**Q.1Imperative Programming:-**

 Imperative programming is the procedure of writing parts of a program while it is already active. ... It thus forms a specific instance of Imperative computation as an extreme opposite to batch processing, where neither writing the program nor its use happens in an Imperative way.Imperative programming is the cornerstone of computing. CPUs primarily work as imperative execution engines, and compilers translate into this language.

Statement-1

Statement-2

.

.

.

Statement-2

**Sequence**

Most prominent to imperative programming is the sequential flow of instructions and branching.

|  |  |
| --- | --- |
| 123456 | var a = 5;var b = calculate(a);if( b > 10 ) { print("Too big"); return;} |

**Mutable variables**

This sequencing is dependent on mutable variables. Variables represent a storage location, not just a value. Bare with me through a few trivial examples, there’s a point to this.

|  |  |
| --- | --- |
| 12 | int a;a = 5; |

The variable a contains an integer. At first it is undefined, or has a default value, depending on the language. We assign the value 5 to it. a now contains a copy of this value, wholly unrelated from the constant value 5.

**Q.1 Object Oriented Programming:-**

 **O**bject-**o**riented **p**rogramming (**OOP**) refers to a type of computer programming (software design) in which [programmers](http://www.webopedia.com/TERM/P/programmer.html) define not only the [data type](http://www.webopedia.com/TERM/D/data_type.html) of a [data structure](http://www.webopedia.com/TERM/D/data_structure.html), but also the types of operations ([functions](http://www.webopedia.com/TERM/F/function.html)) that can be applied to the data structure.

In this way, the data structure becomes an [object](http://www.webopedia.com/TERM/O/object.html) that includes both [data](http://www.webopedia.com/TERM/D/data.html) and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

## The Basic OOP Concepts

* [**Abstraction**](http://www.webopedia.com/TERM/A/abstraction.html)**:** The process of picking out (abstracting) common features of objects and procedures.
* [**Class**](http://www.webopedia.com/TERM/C/class.html)**:** A category of objects. The class defines all the common properties of the different objects that belong to it.
* [**Encapsulation**](http://www.webopedia.com/TERM/E/encapsulation.html)**:** The process of combining elements to create a new entity. A procedure is a type of encapsulation because it combines a series of computer instructions.
* [**Information hiding**](http://www.webopedia.com/TERM/I/information_hiding.html)**:** The process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity.
* [**Inheritance**](http://www.webopedia.com/TERM/I/inheritance.html)**:** a feature that represents the "is a" relationship between different classes.
* [**Interface**](http://www.webopedia.com/TERM/I/interface.html)**:** the languages and codes that the applications use to communicate with each other and with the hardware.
* [**Messaging**](http://www.webopedia.com/TERM/M/message_passing.html)**:** Message passing is a form of communication used in parallel programming and object-oriented programming.
* [**Object**](http://www.webopedia.com/TERM/O/object.html)**:** a self-contained entity that consists of both data and procedures to manipulate the data.
* [**Polymorphism**](http://www.webopedia.com/TERM/P/polymorphism.html)**:** A programming language's ability to process objects differently depending on their data type or class.

Q.2**(a) Orthogonality**:-**orthogonality** in a **programming language** means that a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the **language**" ( Sebesta, Robert. Concepts of **Programming Languages**.

(b)**Portability**:-**Portability** in high-level computer **programming** is the usability of the same software in different environments. The prerequirement for **portability** is the generalized abstraction between the application logic and system interfaces.

**(c) abstraction**:-In object-oriented **programming**, **abstraction** is one of three central principles (along with encapsulation and inheritance). Through the process of **abstraction**, a **programmer** hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

(d) **Reliability:-**Reliability of a programming language means how crash-proof is the code written in that language. Reliability is largely dependent on the code written and very little on the language itself. However reliability does depend on how easy it is to program in that language although it is largely application-specific.

(e) **Program Verification :-**Dynamic verification is also known as test or experimentation. It is further divided into three families ‘Test in the small’, ‘Test in the large’, and ‘acceptance test’. Test in small is where the program tests or checks a single function or class. Test in the large is where the program checks a group of classes like a single module, more than one module, and/or the entire system

## Q.2 Syntax:-

It represents the grammar which tells you the rules for constructing well-formed sentences of the language. If a program contains syntactic errors, it will not pass compilation. For example:
intmultiply\_numbers= a b \*;// syntax error in C

While(1 //synthetically compile time errorparenthesis

{

Printf(“hy); // compile time error

}

 C=a / 0;

 = (Syntax Tree)

 C /

 A 0

**Syntax is a sub-discipline of linguistics that studies the structure of a sentence.** It studies the set of rules, principles, and processes that rule the structure of sentences in any language. Here, the term structure of sentence refers to the word order. The meaning of a sentence can depend on the word order

## Semantics:-

It represents the meaning of the sentences of the language. If a program contains only semantic errors, it means that it may pass compilation, but does not do what it meant to do. For example:

* intmultiply\_numbers= a / b;// semantic error
* float a, b, c; c=a+b\*60; //here convert to int to float(60)
* If an expression involves int, double what should be the casting results, these are semantic questions.

**Semantics is a branch of linguistics that focuses on the study of meaning**. It studies the meaning of words and language. Semantics study ways in which the meanings of words can be related to each other (synonyms, homophony, etc.), ways in which the meanings of sentences can be related to each other, and ambiguity. Ambiguity is one way of studying the meaning of language. A sentence is said to be ambiguous when it has more than one meaning.

Q.2 **Programming Language:-**A programming language is a vocabulary and set of grammatical rules for instructing a [computer](http://www.webopedia.com/TERM/C/computer.html) or computing device to perform specific tasks. The term *programming language* usually refers to [high-level languages](http://www.webopedia.com/TERM/H/high_level_language.html), such as [BASIC](http://www.webopedia.com/TERM/B/BASIC.html), [C](http://www.webopedia.com/TERM/C/C.html), [C++](http://www.webopedia.com/TERM/C/C_plus_plus.html), [COBOL](http://www.webopedia.com/TERM/C/COBOL.html), [Java](http://www.webopedia.com/TERM/J/Java.html), [FORTRAN](https://www.webopedia.com/TERM/F/FORTRAN.html), [Ada](http://www.webopedia.com/TERM/A/Ada.html), and [Pascal](http://www.webopedia.com/TERM/P/Pascal.html).

Each programming language has a unique set of keywords (words that it understands) and a special [syntax](http://www.webopedia.com/TERM/S/syntax.html) for organizing program [instructions](http://www.webopedia.com/TERM/I/instruction.html).

## High-Level Programming Languages

High-level programming languages, while simple compared to human languages, are more complex than the languages the computer actually understands, called [machine languages](http://www.webopedia.com/TERM/M/machine_language.html). Each different type of [CPU](http://www.webopedia.com/TERM/C/CPU.html) has its own unique machine language.

## Converting to Machine Language

Regardless of what language you use, you eventually need to [convert](http://www.webopedia.com/TERM/C/convert.html) your program into machine language so that the computer can understand it. There are two ways to do this:

1) [Compile](http://www.webopedia.com/TERM/C/compile.html) the program.
2) [Interpret](http://www.webopedia.com/TERM/I/interpreter.html) the program.

**Reasons for studying programming language**

1.Increased capacity to express ideas--The depth at which we think is influenced by the expressive power of the language in which wecommunicate our thoughts.

2.Improved background for choosing appropriate languagesThere are mainly 2 types of programmers with two backgroundsSome of them might have learned programming on their own or through in house training programMany other programmer received their formal training in the distant pastIn the first group, training programs often teach one or two language that are directly relevant tothe current work of organization.

3.Increased ability to learn new languages-Software development is an existing profession but it also means that continuous learning isessential.

4.Better understand of significance of implementation-An understanding of implementation issues leads to an understanding of why languages aredesigned the way they are.-This in turn leads to the ability to use a language more intelligently, as it was designed to use.-Certain kinds of program bugs can only be found and fixed by programmer who knows somerelated implementation issues.-Another benefit is it allows us to visualize how a computer executes various languageconstructs.

5.Increased ability to design new languages.

6.Overall advancement of computing-Although it is usually possible to determine why a particular language become popular, it is notalways clear at least in retrospect, that the most popular languages are best available.

Q.3 **A high-level language**

A high-level language is any programming language that enables development of a program in a much more user-friendly programming context and is generally independent of the computer's hardware architecture.

A high-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

A ***h****igh-****l****evel* ***l****anguage* (**HLL**) is a [programming language](https://www.webopedia.com/TERM/P/programming_language.html) such as [C](https://www.webopedia.com/TERM/C/C.html), [FORTRAN](https://www.webopedia.com/TERM/F/FORTRAN.html), or [Pascal](https://www.webopedia.com/TERM/P/Pascal.html) that enables a [programmer](https://www.webopedia.com/TERM/P/programmer.html) to write [programs](https://www.webopedia.com/TERM/P/program.html) that are more or less independent of a particular type of [computer](https://www.webopedia.com/TERM/C/computer.html). Such [languages](https://www.webopedia.com/TERM/L/language.html) are considered high-level because they are closer to human languages and further from [machine languages](https://www.webopedia.com/TERM/M/machine_language.html).

In contrast, [assembly languages](https://www.webopedia.com/TERM/A/assembly_language.html) are considered low-level because they are very close to machine languages.

## Advantages of High-Level Languages

The main advantage of high-level languages over [low-level languages](https://www.webopedia.com/TERM/L/low_level_language.html) is that they are easier to read, write, and maintain. Ultimately, programs written in a high-level language must be translated into machine language by a [compiler](https://www.webopedia.com/TERM/C/compiler.html) or [interpreter](https://www.webopedia.com/TERM/I/interpreter.html).

## Low Level Vs High Level programming languages

### 1) Platform Dependencies

First, you should know about the platform, here platform means Computer Hardware (Computer configuration).

Low Level programming languages are platform dependent that means programs written in Low Level language can run on the same hardware with same configuration, you cannot run them on hardware that has different configuration.

High Level programming languages are platform independent that means programs written in High Level language can run on different hardware with different configuration.

**Remember:** High Level programming languages are platform independent that doesn’t mean they are operating system independent. System hardware configuration may be different but Operating systems should be same.

### 2) Speed

Low Level language programs are faster than High Level language programs as they do not need to convert.

They have less number of syntaxes, functions, keywords, class libraries.

### 3) Easiness

Low Level language programs are not as easy as High Level language. There are only two Low Level programming languages Binary and Assembly. Binary has only 0’s, 1’s, while Assembly has some difficult type symbols which are knows as mnemonics.

But, the High Level languages programs are easy to write, read, modify and understand.

### 4) Performance

Since, Low Level Languages programs are faster, so performance of Low Level languages programs are better than the High Level languages programs.

### 5) Translation

Low Level language Binary does not need translation as Binary codes are Machine codes and computer understands them without any translations.

Assembly needs an Assembler to translate an Assembly program to its equivalent Binary/Machine Code.

High Level Languages are translated by the compilers or interpreters; sometimes (in case of some programming languages) both are required to get the Object/Binary file.

### 6) Flexibilities

High Level languages are flexible to read, edit, debug, understand etc but Low Level Languages are not so easy to handle.

High Level languages have huge libraries with a rich set of Data types, keywords, functions etc so these languages are really good to develop an application with many great features using less effort and resource.

### 7) Support

Low Level languages have less support than High Level Languages. There may be lesser number of professionals (community) in support of Low Level languages as comparisons to High Level Language support.

**Q.3 Symbol Table Management**

The symbol table is accessed by most phases of a compiler, beginning with the lexical analysis to optimization.

|  |  |  |
| --- | --- | --- |
| 1 | A | Int |
| 2 | B | Int |
| 3 | 30 | constant |

 Symbol Table

The symbol table carries the collected information about each named object in the program to other phases of the compiler. As soon as a named token is found, depending on the token type, a call to the symbol table routines must be made.

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.

A symbol table may serve the following purposes depending upon the language in hand:

* To store the names of all entities in a structured form at one place.
* To verify if a variable has been declared.
* To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
* To determine the scope of a name (scope resolution).

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the following format:

<symbolname, type, attribute>

<symbolname, type, attribute>

<symbolname, type, attribute>

<symbolname, type, attribute>

**Error Handler:-**In programming, a development error is one that can be prevented. Such an error can occur in [syntax](http://searchcio-midmarket.techtarget.com/definition/syntax) or logic. Syntax errors, which are typographical mistakes or improper use of special characters, are handled by rigorous proofreading. Logic errors, also called [bug](http://searchsoftwarequality.techtarget.com/definition/bug)s, occur when executed [code](http://whatis.techtarget.com/definition/code) does not produce the expected or desired result. Logic errors are best handled by meticulous program [debugging](http://searchsoftwarequality.techtarget.com/definition/debugging). This can be an ongoing process that involves, in addition to the traditional debugging routine, beta testing prior to official release and customer feedback after official release.

A run-time error takes place during the execution of a program, and usually happens because of adverse system [parameter](http://whatis.techtarget.com/definition/parameter)s or invalid input [data](http://searchdatamanagement.techtarget.com/definition/data). An example is the lack of sufficient [memory](http://searchmobilecomputing.techtarget.com/definition/memory) to run an application or a memory conflict with another program.

Q.4 C= a – b + d \* 25;

 (a) Lexical Analysis:-id1 op1 id2 op2 id3 op3 constant= 07 tokes

No. of tokes generate by lexical analysis is =07

Q.4 **Data objects**

A data object will participate in several bindings in its lifetime.

• Although the attributes of a data object may remain fixed, the bindings may changedynamically.

• These bindings include:

Type

Associates the data object with the allowable set of values.

Location

May be changed by the memory management routine of the operating system.

Value

usually the result of an assignment operation.

Name

can be changed as a result of subprogram calls andreturns

Component

data objects may be composed of several dataobjects. This binding may be represented by a pointer and subsequently changed.

 B (indicate the location of memory)

 10

 Values

 65636 Address

A data object are the physical units with which ABAP statements work at runtime. The contents of a data object occupy memory space in the program. ABAP statements access these contents by addressing the name of the data object and interpret them according to the data type.. For example, statements can write the contents of data objects in lists or in the database, they can pass them to and receive them from routines, they can change them by assigning new values, and they can compare them in logical expressions.

Each ABAP data object has a set of technical attributes, which are fully defined at all times when an ABAP program is running (field length, number of decimal places, and data type). You declare data objects either statically in the declaration part of an ABAP program (the most important statement for this is [DATA](https://help.sap.com/viewer/cfae740a0a21455dbe6e510c2d86e36a/7.31.21/en-US/fceb2ff3358411d1829f0000e829fbfe.html) ), or dynamically at runtime (for example, when you call procedures). As well as fields in the memory area of the program, the program also treats literals like data objects.

**Variable:-**

## **Variables** are used to store information to be referenced and manipulated in a computer program.

 Int a=20;

## [Assigning Value to Variables](https://launchschool.com/books/ruby/read/variables#assigningvaluetovariables)

Naming variables is known as one of the most difficult tasks in computer programming. When you are naming variables, think hard about the names

When you assign a variable, you use the = symbol. The name of the variable goes on the left and the value you want to store in the variable goes on the right.

## [Variable Scope](https://launchschool.com/books/ruby/read/variables%22%20%5Cl%20%22variablescope)

A variable's scope determines where in a program a variable is available for use. A variable's scope is defined by where the variable is initialized or created. In Ruby, variable scope is defined by a block. A block is a piece of code following a method invocation, usually delimited by either curly braces {} or do/end. Be aware that not all do/end pairs imply a block\*.

Variable type:

* 1. Local Variable
	2. Instant Variable
	3. Static variable

Class Student

{

Int age;

String name;

Static int pincode=101;

Void display()

{

Int st\_id;

}

}