Operator Precedence

Precedence order of Python operators in ascending order

Operator	Description
()	Parenthesis
**	Exponentiation
+x, -x, ~x	Positive, negative, bitwise NOT
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
Not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Associativity of Python Operators

- The order in which an expression having multiple operators of same precedence is evaluated
- Almost all the operators have left-right associativity while the exponent '**' operator has right-left associativity

```
>>> # left-right associativity
>>> print(3 * 4 // 5)

# prints 2
```

```
>>> # right-left associativity
>>> print(2**3**2)

# prints 512
# since 2 ** (3**2) = 2 ** 9
```

Keywords in Python ...

- You can use help("keywords") in interactive shell to see a list of keywords
- Python keywords in version 3.7 + are:

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

Quotation in Python

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- String literal must end with same quote as in the beginning of the literal.
- The triple quotes are used for multiple line strings.

```
>>> str1 = 'Single quote example'
>>> str2 = "Double quote example"
>>> str3 = """Triple quote example
                continue in multiple
                lines without any
                escape character
            """
```

Comments in Python

- Python interpreter will ignore the commented part in a python code.
- Hash sign '#' indicates the start of a comment line

```
>>> # This is a comment
>>> print("Welcome to IW")
>>> print("Hey there, How are you?") # This is also a valid comment
```

- If # is used in the middle of the line then the text before it is a valid python expression, while the text following is treated as a comment.
- Python doesn't support multiline comments like c/c++.

.

Virtual Environment

- Python uses the concept of virtual environment during new project setup.
- We may create many python projects in our single system.
- So, instead of installing the python packages globally, we may create the virtual environment for each python project to separate up the packages.
- The advantage of separating the packages is it allows you to work on the packages of different versions in the same system.
- For example, our one project may be in django2.0 and other in django3.0. So, using virtual environment we can install the django2.0 in one environment and django3.0 in another.
- Let's setup our virtual environment

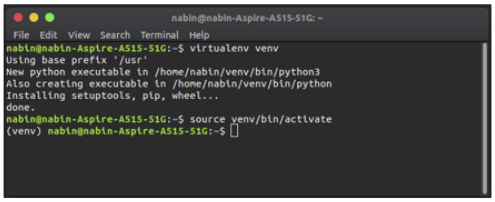
Setting Up the Virtual Environment

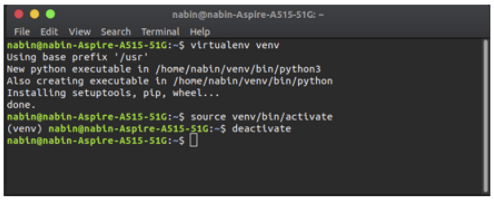
- First install 'pip'. (sudo apt-get install **python3**-pip)
- 'pip' is the package manager for the python packages.
- After the installation of pip, we can install 'virtualenv' using command,

  => pip install virtualenv
- Now navigate to the project folder and enter command

  => virtualenv <virtual_environment_name> ('virtualenv venv')
- The command creates a 'venv' folder where we can install our python packages.
- To install the packages in this environment, we need to activate it using command, (first navigate to the project folder)

  - source venv/bin/activate
- Activated environment displays (venv) at the front. (See in images aside)
- Your project takes the python interpreter and all the python packages from this activated environment.
- To deactivate the environment, simply enter 'deactivate' command.





Python Packages

- Packages can be the third-party packages or your own packages for the project.
- To install the third-party packages, enter the command,

  => pip install <package-name>
- The package will be installed in our activated virtual environment.
- Don't forget to activate the environment else, it would be installed globally.
- To create our own python package, we need to create a folder with '__init__.py' file inside it.
- Remember __init__.py is must to initiate a python package. Without it we can't import the package in other parts of the project.

Operators:

Brief introduction of operators studied today

Table: Math operators from Highest to lowest Precedence

Operator	Operation	Example	Evaluates to
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

The order of operations (also called precedence) of Python math operators is similar to that of mathematics. The operator ** is operated first; the *, /, //, and % operators are evaluated next, from left to right; and the + and - operators are evaluated last (also from left to right). You can also use the parenthesis (small brackets) to override the usual precedence if you need to.

Example:

>> 2 + 3 * 6 => 20

>> (2+3) * 6 => 30

Evaluating an expression going top to bottom reducing it to a single value:

(5 - 1) * ((7 + 1) / (3 - 1))

=> 4 * ((7 + 1) / (3 - 1))

=> 4 * (8) / (3 - 1))

=> 4 * (8) / (2)
=> 4 * 4.0
=> 16.0

Python keeps on evaluating parts of expression until it becomes the single value as in the above expression.
The rules for putting operators and the operands together must be followed strictly, just like the grammar rules that help us to communicate.

Example:

- **I have never visited Kathmandu.**
- **Kathmandu I visited never have**

The second line is difficult to parse as it doesn't follow the rules of english Grammar. Similarly, if you type in a bad Python instruction, Python won't be able to understand it and thus displays the SyntaxError message.

```
>>> 7+
  File "<stdin>", line 1
    7+
     ^
SyntaxError: invalid syntax
```

```
>>> 42 + 5 + * 5
  File "<stdin>", line 1
    42 + 5 + * 5
             ^
SyntaxError: invalid syntax
```

You can always test to see whether an instruction works by typing it into the interactive shell.

The Integer, Floating-Point, and String Data Types:

Expressions are just the values combined with operators, and they evaluate down to the single value. A *data type* is a category for values, and every value belongs to exactly one data type.
The data types we studied today are mentioned in the table below.

Data Type	**Examples**
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
Strings	‘a', 'aa', 'aaa', “Hello!”, “11 cats”

a) Integer:

The values -2 and 30, for example, are said to be integer values. The integer (or int) data type indicates values that are whole numbers.

b) Floating-point numbers:

Numbers with a decimal point, such as 3.14, are called floating-point numbers (or floats). Note that even though the value 2 is an integer, the value 2.0 would be a floating-point number.

c) String:

String value is surrounded with a single quote (‘ ’) or the double quote (“ “) characters. This helps python where the string begins and ends. You can even have a string with no characters in it, ‘ ‘, called as a blank string.

*[**Note:** 30 is an integer;*
30.0 is a floating point value
‘30’ or “30” and ‘30.0’ or “30.0” are the string values]

String Concatenation and Replication:

The meaning of an operator may change based on the data types of the values next to it.

For example, + is the addition operator when it operates on two integers or floating-point values. However, when + is used on two string values, it joins the strings as the string concatenation operator.

Enter the following into the interactive shell:

```
>>> "We" + "Nepali"
'WeNepali'
```

However, if you try to use the **+** operator on a string and an integer value, Python will not know how to handle this, and it will display an error message.

```
>>> "GIT" + 30
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

The error message above “can only concatenate str (not "int") to str” means that the, string and string values can only be concatenated, String and Integer value can not be concatenated implicitly. Your code will have to explicitly convert the integer to the string. The conversion of Integer to string is done below:

```
>>> "GIT" + "30"
'GIT30'
```

The * operator is used for multiplication when it operates on two integer or floating-point values. But when the * operator is used on one string value and one integer value, it becomes the string replication operator. Enter a string multiplied by a number into the interactive shell to see this in action.

```
>>> "GIT" * 5
'GITGITGITGITGIT'
```

The expression evaluates down to a single string value that repeats the original a number of times equal to the integer value.

The * operator can be used with only two numeric values (for multiplication) or one string value and one integer value (for string replication).
Otherwise, Python will just display an error message.

```
>>> "GIT" * 5.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
```

```
>> "GIT" * "GiT"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
```

Both of the errors are seeking for the integer value in order to multiply.

Comments:

Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do. Any text for the rest of the line following a hash mark (#) is part of a comment.

The following line is called a comment:

 # I am a comment. Python ignores me.

Comments can also be used at the end of some executable statements.

```
>>> 2 + 5    # Addition of 2 and 5.
7


>>> "GIT" * 5    # replicating Girls in tech 5 times
'GITGITGITGITGIT'
```

Floating Point Calculations:

Floating-point numbers are not suitable for exact comparison. When you use floats you need to worry about tiny differences between numbers when doing comparisons.

The smallest possible difference between two floats is :

EPSILON

1.8 + 0.1 = 1.9000000000000001 [Epsilon is added]

(1.8 + 0.1) - 1.9 = 2.220446049250313e-16 (EPSILON)

The difference between the 2 numbers is precisely EPSILON.

[Note:
You can check epsilon value with python

```
import sys
print(Sys.float_info.epsilon )  # 2.220446049250313e-16

(1.8 + 0.1) - 1.9 == Sys.float_info.epsilon   # True
```
]

This EPSILON’s worth of inaccuracy is often too small to matter, but it does when you’re doing comparisons.