.names = c("Matrix1","Matrix2")
print(A)
A2 = array(c('green','yellow','red'),dim=c(3,2,2))
print(A2)
```

Array

Addition and Subtraction

```
V1 = c(5,9,3)      V2 = c(10,11,12,13,14,15)      A1 = array(c(V1, V2),dim = c(3,3,2))

V3 = c(9,1,0)      V4 = c(6,0,11,3,14,1,2,6,9)    A2 = array(c(V1, V2),dim = c(3,3,2))

A3 = A1 + A2       print(A3)

Ar1 = A1 [1,1,1]                  Ar2 = A2 [2,1,3]
print(Ar1)                        print(Ar2)

M1 = A1 [,,2]                     M2 = A2 [,,2]
result = M1+ M2                   print(result)
```

Examples

```
x=10
if(x==12) {
        print("False")

} else if(x==30) {
        print("No")

} else {
        print("True")
}
```

```
x = switch (4, “A”, “B”, “C”, “ranjan”)
print(x)
```

```
a1=1:10
for(i in a1) {
        print(i)
        }
```

```
a1=1
while(a1<15) {
        print("hello")
        a1=a1+1
        }
```

```
count=2
repeat{
        print("hello")
        count=count+1
        if (count>7)
        {break}
        }
```

```
v =LETTERS[1:6]
for ( i in v){
        if (i == "D")
        {next}
        print(i)
        }
```

1.11.3 Function in R Language

- A function is a set of statements organized together to perform a specific task.
- R has many **in-built** functions and also users can create their own **user-defined** functions.

Function Definition

```
function_name =function(arg_1, arg_2, ...) {
                                        function body
                                        }
```

Function Components

- **Function Name:** Actual name of function stored in R environment as an object with the name.
- **Arguments:** When a function is invoked, we pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body:** Contains a collection of statements that defines what the function does.
- **Return Value:** The last expression in the function body to be evaluated.

Function Definition

Syntax

```
def functionname(parameters) {
        function_suite
        return expression
        }
```

#By default, parameters have positional behavior and need to inform them in the same order that they were defined.

Example: #Following function takes a string as input parameter and prints it

```
def printme(str) {
        print str
        return
        }
```

Function Calling

With an Argument

```
# Create a function with an argument.
        func = function(a) {
                        for(i in 1:a) {
                                b = i^2
                                print(b)
                                }
                        }
```

```
# Call the function func supplying 4 as an argument.
        func(4)
```

Calling a Function without an Argument

```
# Create a function without an argument.
        func =function() {
                        for(i in 1:3) {
                                print(i^2)
                                }
                        }
```

```
# Call the function without an argument.
        func()
```

Function Calling

With Argument Values (by position & name)

```
# Create a function with arguments.
        func = function(a,b) {
                        result <- a*b
                        print(result)
                        }


# Call the function by position of arguments.
        func(5,3)

# Call the function by names of the arguments.
        func(a=11,b=5)
```

With Default Argument

```
# Create a function with arguments.
        func=function(a = 3,b =6) {
                        result <- a*b
                        print(result)
                        }


# Call the function without giving any argument.
        func()

# Call the function giving new values of argument.
        func(9,5)
```

Function Types

In Built Functions

- Built-in functions which can be directly called in the program without defining them first.
- Simple examples of in-built functions are seq(), mean(), max(), sum(x)and paste(...) etc.

```
print()
seq(32,44, by=2)
mean(25:82)        sum(41:68)        factorial(5)        as.Date("2019-06-05")
length(V)
```

User Defined Functions

- They are specific to what a user wants and once created they are used like the built-in functions.

```
func = function(a) {
                    for(i in 1:a) {
                                b = i^2
                                print(b)
                                }
}
```

Example

```
def sum() {
        a =  random.random()
        b =  random.random()
        return a+b          #Functions in R takes multiple inputs but return only one output
        }
sum()
```

Vectors

- Vectors are the most basic and simplest R data objects.
- Other R-Objects are built upon the atomic vectors.
- Vectors do not have a dimension, meaning there is no such thing as a column vector or row vector.

- There are six data types of atomic vectors termed as six classes of vectors - logical, integer, double, complex, character and raw.
- In R the number of classes is not confined to only the above six types.
- For example, using many atomic vectors an array is created whose class will become array.

- R is a vectorised language that means operations are applied to each element of the vector automatically, without the need to loop through the vector.

Vectors

Single Element Vector

- Write just one value in R, it becomes a vector of length 1
- It belongs to one of the above vector types.

```
print("abc")                    # Atomic vector of type character.
print(12.5)                     # Atomic vector of type double.
print(63L)                      # Atomic vector of type integer.
print(TRUE)                     # Atomic vector of type logical.
print(2+3i)                     # Atomic vector of type complex.
print(charToRaw('hello'))       # Atomic vector of type raw.
```

Vectors

Vector arithmetic

```
V=c(2:5)                V=c(2,3,4,5)            print(V*2)      print(V^2)      sqrt(V)
V1 =c(3,8,4,5,0,11)     V2= c(4,11,0,8,1,2)
V1=c(100:200)           V2=c(300:400)           result = V1+V2
```

Vector Element Recycling

```
V1 =c(3,8,4,5,0,11)             V2 = c(4,11)            # V2 becomes c(4,11,4,11,4,11)
result = V1+V2
```

1.11.4 Vector and Lists

Vectors

Vector Element Sorting

V =c(3,8,4,5,0,11, -9, 304)

V = c("Red","Blue","yellow","violet")

result = sort(V)

nchar(V) length(V)

result =sort(V, decreasing = TRUE)

Vector Comparison

V1 = 10:1 V2= -4:5

any(V1<V2) all(V1<V2)

Vectors

Missing Data Handling

- R has two types of missing data: **NA**and **NULL**
- NA seen as just another element of a vector.
- is.na tests each element of a vector for missingness.

- NULL is not exactly missingness, it is nothingness,
- If used inside a vector it simply disappears.

```
z1 =c(1, 2, NA, 8, 3, NA, 3)
is.na(z1)
print(z1)
```

```
z2 =c(1, 2, NULL, 8, 3, NULL, 3)
is.null(z2)
print(z2)
```

List

Introduction

- R object containing different elements − number, string, vector, function and another list inside it.
- A list can also contain a matrix or a function as its elements.

Creation

```
L =list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1, sin)
print(L)
```

Naming list element

```
L = list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),list("green",12.3))
names(L) =c("1st Quarter", "A_Matrix", "A Inner list")
print(L)
```

List

Manipulating List Elements

```
L = list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))
names(L) =c("1st Quarter", "A_Matrix", "A Inner list")
L[4] = "New element"                # Add a new element 4th at the end of the list.
L[4] = NULL                         # Remove the last element 4th .
L[3] = "updated element"            # Update the 3rd Element.
L[[1]] [3] = "ranjan"               # Update the 4th element of 1st Element.
print(L)
```

Merging List

```
L1 = list(1,2,3)
L2 = list("Sun","Mon","Tue")
L3 =c(L1, L2)
print(L3)
```

List

Converting List into Vector

```
L1 =list(1:5)
L2 = list(10:14)

L3 =c(L1, L2)
print(L3)

V1 =unlist(L1)
V2 =unlist(L2)

V3=c(V1, V2)
print(V3)
```

1.11.5 Matrices and Data Frames

Matrix

- R objects in which same type of elements are arranged in a two-dimensional rectangular data set.
- We use matrices containing numeric elements to be used in mathematical calculations.
- We can create a matrix containing only characters or logical values, they are not of much use.

Matrix Creation

- A Matrix is created using the **matrix()** function.

Syntax: matrix(data, nrow, ncol, byrow, dimnames)

```
M = matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
print(M)
M2 = matrix(c('green','yellow','red'), byrow=TRUE)
print(M2)
```

Special Matrix Creation

```
U = matrix(k, 3, 3)
D = diag(k, 3, 3)
I = diag(1, 3, 3)
```

```
> U = matrix(5, 3, 3)
> D = diag(5, 3, 3)
> I = diag(1, 3, 3)
> U
     [,1] [,2] [,3]
[1,]    5    5    5
[2,]    5    5    5
[3,]    5    5    5
> D
     [,1] [,2] [,3]
[1,]    5    0    0
[2,]    0    5    0
[3,]    0    0    5
> I
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
>
```

Matrix

Matrix's Matrices

```
M = matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
print(M)
```

Matrices

```
dim(M)
nrow(M)
ncol(M)
Length(M)
```

Matrix

Matrix Algebra

```
M1 = matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
M2 = matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
```

Matrix Addition & Subtraction #Matrix Compatibility should be there

```
M3 = M1+ M2          print(M3)
M3 = M1- M2          print(M3)
```

Matrix Multiplication & Division #Matrix Compatibility should be there

```
M3 = M1 * M2         print(M3)      #Regular matrix Multiplication
M3 = M1 %*% M2       print(M3)      #Element wise Multiplication
M3 = M1 / M2         print(M3)
```

Matrix

Matrix Concatenation

- Merging of a row or column to a matrix #Matrix Compatibility should be there
- Concatenation of a row is done by rbind()
- Concatenation of a column is done by cbind()

```
M1 = matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
M2 = matrix(c(5, 2, 0), nrow = 1)
C = rbind(A, B)
```

```
M1 = matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
M2 = matrix(c(5, 2), byrow=FALSE, nrow = 1)
C = cbind(A, B)
```

Data frame

- A table or a two-dimensional array like structure in which each column contains values of one variable and each row contains one set of values from each column.
- Unlike an array or matrix, in data frame each column can contain different modes of data.
- First column can be numeric, second column can be character, third column can be logical.

- Following are the characteristics of a data frame -
  - It is a list of vectors of equal length.
  - The column names should be non-empty.
  - The row names should be unique.
  - The data stored in a data frame can be of numeric, factor or character type.
  - Each column should contain same number of data items.

Data frame

Creation

```
D = data.frame (
        std_id = c (1:4),
        std_name = c("Chandan", "Esther","Kikru","Ranjan"),
        std_roll = c(10, 12, 11,18),
        start_date = as.Date(c("2019-05-01", "2019-05-02", "2019-05-05", "2019-05-03")),
        dept = c("IT","BioTech","BioTech","IT")
)
print(D)
```

Create using data from a file

```
D = read.table("path of the file")
```

Data frame

What is factor issue

- When character columns are created in a data frame they become factors
- Factor variables are those where the character column is split into categories or factor levels

D[3,1]="others"

#warning→ invalid factor level, NA is generated

Resolving factor issue

- New entries need to be consistent with factor levels which are fixed when the data frame is first created

D=data.frame(column1, column2, column3, stringAsFactors=F)

#Now trying the same manipulation

D[3,1]="others"

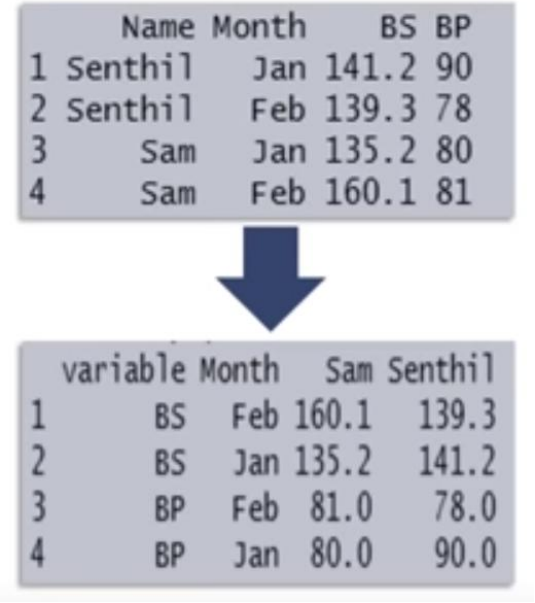
Data frame

Recasting

- Process of manipulating a data frame in terms of its variables.
- Reshaping the data – View the insights.

Step

- Melt
- Cast



	Name	Month	BS	BP
1	Senthil	Jan	141.2	90
2	Senthil	Feb	139.3	78
3	Sam	Jan	135.2	80
4	Sam	Feb	160.1	81

	variable	Month	Sam	Senthil
1	BS	Feb	160.1	139.3
2	BS	Jan	135.2	141.2
3	BP	Feb	81.0	78.0
4	BP	Jan	80.0	90.0

Identifier (Discrete Variable) → Name , Month
Measurements (Numeric Variable) → BS , BP

1.11.5 Matrices and Data Frames

Data frame

Melt

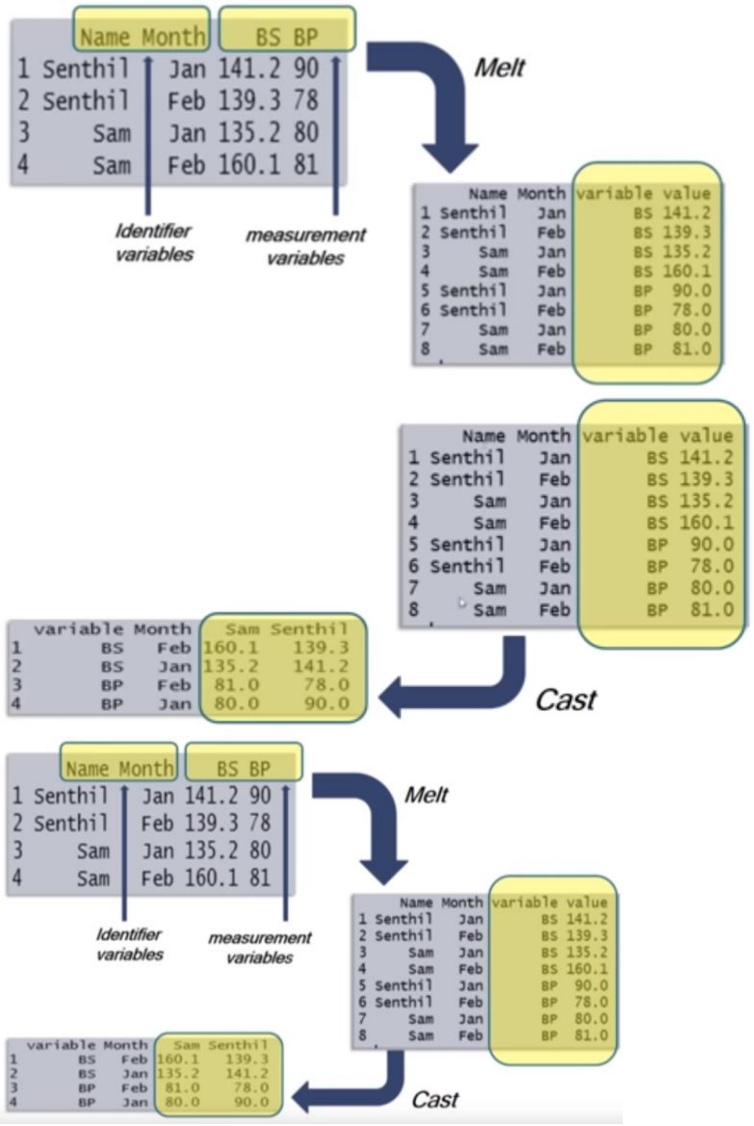
```
library (reshape2)
D1 = melt (D, id.var=c("Name","Month"), measures.vars=c("BS", "BP"))
print(D1)
```

Cast

```
library (reshape2)
D2 = dcast(D, variable+month ~Name, value.var= “value")
print(D1)
```

Recast – melt and cast together

```
Recast (D, variable+Month~Name, id.var=c("Name","Month"))
```



Data frame

Add New Variable Based on Existing One

```
Library(dplyr)
D1 = mutate(D, log_BP=log(BP))
print(D1)
```

	Name	Month	BS	BP
1	Senthil	Jan	141.2	90
2	Senthil	Feb	139.3	78
3	Sam	Jan	135.2	80
4	Sam	Feb	160.1	81

	Name	Month	BS	BP	Department
1	Senthil	Jan	141.2	90	PSE
2	Senthil	Feb	139.3	78	PSE
3	Sam	Jan	135.2	80	PSE
4	Sam	Feb	160.1	81	PSE

Joining Two Data Frames

- The Two Data Frame are –

	Name	Department
1	Senthil	PSE
2	Ramesh	Data Analytics
3	Sam	PSE

```
Library(dplyr)
D1 = left_join(D1, D2, by= "Name")          #Left join
D1 = right_join(D1, D2, by= "Name")         #Right join
D1 = inner_join(D1, D2, by= "Name")         #Inner join
```

	Name	Month	BS	BP	Department
1	Senthil	Jan	141.2	90	PSE
2	Senthil	Feb	139.3	78	PSE
3	Ramesh	<NA>	NA	NA	Data Analytics
4	Sam	Jan	135.2	80	PSE
5	Sam	Feb	160.1	81	PSE

	Name	Department	Month	BS	BP
1	Senthil	PSE	Jan	141.2	90
2	Senthil	PSE	Feb	139.3	78
3	Sam	PSE	Jan	135.2	80
4	Sam	PSE	Feb	160.1	81

1.11.6 Import Data into R: read CSV, Excel

- Using R we can read data from files stored outside the R environment. R can read and write into various file formats like csv, excel, xml etc.

Setting the working directory

```
# Get and print current working directory.
Print(getwd ())
# Set new working directory.
setwd ()
```

Reading a csv file

```
data <- read.csv("input.csv")
print(data)
```

Input Excel file

```
Install. Package("xlsx")
# Load the library into R workspace.
library("xlsx")
```

Reading the Excel file

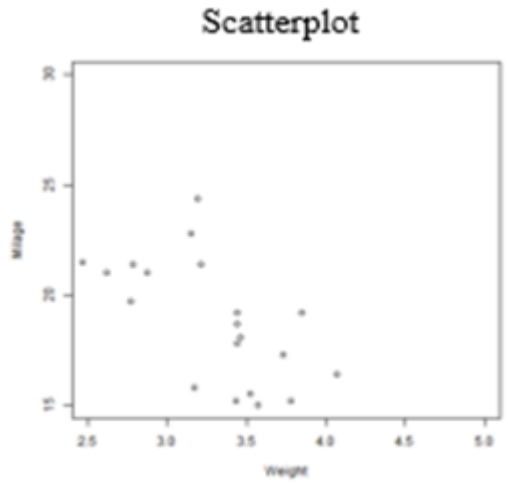
```
#Read the first worksheet in the file input.xlsx.
Data <- read.xlsx("input.xlsx", sheetIndex =1)
```

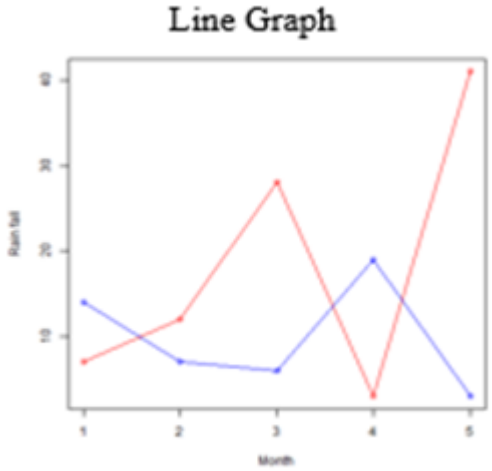
Reading the xml file

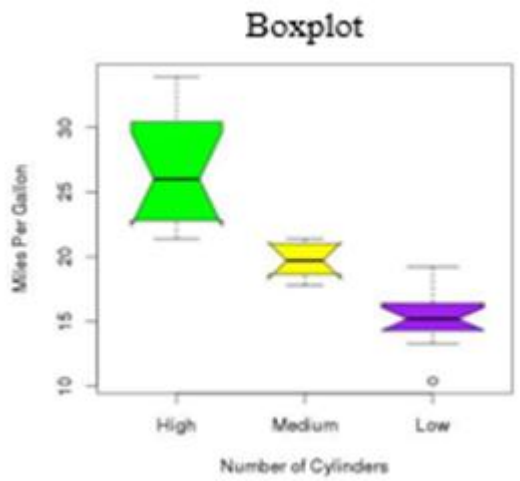
```
Install.package ("xml")
library("xml")
# Also load the other required package.
library("methods")
# Give the input file name to the function.
Result <- xmlparse(file= "input.xml")
Print(result)
```

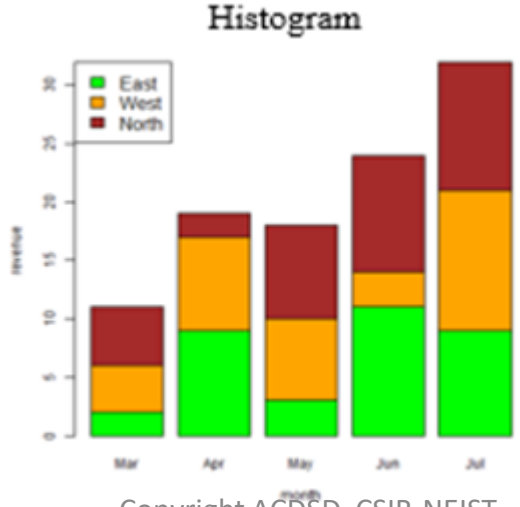
1.11.7 Data Analysis

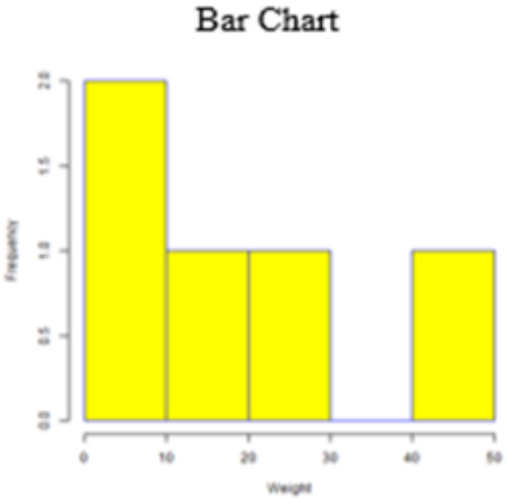
- R Programming language has numerous libraries to create different charts and graphs.











1.11.7.1 Pie Charts

- A pie-chart is a representation of values as slices of a circle with different colors.
- In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

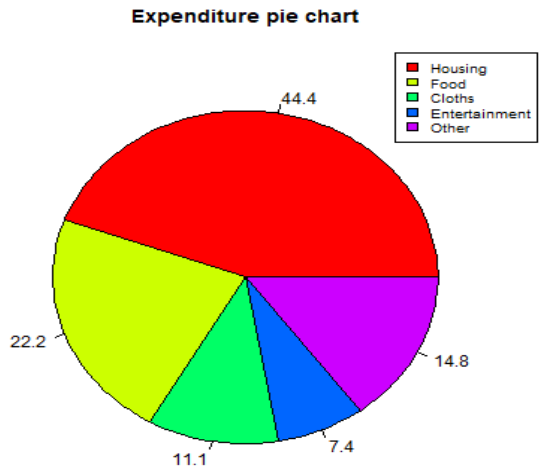
The basic syntax for creating a pie-chart using the R is −

```
Pie(x, labels, radius, main, col, clockwise)
```

Example: The monthly expenditure breakdown of an individual

Input file -

```
> expenditure
Housing    Food    Cloths    Entertainment    Other
  600       300     150          100            200
```



Codes -

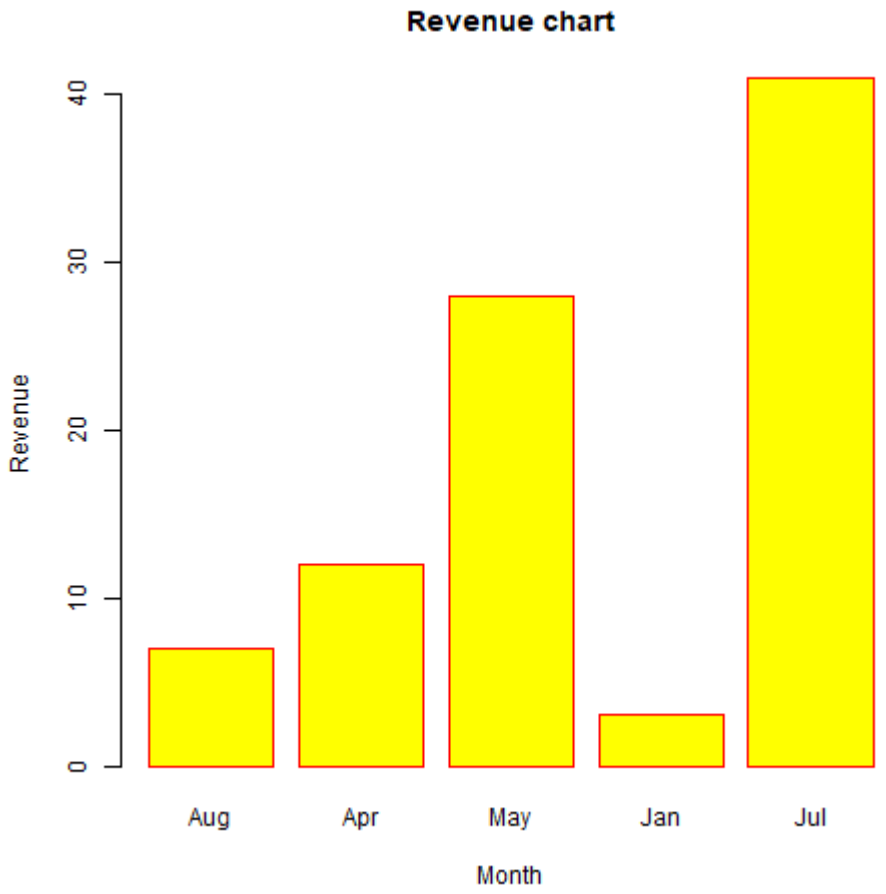
```
#Create data for the graph.
x <-  c(600, 300, 150,100, 200)
labels <-
c("Housing","Food","Cloths","Entertainment","Oth
er")
piepercent<- round(100*x/sum(x), 1)
# Give the chart file a name.
png(file = "Monthly expenditure.jpg")
# Plot the chart.
pie(x, labels = piepercent, main = "Expenditure pie
chart",col = rainbow(length(x)))
legend("topright",
c("Housing","Food","Cloths","Entertainment","Oth
er"), cex = 0.8,
       fill = rainbow(length(x)))
# Save the file.
dev.off()
```

1.11.7.2 Bar charts

- A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable.
- R uses the function **barplot()** to create barcharts.

Example:

```
# Create the data for the chart
A <- c(7,12,28,3,41)
B <- c("Aug","Apr", "May", "Jan", "Jul")

# Give the chart file a name
png(file = "barchart_months_revenue.png")

# Plot the bar chart
barplot(A,names.arg=B,xlab="Month", lab="Revenue",col="yellow",
        main="Revenue chart", border="red")
# Save the file
dev.off()
```

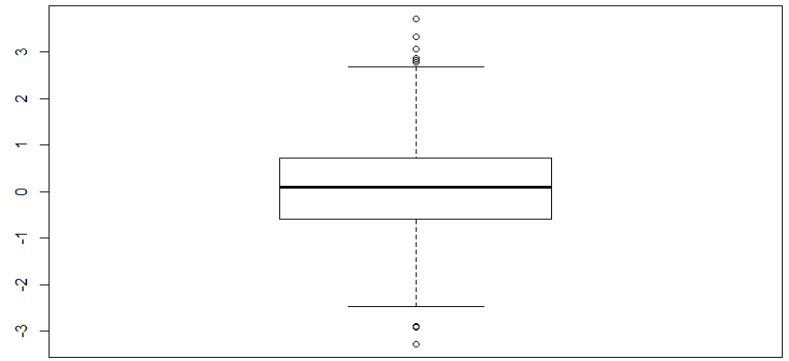


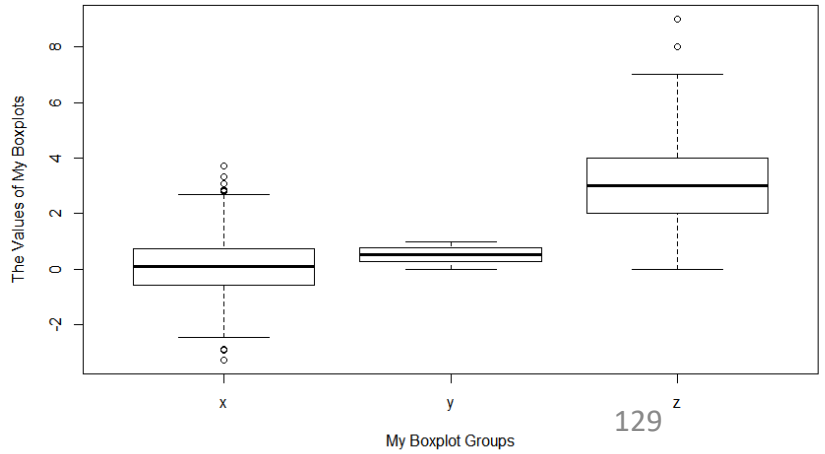
1.11.7.3 Boxplots

- Boxplots are a measure of how well distributed is the data in a data set.
- This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.
- Boxplots are created in R by using the **boxplot()** function.

Example-

```
# Create random data
x <- rnorm(1000)
set.seed(8642)
# Basic boxplot in R
boxplot(x)
#Multiple Boxplots in Same Plot#
# Create more variables
y <- runif(1000)
z <- rpois(1000, 3)
# Combine variables in data frame
data <- data.frame(values = c(x, y, z),
                   group = c(rep("x", 1000),
                             rep("y", 1000),
                             rep("z", 1000)))
# First six rows of data
head(data)
# Multiple boxplots
boxplot(values ~ group, data) in same graph
#Boxplot with User-Defined Title & Labels#
# Change main title#
boxplot(values ~ group, data, and axis labels
        main = "My Boxplots",
        xlab = "My Boxplot Groups",
        ylab = "The Values of My Boxplots")
```





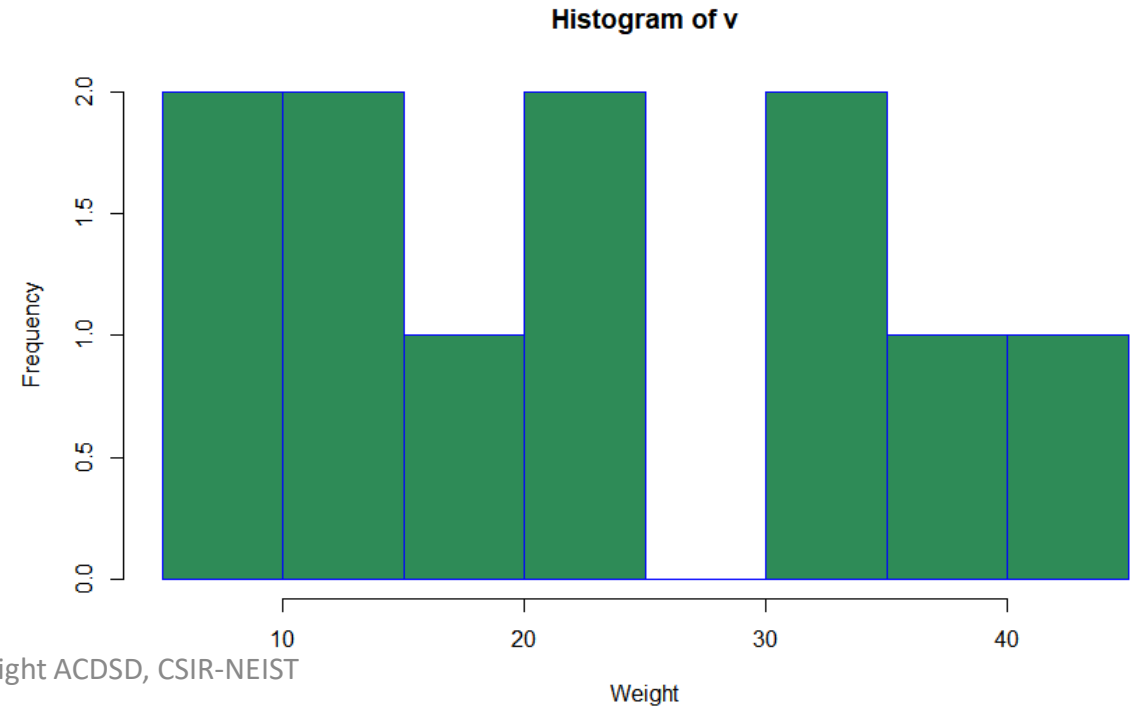
1.11.7.4 Histogram

- A histogram represents the frequencies of values of a variable into ranges.
- Histogram is similar to bar chat but the difference is it groups the values into continuous ranges.
- R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Example-

A simple histogram is created using input vector, label, col and border parameters.

```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "sea green",border = "blue")

# Save the file.
dev.off()
```



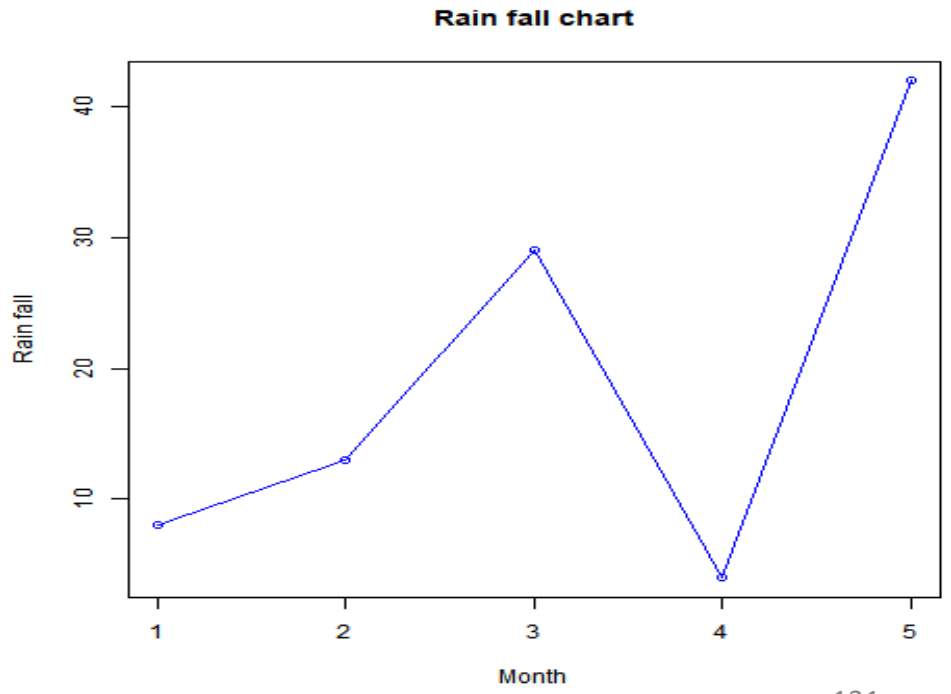
1.11.7.5 Line Graphs

- Line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of the coordinate value.
- Usually used in identifying the trends in data.
- The **plot()** function in R is used to create the line graph.

Example-

A multiple line chart is created using the input vector and the type parameter as "O" . The below script will create and save a line chart in the current R working director..

```
# Create the data for the chart.
v <- c(8,13,29,4,42)
# Give the chart file a name.
png(file = "line_chart_label_colored.jpg")
# Plot the bar chart.
plot(v,type = "o", col = "blue", xlab = "Month", ylab = "Rain
fall",main = "Rain fall chart")
# Save the file.
dev.off()
```



1.11.7.6 Scatter plots

- Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.
- The simple scatterplot is created using the **plot()** function.

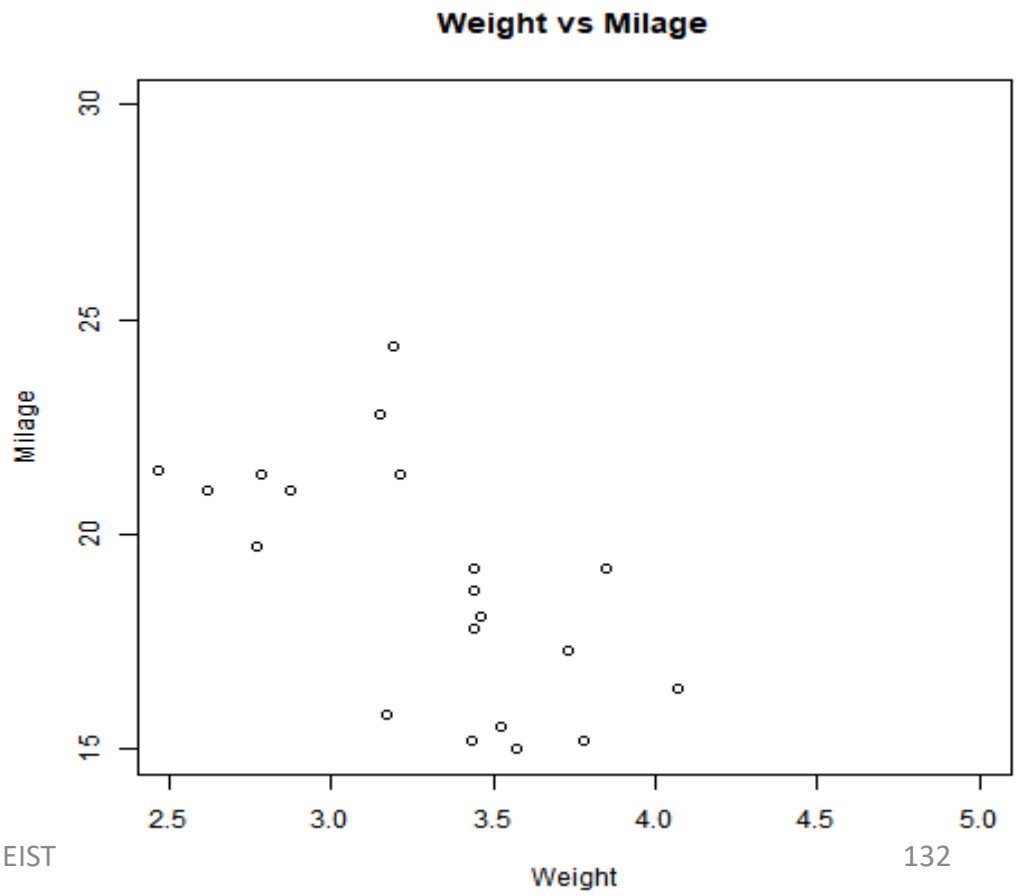
Example- Using the data set **"mtcars"** available in the R environment to create a basic scatterplot.

Input data:

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

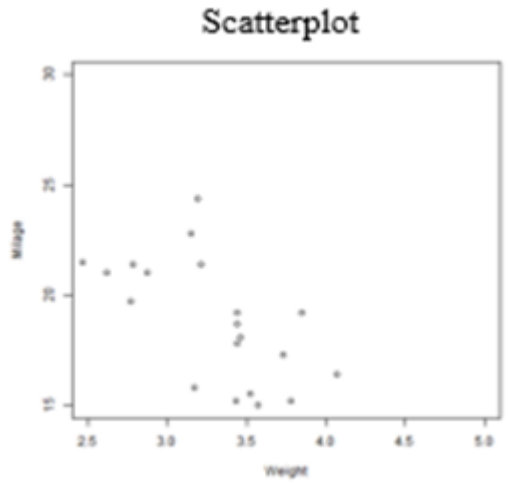
Creating Scatterplot:

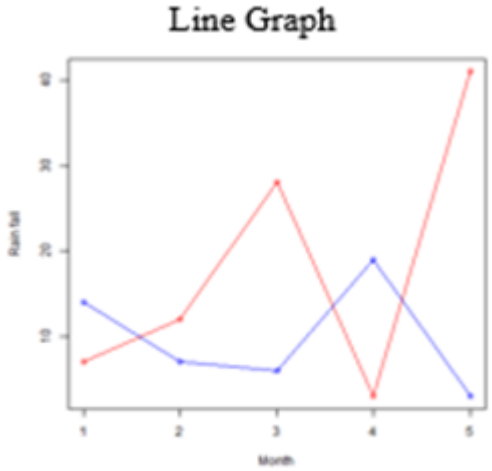
```
# Get the input values.
input <- mtcars[,c('wt','mpg')]
# Give the chart file a name.
png(file = "scatterplot.png")
# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15
and 30.
plot(x = input$wt,y = input$mpg,
    xlab = "Weight",
    ylab = "Milage",
    xlim = c(2.5,5),
    ylim = c(15,30),
    main = "Weight vs Milage"
)
```

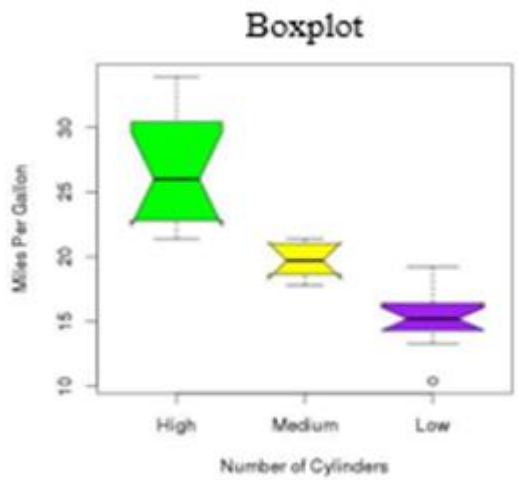


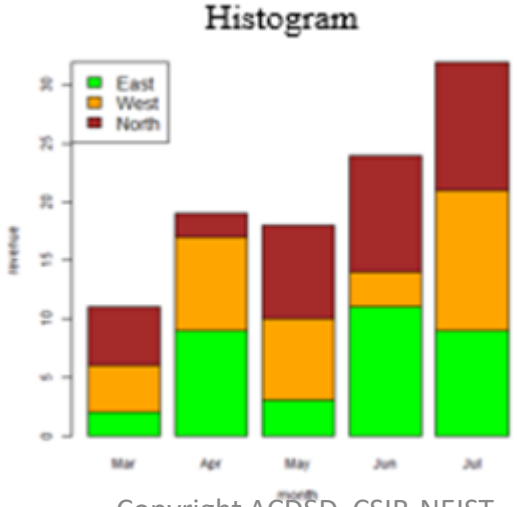
1.11.8 R Statistics Examples

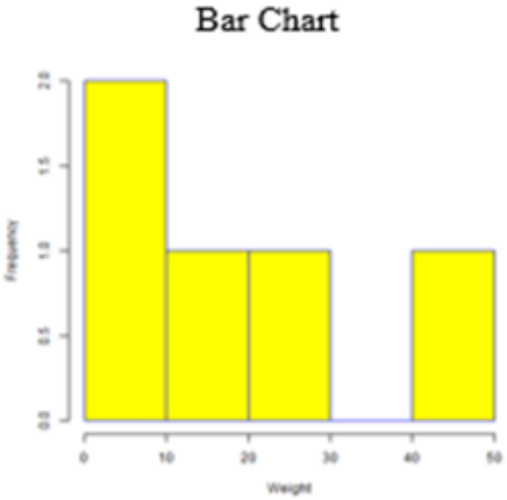
- R Programming language has numerous libraries to create different charts and graphs.











1.11.8 R Statistics Examples

- Scatterplots show many points plotted in the Cartesian plane.
- Each point represents values of two variables - One variable is chosen in horizontal and other in vertical axis.

Syntax: plot(X, Y, xlab, ylab, xlim, ylim, main)

Syntax: plot(x = D$X, y = D$Y, xlab = "Domain", ylab = "Members", xlim = c(2.5,5), ylim = c(15,30), main = "ACDS Team")

X = c(3, 7, 5, 4)

Y = c(37, 72, 54, 46)

plot(x = X, y = Y, xlab = "Domain", ylab = "Members", xlim = c(2.5,7), ylim = c(35,80), main = "ACDS Team", las=1, col=2, cex=2, pch=8)

pairs(~X+Y+Z+K, data = mtcars, main = "Scatterplot Matrix")

1.11.8 R Statistics Examples

- A line graph is a line that connects points by drawing line segments between them.
- Line charts are usually used in identifying the trends in data.

Syntax: plot(X, Y, type = “l", xlab, ylab, main)

Syntax: plot(DX, DY, xlab = "Domain", ylab = "Members", main = "ACDSD Team")

```
X= c()
Y = c(7,12,28,3,41)
plot(X, Y, xlab = "Domain", ylab = "Members", main = "ACDSD Team", type = "o", col = "red")
```

1.11.8 R Statistics Examples

- Boxplots are a measure of how well distributed is the data in a data set.
- It divides the data set into three quartiles.
- It represents minimum, maximum, median, first quartile and third quartile in the data set.

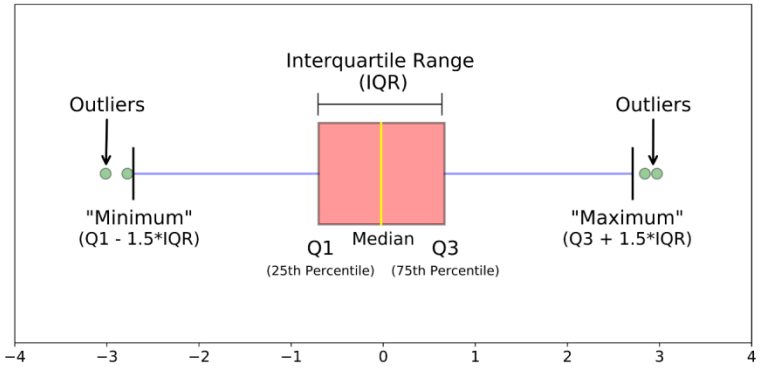
Syntax: boxplot(x, data, notch, varwidth, names, main)

Syntax: boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main = "Mileage Data")

X = c(54, 67, 89, 31, 53, 68, 88, 34)

Y = c(7, 12, 4, 6, 9, 13, 3, 8)

boxplot(X ~ Y, xlab = "Age", ylab = "Members", main = "ACDS Team")



1.11.8 R Statistics Examples

- A pie-chart is a representation of values as slices of a circle with different colours.
- The slices are labelled and the numbers corresponding to each slice is also represented.

Syntax: pie(P, labels, main)

Syntax: pie(D$P, labels = D$lbls, main = "ACDSD Team")

```
P = c(21, 62, 10, 53)
lbls= c("Biology", "Chemistry", "IT", "Bioinformatics")
pie(P, labels=lbls, main = "ACDSD Team", col = rainbow(length(P)))
pie3D(P, labels = lbls, main = "ACDSD Team", explode = 0.1)  #library(plotrix)
pie(P, labels = round(100*P/sum(P)), main = "ACDSD Team", col = rainbow(length(P)))
legend("topright", c("Biology", "Chemistry", "IT", "Bioinformatics"), cex= 0.8, fill =
rainbow(length(P)))
```

1.11.8 R Statistics Examples

- A histogram represents the frequencies of values of a variable bucketed into ranges.
- Similar to bar chat but difference is it groups the values into continuous ranges.
- Each bar in histogram represents the height of the number of values present in that range.

Syntax: hist(H, xlab, ylab, xlim, ylim, main)

Syntax: hist(D$H, xlab = "Weight", main="ACDSD Team")

H = c(36,22,12,41,31,33,19, 90, 65)

hist(H, xlab = "Weight", main="ACDSD Team", col = "green", border = "red", xlim = c(0,40), ylim = c(0,5))

1.11.8 R Statistics Examples

- Represents data in rectangular bars with length of bar proportional to value of variable.
- R can draw both vertical and horizontal bars in the bar chart.

Syntax: **barplot(names.arg=X, Y, xlab, ylab, main)**

Syntax: **barplot(names.arg=DX, DY, xlab="Domain", ylab="Members", main="ACDSD Team")**

```
X = c("Biology", "Chemistry", "IT", "Bioinformatics")
Y = c(7, 12, 4, 6)
barplot(names.arg=X, Y, xlab="Domain", ylab="Members", main="ACDSD Team", col="blue", border="red")
```

1.11.9 R Statistics Summary


```
plot(x = X, y = Y, xlab="Domain", ylab="Members", xlim=c(2,5), ylim=c(15,30))

plot(L, xlab = "Domain", ylab = "Members")

boxplot(mpg ~ cyl, data = mtcars, xlab = "NoC", ylab = "MPG")

pie(P, labels = lbls)

hist(H, xlab = "Weight", ylab="Frequency")

barplot(names.arg=X, Y, xlab="Domain", ylab="Members")
```

1.12 Python

Introduction

- Used on a server to create web applications.
- Major Python versions: Python2 and Python 3.

History

- It was initially designed by Guido Van Rossum in 1991 and developed by Python Software Foundation.

Uses

- Machine Learning
- Web Frameworks
- Image Processing
- Multimedia
- Text Processing

Applications

- GUI based Desktop applications(Games, Scientific applications).
- Web frameworks and applications.
- Enterprise and Business applications
- Operating Systems
- Language Development
- Prototyping

Python Features

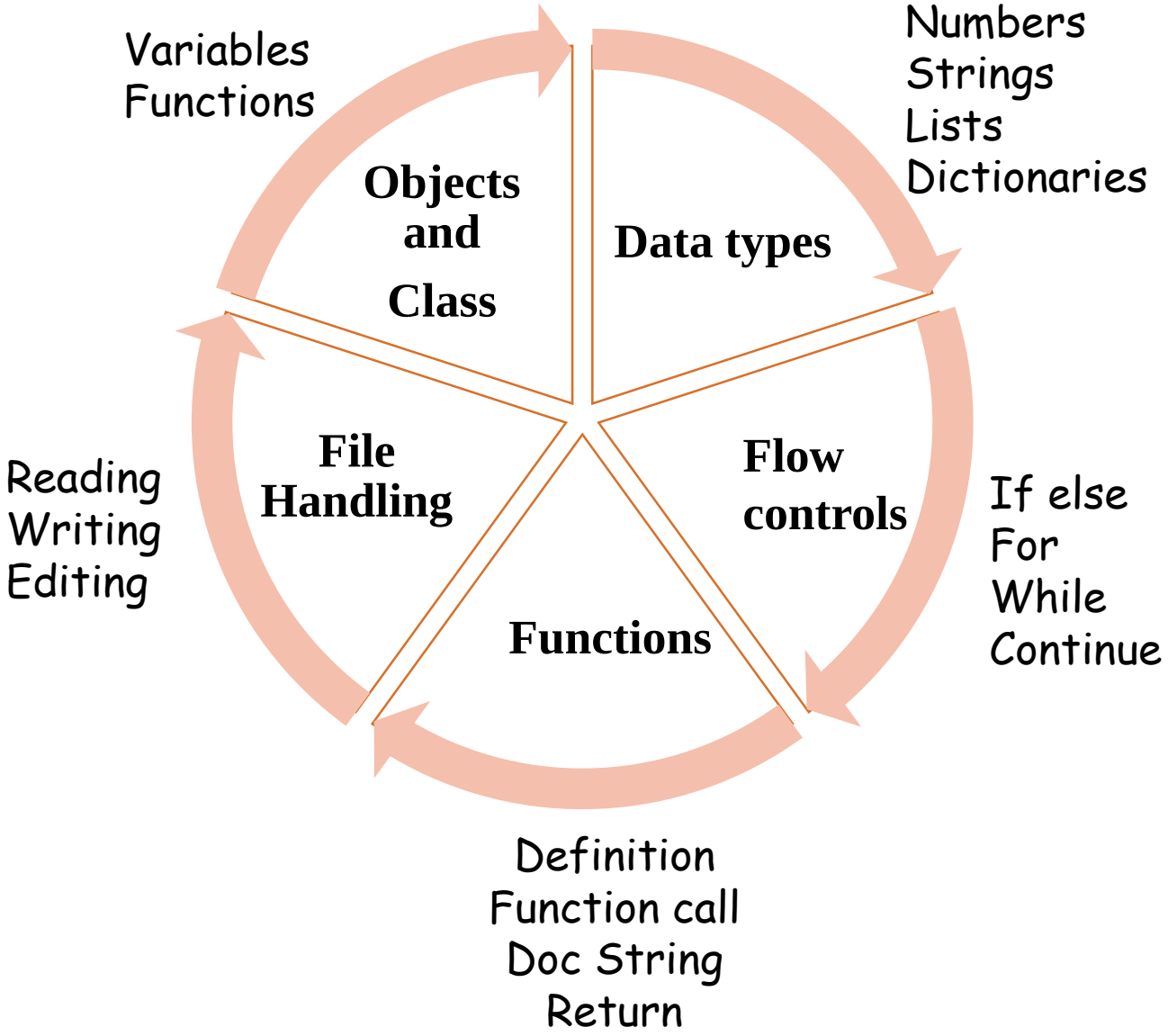
- **Simple and Open source –** One of the simplest and open source.
- **Interpreted** − Processed at runtime by the interpreter, no need to compile before executing it.
- **Dynamic Type Checking –** Type of variable known at run time i.e. type need not be declared.
- **Object-Oriented** − Supports Object-Oriented programming that encapsulates code within objects.
- **Integrated** – Can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python Editions

- Python 1.0 – 1991
- Python 2.0 – 2000
- Python 3.0 – 2008 (python 3.7 - latest)
- Stable Version – Python 3.7.4 (2019)

Common IDEs for Python

- PyCharm (Text Editor)
- Atom (Text Editor)
- Jupyter Notebook
- Spyder



Library	Stands For	Purpose
Numpy	Numerical Python	N-dimensional Array and Matrices
Scipy	Scientific Python	Linear algebra, Integration-Differential equation, Optimization
Pandas	Python Data Analysis (Panel Data)	Data frame (Operation on CSV file by creating data frame)
Matplotlib	Matlab Plot Library	Visualization (plot)
Tkinter	Tk Interface	GUI Programming

1.12.3 Flow Control

- Need to control the flow of normal sequential execution of the program many times.

- There are two types of control statements –
  - **Decision Statements** - Execute certain commands only when certain conditions satisfied.

  - **Loop Statements -** Execute certain commands repeatedly with some logic to stop it.
  - **Loop Control Statements -** Control the normal flow of loops by introducing some keyword.

Note: Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Decision Statements

If statement:

```
if(boolean_expressions):
        # statement(s) will execute if the Boolean expressions are true
```

If-else statement:

```
if(boolean_expressions):
        # statement(s) will execute if the boolean expression is true
else:
        # statement(s) will execute if the boolean expression is false
```

If-else-if Statement:

```
if(boolean_expression 1):
        # Executes when the boolean expression 1 is true
elseif( boolean_expression 2):
        # Executes when the boolean expression 2 is true
else:
        # executes when none of the above condition is true
```

Loop Statements

- Sometimes, need to execute a block of code several number of times.
- Python provides the following kinds of loop to handle looping requirements -

For loop: for i in range[k]:

#statements

While loop: while (test_expression)

#statements

Loop Control Statements

Break Statement (*keyword: break****)***

- Terminates loop statement and transfers execution to the statement immediately following the loop.

Continue Statement (*Keyword: continue****)***

- Causes loop to skip the remainder portion and immediately retest its condition prior to reiterating.

Pass statement (*Keyword: pass****)***

- Used when a statement is required syntactically but do not want any command or code to execute.

Examples

if-else →

```
i=5
if (i<10):
        print ("he")
else:
        print (“me")
```

for →

```
for i in range[10]:
        print (i)
```

while →

```
i=10
while (i<100) :
        print (i)
        i=i+2
```

Repeat →

```
count=2
repeat{
        print("hello")
        count=count+1
        if (count>7)
        {break}
        }
```

Next →

```
v =LETTERS[1:6]
for ( i in v){
        if (i == "D")
        {next}
        print(i)
        }
```

Function

- A function is a set of statements organized together to perform a specific task.
- Python has many **in-built** functions and also users can create their own **user-defined** functions.

Function Definition

```
def function_name (arg_1, arg_2, ...):
                    #function body
                    return expression # optional statement
```

Syntax

```
def functionname(parameters):
        function_suite
        return expression
```

Example: #Following function takes a string as input parameter and prints it

```
def printme(str):
        "This prints a passed string into this function"
        print str
        return
```

Function

- Once the basic structure of a function is finalized, we can execute it by calling it from another function or directly from the Python prompt.

```
# Function definition is here
def printme(str):
        "This prints a passed string into this function"
        print str
        return

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Function

In Built Library Function

```
import random
random.random()
random.randrange (10,20)
random.randint(10,20)
```

User Defined Function

```
def sum() :
        a =  random.random()
        b =  random.random()
        return a+b                    #Functions in Python takes multiple inputs but return only one
```

Function

Example 1

```
def sum() :
        a =  random.random()
        b =  random.random()
        return a+b          #Functions in Python takes multiple inputs but return only one output
sum()
```

Example 2

```
def numbers(a, b) :
        sum =  a+b
        substract =  a-b
        return [sum, subtract]        #list, tuple, dictionary are used to return multiple outputs
numbers()
```

String

- A contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings taken using slice operator ([ ] & [:]) with indexes starting at 0 of string beginning

- Python does not support a character type; these are treated as strings of length one.

```
S1 = 'Hello World!'
S2 = "Python Programming“
word = 'word'
sentence = "Multiple Words"
paragraph = """Multiple Sentences"""
```

String

Operator	Description
+	Concatenation
*	Repetition
[]	Slice
[:]	Range Slice
in	Membership
not in	Membership
r/R	Raw String
{}, .format()	Format - ***Performs String formatting***

```
print ("My name is {} and weight is {} kg!"  .format ('Ranjan',66))
```

List

- List is a ordered collection of different types non-distinct objects.
- Enclosed by []
- Mutable – Once created they can be modified.

```
L1= [10,20,30, 'Ranjan', 'Rohitesh', 'Barnali']
```

```
L= []
L.append (10,20,30, 'Ranjan', 'Rohitesh', 'Barnali')
```

```
id = [1,2,3,4]
name = ['Ranjan', 'Rohitesh', 'Barnali']
L2 = [id, name]
```

Dictionary

- Unordered collection of objects of different types (holds **key:value** pair)
  - Keys must be unique and *immutable* data type such as strings, numbers and tuples.
  - Key-values can be repeated and be of any type like strings, numbers and tuples as well as list.

- Its like Hash table where key-value pairs map to values.

```
D1 = {key1 : val1 , key2 : val2}
D2 = {   'ID' : [1, 2]
         'Names' : ['Ranjan', 'Rohitesh']
         'DOB' : ['6/11/1991', '6/7/1989']
         }
```

Note – Dictionary keys are case sensitive, same name but different cases of Key treated distinctly.

1.12.6 Python Database Interaction


```
#Connect to Database
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW DATABASES")
mycursor.execute("CREATE DATABASE mydatabase")


#try to access the database when making the connection
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

1.12.6 Python Database Interaction


```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255),
    address VARCHAR(255))")
```

Basic Operational Library

Library	Stands For	Purpose
Numpy	Numerical Python	N-dimensional Array and Matrices
Scipy	Scientific Python	Linear algebra, Integration-Differential equation, Optimization
Pandas	Python Data Analysis (Panel Data)	Data frame (Operation on CSV file by creating data frame)
Matplotlib	Matlab Plot Library	Visualization (plot)
Tkinter	Tk Interface	GUI Programming

Machine Learning and Data Science Libraries

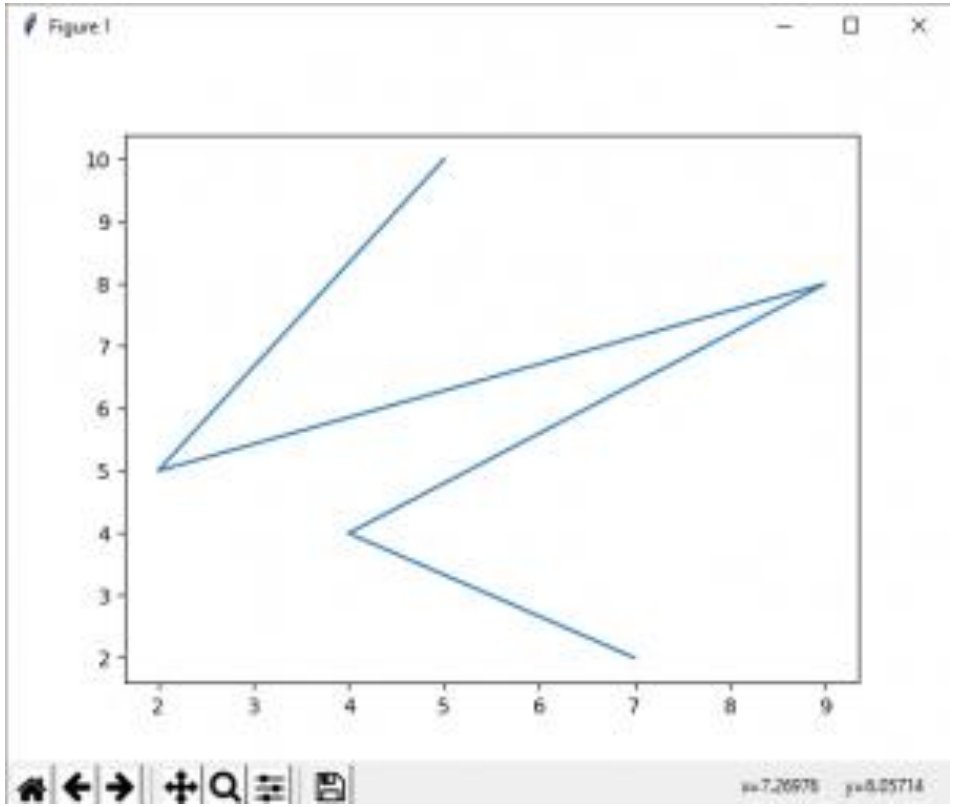
Library	Purpose
Scikit-learn (Sklearn), Pytorch	Machine Learning
Caffe, Keras, Theano, Tensorflow	Deep Learning
OpenCV	Computer Vision
NLTK	Natural Language Processing
NetworkX	Network Science

Matplotlib

- Matplotlib is an visualization library in Python for 2D plots of arrays.
- It allows visual access to huge amounts of data in easy visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Example

```
# importing matplotlib module
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
 # Y-axis values
y = [10, 5, 8, 4, 2]
 # Function to plot
plt.plot(x,y)
  # function to show the plot
plt.show()
```



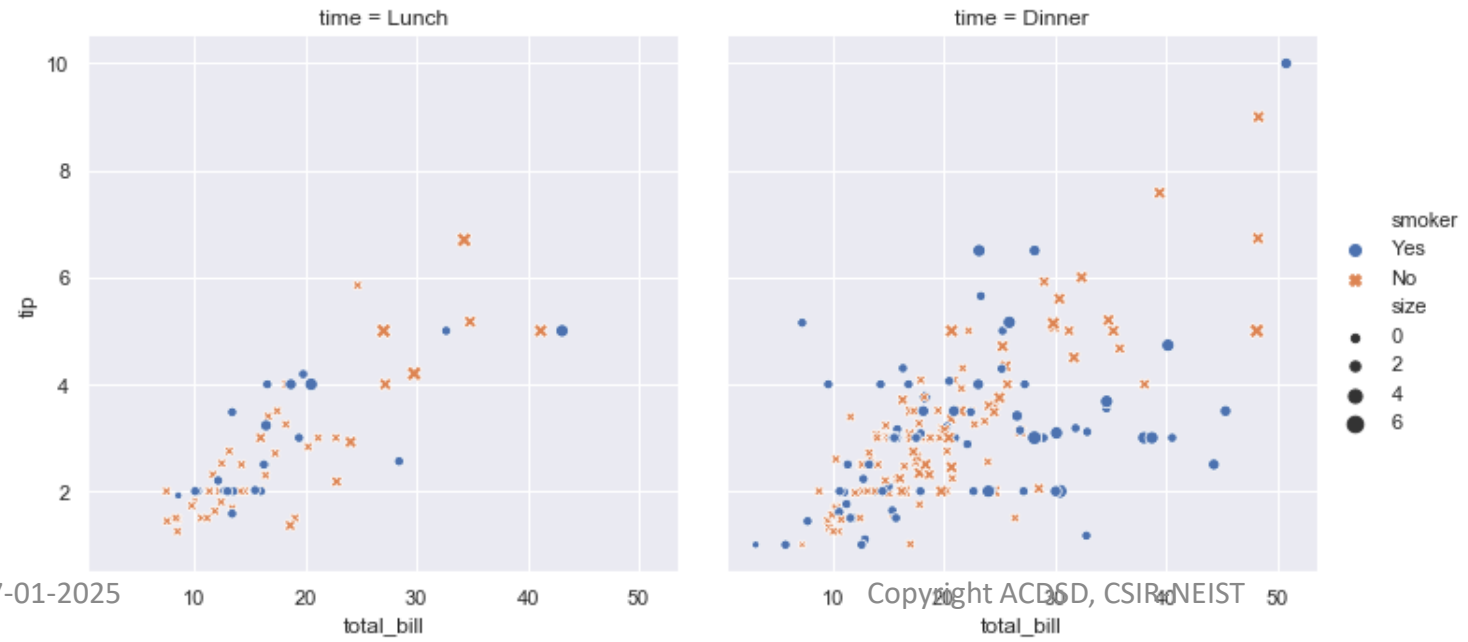
Ref: https://www.geeksforgeeks.org/python-introduction-matplotlib/

Seaborn

- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

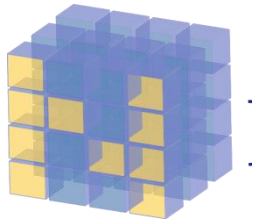
Example:

```
import seaborn as sns
sns.set()
tips = sns.load_dataset("tips")
sns.relplot(x="total_bill", y="tip", col="time", hue="smoker", style="smoker", size="size", data=tips);
```



Ref: https://seaborn.pydata.org/introduction.html

NumPy



NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

- **Interactive:** Numpy is very interactive and easy to use.
- **Mathematics:** Makes complex mathematical implementations very simple.
- **Intuitive:** Makes coding real easy and grasping the concepts is easy.
- **Lot of Interaction:** Widely used, hence a lot of open source contribution.
- It can be utilized for expressing images, sound waves, and other binary raw streams as an array of real numbers in N-dimensional.

NumPy

```
# Python program to demonstrate  # array creation techniques
import numpy as np
# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)
# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)
# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)
# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s."
        "Array type is complex:\n", d)
```


Ref: https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/

TensorFlow

- TensorFlow works like a computational library for writing new algorithms that involve a large number of tensor operations, since neural networks can be easily expressed as computational graphs they can be implemented using TensorFlow as a series of operations on Tensors.
- **Responsive Construct:** With TensorFlow, we can easily visualize each and every part of the graph which is not an option while using *NumPy* or *SciKit*.

2. **Flexible: One** of the very important Tensorflow Features is that it is flexible in its operability, meaning it has modularity and the parts of it which you want to make standalone, it offers you that option.
3. **Easily Trainable:** It is easily trainable on CPU as well as *GPU* for distributed computing.
4. **Parallel Neural Network Training:** TensorFlow offers pipelining in the sense that you can train multiple *neural networks* and multiple GPUs which makes the models very efficient on large-scale systems.

Scikit Learn

The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Features Of Scikit-Learn

1. **Cross-validation:** There are various methods to check the accuracy of supervised models on unseen data.
2. **Unsupervised learning algorithms:** Again there is a large spread of algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.
3. **Feature extraction:** Useful for extracting features from images and text (e.g. Bag of words).

- It contains a numerous number of algorithms for implementing standard machine learning and data mining tasks like reducing dimensionality, classification, regression, clustering, and model selection.

Scikit Learn

```
# Sample Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```


Ref: https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library

Keras

It provides an easier mechanism to express neural networks. Keras also provides some of the best utilities for compiling models, processing data-sets, visualization of graphs.

Features of Keras

- It runs smoothly on both CPU and GPU.
- Keras supports almost all the models of a neural network – fully connected, convolutional, pooling, recurrent, embedding, etc. Furthermore, these models can be combined to build more complex models.
- Keras, being modular in nature, is incredibly expressive, flexible, and apt for innovative research.
- Keras is a completely Python-based framework, which makes it easy to debug and explore.
- Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers and a host of tools to make working with image and text data easier.

PyTorch

- It allows developers to perform tensor computations with ith acceleration of GPU, creates dynamic computational graphs, and calculate gradients automatically. It also offers rich APIs for solving application issues related to neural networks.
- **Hybrid Front-End:** A new hybrid front-end provides ease-of-use and flexibility in eager mode, while seamlessly transitioning to graph mode for speed, optimization, and functionality in C++ runtime environments.
- **Libraries and Tools :** An active community of researchers and developers have built a rich ecosystem of tools and libraries for extending PyTorch and supporting development in areas from computer vision to reinforcement learning.
- PyTorch is primarily used for applications such as natural language processing.

SciPy

- SciPy library contains modules for optimization, linear algebra, integration, and statistics.
- The main feature of SciPy library is that it is developed using NumPy, and its array makes the most use of NumPy.
- It provides all the efficient numerical routines like optimization, numerical integration, and many others using its specific sub modules.
- SciPy is a library that uses NumPy for the purpose of solving mathematical functions. SciPy uses NumPy arrays as the basic data structure, and comes with modules for various commonly used tasks in scientific programming like linear algebra, integration (calculus), ordinary differential equation solving and signal processing.

Example-

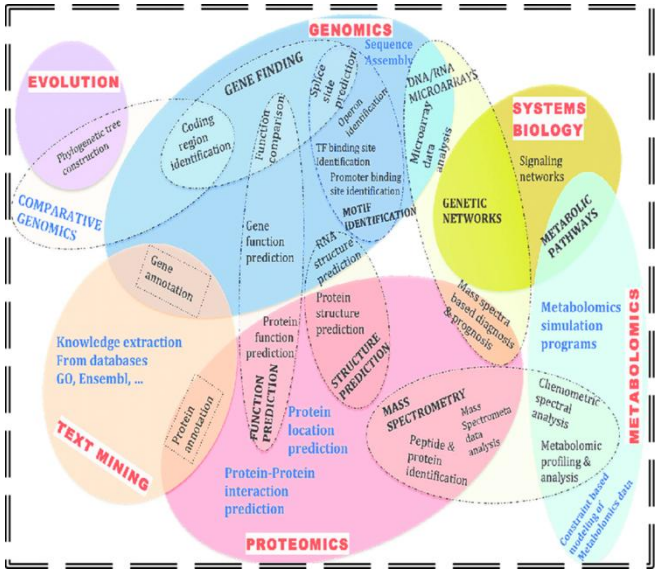
$$\int_0^1 12x\,dx$$

```
import scipy.integrate
f= lambda x: 12*x
i = scipy.integrate.quad(f, 0, 1)
print (i)
```

Ref: https://intellipaat.com/blog/tutorial/python-tutorial/python-scipy/

1.12.7 Machine Learning

- Machine learning is a method of data analysis that automates analytical model building.
- Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.



Data → Data Pre-processing (Cleaning; Transformation; Data Reduction) → Feature Selection → Learning (Supervised; Unsupervised; Reinforcement; Deep Learning) → Analysis (Outliers; Overfitting; Validation) → Model

1.12.8 Examples

1. Python script to read a given CSV files with initial spaces after a delimiter and remove those initial spaces.

```
import csv
print("\nWith initial spaces after a delimiter:\n")
with open('departments.csv', 'r') as csvfile:
   data = csv.reader(csvfile, skipinitialspace=False)
   for row in data:
     print(', '.join(row))
print("\n\nWithout initial spaces after a delimiter:\n")
with open('departments.csv', 'r') as csvfile:
   data = csv.reader(csvfile, skipinitialspace=True)
   for row in data:
     print(', '.join(row))
```

2. Python script that reads each row of a given csv file and skip the header of the file. Also print the number of rows and the field names.

```
import csv
fields = []
rows = []
with open('departments.csv', newline='') as csvfile:
 data = csv.reader(csvfile, delimiter=' ', quotechar=',')
fields = next(data)
 for row in data:
   print(', '.join(row))
print("\nTotal no. of rows: %d"%(data.line_num))
print('Field names are:')
print(', '.join(field for field in fields))
```

1.12.8 Examples

3. Python script to append text to a file and display the text.

```
def file_read(fname):
        from itertools import islice
        with open(fname, "w") as myfile:
                myfile.write("Python Exercises\n")
                myfile.write("Java Exercises")
        txt = open(fname)
        print(txt.read())
file_read('abc.txt')
```

4. Python script to find the longest words.

```
def longest_word(filename):
    with open(filename, 'r') as infile:
            words = infile.read().split()
    max_len = len(max(words, key=len))
    return [word for word in words if len(word) ==
max_len]
print(longest_word('test.txt'))
```

5. Python script to count the frequency of words in a file.

```
from collections import Counter
def word_count(fname):
        with open(fname) as f:
                return Counter(f.read().split())
print("Number of words in the file
:",word_count("test.txt"))
```

6. Python script to copy the contents of a file to another file .

```
from shutil import copyfile
copyfile('test.py', 'abc.py')
```

7. Python script to write a Python dictionary to a csv file. After writing the CSV file read the CSV file and display the content.

```
import csv
csv_columns = ['id','Column1', 'Column2', 'Column3', 'Column4', 'Column5']
dict_data = {'id':['1', '2', '3'],
    'Column1':[33, 25, 56],
    'Column2':[35, 30, 30],
    'Column3':[21, 40, 55],
    'Column4':[71, 25, 55],
    'Column5':[10, 10, 40], }
csv_file = "temp.csv"
try:
   with open(csv_file, 'w') as csvfile:
       writer = csv.DictWriter(csvfile, fieldnames=csv_columns)
       writer.writeheader()
       for data in dict_data:
           writer.writerow(dict_data)
except IOError:
   print("I/O error")
data = csv.DictReader(open(csv_file))
print("CSV file as a dictionary:\n")
for row in data:
   print(row)
```

1.12.8 Examples

8. Python script to count the number of rows of a given SQLite table

```
import sqlite3
from sqlite3 import Error
def sql_connection():
    try:
      conn = sqlite3.connect('mydatabase.db')
      return conn
    except Error:
      print(Error)

def sql_table(conn):
    cursorObj = conn.cursor()
# Create the table
    cursorObj.execute("CREATE TABLE
salesman(salesman_id n(5), name char(30), city char(35),
commission decimal(7,2));")
    print("Number of records before inserting rows:")
    cursor = cursorObj.execute('select * from salesman;')
    print(len(cursor.fetchall()))
# Insert records
    cursorObj.executescript("""
    INSERT INTO salesman VALUES(5001,'James Hoog',
'New York', 0.15);
INSERT INTO salesman VALUES(5002,'Nail Knite',
'Paris', 0.25);
    INSERT INTO salesman VALUES(5003,'Pit Alex',
'London', 0.15);
    INSERT INTO salesman VALUES(5004,'Mc Lyon',
'Paris', 0.35);
    INSERT INTO salesman VALUES(5005,'Paul Adam',
'Rome', 0.45);
    """)
    conn.commit()
    print("\nNumber of records after inserting rows:")
    cursor = cursorObj.execute('select * from salesman;')
    print(len(cursor.fetchall()))

sqllite_conn = sql_connection()
sql_table(sqllite_conn)

if (sqllite_conn):
  sqllite_conn.close()
  print("\nThe SQLite connection is closed.")
```

1.12.8 Examples

9. Python script to create a SQLite database and connect with the database and print the version of the SQLite database.

```
import sqlite3
try:
   sqlite_Connection = sqlite3.connect('temp.db')
   conn = sqlite_Connection.cursor()
   print("\nDatabase created and connected to SQLite.")
   sqlite_select_Query = "select sqlite_version();"
   conn.execute(sqlite_select_Query)
   record = conn.fetchall()
   print("\nSQLite Database Version is: ", record)
   conn.close()
except sqlite3.Error as error:
   print("\nError while connecting to sqlite", error)
finally:
   if (sqlite_Connection):
      sqlite_Connection.close()
      print("\nThe SQLite connection is closed.")
```

10. Python script to connect a database and create a SQLite table within the database.

```
import sqlite3
from sqlite3 import Error
def sql_connection():
   try:
     conn = sqlite3.connect('mydatabase.db')
     return conn
   except Error:
     print(Error)
def sql_table(conn):
   cursorObj = conn.cursor()
   cursorObj.execute("CREATE TABLE
agent_master(agent_code char(6),agent_name
char(40),working_area char(35),commission
decimal(10,2),phone_no char(15) NULL);")
   print("\nagent_master file has created.")
   conn.commit()
sqllite_conn = sql_connection()
sql_table(sqllite_conn)
if (sqllite_conn):
 sqllite_conn.close()
 print("\nThe SQLite connection is closed.")
```

Problem Description:

The program takes in the marks of 5 subjects and display the grade.

Problem Solution:

1. Take in the marks of 5 subjects from the user and store in different variables.
2. Find the average of the marks.
3. Use an else condition to decide the grade based on the average of the marks.
4. Exit.

1.12.8 Examples


```
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
avg=(sub1+sub2+sub3+sub4+sub4)/5
if(avg>=90):
  print("Grade: A")
elif(avg>=80&avg<90):
  print("Grade: B")
elif(avg>=70&avg<80):
  print("Grade: C")
elif(avg>=60&avg<70):
  print("Grade: D")
else:
  print("Grade: F")
```

Run time Test Cases

```
Enter marks of the first subject: 85
Enter marks of the second subject: 95
Enter marks of the third subject: 99
Enter marks of the fourth subject: 93
Enter marks of the fifth subject: 100
Grade: A

Enter marks of the first subject: 81
Enter marks of the second subject: 72
Enter marks of the third subject: 94
Enter marks of the fourth subject: 85
Enter marks of the fifth subject: 80
Grade: B
```

1.13 References

1) Scott, M.L. Programming Language Pragmatics. *Elsevier/Morgan Kaufmann*, **2006.**

This book is an excellent treatment of programming languages and their implementation. It has been the first exposure to the field of computer science. It has given different types of exercise plan for each and every topic. ★ ★ ★

2) Knuth, D.E. The Art of Computer Programming. *Addison-Wesley*,**1998**.

There are four volumes of this book are available. The fundamental algorithms and a brief explanation of all algorithms have been addressed properly. ★ ★ ★ ★

3) Klemens, B. 21st Century C. *OReilly Media*, *Inc.*, **2015**.

The C programming with all the exercises have been described in this book. ★ ★

4) Monk, S. Programming arduino: Getting started with sketches. *McGraw-Hill education*, **2016**.

5) Deitel, P. , Deitel, H. C for Programmers with an Introduction to C11 (Deitel Developer Series). *Prentice Hall*, **2013**.

6) Patt, Y. N., Patel, S. J. Introduction to computing systems: from bits and Gates to C and beyond. *Boston: McGraw-Hill Higher Education*,**2004.**

7) Lee, J., et al. Beginning Perl. *Apress*, **2004.**

8) Cozens, S., Peter, W. Beginning Perl. *Wrox Press*, **2002.**

9) Schwartz, R. L., et al. Intermediate Perl. *OReilly*, **2012.**

10) Kanetkar, Y.P, Let Us C, *BPB Publications*, **2018.**

Web Links

- https://www.tutorialspoint.com/r/index.htm
- https://www.datamentor.io/r-programming/
- https://www.learnpython.org/
- https://github.com/python#start-of-content

THANK YOU