Teaching guide: Random numbers

This resource will help with understanding random number generation in a programming language. It supports Section 3.2.9 of our current GCSE Computer Science specification (8520). The guide is designed to address the following learning aims:

- Analyse situations where random number generation could be used in programs.
- Apply different forms of random number generation within programs.

Programs that run on computer systems are deterministic – with exactly the same inputs they should produce exactly the same outputs. If they did not do this then it would be impossible to write a correct program, so this is no bad thing. However, it does limit the ability to create random events in programs which are useful in a wide range of circumstances. For example:

- shuffling a playlist
- modelling flipping a coin
- making computer games less predictable
- picking a name out of a group of students.

Randomness is easy to produce in the real world – spinning a wheel, rolling a dice and so on are millennia-old techniques but producing the same randomness in a computer program is actually rather tricky. In actual fact computers do not produce random numbers at all – they use complex mathematical techniques to produce a series of numbers that may appear random but are really only an approximation to randomness (called pseudo-random numbers).

However, it is very unlikely that this will have a noticeable impact on any of the programs your students will write for this course. Fortunately, programming languages have random number generation subroutines that simplify the process.

They fall into three broad categories:

1. generating a random integer between a lower and upper value.
2. generating a random real between a lower and upper value, commonly set as 0 and 1.
3. randomly choosing an element from a data structure.

Some languages support all three of these (and more) whereas other languages may just support the first one or two. That said, only the first one or the second one are required to create the other two because:

- a random real number can be produced from dividing one random integer by another
- a random integer can be produced by rounding (and possibly multiplying by a constant factor) a random real
- a random element can be chosen from a data structure by using a random integer as an index where the lower value is the first index of the structure and the upper value is the final index.

The following example shows how a coin toss could be modelled using random integer generation:

```
random_number ← RANDOM_INT(0, 1)
IF random_number = 0 THEN
   result ← 'heads'
ELSE # only other value is 1
   result ← 'tails'
ENDIF
```

This example shows how a student's name could be picked at random from an array of names again using random integer generation:

```
names ← ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve']
random_number ← RANDOM_INT(0, LEN(names)-1)
random_name ← names[random_number]
```

If your language only supports random real generation between 0 and an upper limit of 1 then this example could be rewritten as:

```
names ← ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve']
random_real_number ← RANDOM_REAL(0, 1)
# multiply real by the number of elements in the array
random_real_number ← random_real_number * LEN(names)
# Round down the real number to its whole number part to
# get an integer between 0 and 4 inclusive
random_int_number ← ROUND_TO_INT(random_real_number)
random_name ← names[random_int_number]
```

Many languages will support randomly choosing an element from a data structure directly:

```
names ← ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve']
random_name ← RANDOM_CHOICE(names)
```

Although the specification requires students to be able to use their programming language to generate random numbers, they do not need to understand the pseudo-randomness of these numbers and how they are actually produced.