{:title “Python - Numbers” :layout :post :date “2016-10-06” :tags [“Python” “Code” “Guide”]}

We’ll learn about different numbers and doing math in python. So open up a python interpreter if you forgot how to just look at the previous post. Just like before try out each line one by one into the python interpreter.

```
5 + 2

5 - 2

5 * 2

5 / 2

5 % 2

5 ** 3

5 ** (1/2)

5 ** (1/2.0)

float(2)

float(5/2)

float(5) / 2

5 / float(2)

5 / 2.0

5 - 2 * 5

(5 - 2) * 5
```

Explanations below:

Line 1-7:

```
5 + 2

5 - 2

5 * 2
```

```
5 / 2
```

Basic math that you should've learned in school. Nothing interesting per say. Expect you may notice that the 5/2 gives you 2 which is odd but I'll explain that in the explanation on line 13 and 15. Line 9:

```
5 % 2
```

Ah we see the % symbol once again but this time it is not in a string. It is used as an operator just like +, -, *, or /. However this time it is not a string formator in this context and is called a modulo and returns the reminder of the division of 5 / 2. Which is what you get 1. This may be a little weird having not experienced such an operator before but it is quite useful for determining the divisbility of a number and etc. Line 11:

```
5 ** 3
```

Using ** is how you raise an integer to an exponent. In this cause you are raising 5 to the third power. Line 13:

```
5 ** (1/2)
```

Huh. Why did I get 1? That doesnt make sense. This is because the result of doing 1/2 is 0. So you are essentially doing 5 raised to the 0 power which is of course anything raised to the 0 power is 1. But why did 1/2 get evaluated to 0 and not 0.5? I'll talk about that in the explanation for line 15 in more depth. Line 15:

```
5 ** (1/2.0)
```

This time the result is 2.23 which is the result I was hoping. So what gives by just adding a decimal point to the 2? Yes. Since you made 2 into a float there are more decimal points in the calculation so that 1/2.0 is evaluated to 0.5 however when doing 1/2 you don't have the significant figures and thus rounded down to 0. So when doing calculations involving precision use floats. Line 17:

```
float(2)
```

Converts the integer 2 into a float 2.0 without actually adding to the decimal point yourself. Line 19:

```
float(5/2)
```

This gives 2.0 which makes sense because 5 / 2 is 2 and than the 2 is converted to 2.0 as a float. Line 21:

```
float(5) / 2
```

The 5 is turned into a float so it is 5.0 and than 5.0 / 2 will give you 2.5 which is what we expect. Line 23:

```
5 / float(2)
```

Showing you that it can be done with the other digit and that only one has to be a float in order for the extra decimal point to appear in the final result. Line 25:

```
5 / 2.0
```

This is equivilant to that of line 23. Line 27:

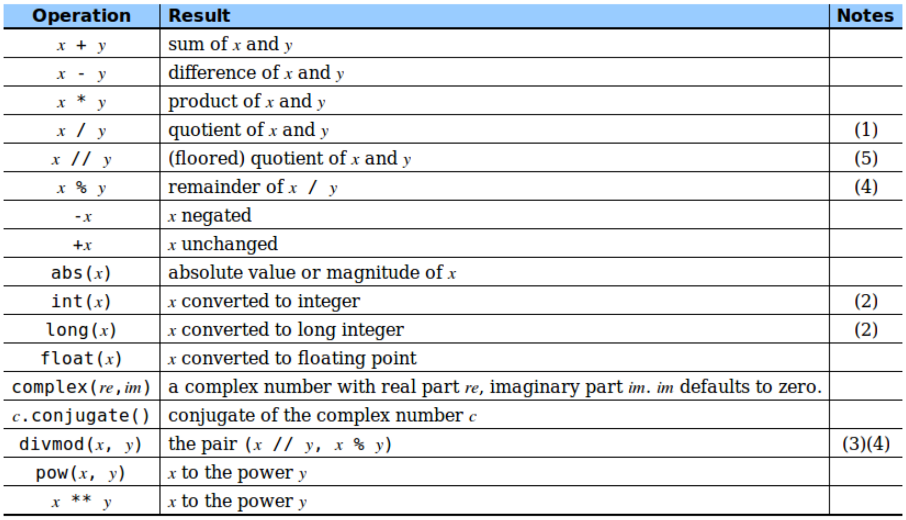
```
5 - 2 * 5
```

The order of operations in python is done by PEMDAS. Thus multiplication is done first so 2 * 5 is 10 and than 5 - 10 is -5. Which is what you get. Line 29:

```
(5 - 2) * 5
```

This time we had parantheses around 5 - 2 so that is the first thing done in the oder of operations. So 5 - 2 gives us 3 and than 3 * 5 is 15.

For those wanting all the numeric types and learn more about them here is a table from python's documentation.



Operation	Result	Notes
`x + y`	sum of *x* and *y*	
`x - y`	difference of *x* and *y*	
`x * y`	product of *x* and *y*	
`x / y`	quotient of *x* and *y*	(1)
`x // y`	(floored) quotient of *x* and *y*	(5)
`x % y`	remainder of `x / y`	(4)
`-x`	*x* negated	
`+x`	*x* unchanged	
`abs(x)`	absolute value or magnitude of *x*	
`int(x)`	*x* converted to integer	(2)
`long(x)`	*x* converted to long integer	(2)
`float(x)`	*x* converted to floating point	
`complex(re,im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	
`c.conjugate()`	conjugate of the complex number *c*	
`divmod(x, y)`	the pair `(x // y, x % y)`	(3)(4)
`pow(x, y)`	*x* to the power *y*	
`x ** y`	*x* to the power *y*	

Figure 1: numerictypes