INTPROG
Introduction to Programming

poolem.myweb.port.ac.uk/intprog

Practical Worksheet P02: Working with Numeric Types

Introduction

This worksheet is designed to acquaint you with Python's numerical types, and to develop your numerical problem solving skills. It assumes that you have a good understanding of the material covered in Worksheet P01, and have managed to complete at least the first few of its programming exercises. If this is not the case, make sure that you spend some extra time on Worksheet P01 and, if necessary, get some help from a member of staff in the practical class, the unit textbook (Zelle), your fellow student programmers, or the Tutor Centre (room BK1.16).

Work through this worksheet at your own pace and, as always, feel free to experiment with the Python language using the shell until you fully understand each concept.

You should try to complete this worksheet by next week's practical session. We will take a look at your solutions in next week's practical and give you some feedback to help you improve your programming skills and to correct any misunderstandings you may have. We will continue to do this each week, so it is important that your attempted solutions are available (electronically) in the following week's practical. (If you do most of your work at home, make sure that you copy the solutions file to the N: drive, or copy them to a USB stick, before the following practical.) In week 3 you will have your first in-class assessment, and this will be based on these first two worksheets.

Start PyScripter

Start PyScripter from MyApps. Before starting the exercises, we'll first run through the main concepts covered in Lecture P02 (and P03). This is best done using the shell (the bottom region of the PyScripter window.)

Python's data types

Python has a special function called `type` which gives the ***data type*** of any value, variable, or indeed any expression. First, let's find the type of some `int`, `float`, `str` and `bool` data values; enter:

```
type(3)
type(-64)
type(53.2173)
type(3.0)
type(123456789123456789)
type("hello")
type("123")
type('123')
type(True)
type("False")
```

Now, let's experiment with some variables:

```
x = 2
type(x)
y = 2.3
type(y)
x = 3.2
type(x)
x = "hello"
type(x)
x = True
type(x)
```

We see that the type of a variable is simply the type of its current value. Notice also, as seen in the case of `x`, that the type of a variable can change. Finally, let's try some longer (numerical) expressions:

```
x = 10
x + 27
type(x + 27)
x - 3.2
type(x - 3.2)
12 * x
type(12 * x)
x / 5
type(x / 5)
x / 2.5
type(x / 2.5)
```

Notice here that the type of an expression is simply the type of whatever it evaluates to (i.e. the type of its value). Also, where we have an operation with both an `int` and a `float` operand (e.g. as in `x - 3.2`) the type of the result is a `float` (the `int` value is automatically converted into a `float` value before the operation is performed). Also notice that divisions using `/` are always `float` divisions, even if both the operands are integers.

Experiment with the `type` function until you are satisfied that you understand the concept of a data-type, and how the arithmetic operators work with respect to the types of their operands.

Operators and precedence

The arithmetic operators `+`, `-`, `*`, `/`, and `**` (power) obey the standard mathematical precedence rules. Try to predict the results of the expressions below before entering them:

```
x = 3
y = 4
1 + x * y
(1 + x) * y
3 ** 2
2 * 3 ** 2
15 - 12 / y * x
15 - 12 / (y * x)
```

Type conversions and built-in numeric functions

We can convert between `int`s, `float`s and `str`ings as follows:

```
float(3)
int(5.6)
str(77.1)
int("456")
```

Try also:

```
x = 4.8
int(x)
round(x)
str(x)
```

It is important to realise that none of these functions changes the value (or type) of the variable `x` itself; they just give a new value of a different type. To check this, look at the value of `x` now; enter:

```
x
```

Integer division and remainder

Enter the following expressions:

```
23 / 5
23 // 5
23 % 5
```

The first expression here is a standard `float` division. The second is an ***integer division***; and the final one gives the ***remainder*** for this integer division. We see that 5 goes into 23 four times, with remainder 3.

Using the math module

Many useful functions and constants are not built into the Python language, but appear in external "modules" (other files of Python program code). A good example is the math module. To be able to use the facilities in this module, we first need to ***import*** it:

```
import math
```

Now, try entering:

```
math.pi
math.sqrt(2)
math.sqrt(-1)
math.sin(0)
math.sin(math.pi / 2)
math.log(1)
math.log(math.e)
```

Note: `sqrt` gives the square root of a number, `sin` gives the sine of an angle expressed in radians, and `log` gives the natural log of a number. Don't worry too much here if you don't remember much about trigonometric functions!

Warm-up exercise

For some practice using the numerical operators, first type the following at the shell prompt:

```
x1 = 1
y1 = 2

x2 = 4
y2 = 6
```

in order to represent two points with coordinates $(1, 2)$ and $(4, 6)$ in two-dimensional space (see Figure **??**).

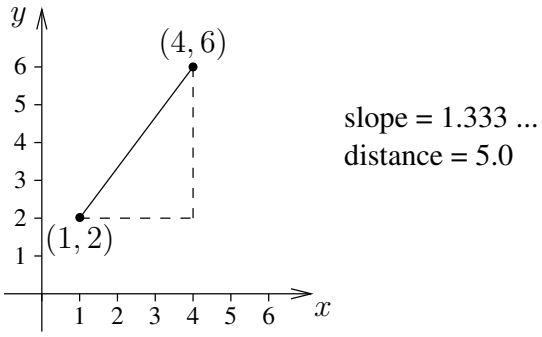

Figure 1: . Slope and distance between two points.

Now, try to compose Python expressions that are equivalent to the following mathematical expressions. The first one gives the ***slope*** of the line that passes through the two points:

$$\frac{y_2 - y_1}{x_2 - x_1}$$

The next expression uses Pythagoras' theorem to give the ***distance*** between the two points:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If you have translated the expressions correctly, they should give values 1.333... and 5.0 respectively.

Programming exercises

The following exercises should be solved using PyScripter's editor and saved to a file called pract02.py, which should include some comments (lines beginning with the # symbol) that include your name and student number.

1. Write a function `circumferenceOfCircle` that asks the user for the radius of a circle, and then outputs its circumference. (Use the formula $circumference = 2\pi r$. For π, use `math.pi` from the math module.)

2. Write a function `areaOfCircle` that asks the user for the radius of a circle, and then outputs its area (using the formula $area = \pi r^2$).

3. Write a function `costOfPizza` that asks the user for the diameter (not the radius) of a pizza (in cm), and then outputs the cost of the pizza's ingredients (based on its area) in pence. Assume that the cost of the ingredients is 1.5p per square cm.

4. Write a function `slopeOfLine` that first asks the user for four values `x1`, `y1`, `x2` and `y2` that represent two points in two-dimensional space (i.e. points with coordinates (x_1, y_1) and (x_2, y_2)). The function should then output the slope of the line that connects them. (Hint: copy the relevant expression from the shell — see the warm-up exercises!)

5. Write a function `distanceBetweenPoints` that asks the user for four values `x1`, `y1`, `x2` and `y2` that represent two points in two-dimensional space, and then outputs the distance between them. (Hint: copy the relevant expression from the shell see the warm-up exercises!)

6. Write a function `travelStatistics` which asks the user to input the average speed (in km/hour) and duration (in hours) of a car journey. The function should then output the overall distance travelled (in km), and the amount of fuel used (in litres) assuming a fuel efficiency of 5 km/litre.

7. Write a function `sumOfNumbers` that outputs the sum of the first n positive integers, where n is provided by the user. For example, if the user enters 4, the function should output 10 (i.e. 1 + 2 + 3 + 4). (Hint: This function should use a loop.)

8. [harder] Write a function `averageOfNumbers` which outputs the average of a series of numbers entered by the user. The function should first ask the user how many numbers there are to be inputted.

9. [harder] Write a function `selectCoins` that asks the user to enter an amount of money (in pence) and then outputs the number of coins of each denomination (from $£2$ down to 1p) that should be used to make up that amount exactly (using the least possible number of coins). For example, if the input is 292, then the function should report: $1 \times £2$, $0 \times £1$, 1×50p, 2×20p, 0×10p, 0×5p, 1×2p, 0×1p. (Hint: use integer division and remainder).