**Differences between Python 2 Python 3**

If you are new to Python, you might be confused about the different versions that are available. Although Python 3 is the latest generation of the language, many programmers still use Python 2.7, the final update to Python 2, which was released in 2010.

There is currently no clear-cut answer to the question of which version of Python you should use; the decision depends on what you want to achieve. While Python 3 is clearly the future of the language, some programmers choose to remain with Python 2.7 because some older libraries and packages only work in Python 2.

Why Are There Different Versions of Python?

Programming languages constantly evolve as developers extend the functionality of the language and iron out quirks that cause problems for developers. Python 3 was introduced in 2008 with the aim of making Python easier to use and change the way it handles strings to match the demands placed on the language today. Programmers who first learned to program in Python 2 sometimes find the new changes difficult to adjust to, but newcomers often find that the new version of the language makes more sense.

Python 3.0 is fundamentally different to previous Python releases because it is the first Python release that is not compatible with older versions. Programmers usually don’t need to worry about minor updates (e.g. from 2.6 to 2.7) as they usually only change the internal workings of Python and don’t require programmers to change their syntax. The change between Python 2.7 (the final version of Python 2) and Python 3.0 is much more significant — code that worked in Python 2.7 may need to be written in a different way to work in Python 3.0.

### Key Differences Between Python 2 and Python 3

Here are some key differences between Python 2 and Python 3 that can make the new version of the language less confusing for new programmers to learn:

* Print: In Python 2, “print” is treated as a statement rather than a function. There is no need to wrap the text you want to print in parentheses, although you can if you want. This can be confusing, as most other actions in Python use functions that require the arguments to be placed inside parentheses. It can also lead to unexpected outcomes if you put parentheses around a comma-separated list of items that you want to print. In contrast, Python 3 explicitly treats “print” as a function, which means you have to pass the items you need to print to the function in parentheses in the standard way, or you will get a syntax error. Some Python 2 programmers find this change annoying, but it can help to prevent mistakes.
* Integer Division: Python 2 treats numbers that you type without any digits after the decimal point as integers, which can lead to some unexpected results during division. For example, if you type the expression 3 / 2 in Python 2 code, the result of the evaluation will be 1, not 1.5 as you might expect. This is because Python 2 assumes that you want the result of your division to be an integer, so it rounds the calculation down to the nearest whole number. In order to get the result 1.5, you would have to write 3.0 / 2.0 to tell Python that you want it to return a float, that is, to include digits after the decimal point in the result. Python 3 evaluates 3 / 2 as 1.5 by default, which is more intuitive for new programmers.
* List Comprehension Loop Variables: In previous versions of Python, giving the variable that is iterated over in a list comprehension the same name as a global variable could lead to the value of the global variable being changed — something you usually don’t want. This irritating bug has been fixed in Python 3, so you can use a variable name you already used for the control variable in your list comprehension without worrying about it leaking out and messing with the values of the variables in the rest of your code.
* Unicode Strings: Python 3 stores strings as Unicode by default, whereas Python 2 requires you to mark a string with a “u” if you want to store it as Unicode. Unicode strings are more versatile than ASCII strings, which are the Python 2 default, as they can store letters from foreign languages as well as emoji and the standard Roman letters and numerals. You can still label your Unicode strings with a “u” if you want to make sure your Python 3 code is compatible with Python 2.
* Raising Exceptions: Python 3 requires different syntax for raising exceptions. If you want to output an error message to the user, you need to use the syntax: