M30299 – Programming
Lecture 03 – Computing with Numbers

Matthew Poole & Nadim Bakhshov

moodle.port.ac.uk

School of Computing
University of Portsmouth

2020/21

Introduction to lecture

- In the previous lecture we:
  - introduced the concept of a **data type**; and
  - covered the basics of Python's numeric data types, `int` and `float`.
- In this lecture, we will:
  - review some of the ideas from that lecture.
  - cover the remaining important details of the numeric types.
- We will cover the use of some more advanced data types in the following few lectures.

Review of Python's data types

- Recall Python's `int`, `float` and `str` data types:

```
>>> type(3)
<class 'int'>
>>> type(-42)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type("Hello")
<class 'str'>
>>> type(3 + 4.2)
<class 'float'>
```

Review of Python's data types

- Consider the following version of our weight conversion program:

```
kilosInput = input("Enter a weight in kilos: ")
kilos = int(kilosInput)
pounds = 2.2 * kilos
print("The weight in pounds is", pounds)
```

- If the user enters 10 at the prompt, what will be the **values** and **types** of the following variables at the end of the program?
    - `kilosInput`
    - `kilos`
    - `pounds`

Obtaining user input ... revisited

- What do you think will happen if we execute the following code and the user enters 10?

```
kilos = input("Enter a weight in kilos: ")
pounds = 2.2 * kilos
```

- What about the following code if the user enters their name?

```
name = int(input("Enter your name: "))
print("Hello,", name)
```

Arithmetic involving ints and floats

- We saw last lecture that the `int` and `float` data types include the arithmetic operators +, −, ∗, / and ∗∗ (power).
- Each of these operators takes two **operands**. Usually:
  - if both operands are `ints` (e.g. `3 + 4`), the result is an `int` (7).
  - if both are `floats` (e.g. `3.1 + 4.2`), the result is an `float` (7.3).
- If the operands are of different types, e.g.:

```
3 + 4.2
```

then the int (3) is **automatically converted** into a float (3.0), and then a floating-point operation is performed (to give 7.2).

Floating point and integer division

- There is one exception to the above – the division operator / always performs floating point division:

```
>>> 11 / 4
2.75
```

- There is a separate **integer division** operator //:

```
>>> 11 // 4
2
```

- (When given two `int` operands, this gives an `int` result.)

Integer division and remainder

- Finally, the operator % that gives the **remainder** of an integer division:

```
>>> 11 % 4
3
```

- Integer division and remainder can often be useful; e.g.:

```
>>> children = 3
>>> sweets = 14
>>> sweetsEach = sweets // children
>>> sweetsLeftForMe = sweets % children
```

- What are the values of `sweetsEach` and `sweetsLeftForMe`?

Dangers of floating-point arithmetic

- Recall that we've said that `float` values are represented by the computer during a fixed amount of space (64 binary digits).
- This limits the accuracy at which some real-world values can be represented and sometimes leads to some surprising issues.
- To see this, let's first pretend that computers worked with decimal (base 10) numbers instead of binary (base 2).
- Assume also that we **have to** represent values in decimal using **at most 5 figures**.
- The number $\frac{1}{3}$ would be represented as 0.3333; this is clearly an **approximation** to its true value.
- We know, for example, that $\frac{1}{3} \times 3$ is equal to 1, but 0.3333×3 gives a **different** value; i.e. 0.9999.

Dangers of floating-point arithmetic

- Some numbers such as 0.1 can be represented accurately using a limited number of digits in decimal, but they cannot in binary. Try:

```
>>> x = 0
>>> x = x + 0.1  (repeat this statement ten times)
>>> x
0.9999999999999999
```

- We expected the final result to be 1, but it isn't!.
- These problems are true of all programming languages that use floating point numbers — Python, C, Java ...
- (This sometimes needs to be kept in mind when using `float` — e.g. when testing whether two numbers are equal.)

Dangers of floating-point arithmetic

- These problems are true of all programming languages that use floating point numbers — Python, C, Java, Pascal . . .
- (This sometimes needs to be kept in mind when using `float` — e.g. when testing whether two numbers are equal.)

Built-in numeric functions

- There are a few other built-in functions that we may find useful:
- The `round` function rounds a `float` to the nearest `int`:

```
>>> round(3.8)
4
>>> round(6.2)
6
```

- The `round` function can also **restrict** the number of digits after the decimal point in a float:

```
>>> round(1.237143225, 2)
1.24
```

Built-in numeric functions

- The `abs` function returns the **absolute** (i.e. positive) value of a number:

```
>>> abs(-3)
3
>>> abs(3)
3
```

- The `pow` function does the same thing as `**`:

```
>>> pow(2, 3)
8
>>> pow(2, 0.5)
1.4142135623730951
```

Using the math module

- Other mathematical functions are not built-in to the language, but are defined in a **module** called the math module.
- A module is often just another Python file. (Large programs are usually split into several separate modules.)
- There are many Python modules available for doing graphics, Internet programming, connecting to databases, etc.
- To use a module, we first have to **import** it:

```
>>> import math
```

Using the math module

- Among the things the math module provides are **constants**; a constant is simply a "variable" whose value should not be changed.
- To use a constant or function from another module, we need to prefix its name with the module's name and a dot; e.g.

```
>>> math.pi
3.1415926535897931
```

- Functions in the math module include one for finding square roots:

```
>>> math.sqrt(2)
1.4142135623730951
```

Translation of mathematical formulae

- To summarise the use of numerical data types, let's take a mathematical formula and translate it into a Python assignment.
- Let's take the formula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

  which gives one of the roots of a quadratic equation $ax^2 + bx + c = 0$.
- As a Python assignment, this can be written as follows:

```
>>> x1 = (-b + math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
```