**Times tables practice**

Worked example . Countdown

The program below displays a sequence of numbers, starting from 10 and counting down to 1.

|  |  |
| --- | --- |
| 1  2  3  4  5 | count = 10  while count >= 1:  print(count)  count = count-1  print("Lift off") |

The count variable keeps track of the current number. It is assigned an initial value of 10 (line 1) and decreased by 1 at the end of each iteration (line 4).

The condition in the while statement checks the value of count (line 2) to make sure that the iteration will continue as long as count is at least 1.

Worked example . Ten sixes

The program below simulates an experiment in which a dice is rolled repeatedly, until the number six has been rolled ten times.

Two counter variables are used: rolls keeps track of the total number of dice rolls and sixes keeps track of the number of sixes rolled. The former is increased in every iteration, whereas the latter is only increased when a six is rolled.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | from random import randint  rolls = 0  sixes = 0  while sixes < 10:  dice = randint(1,6)  print(dice)  if dice == 6:  sixes = sixes + 1  rolls = rolls + 1  print("Ten sixes in", rolls, "dice rolls") |

**Times tables practice**

In Key Stage 2, pupils learn their times tables through constant practice. You will create a program that will help with their practice by producing random times tables questions and providing immediate feedback.

You will start with a version that poses a single question and then extend it to ask multiple questions.

Task 1 . A practice question

Open the [Python program below](https://ncce.io/py-times-50) (ncce.io/py-times-50) in your development environment. It generates a single random times tables question and checks the user’s answer to provide appropriate feedback.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from random import randint   |  | | --- | | a = randint(2,12)  b = randint(2,12)  print(a, "times", b, "=")  answer = int(input())  product = a \* b  if answer == product:  print("That is correct")  else:  print("I am sorry")  print(a, "times", b, "is", product) | |

**Step 1**

In order to generate **multiple questions**, insert all of the statements in the rectangle (lines 2 to 11) into a while statement, so that they are repeated.

Use True as the condition in the while statement. This means ‘repeat forever’.

|  |  |  |
| --- | --- | --- |
|  | while True:   |  | | --- | | code for a single question | |

**Step 2**

**Run** your program. It will never stop asking questions, so you will need to **terminate** it (look for a ‘Stop’ button or try pressing Ctrl+C).

**Syntax checklist**

If you encounter an **error message**, read it and try to fix the problem. Use the list below to check for common errors (and tick ✓ if you find yours).

|  |  |
| --- | --- |
|  | misspelt while or True |
|  | forgot the colon : after the condition in while |
|  | forgot to **indent** the statements in the while block |

**Step 3**

Introduce a variable called questions to keep track of the number of questions that have been posed to the user.

There are **two modifications** that you will need to make to your program:

* Assign an initial value to questions.
* Increase the value of questions by 1 in each iteration.

To make sure that questions is initialised and modified properly, use print to display the value of questions, anywhere within the while block.

|  |  |
| --- | --- |
|  | print("Question", questions) |

**Tip**

The value of questions must be increased in **every** iteration, so the corresponding statement must be **inside** the while block. Be careful with **indentation**.

**Step 4**

Modify your program so that it asks exactly three questions.

There is only **one modification** that you will need to make to your program:

* Replace the True condition with a condition that checks the value of questions. The iteration should only continue if the number of questions posed does not exceed three.

Explorer task . Measure performance

Introduce a variable called correct to keep track of the number of questions that the user has answered correctly.

There are **two modifications** that you will need to make to your program:

* Assign an initial value to correct.
* Increase the value of correct by 1 every time the user answers a question correctly.

At the end of the game, display the number of correct answers.

|  |  |
| --- | --- |
| **Example** |  |
| **Note:** The numbers here are examples and they will be different every time the program is executed. | |
| At the end of the game, the program displays a message with the number of correct answers. | You answered 2 out of 3 correctly |

Explorer task . Measure performance

Modify the program so that it keeps asking questions until the user has answered three of them correctly.

There is **one modification** that you will need to make to your program:

* Check the value of the correct variable in the condition of the while statement.

|  |  |
| --- | --- |
| **Example** |  |
| **Note:** The numbers here are examples and they will be different every time the program is executed. | |
| At the end of the game, the program displays a message with the number of correct answers. | You answered 3 out of 5 correctly |

Explorer task . The user sets the limits

The three-question limit is arbitrary. **Extend** the program to ask the user, before the game begins, what the total number of questions or the total number of correct answers should be.

Resources are updated regularly — the latest version is available at: [ncce.io/tcc](http://ncce.io/tcc).

This resource is licensed under the Open Government Licence, version 3. For more information on this licence, see [ncce.io/ogl](http://ncce.io/ogl).