| Differences between Interpreter and Compiler |
| --- |
| Interpreter translates just one statement of the program at a time into machine code. | Compiler scans the entire program and translates the whole of it into machine code at once. |
| An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower. | A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster. |
| An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory. | A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed. |
| Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy. | A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler. |
| Interpreters are used by programming languages like Ruby and Python for example. | Compliers are used by programming languages like C and C++ for example. |

| **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. |
| 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. |

#### 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()

Keywords:

#### The if Keyword

The **if** keyword is used to start a [conditional statement](https://realpython.com/python-conditional-statements/). An if statement allows you to write a block of code that gets executed only if the expression after if is truthy.

The syntax for an if statement starts with the keyword if at the beginning of the line, followed by a valid expression that will be evaluated for its truthiness value:

if <expr>:

<statements>

#### The elif Keyword

The **elif** statement looks and functions like the if statement, with two major differences:

1. Using elif is only valid after an if statement or another elif.
2. You can use as many elif statements as you need.

In other programming languages, elif is either else if (two separate words) or elseif (both words mashed together). When you see elif in Python, think else if:

if <expr1>:

<statements>

elif <expr2>:

<statements>

elif <expr3>:

<statements>

Python doesn’t have a [switch statement](https://en.wikipedia.org/wiki/Switch_statement). One way to get the same functionality that other programming languages provide with switch statements is by using if and elif.

#### The else Keyword

The **else** statement, in conjunction with the Python keywords if and elif, denotes a block of code that should be executed only if the other conditional blocks, if and elif, are all falsy:

if <expr>:

<statements>

else:

<statements>

Notice that the else statement doesn’t take a conditional expression.

#### The for Keyword

The most common loop in Python is the for loop. It’s constructed by combining the Python keywords **for** and **in** explained earlier. The basic syntax for a for loop is as follows:

for <element> in <container>:

<statements>

A common example is looping over the numbers one through five and [printing](https://realpython.com/python-print/) them to the screen:

>>>

>>> for num in range(1, 6):

... print(num)

#### The while Keyword

Python’s [while loop](https://realpython.com/python-while-loop/) uses the keyword **while** and works like a while loop in other programming languages. As long as the condition that follows the while keyword is truthy, the block following the while statement will continue to be executed over and over again:

while <expr>:

<statements>

#### The break Keyword

If you need to exit a loop early, then you can use the **break** keyword. This keyword will work in both for and while loops:

n example of using the break keyword would be if you were summing the integers in a list of numbers and wanted to quit when the total went above a given value:

>>>

>>> nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> sum = 0

>>> for num in nums:

... sum += num

... if sum > 10:

... break

...

>>> sum

15

#### The continue Keyword

Python also has a **continue** keyword for when you want to skip to the next loop iteration. Like in most other programming languages, the continue keyword allows you to stop executing the current loop iteration and move on to the next iteration:

for <element> in <container>:

if <expr>:

continue

#### The and Keyword

The Python keyword **and** is used to determine if both the left and right operands are truthy or falsy. If both operands are truthy, then the result will be truthy. If one is falsy, then the result will be falsy:

<expr1> and <expr2>

#### The or Keyword

Python’s **or** keyword is used to determine if at least one of the operands is truthy. An or statement returns the first operand if it is truthy and otherwise returns the second operand:

<expr1> or <expr2>