Mathematical Data Operators

Programming often requires numbers to be manipulated. This usually involves standard mathematical operators: addition, subtraction, multiplication and division. It can also involve some more complex operations.

1. Adding, subtracting and multiplying

These are the easiest operators to use. The symbols used for adding (+) and subtracting (−) are exactly the same as in normal maths. For multiplication we use the symbol * - an x might mean something different.

✎Exercise 1.1

Evaluate the pseudo-code mathematical expressions below, giving the value of `answer` in each case

```
a) answer ← 7 + 4
b) answer ← 5 + 0
c) answer ← 7 - 4
d) answer ← 7 * 4
e) answer ← 7 * 1
f) answer ← 7 * 0
```

✎Exercise 1.2

If the following pseudo-code is run first, what is the value returned by each of the expressions below. Assume the values are always as initially assigned by the pseudo-code.

```
varOne ← 6
varTwo ← 3
varThree ← 2

a) varOne + varTwo
b) varThree + varTwo + varTwo
c) varOne - varTwo
d) varTwo - varThree
e) varThree * varOne
f) varTwo - varOne
```

The basic operators are used in exactly the same way when programming with Python:

```
valueOne = 4
valueTwo = 2
answer = valueOne + valueTwo
print(answer)
```

💻Programming exercise 1.1

Write a program which multiplies two numbers together.

a) What happens when you make one of the numbers negative?
b) What happens when you make both of the numbers negative?
c) What happens when you make one of the numbers 0?

---

2 Division

Division is a little more complex because decimal numbers can easily get involved.

The operator for division is /

```
answer ← 8 / 2
```

This expression evaluates to 4 as 8 divided by 2 is 4.

✎Exercise 2.1

Give the values for answer in each case:

```
a) answer ← 8 / 4
b) answer ← 6 / 2
c) answer ← 7 / 1
d) answer ← 3 / 2
```

Don't forget that you cannot divide by 0. Any attempt to do something such as:

```
answer ← 8 / 0
```

will end with an error.

💻Programing exercise 2.1

Try this program and check the output.

```
# program to show division
valueOne = 5
valueTwo = 2
valueThree = 8
answerOne = valueThree / valueTwo
answerTwo = valueThree / valueOne
print(answerOne)
print(answerTwo)
```

You should see that both answers are given as decimals. You would expect this for answerTwo – 8 / 5 is 1.6 (one and three fifths remainder). But answerOne is written as 4.0 rather than simply as 4.

This will always happen in Python. Every time you divide two numbers you'll get a decimal number, even if they divide perfectly. There are really good reasons for this and we'll come back to it at another time.

💻Programming exercise 2.2

```
# long decimal numbers
answer = 8 / 3
```

The answer is 2 and two thirds. In maths we would usually write this as 2.666 recurring (or 2.667 if we're rounding values). Python deals with long decimal numbers by only going to a set number of decimal places.

This can effect the accuracy of calculations as very long numbers will always be rounded. This is something to be aware of when dividing.

💻Programming exercise 2.3

Write a program which divides a number by 0. What error message do you get?

3 The Exponential Operator

Using the exponential operator raises a value to a power. This is what happens when you square or cube a number – 4 exponential 2 is 4 squared.

There is no pseudo-code version of the exponential operator. In Python the symbol ^ is used (shift + 6 on your keyboard).

`2^2` is 2 squared - in other words, `2*2`

💻Programming exercise 3.1

```
# program to square and cube numbers
varOne = 2
varTwo = 4
print(varOne ^ 2)
print(varTwo ^ 3)
print(varTwo ^ varOne)
```

Answers are provided for this programming exercise.

---

4 Integer Division

Sometimes you want the whole number that results when you divide two numbers, ignoring any decimal values or remainders. This is known as **Integer Division**. The pseudo-code operator for integer division is **DIV**.

```
answer ← 3 DIV 2
```

gives a value for answer of 1

The value returned when the DIV operator is applied is the **whole number** without any remainder.

The expression 3/2 would return a value of 1.5. When DIV is used any remainder is discarded and the value 1 is returned.

```
answer ← 8 DIV 3
```

gives a value for answer of 2. 8/3 is 2.667, so the remainder is discarded and the whole number returned.

DIV always gives just the whole number.

Exercise 4.1

Give the values returned by each of the following:

a) 7 DIV 4
b) 8 DIV 2
c) 9 DIV 4
d) 15 DIV 2
e) 7 DIV 7
f) 3 DIV 8

Integer division can seem confusing to begin with. You need to remember to just get rid of any remainder and always round down to the lower whole number.

In Python, integer division is handled by using the symbol //

Programming exercise 4.1

Try this program

```
# integer division program
varOne = 7
varTwo = 3
varThree = 2
print(varOne / varTwo) # dividing normally
print(varOne // varTwo) # integer division
print(varOne / varThree) # dividing normally
print(varOne // varThree) # integer division
print(varThree // varOne) # integer division
```

Answers are provided for this programming exercise.

5 The Modulus Operator

Sometimes you want just the **remainder** from division. This is handled by using the **modulus** operator. In pseudo-code this uses the operator **MOD**.

```
answer ← 7 MOD 4
```

7/4 would be 1.75, or 1 remainder 3. MOD returns just the remainder value, so 7 MOD 4 evaluates to 3

This will probably seem strange to begin with…

```
answer ← 3 MOD 2
```

5/2 = 2.5 - or 2 remainder 1. So, 3 MOD 2 evaluates to 1

The trick to using MOD is to get out of the habit of using decimals and think about remainders instead. When you first learned to divide numbers that's what you did. It's only later that people have made things more complicated by using decimals.

Exercise 5.1

Evaluate the following expressions.

a) 7 MOD 2
b) 8 MOD 3
c) 9 MOD 4
d) 5 MOD 3
e) 6 MOD 2

---

2 MOD 5 is interesting

2 / 5 = 0.4. This means that 2 goes into 5 0 times with ⅖ remainder. So the remainder is 2. So 2 MOD 5 evaluates to 2.

Whenever the number to the right of MOD is greater than the number to the left, the answer is the number on the left.

- 2 MOD 9 = 2
- 5 MOD 7 = 5
- 324 MOD 398 = 324

Exercise 5.2

Evaluate each of

a) 5 MOD 6
b) 9 MOD 9

The Modulus operator allows you to check if a number is perfectly dividible by another number. If this is the case then the answer to valOne MOD valTwo is 0. This is surprisingly helpful when you get into more complex programming problems.

In Python the operator % is used to signify MOD.

Programming exercise 5.1

```
# program using modulus division
varOne = 7
varTwo = 3
varThree = 2
print(varOne % varTwo)
print(varOne % varThree)
print(varThree % varOne)
```

Answers are provided for this programming exercise.

6 User Inputted Numbers

All numbers entered by users in pseudo-code or in Python are initially stored as **strings**. This causes a problem when we want to do maths with a value as computers will think of "42" as a sequence of keyboard characters – as a string. If we try and add "72" to it the computer will just **concatenate** the values.

This means that "42" + "72" evaluates as "4272" – as a string with the two numbers joined together.

To deal with this we need to **convert** strings into numbers before we do anything mathematical with them. This can be done in both pseudo-code and in Python.

We will assume that all numbers are going to be entered as whole numbers – as **integers**.

The pseudo-code to do this is:

```
valueOne ← USERINPUT
valueOne ← STRING_TO_INT(valueOne)
```

In Python we can include the conversion on the same line as the data input:

```
valueOne = int(input("Enter a number: "))
```

The `int()` command converts the string entered at input into an integer.

💻 Programming exercise 6.1

Write a program which asks the user to enter two numbers. The program should then multiply the two numbers together and report the answer.

Solutions to exercises

Exercise 1.1

a) 11
b) 5
c) 3
d) 28
e) 7
f) 0 – don't forget that if you multiply by 0 you get 0

Exercise 1.2

a) 9
b) 8
c) 3
d) 1
e) 12
f) -3 – minus numbers will work just fine

Exercise 2.1

a) 2
b) 3
c) 7
d) 1.5

Programming exercise 3.1

- varOne ^ 2 is 2 ^ 2 or 2 squared (2 * 2) = 4
- varTwo ^ 3 is 4 ^ 3 or 4 cubed (4 * 4 * 4) = 16 * 4 = 64
- varTwo ^ varOne is 4 ^ 2 or 4 squared (4 * 4) = 16

Exercise 4.1

a) 1
b) 4
c) 2
d) 7
e) 1
f) 0 – 3/8 is less than 1, so the value is rounded down to 0

Programming exercise 4.1

- varOne / varTwo = 2.333
- varOne // varTwo = 2 – drop the remainder
- varOne / varThree = 3.5
- varOne // varThree = 3 – drop the remainder
- varThree // varOne = 0 – drop the remainder

Exercise 5.1

a) 1 – the remainder is 1
b) 2 – the remainder is 2
c) 1 – the remainder is 1
d) 2 – the remainder is 2
e) 0 – the remainder is 0 as 2 goes perfectly into 6

Exercise 6.2

a) 5 – the value to the right is larger, so use the value to the left
b) 0 – 9 goes into 9 once with no remainder

Programming exercise 5.1

- varOne % varTwo = 1 – the remainder is 1
- varOne % varThree = 1 – the remainder is 1
- varThree % varOne = 2 – varThree is smaller than varOne