National Centre for Computing Education

Raspberry Pi

Count

Worked example

What will this program display when it is executed?

```
1 count = 3
2 print(count)
3 count = count-1
4 print(count)
```

When executing a program, it is useful to use a **trace table**, to keep track of the current line that is being executed, the values of variables, and the output produced.

line	`count`	output
1	3	
2		3
3	2	
4		2

The answer to the question is in the rightmost column of the table: **the program will display 3 and 2.**

Task 1

This program **extends** the one from the **worked example** on the previous page.

```
1  count = 3
2  print(count)
3  count = count-1
4  print(count)
5  count = count-1
6  print(count)
7  count = count-1
```

When executing a program, it is useful to use a **trace table**, to keep track of the current line that is being executed, the values of variables, and the output produced.

The first rows of the trace table have been filled in (from the worked example on the previous page). **Complete the rest of the rows**.

line	`count`	output
1	3	
2		3
3	2	
4		2
5		
6		
7		

What will this program display when it is executed? The answer to the question is in the rightmost column of the table.

Task 2

This program seems to be **repeating** the same actions over and over again.

```
1  count = 3
2  print(count)
3  count = count-1
4  print(count)
5  count = count-1
6  print(count)
7  count = count-1
```

Step 1

In your programming environment, **type** the incomplete program below, which will use `while` to repeat the block of actions.

Note: You won't be able to run the program successfully until you fill in the missing condition in the next step.

```
1  count = 3
2  while              :
3      print(count)
4      count = count-1
```

Step 2

The value of `count` is initialised to **3** and decreased by one in every iteration.

Fill in the missing condition in the `while` loop using one of the options below, so that the last value printed by the program is **1**.

A. `while count > 1 :`

B. `while count >= 1 :`

C. `while count < 1 :`

D. `while count == 1 :`

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`
- [ ] forgot the colon `:` after the condition in `while`
- [ ] forgot to **indent** the statements in the `while` block

Task 3 Countdown

Step 1

Modify a single line in your current program so that the countdown starts from `10` instead of starting from `3`.

Step 2

Insert a single line in your current program so that the message `Lift off!` is displayed when the countdown reaches zero.

Tip

This action needs to be executed **after** the iteration, so it should **not** be part of the `while` block. Be careful with **indentation**.

Task 4 Skip counting upwards

Modify your current program so that it starts from `1` and skip counts over every 3 numbers, until it exceeds `19` (i.e. the program should print `1`, `4`, `7`, `10`, `13`, `16`, and `19`).

Tip

There are **three statements** that you will need to modify in your program:

- [ ] The assignment that determines where the counting starts
- [ ] The assignment that determines how `count` is modified in each iteration

- ☐ The condition that determines whether or not the iteration should continue

Tip

If your changes are incorrect, your program may keep displaying values forever! In that case, **terminate** your program (look for a 'Stop' button or try pressing Ctrl+C).

Explorer task The element of time

The `time` module includes a function called `sleep` that allows your program to wait for a given number of seconds. Import `sleep` at the start of your program:

```
from time import sleep
```

Use `sleep` in your counting program, so that there is a one-second delay between each of the numbers being printed.

```
  sleep(1)
```

National Centre for Computing Education

Raspberry Pi

Times tables practice

Worked example Countdown

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

```
1 count = 10
2 while count >= 1:
3   print(count)
4   count = count-1
5 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 from random import randint
2 rolls = 0
3 sixes = 0
4 while sixes < 10:
5   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 (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.

```
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.