# Differences between compiled and Interpreted Languages

**Manikandan10**, 14 Dec 2013 BSD

★★★★★ 4.65 (20 votes)

Rate this:

The article describles the differences between compiled and interpreted languages

**Download Python Scripting Sample - 25.6 KB**

## Introduction

There are thousands of programming languages in this world. Some of them are called compiled languages and the software we use for the language is called compiler. Some of them are called as interpreted languages. The softwares we use for them are called interpreter. Another interesting fact is that for every compiled language, an interpreter can be built but the reverse is impossible. That is, all the interpreted languages can not be a compiled language. Either Being interpreted or compiled is not the property of the programming languages but, design of some languages make them unsuitable for native code generation. This article describes the advantages and disadvantages of both natively compiled and those are not natively compiled. This is my first article and I have not well practiced in writting yet.

## Compiled and Interpreted languages

### Natively Compiled Languages

A natively compiled language is a programming language that have compiler(s) built for it which can compile the source code to native code. A natively language can always be an interpreted language. Take for an instance C++ is a natively compiled language. But it also has a number interpreters like CINT, ch interpreter etc., This is because of the constrains of the native code. The natively compiled languages are created in such a way to suit the constrains but the interpreted languages are not.

### Interpreted Languages and JIT Compiled (Not compiled to native code)

In some cases, the source code is executed line by line by a software called interpreter. Interpreted languages are often slow than the compiled languages because of a number of reasons. It is because the source should be executed line by line. Also the interpreted languages like python and even Java, C# have a number of advanced facilities like dynamic typing, type checking, extensive type information which would be stored in the interpreter reduces the performance of the interpreted applications. Another disadvantage is that one should have the interpreter installed in the computer unless the interpreter is embedded in the software which may increase the size of the package. For example, Blender has embedded python interpreter.

#### Direct execution of the source code

This is where the source files of interpreted languages are directly executed by the interpreters without any step of changing it to an intermediate or bytecode form. Some web scripting languages like PHP, JScript and VBScript works based on direct source execution. Some of them may have Just-In-Time compilation but they only don't get their source converted to a bytecode form.

#### Compiling to a Bytecode

There are some compilers or frameworks which convert source code of certain programming languages to an intermediate language that is embedded in a executable or any sort of file. Then the virtual machine or interpreter executes the intermediate language during the runtime of the application. These kind of applications are rather fast than that have it source interpreted. Sometimes, there is another sort of execution known as Just-In-Time execution(JIT) where the bytecode is converted to native code during runtime by the interpreters and executed. This methodology proved to improve the peformance of interpreted languages. Even though then they are slower than natively compiled languages. Microsoft's .NET Framework and some implementations of Java work based on this principle. Compiling to a Bytecode comes under the category of interpreted languages

## Advantages of Interpreted and JIT Compiled languages

### General Advantages

We have already spoken about the disadvantages of the interpreted languages and JIT Compiled languages. Even though they have many disadvantages they have a number of advantages. Firstly interpreted languages saves compile time resulting in faster development. This makes them ideal for scientific and mathematical computing. They also provide many features that aren't accessible in compiled languages. They are also very suitable for scripting. For example Microsoft Word provides scripting capabilities by interpreting Visual Basic code. If you write an application which you need to seamlessly interact with the user, then scripting is one of the choices. It also enable them to automate tasks reducing their appetite. Interpreted languages like JScript, VBScript, PHP are good choice for Web Programming. This is because they can't be compiled for each and every platform and device while its loading. Suppose if a web page is loaded in a Linux Machine, and it contains C++ code, it can't be compiled for that machine. So interpreted languages play an inevitable role in web programming. They run whatever the device or machine is if viewed in a browser supporting them. For example viewing a web page containing JScript in Google Chrome in computer and in an Android Tablet produces the same results. Java is often referred to as an interpreted language. Java source code is converted to byte code. Java is not popularly used as a desktop programming language but it finds its use in many other places like Web, Mobile phone programming, Android Programming etc., For instance, Take Google's Android OS. It has a Linux kernel and so one can create software using ARM compilers. But Google did not do that. Android has a Virtual Machine named Dalvik which executes Dalvik byte code. For some reason they didn't embed a Java VM. They provided tools to convert Java bytecode to Dalvik Bytecode. The reason for why they prefer Java over C++ is that Java has a large community which may lead to the rapid increase of apps in their store. Java is also a powerful Object Oriented Programming language. The Android apps running on Dalvik VM does not provide full performance. Although there were no visible effects on performance, Google has released Android Native Development Kit(NDK) which allows the users to speed up the critical pieces of code by the Native Code. Android NDK provides a way for creating parts of apps using C and C++ ARM Compilers.

### Extending your application

Scripting languages also provide a great platform for writing extension or plugins for your applications without loading Dynamic link libraries or Shared Objects dynamically. This can reduce the difficulty in writing a plugin for a particular application. The traditional way of writing a plugin involves creating a DLL which requires the compiler and also the libraries used by the applications. Many modern applications provide way for creating plugins using scripting languages like Python.

### Cross platform support

Cross platform programming is always been difficult and a software clinging to a single platform can seriously affect the range of audience. If you create a software in a compiled language you have to move the source from machine to machine and compiling the source in that machine. But executing the source by the interpreter does not stick to a single platform only if the code itself is not platform dependent. Producing platform independent code has not always been an issue. Suppose if you create a python application using Windows and want it to be platform independent then don't use platform specific features like Component Object Model(COM) which is available only under windows. Java follows the policy of "Write Once, Run anywhere". But .NET is not cross platform. It is much more suited for Windows but there are many variants of .NET like dotGNU that are cross platform. Even there is a .NET based framework available for Android!!

### Java and .NET supports Reflection

Java and .NET languages support an excellent feature called reflection. Using reflection one even can create a method or class in runtime and use it. Even a dozen of .NET languages rose only because of reflection feature. They support seamless object serialization without any external library. Even an assembly can be disassembled using reflection. But one should be aware that every .NET and Java assembly can be re engineered using the same reflection feature. This danger can be overcome by using obsufcators. See this page for a list of .NET obfuscator.

### Wide Range of Devices

Usage in wide range of devices where performance limits is another advantage of languages that aren't converted to native code. Take Java Micro Edition, it is available in wide range of devices even in mobiles that aren't smart phones. It allows the use of a powerful object oriented programming language to create apps for an unadvanced device. The Java installer claims that Java runs on more than 3 billion devices. Python interpreter is available for Android devices(QPython for Android) enabling the users to perform advanced calculations using a handy device. No one can compile an applicaton to native code and run in a J2ME Device where there is actually no Operating System.

### Smaller executables and packages

It is obvious that the executables of interpreted applications are of very low size than that of the compiled languages. It is interesting to note that in a C# application is embedded the Intermediate language which is not machine language. It is the same case in Java and this dramatically can reduce the size of the executables. No one should deny that Python libraries like wxWidgets are created using C++ and connected by SWIG. But the size of wxWidgets library for Python is less than for that of C++.

### Easily debuggable

Bugs are often the headaches of programmers. .NET Languages and other interpreted languages provides facilities so that bugs can be caught easily. They throw an exception and the details of the exception will be clearly furnished in a dialog box. Unlike native code which only indicates an error is there managed code even show an error even that an object that is not initialized and also indicates the name of the object. This easily eradicates bugs. For an instance if your application is distributed with a bug and the user sees an error message then the user can be intimated to send the error report. It enables us to easily locate the error in the source code rather than searching for it.

### Side by Side code execution

This is one of the features that are unique to the interpreted languages. This feature is unavailable in C# or Java. Programming languages like Python, J, R have the console or GUI command prompt based softwares where code is executed on the fly. They are also referred to as "shell". They also report errors on the fly. This can provide a concrete facility for learning.

## Advantages of the compiled languages

### Speed

Compiled languages are always supposed to be fast because of their direct execution by the computer. Speed and performance can change programmers preference. Especially for large projects, speed and performance is indispensable. Poor speed can crush user experience and can annoy them. C is claimed to be the fastest programming language next to assembly code. It is believed that C is faster than C++ is some instances. One website even says that some algorithms implemented in python may run ten times slower than its equivalent written in C++. It is also obvious that speed always does not matter. There can be places where speed is secondary.

### Native applications are secured

Even though native applications can be disassembled, the assembly code is not so clear and the source code can not be that easily obtained. For example it is possible to generate C# code from a .NET assembly using some tools but it is impossible to generate C++ code from an executable.

### Large softwares are written in Compiled Languages

Large softwares and million dollar projects are often written in compiled languages because of the speed and performance they offer. Many large softwares with huge code bases ranging from Office suites, IDEs, Compilers, Games, time and mission critical applications were written in compiled languages. You can even observe in your computer many softwares are natively compiled. Even the web browser you are viewing this page may be written in compiled languages. Browsers are compiled natively because they need speed. But the parts of a large software may be written in interpreted languages. It is to be noticed that your browser uses many interpreted languages like JScript, VBScript, PHP etc., to view web pages.

### Reflection is not impossible

Reflection is not impossible in compiled languages. It is possible with third party libraries like GNU lightning, Boost Serialization etc., Thus it is not true that reflection is impossible in compiled languages. It saves us from creating XML or any other sort of IO routines which saves our data textually for every persistent data which is time consuming. Some programmers prefer saving their data in binary form rather than simply saving in a textual manner. But writing compilers for native languages that can emit machine code is an advanced step though then there are some libraries like ASMJIT that can be used for real time machine code generation.

### Interoperability is possible with .NET, Java and Python

It is possible for a natively compiled application to use .NET(through COM), Java libraries and Python source codes. So if you have a large investment in interpreted languages and planning to move to compiled languages, you can still use your libraries that you have created for .NET, Java and python. There is an option of using .NET controls in your MFC Applications. .NET Assemblies can be accessed by creating a COM wrapper from any program.

## Why aren’t there compilers for some interpreted languages?

Already the article discussed about the impossibility of creating compilers for some interpreted languages due to the some limitations in converting them to machine language. It does not mean that there can't be a compiler for every interpreted language. Java has a native code compiler(GNU Java Compiler). Python does not have native compilers that can compile native code to machine language. Python converts the source to a byte code file with an extension (.pyc). But the source can be converted to executables and though then, it is interpreted. In this case the code is embedded in the executable and then executed by the Python interpreter embedded in it. It doesn't have any effect but helps the program to be closed source. Some of the features that limit the compilation to native code are:-

In Python, Functions do not indicate their return types until the end of the function is executed. It is not even known by interpreter whether an object will be returned or not before the actual execution of the routine. This sort of *dynamic typing* can be impossible in machine code. Consider the following program

Hide Copy Code

```
import random;
def method():
    generated = random.randrange(6);

    if(generated / 2 == 0):
      return "It is even";
    if(generated / 2 == 1):
      return 1;
    return 0; ## Impossible

print(method());
print(method());
print(method());
```

The output came as follows :

Hide Copy Code

```
It is even
It is even
1
```

The output may *differ* every time you execute it and so it is impossible to generate an assembly language representation of this program.

***What about the templates in the C++ or D programming language?***

Templates in C++ or D are different from the dynamic typing as done in python programs. They are nothing but similar to a macro. The type of the object is defined in the compile time and not in the runtime as done in the above program. Consider the following C++ source code.

Hide Shrink ▲ Copy Code

```
#include <iostream>
#include <conio.h>
#include <typeinfo>

using namespace std;

#define INT 0 // Return an integer
#define STR 1// Return a string

template <class T>
void method(T val)
{
   if(!strcmp(typeid(val).name(), "int"))
   {
      cout<<"The variable is an integer"<<endl;
   }
   else if(!strcmp(typeid(val).name(), "double"))
   {
      cout<<"The variable is a double"<<endl;
   }
}

void main()
{
   method<int>(3);
   method<double>(2.4);
   getch();
}
```

The above program is an example of function templates in C++ which would yield the following output,

The variable is an integer

The variable is a double

- The first time the method is called the template parameter is "int" and so while compiling the function a C++ compiler would change all the objects whose type names are declared T to "int" and so the type identification function informed the type the variable "var" as "int". These all occur only in the compile time.

- Some python functions like "eval()" always require the Python parsing system and runtime system.

## Conversion of Class - The C++ Way

- In C++, a class is compiled in entirely different way unlike as in C# or Java. Consider the following C++ code

Hide   Copy Code

```
class ABC
{
public:
    int a;
    ABC()
    {
    }
    void Method1()
    {

    }
    void Method2()
    {

    }
};
int main()
{
    ABC abc;
    abc.a = 10;
    abc.Method1();
    return 0;
}
```

The code contains a class named "ABC" which have "Method1" and the class has been instantiated in the main() function. I converted the source to an assembly representation by the MinGW C++ Compiler using "-S" Option(Case Sensitive) and the assembly looked as the following,

Hide   Shrink ▲   Copy Code

```
.file   "m.cpp"
    .def    ___main;    .scl    2;  .type   32; .endef
    .text
    .align  2
.globl  _main
    .def    _main;  .scl    2;  .type   32; .endef
_main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    andl    $-16, %esp
    movl    $0, %eax
    addl    $15, %eax
    addl    $15, %eax
    shrl    $4, %eax
    sall    $4, %eax
    movl    %eax, -8(%ebp)
    movl    -8(%ebp), %eax
    call    __alloca
    call    ___main
    subl    $12, %esp
    leal    -4(%ebp), %eax
    pushl   %eax
    call    __ZN3ABCC1Ev
    addl    $16, %esp
    movl    $10, -4(%ebp) // Create a local variable which is actually "abc.a"
    subl    $12, %esp
    leal    -4(%ebp), %eax
    pushl   %eax
    call    __ZN3ABC7Method1Ev // call "abc.Method1()" This includes the method in the assembly
    addl    $16, %esp
    movl    $0, %eax
    leave
    ret
    .section    .text  $_ZN3ABC7Method1Ev,"x"
    .linkonce discard
    .align  2
.globl __ZN3ABC7Method1Ev
    .def    __ZN3ABC7Method1Ev; .scl    2;  .type   32; .endef
__ZN3ABC7Method1Ev:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $4, %esp
    movl    $10, -4(%ebp)
    leave
    ret
    .section    .text   $_ZN3ABCC1Ev,"x"
    .linkonce  discard
    .align   2
.globl __ZN3ABCC1Ev
    .def    __ZN3ABCC1Ev;   .scl    2;  .type   32; .endef
__ZN3ABCC1Ev:
    pushl   %ebp
    movl    %esp, %ebp
    leave
    ret
```

You may observe that there is no class definition in the assembly code. The class definition is only stored in the symbol table of the C++ compiler. You may see that the variable "a" is used in the main function and yet there is no variable of the class ABC. The variable "a" is just stored in a local variable(see the comments). The method is stored as "__ZN3ABC7Method1Ev" as we called it from the main function; you may be surprised to see that there is no definition of Method2. It is because we didn't use it. The compiler decides to include a field or function only if it is used. It is clear that class "ABC" is stored in the symbol table of the compiler and then if a field is used the compiler includes it in the assembly. Here we used Method1 so the compiler included the Method in the assembly with the name "__ZN3ABC7Method1Ev". And when we called the function you may note the statement "`call` __ZN3ABC7Method1Ev". This is the actual method by which OOP native compilers compile code to native code.

## Conversion of Class - The C# Way

Above we discussed about how a native C++ compiler compiles a class. The following is the way a C# compiler compiles a class. Consider a class named "ABC" which has a variable "a" is compiled to a .NET executable then the following is the disassembly. The namespace name is "ClassConversion" and it is a simple console program.

## The demo program

The demo program provided shows simple interfacing between Python and C++. It in fact, implements a simple python scripting system.
It defines two Python methods namely "DisplayDayNames()" which displays seven days of a week and "Distance(x1,y1,x2,y2)" which finds the distance between two coordinates using a simple mathematical formula.

Hide   Copy Code

```
PyRun_SimpleString("import DemoModule");
PyRun_SimpleString("from DemoModule import *");
PyRun_SimpleString("print Distance(2,3,4,5);"); // Call the method "Distance" from python
PyRun_SimpleString("DisplayDayNames();");
```

The above 4 lines executes Python code line by line using the embedded interpreter.

## Conclusion

This article, by describing the advantages and disadvantages, does not advocate to use only compiled languages or only interpreted languages. It is also fully possible to use both. But you've to be very clear which to use where. For example, if you opt Python for developing an application, you can still use C++ to speed up the critical parts of your code where speed is necessary. Rather than using the library provided with Python distribution, one can use Boost.Python Library to seamlessly interporate the both languages. Download the full Boost distribution which will be useful in all sorts of development.

## More Information

### Bibliography

The D Programming Language by Andrei Alexandrescu - Buy the book

The Dragon Book Principles, Techniques and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman - Buy the book

### Online Reference

Interpreted Language - go to the page

Compiled Language - go to the page

Managed Code - go to the page

### Websites

Boost libraries site - www.boost.org

Python website - www.python.org

D Programming website - www.dprogramming.com

## History

## License

## Share

EMAIL TWITTER

## About the Author

## You may also be interested in...

Lua Interpreter

Smart Stove Top in Python

Compiler and language

SAPrefs - Netscape-like Preferences Dialog

Watering System in Python

Generate and add keyword variations using AdWords API

## Comments and Discussions

**You must Sign In to use this message board.**

**Search Comments** Go

First Prev Next

**Nice!**
**Volynsky Alex 3-Jan-14 6:16**

**Too simplistic**
**terrycolligan 16-Dec-13 6:42**

Re: Too simplistic
**PIEBALDconsult** 16-Dec-13 7:23

Re: Too simplistic
**terrycolligan** 16-Dec-13 10:21

Re: Too simplistic
**PIEBALDconsult** 16-Dec-13 11:14

**Honest feedback**
**Sentenryu 16-Dec-13 5:57**

**Some comments.**
**PedroMC 16-Dec-13 4:45**

Re: Some comments.
**PIEBALDconsult** 16-Dec-13 5:09

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.