Python, Day 2.5: Operators

Andrew Bydlon

January 6, 2019

Arithmetic operators

Many common operators are loaded by default into python:

- '+' Addition
- '-' Subtraction
- '*' Product
- '/' Division
- '**' Exponentiation
- '//' Integer division
- '%' Remainder or mod

Let's do some demonstrations!

Examples

Example

```
>>> 46-33
    13
>>> 26/3
    8.666666666666666    # note the conversion to float type!
>>> 47//7
    6
>>> 47% 7
    5    # note 47 = 6*7+ 5, or 47/7 = 6 + 5/7
>>> 66/3*2
    44.0
```

Order of operations

An important note is in which order operations are performed by python. Here is the hierarchy:

1. Parenthesis: ()
2. Exponentiation: **
3. Positives/negatives +x (or) -x (for the beginning of an expression)
4. Product and Division Operators: *, /, //, %
5. Addition and Subtractions: +, -
6. Logical operators: not > and > or

If two operations are on the same level, then it defaults to **left to right** operations.

What would be the result of the following operation?

$$3 * 4/6 * 3 + 6\%4//3$$

Compounded Operators

One can save a few keystrokes while redeclaring a variable. For example, we can replace ‘Var = Var + 4’ with ‘Var += 4’.

In general, any of the previously discussed operators (+,-,*,**,/,//,%) can be adjoined to the left of ‘=’.

Example

```
> > > MyVar = 3
> > > MyVar *= 7
> > > print(MyVar)
21
> > > MyVar //= 5
> > > print(MyVar)
4
```

Dynamic Type Conversion

Just a small note; if an *int* type variable interacts with a *float* type, the result will be made floating type.

Example

```
> > > 3*3.0
9.0
> > > type(3*3.0)
float
```

Similarly, for a *float* or *int* interacting with a *complex*, they are all made complex type.

Example

```
> > > type(3*(1+2j))
complex
```

Boolean Variables

Boolean variables take values either True or False. They can appear via *relational operators*:

>, <, >=, <=, !=, and ==.

Example

```
> > > 5 > 3
True
> > > 5 <= 3
False
> > > 5 != 3
True
> > > 5 == 3
False
```

Note: Be careful about == (checks equal) vs = (declaration)

and, or, not

Similar to how + or * are operators on numeric variables, we have a few operators for boolean variables:

1. and: Bool1 and Bool2 returns True if both Bool1 and Bool2 are true statements. Otherwise it returns False.
2. or: Bool1 or Bool2 returns False if both Bool1 and Bool2 are false statements. Otherwise it returns True.
3. not: not as an operator interchanges True and False statements.

Assignment 3

Let's continue from Assignment 2 (you may copy and paste your code) and make a financial advisory program!

- Previous code excluding the output to the user.
- Ask the user how much money they require after t years.

Then output the following info:

It is (Some Boolean Var) that you will have the money after (t) years.
The required rate to achieve your goal is r.

r can be acquired by solving for it in the previous equation with A the newly acquired information of the second bullet.

Remember to upload your .py file when you finish!