Comparing Python 2.x and Python 3.x

Although development of Python started in the late 80's, Python 1.0 was published in January 1994. Some important features like cycle-detecting garbage collector and Unicode support were added in Python 2.0 which was released in October 2000. Python 3.0 was released in December 2008. It was designed to rectify certain flaws in earlier version. The guiding principle of Python 3 was: "reduce feature duplication by removing old ways of doing things". Python 3.0 doesn't provide backward compatibility. That means a Python program written using version 2.x syntax doesn't execute under python 3.x interpreter. Since all the Python libraries are not fully ported to Python 3, Python Software Foundation continues to support Python 2. The foundation has recently announced it will discontinue Python 2 support by year 2020.

Some of the obvious differences between the Python 2.x variants and Python 3.x variants are displayed below.

Print

Python 2.x	**Python 3.x**

It is not mandatory to use parenthesis with the in-built print function.
Valid statements
`print "Hello World"`
`print ("Hello World")`
	Parentheses are mandatory with the print function.
	Valid statement
	`print ("Hello World")`
	If the parentheses are missing, the interpreter displays the following error:
	SyntaxError: Missing parentheses in call to 'print'

Input

Python 2.x	**Python 3.x**

Two types of input functions are available to read data from the console. The raw_input() function always treats the input as string, while the input() function evaluates the input and accordingly decides the data type of variable.	The raw_input() function has been deprecated and input() function more or less behaves as raw_input() thereby input is always read as string.

Integer division

Python 2.x	**Python 3.x**
Numbers that you type without any digits after the decimal point are treated as integers during division.	Evaluates the result of a division as decimals even if the input number are not specified with decimals.
3/2 Output = 1	3/2 Output = 1.5

Unicode strings

Python 2.x	**Python 3.x**
Strings are stored as ASCII by default. To store strings as Unicode, they have to be marked with a ‘u’. Example: u’hello’	Stores strings as Unicode by default.

Long integer

Python 2.x	Python 3.x
Arithmetic operations over normal integers may overflow the memory allocated to them. To allocate more memory to an integer object, a trailing L is attached. Example: 100L	Integer objects are long by default and do not require the trailing L. Example: 100