1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

4 Numeric Values and Types In Python

In this section we introduce the concepts of values, types, literals, operators, and expressions in the context of numbers in Python.

4.1 Values, types, and literals

These three concepts need to be introduced together before we can demonstrate anything.

> *Definition*: A value is a piece of information in a program, for example a number.

> *Definition*: A type is a name that describes the kind of information that a value represents.

> *Definition*: A literal is a textual representation of a value in its simplest form.

Now we can demonstrate these three concepts in the Python REPL (read-eval-print-loop).

```
$ python3
>>> 42
42
>>>
```

At the prompt >>>, we typed the characters `'4'` and `'2'` which make up a numeric literal representing the value 42.

We press the enter key, and the REPL reads the the literal, parses it, converts it to its internal representation, then prints it out again, reconverting it to the textual literal.

Now we can ask the REPL about which internal representation it used with the `type` function.

```
>>> type(42)
<class 'int'>
>>>
```

The Python type for whole numbers is `int`, short for integer.

```
>>> type(4.2)
<class 'float'>
>>>
```

The Python type for fractional numbers is `float`, short for "floating point".

4.1.1 int

Python only provides type `int` for whole numbers.

They can be as big as memory permits.

```
>>> 10000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000
>>>
```

So we don't need other types for bigger numbers like `long` in C and Java.

`int` literals can be in decimal, octal, or hexadecimal.

```
>>> 123
123
>>> 0o123
83
>>> 0x123
291
>>>
```

`123` means $123_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 1 \times 100 + 2 \times 10 + 3 \times 1 = 123$.

`0o123` means $123_8 = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 1 \times 64 + 2 \times 8 + 3 \times 1 = 83$.

`0x123` means $123_{16} = 1 \times 16^2 + 2 \times 16^1 + 3 \times 16^0 = 1 \times 256 + 2 \times 16 + 3 \times 1 = 291$.

4.1.2 float

Python only provides type `float` for fractional numbers.

In other languages we usually see at least two, `float` and `double`, short for "double precision floating point".

Python's `float` is already the highest precision the current computer supports.

`float` literals must have either a decimal point or an exponent to distinguish them from `int`s.

```
>>> 4
4
>>> 4.
4.0
>>> 4e0
4.0
>>>
```

`4e0` means $4 \times 10^0 = 4 \times 1 = 4$.

`6.02e23` means $6.02 \times 10^{23} = 602000000000000000000000$.

`9.10938356e-31` means $9.10938356 \times 10^{-31} =$
0.000000000000000000000000000000910938356.

4.1.3 int and float are classes

Recall:

```
>>> type(42)
<class 'int'>
>>> type(4.2)
<class 'float'>
>>>
```

Python is telling us that `int` and `float` are classes.

This is our first clue that Python is an object-oriented programming language.

4.2 Arithmetic operators and expressions

Definition: An operator is a symbol which represents a computational operation, For example the plus sign, +, represents addition.

Definition: An expression is sequence of symbols that represent a value.

An expression may be as simple as a single literal, or contain many literals, operators and other symbols that represent values.

4.2.1 Arithmetic operators

Most arithmetic operators in Python are *binary, infix* operators.

That is, they are written in between the two values that they are operating on, just as in Mathematics.

The binary, infix operators in Python are:

operator	*meaning*
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	fractional division
`//`	floored division
`%`	modulo (remainder)
`**`	exponentiation

These operators are *unary, prefix* operators.

They are written before the single value they operate on.

operator	*meaning*
+	positive
-	negation

4.2.2 Mixed-mode arithmetic

In Python you may mix the types of numbers you use in expressions.

This is not true in all languages.

It is easy to predict what type of result you will get.

Python will automatically convert `int`s to `float`s before combining them, but it will not convert `float`s to `int`s.

Automatic conversion like this is called type *coercion*.

4.2.3 Arithmetic examples with explanations

```
>>> -1
-1
>>> - 1
-1
>>> + 4
4
>>>
```

A unary minus, -, negates the value that follows it.

A unary plus, +, does nothing.

```
>>> 1 + 2
3
>>> 1.1 + 2.2
3.3000000000000003
>>> 1 + 2.2
3.2
>>>
```

Binary plus works like you would expect.

Adding two `int`s gives an `int` result.

Adding a `float` to anything yields a `float`.

Note particularly that floating point arithmetic is not always perfectly accurate due to the finite number of bits in the binary representation of floating point numbers.

```
>>> 3 - 1
2
>>> 1 - 3
-2
>>> 2 * 3
6
>>> 2.5 * 3
7.5
>>> 3 ** 2
9
>>> 3.3 ** 2.2
13.827086118044146
>>>
```

No surprises here.

There are two kinds of division.

Fractional division, `/`, always returns a `float`.

```
>>> 10 / 2
5.0
>>> 10 / 3
3.3333333333333335
>>> 10.0 / 3.0
3.3333333333333335
>>>
```

Floored division, `//`, does a division, and then the *floor* operation.

To floor a number is to round the number off to the closest whole number, but only ever rounding *down*, towards $-\infty$.

For example the floor of 7 is $\lfloor 7 \rfloor = 7$.

The the floor of 7.7 is $\lfloor 7.7 \rfloor = 7$.

The the floor of -7.7 is $\lfloor -7.7 \rfloor = -8$.

Floored division is most often used only with `int`s, where it behaves like div.

```
>>> 10 // 2
5
>>> 10 // 3
3
>>>
```

The results when negatives are involved can be surprising.

```
>>> -10 // 3
-4
>>> 10 // -3
-4
>>>
```

Floored division can be used with `float`s too, and the results are always whole numbers, but still with type `float`.

```
>>>  10.0 // 3.0
3.0
>>>
```

The modulo operator, `%`, returns the remainder after floored division.

```
>>> 10 // 3
3
>>> 10 % 3
1
>>> -10 // 3
-4
>>> -10 % 3
2
>>>
```

$(10 \operatorname{div} 3) \times 3 + 10 \bmod 3 = 3 \times 3 + 1 = 9 + 1 = 10.$

$(-10 \operatorname{div} 3) \times 3 + -10 \bmod 3 = -4 \times 3 + 2 = -12 + 2 = -10.$

It can be used with `floats` as well.

4.2.4 Operator precedence

The arithmetic operators in Python may be combined to make more complex expressions.

Expressions are evaluated from left-to-right, but the normal order of precedence applies.

highest precedence	**
	unary +, unary -
	*, /, //, %
lowest precedence	binary +, binary -

Use parentheses to override the normal precedence.

Example expressions, using the REPL as a calculator:

What is the molecular weight of water? The atomic weight of Hydrogen is 1.00794, the atomic weight of Oxygen is 15.9994, and the formula for water is H_2O.

```
>>>  2 * 1.00794 + 15.9994
18.01528
>>>
```

How much will be in my savings account if I start with $100.00, compound interest is paid monthly at 3.1% per annum, and I wait 18 months?

```
>>> 100.0 * (1.0 + 3.1 / 100 / 12) ** 18
104.753526774703
>>>
```

Section summary and further reading

This section covered:

- the concepts of: values, types, literals, operators, and expessions;
- the Python numeric types `int` and `float`;
- writing Python expressions with literal values, and operators; and
- using the Python REPL as a calculator.

For this section you should also read:

- Using Python as a Calculator from The Python Tutorial.