## Difference Between Compiler and Interpreter

**Interpreter Vs Compiler**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Compiler** | **Interpreter** |
| Steps of Programming | * Creation of the program. * The Compiler analyses all the language statements and throws an error when it finds something incorrect. * If there’s zero error, the compiler converts the source code to machine one. * It links various code files into a runnable program (exe). * It runs the program. | * Creation of the program. * It doesn’t require the linking of files or generation of machine code. * It executes the source statements line by line during the execution. |
| Advantage | The code execution time is comparatively less because the program code already gets translated into machine code. | They are fairly easy to use and execute, even for a beginner. |
| Disadvantage | One can’t change a program without getting back to the source code. | Only computers with the corresponding Interpreter can run the interpreted programs. |
| Machine Code | It stores the machine language on the disk in the form of machine code. | It doesn’t save the machine language at all. |
| Running Time | The compiled codes run comparatively faster. | The interpreted codes run comparatively slower. |
| Model | It works on the basis of the language-translation linking-loading model. | It works on the basis of the Interpretation method. |
| Generation of Program | It generates an output program in the exe format. A user can run it independently from the originally intended program. | It doesn’t generate an output program. Meaning, it evaluates the source program every time during individual execution. |
| Execution | One can separate the program execution from the compilation. Thus, you can perform it only after completing the compilation of the entire output. | Execution of the program is one of the steps of the Interpretation process. So, you can perform it line by line. |
| Memory Requirement | Target programs execute independently. They don’t require the Compiler in the memory. | Interpreter originally exists in the memory at the time of interpretation. |
| Best Fitted For | You cannot port the Compiler because it stays bound to the specific target machine. The compilation model is very common in programming languages like C and C++. | They work the best in web environments- where the load time is very crucial. Compiling takes a relatively long time, even with small codes that may not run multiple times due to the exhaustive analysis. Interpretations are better in such cases. |
| Optimization of Code | A compiler is capable of seeing the entire code upfront. Thus, it makes the codes run faster by performing plenty of optimizations. | An interpreter sees a code line by line. The optimization is, thus, not very robust when compared to Compilers. |
| Dynamic Typing | Compilers are very difficult to implement because they can’t predict anything that happens during the turn time. | The Interpreted language supports Dynamic Typing. |
| Use | It works best for the Production Environment. | It works the best for the programming and development environment. |
| Execution of Error | A Compiler displays every error and warning while compiling. So, you can’t run this program unless you fix the errors. | An Interpreter reads every statement, then displays the errors, if any. A user must resolve these errors in order to interpret the next line. |
| Input | A Compiler takes a program as a whole. | An Interpreter takes single lines of a code. |
| Output | The Compilers generate intermediate machine codes. | The Interpreters never generate any intermediate machine codes. |
| Errors | This translator displays all the errors after compiling- together at the same time. | It displays the errors of every single line one by one. |
| Programming Languages | Java, Scala, C#, C, C++ use Compilers. | Perl, Ruby, PHP use Interpreters. |

## What is a Compiler?

A compiler is a translator that produces an output of low-level language (like an assembly or machine language) by taking an input of high-level language. It is basically a computer program used to transform codes written in a programming language into machine code (human-readable code to a binary 0 and 1 bits language for a computer processor to understand). The computer then processes the machine code for performing the corresponding tasks.

* Compilers check all types of errors, limits, and ranges. Thus, it’s more intelligent.
* The run time of its program is longer, and it occupies more memory.

## What Is an Interpreter?

It is a program that functions for the translation of a programming language into a comprehensible one. It is a computer program used for converting high-level program statements into machine codes. It includes pre-compiled code, source code, and scripts.

* An interpreter translates only one statement at a time of the program.
* They create an exe of the programming language before the program runs.

**Difference between python 2 and python 3 –**

|  |  |  |
| --- | --- | --- |
| **Comparison Parameter** | **Python 2** | **Python 3** |
| “Print” Keyword | In Python 2, print is considered to be a statement and not a function. | In Python 3, print is considered to be a function and not a statement. |
| Storage of Strings | In Python 2, strings are stored as ASCII by default. | In Python 3, strings are stored as UNICODE by default. |
| Division of Integers | On the division of two integers, we get an integral value in Python 2. For instance, 7/2 yields 3 in Python 2. | On the division of two integers, we get a floating-point value in Python 3. For instance, 7/2 yields 3.5 in Python 3. |
| Exceptions | In Python 2, exceptions are enclosed in notations. | In Python 3, exceptions are enclosed in parentheses. |
| Variable leakage | The values of global variables do change in Python 2 if they are used inside a for-loop. | The value of variables never changes in Python 3. |
| Iteration | In Python 2, the xrange() function has been defined for iterations. | In Python 3, the new Range() function was introduced to perform iterations. |
| Libraries | A lot of libraries of Python 2 are not forward compatible. | A lot of libraries are created in Python 3 to be strictly used with Python 3. |
| Ease of Syntax | Python 2 has more complicated syntax than Python 3. | Python 3 has an easier syntax compared to Python 2. |
| Backward compatibility | Python 2 codes can be ported to Python 3 with a lot of effort. | Python 3 is not backward compatible with Python 2. |
| Application | Python 2 was mostly used to become a DevOps Engineer. It is no longer in use after 2020. | Python 3 is used in a lot of fields like Software Engineering, Data Science, etc. |

**Understand and explain about 10 different keywords**

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

[finally](https://www.w3schools.com/python/ref_keyword_finally.asp) - Used with exceptions, a block of code that will be executed no matter if there is an exception or not

[assert](https://www.w3schools.com/python/ref_keyword_assert.asp) For debugging

[lambda](https://www.w3schools.com/python/ref_keyword_lambda.asp) To create an anonymous function

[nonlocal](https://www.w3schools.com/python/ref_keyword_nonlocal.asp) To declare a non-local variable

yield To end a function, returns a generator

[while](https://www.w3schools.com/python/ref_keyword_while.asp) To create a while loop

[import](https://www.w3schools.com/python/ref_keyword_import.asp) To import a module

[return](https://www.w3schools.com/python/ref_keyword_return.asp) To exit a function and return a value

[pass](https://www.w3schools.com/python/ref_keyword_pass.asp) A null statement, a statement that will do nothing.

[nonlocal](https://www.w3schools.com/python/ref_keyword_nonlocal.asp) To declare a non-local variable.

**Variable creation rules ===> programs for this one.**

Naming variables

When you name a variable, you need to adhere to some rules. If you don’t, you’ll get an error. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

The following are the variable rules that you should keep in mind:

* Variable names can contain only letters, numbers, and underscores (\_). They can start with a letter or an underscore (\_), not with a number.
* Variable names cannot contain spaces. To separate words in variables, you use underscores for example sorted\_list.
* Variable names cannot the same as keywords, reserved words, and built-in functions in Python.

The following guidelines help you define good variable names:

* Variable names should be concise and descriptive. For example, the active\_user variable is more descriptive than the au.
* Use underscores (\_) to separate multiple words in the variable names.
* Avoid using the letter l and the uppercase letter O because they look like the number 1 and 0.

Summary

* A variable is a label that you can assign a value to it. The value of a variable can change throughout the program.
* Use the variable\_name = value to create a variable.
* The variable names should be as concise and descriptive as possible. Also, they should adhere to Python variable naming rules.

A variable can have a short name (like x and y) or a more descriptive name (age, car name, total\_volume). When you develop a program, you need to manage values, a lot of them. To store values, you use variables.

In Python, a variable is a label that you can assign a value to it. And a variable is always associated with a value.

Python variable ex. - num = 100

str = "BeginnersBook"

print(num)

print(str)

**-Regards,**

**Renuka Madane.**