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.

```
count = 3
print(count)
count = count-1
print(count)
count = count-1
print(count)
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.

```
count = 3
while              :
    print(count)
    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)
```