Q what do you mean by evolution of data type concept?

Early languages, such as FORTAN and COBOL, limit the creation of new data types to subprograms definations. As the concept of data type has evolved, new language designs have provided better facilities for specifing and implementing entire abstract data types, such as the Ada Package and C++ or Java class.

Higher level languages provide a set of basic data types, such as real, integer and character string. Type checking is provided to ensure that operations such as + or X are not applied to data of the wrong type. The early notion of data types defines a data type as a set of values that a variable might take on. Data types are directly associated with individual variables so that each declaration means a variable and defines its types. If a program uses several arrays, each containing 20 real numbers, each array is declared seprately and the entire array description is repeated for each.

Around 1970, Pascal extended the concept to a general type defination applicable to a set of variables. A type defination gives the structure of a data type with its possible value bindings. To get a particular data object of the defined type, a declaration requires only the variable name and the name of the type to be given.

During the 1970s, the concept of data type was extended beyond just a set of data objects to also include, the set of operations that manipulate those data objects. For primitive types, such as real and integer, a language provides a facility to declare variables of that type and a set of operations on reals and integers that represent the only way that reals and integer can be manipulated by the programmers, Thus, the storage represent of reals and integers is effectively encapsulated. All the programs sees is the name of the type and the list of operations available for manipulating data objects of that type.

Q What do you mean by Abstraction, Encapsulation and Information Hiding? Explain briefly

ABSTRACTION

Abstraction is the process of taking away or removing characterstics from something in order to reduce it TO a set of essential characterstics. In object-oriented programming, abstraction is one of the three principles. Though the process of abstraction, a programmer hides all the three principles. Though the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase effeciency. The resulting object can be refred to as an abstraction, meaning a named entity made up of selected attributes and behavior specific to a particular usage of the originating entity. In the process of abstraction, the programmer tries to ensure that the entity is named in a manner that will make sense and that it will have all the relevant aspects included and none of the extraneous ones.

ENCAPSULATION

Encapsulation is binding the code and data together to hide the complexity of a class. Encapsulation has less to do with access specifiers (private, public, protected). In encapsulation, member inside a class be private, public or protected.

The private members of a class are only accessible to the objects of that class only, and the public members are accesible to the objects of the class as well as they are accessible to outside the class.

..A HIDING

Data Hiding is a concept in object-oriented programming which confirms the security of members of a class from unauthorized access. Data Hiding is a technique of protecting the data members from being manipulated or hacked from any other source. Data is the most sensitive and volatile content of a program which if manipulated can result in an incorrect output and it also harms the integrity of the data.

Data Hiding is controlled with the help of access specifiers (private, public or protected). The data which is accessible from outside the class hence if you want to hide your data or restrict it from being accessed from outside, declare your data private. Private data is only accessible to the objects of that class only.

Key differences between data Hiding and Encapsulation

		DATA HIDING	ENCAPSULATION
1.	Basic	Data Hiding concerns about data security along with hiding complexity.	Encapsulation concerns about wrapping data to hide the complexity of a system.
2.	Focus	Data Hiding focuses on restricting or permitting the data inside the capsule.	Encapsulation focuses on enveloping or wrapping the complex data.
3.	Access specifier	The Data under data hiding is always private and inaccessible.	The data under it maybe private or public.
4.	Process	Data Hiding is process as well as technique	Encapsulation is a sub-process in data hiding.

Q Explain sub-program.

A subprogram is an abstract operation defined by the programmer. Subprograms form the basic building block of which most programs are constructed, and facilities for their defination and innvocation are found in almost every language. Two views of subprograms are here.

At the program design level, a subprogram represents an abstract operation that the programmer defines, as opposed to the primitive operations that are build into the language.

At the language design level, the concern is with the design and implementation of the general facilities for subprogram defination and invocation.

A subprogram defination has two parts: a specification and an implementation. However, for a subprogram, both parts are provided by the programmer when the subprogram is defined.

Specification of a subprogram :- Because a subprogram represents an abstract operation, we should be able to understand its specification without understanding how it is implemented. The specification for a subprogram is the same as that for a primitive operation. It includes :-

1) The name of a subprogram

2). The signature (also called prototype) of the subprogram giving the number of arguments, their order, and the data type of each, as well as number of results, their order, and the data type of each; and

3) The action performed by the subprogram (i.e a description of the the function it computes)

ementation of sub- program

Sub program represents an operation of the virtual computer layer constructed by the programmer, and thus a subprogram is implemented using the data structures and operations provided by the programming language. The implementation is defined by the subprogram body, which consist of local data declarations defining the data structures used by the subprograms and statements defining the actions to be taken when the subprogram is executed. The declarations and statements are usually encapsulated so that neither the local data ~~more~~ than statements are accessible seperately to the user of the subprogram; the user may only invoke the subprograms with a particular set of arguments and receive the computed results.

Topic — Overloaded Subprograms

A subprogram is a program inside any larger program that can be reused any number of times.

An overloaded subprogram is a subprogram that has the same name as another subprogram in the same code/program. A subprogram must be different from the others in the number, order, or types of its parameters, and possibly in its return type if it is a function.
The meaning of a call to an overloaded subprogram is determined by actual parameters list.
Overloaded subprograms ~~can~~ have same name but not necessary have same process.

For example overloaded subprogram in C++:

```
#include <iostream>

 using namespace std;

void show (int a)
{
     cout << a;
 }
```

```
void show (int a, int b)
{
    cout << a << endl;
    cout << "*" << b;

}

int main ()
{
    int p = 20;
    int q = 50;
    show (p);
    show (p, q);
    return 0;
}
```

In this program function [illegible] 'show' is overloaded. Both function has different number of arguments / paraments.
Hence, it shows overloaded subprogram

Topic — Generic Subprograms :

Subprograms are of two types :-
(i) Procedures (ii) Functions.

A generic subprogram is a subprograms which have parametric polymorphism.
A generic subprogram can accept different types of values of same single memory location.
Parametrically polymorphic subprograms are often called generic subprograms.
C++ provide a kind of compile - time parametric polymorphism
For example :

```
#include <iostream>
using namespace std;

template <class T>
void show (T a)
{
    Cout << a ;
}
int main()
{
    show (10);            // calls program for int
    show ('a');           // calls program for char
    show ("22.150"); // calls program for double
    return 0;
}
```