1. **Difference between compiler and interpreter**
2. [**Compiler**](https://www.geeksforgeeks.org/introduction-of-compiler-design/)**:**

* It is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e. machine or assembly language.
* A compiler is more intelligent than an assembler it checks all kinds of limits, ranges, errors, etc.
* But its program run time is more and occupies a larger part of memory. It has slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.

Lightbox

1. [**Interpreter**](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**:**

* An interpreter is a program that translates a programming language into a comprehensible language. –
* It translates only one statement of the program at a time.
* Interpreters, more often than not are smaller than compilers.

https://media.geeksforgeeks.org/wp-content/uploads/20200411231903/Interpreter.jpg

|  |  |
| --- | --- |
| **Complier** | [**Interpreter**](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/) |
| Compiler scans the whole program in one go. | Translates program one statement at a time. |
| As it scans the code in one go, the errors (if any) are shown at the end together. | Considering it scans code one line at a time, errors are shown line by line. |
| Main advantage of compilers is it’s execution time. | Due to interpreters being slow in executing the object code, it is preferred less. |
| It converts the source code into object code. | It does not convert source code into object code instead it scans it line by line |
| It does not require source code for later execution. | It requires source code for later execution. |
| Eg- C, C++, C# etc. | Python, Ruby, Perl, SNOBOL, MATLAB, etc. |

1. **Difference between python 2 and python 3**

| **Comparison Parameter** | **Python 2** | **Python 3** |
| --- | --- | --- |
| Year of Release | Python 2 was released in the year 2000. | Python 3 was released in the year 2008. |
| “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. |
| Ease of Syntax | Python 2 has more complicated syntax than Python 3. | Python 3 has an easier syntax compared to Python 2. |
| 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. |
| Usage in today’s times | Python 2 is no longer in use since 2020. | Python 3 is more popular than Python 2 and is still in use in today’s times. |
| 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. |

**Example:**

|  |  |
| --- | --- |
| **Python 2**  def main():  print "Hi! This is Python 2"  if \_\_name\_\_== "\_\_main\_\_":  main() | **Python 3**  def main():  print ("Hi! This is Python 3")    if \_\_name\_\_== "\_\_main\_\_":  main() |

1. **Explain about 10 different keywords**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Description** | **Example** |
| break | A break is a [loop control statement](https://www.educba.com/control-statement-in-c-plus-plus/). It helps to control the execution of the loops. Specifically, the break is responsible for terminating the execution of the loop. | for i in range(1, 11):        # print the value of i      print(i)        # check the value of i is less then 5      # if i lessthen 5 then continue loop      if i < 5:          continue        # if i greater then 5 then break loop      else:          break |
| continue | Continue is a [loop control statement](https://www.educba.com/control-statements-in-c/). It helps to control the execution of the loops. Specifically, Continue is responsible for switching the loop control to the condition statement again. |
| def | The def keyword is used for defining a function or method in [python programming](https://www.educba.com/python-programming-beginners-tutorial/). The function is a block of code that can be executed. | # define GFG() function using def keyword  def GFG():      i=20      # check i is odd or not      # using if and else keyword      if(i % 2 == 0):          print("given number is even")      else:          print("given number is odd")    # call GFG() function  GFG() |
| if | The if keyword represents a condition instance in python. |
| else | The else keyword is used to represent the false execution of an[if statement](https://www.educba.com/if-statement-in-python/). |
| true | This keyword represents the Boolean value ‘true’. | check\_string = '123' print(check\_string.isdigit()) |
| false | This keyword represents the Boolean value ‘false’. | check\_string = 'asd' print(check\_string.isdigit()) |
| not | The word ‘not’ is also reserved for Boolean or logical operations. When implying this keyword, it means that an operation will be applied when the given conditional expression is not satisfied.   Popular Course in this category | check\_value = 5 if check\_value not in [1,7,4,6]: print("Hello World!\n") else: print("Nothing to print") |
| and | The word ‘and is reserved for Boolean or logical operations. When implying this keyword, it means that an operation will be applied only when both conditions stand true. | check\_value = 5 if check\_value > 1 and check\_value < 10: print("Hello World!\n") else: print("Nothing to print") |
| or | The word ‘or’ is also reserved for Boolean or logical operations. When implying this keyword, it means that an operation will be applied even when one of the conditions stands true. | check\_value = 5 if check\_value > 1 or check\_value < 10: print("Hello World!\n") else: print("Nothing to print") |

1. **Variable creation rules (programs for this one)**

Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

**#Legal variable names:**  
myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"  
  
**#Illegal variable names:**  
2myvar = "John"  
my-var = "John"  
my var = "John"