Numeric operations in Python

The following operations are defined for numeric types (i.e. `int` and `float`):

- `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`
- `x // y`: floored quotient of `x` and `y`
- `x % y`: remainder of `x / y`
- `-x`: `x` negated
- `+x`: `x` unchanged
- `abs(x)`: absolute value or magnitude of `x`
- `int(x)`: `x` converted to integer
- `float(x)`: `x` converted to floating point
- `x ** y`: `x` to the power `y`

A complete list of operations and more detailed discussion can be found in the Python documentation.

Complex numbers

Python has built-in complex numbers:

```
c = 3+6j
print(c)
print(type(c))
```

Access real and imaginary parts:

```
print(c.real)
print(c.imag)
```

The parts have type `float`:

```
print(type(c.real))
print(type(c.imag))
```

Numeric conversions

It is often useful to convert between integers and floating point numbers. Python fully supports mixed arithmetic: when a binary arithmetic operator (such as `+` or `*`) has operands of different numeric types, the operand with the "narrower" type is widened to that of the other, where integer is narrower than floating point. Comparisons between numbers of mixed type use the same rule. The constructors `int()` and `float()` can be used to produce numbers of a specific type.

Let's see some examples:

```
x = 1 + 2.0
print(x)
print(type(x))
```

In this case the integer `1` is "widened" or converted to the floating point number `1.0` before the addition.

It is possible to manually convert from `int` to `float`:

```
x = float(3)
print(x)
print(type(x))
```

It is also possible to convert from `float` to `int`:

```
x = int(4.7)
print(x)
print(type(x))
```

Let's see what happens with negative numbers:

```
x = int(-8.9)
print(x)
print(type(x))
```

The Python `int()` constructor rounds floating point numbers towards `0`.

Converting to and from strings

Python makes it very easy to convert numbers to and from strings. This is a useful feature when trying to read numbers from a text file. Let's see it in action:

```
my_num_str = "42"
print(type(my_num_str))

my_num_int = int(my_num_str)
my_num_float = float(my_num_str)

print(my_num_int)
print(type(my_num_int))

print(my_num_float)
print(type(my_num_float))
```

It also easy to convert from a numeric type back to a string:

```
print("exam score:" + str(95) + "%")
```

An attempt to concatenate a string with a numeric type is an error:

```
print("exam score:" + 95 + "%")
```

It is better to use string formatting for this:

```
print("exam score: {}%".format(95))
```