Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students’ responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students’ scripts: alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Assessment Writer.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students’ reactions to a particular paper. Assumptions about future mark schemes on the basis of one year’s document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

Further copies of this Mark Scheme are available from [http://www.aqa.org.uk/](http://www.aqa.org.uk/)
Level of response marking instructions

Level of response mark schemes are broken down into a number of levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are a range of marks in each level. The descriptor for the level represents a typical mid-mark performance in that level.

Before applying the mark scheme to a student’s answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student’s answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level. ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student’s answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner’s mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.
Examiners are required to assign each of the candidates’ responses to the most appropriate level according to **its overall quality**, then allocate a single mark within the level. When deciding upon a mark in a level examiners should bear in mind the relative weightings of the assessment objectives. This will be exemplified and reinforced as part of examiner training and standardisation.
### Marking guidance

<table>
<thead>
<tr>
<th>Qu</th>
<th>Part</th>
<th>Marking guidance</th>
<th>Total marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td>Mark is for AO2 (apply)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: B;</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>All marks AO2 (analyse)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nathan was not killed with poison (rule a); therefore Peter was not in the kitchen (rule c); therefore Martin was not in the dining room (rule e); therefore Suzanne was in the dining room (rule b); therefore Steve murdered Nathan (rule d).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mark as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Any correct point from the list above;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Any two further correct points from the list above;</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>1</td>
<td>Mark is for AO1 (understanding)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Original state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Table completed as above I. order of rows</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>All marks AO2 (analyse)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0</td>
<td>1)*((00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mark as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: (0</td>
<td>1)* at start;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: (00)</td>
<td>(11);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: (0</td>
<td>1)* at end;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alternative answer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0</td>
<td>1)*((11(0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mark as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: (0</td>
<td>1)* at start;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: (11(0</td>
<td>1)*);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark:</td>
<td>(00(0</td>
</tr>
<tr>
<td>Maximum 2 marks: If final answer not correct.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A any regular expression that correctly defines the language.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
02  3  Mark is for AO2 (apply)

<table>
<thead>
<tr>
<th>Rule number (given in Figure 2)</th>
<th>Could be defined using a regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>Y</td>
</tr>
</tbody>
</table>

1 mark: All values in the table have been completed correctly.

02  4  1 mark for AO2 (analyse) and 1 mark for AO3 (design)

1 mark for AO2 (analyse): There is no non-recursive / base case;
1 mark for AO3 (design): `<word> ::= <char><word>|<char>;;`

03  1  Mark is for AO1 (understanding)

It contains a cycle / cycles;

03  2  All marks AO2 (apply)

<table>
<thead>
<tr>
<th>Vertex (in Figure 3)</th>
<th>Adjacent vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>1, 3, 4</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Mark as follows:
1 mark: Three correct rows;
1 mark: All rows correct;
1 Order of items within each list/row.

03  3  All marks AO1 (understanding)

Adjacency list appropriate when there are few edges between vertices // when graph/matrix is sparse;
when edges rarely changed;
when presence/absence of specific edges does not need to be
<table>
<thead>
<tr>
<th>tested (frequently);</th>
<th>Max 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Alternative words which describe edge, eg connection, line</td>
<td></td>
</tr>
</tbody>
</table>
Mark SCHEME – A-LEVEL COMPUTER SCIENCE PAPER 1 – 7517/1A/1B/1C/1D/1E – SPECIMEN

### All marks AO2 (apply)

<table>
<thead>
<tr>
<th>NoOfCats</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**Mark as follows:**
1 mark: A is set the sequence indicated in the table;
1 mark: B is set the sequence indicated in the table;
1 mark: C is set the sequence indicated in the table;
1 mark: NoOfCats is set to 5, Cat[1] is set to 1;
1 mark: Cat[2] is set to 2 and Cat[3] is set to 3;
1 mark: Cat[4] is set to 1 and Cat[5] is set to 1;

**Info for examiner:** Ignore the empty cells in the sequences - values do not need to be set in the rows indicated in the table.

### Mark is for AO2 (analyse)

To work out which cats will travel together to the show //
To plan which cats will be in the van on which journey to the cat show //
To colour the vertices of a graph //
To create a decomposition of a graph;

Max 1

10 of 44
<table>
<thead>
<tr>
<th>03</th>
<th>6</th>
<th>All marks AO1 (knowledge)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>1 mark (1 from):</strong> The problem can be solved // algorithm exists for problem; But it cannot be solved in polynomial time // but not quickly enough to be useful; <strong>Max 2</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>1 mark:</strong> It takes an unreasonable amount of time; to solve; <strong>A Too long time but R Long time</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>03</th>
<th>7</th>
<th>All marks AO1 (understanding)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>1 mark:</strong> Use of heuristic; algorithm that makes a guess based on experience; That provides a close-to-optimal solution/approximation; that only works in some cases; <strong>A non-optimal</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example of heuristic method eg hill-climbing/stochastic/local improvement/greedy algorithms/simulated annealing/trial and error/any reasonable example; <strong>1 mark:</strong> Relax some of the constraints on the solution; <strong>A Solve simpler version of problem</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>1</th>
<th>Mark is for AO1 (understanding)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>False;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>2</th>
<th>Mark is for AO1 (understanding)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>THEN Failed ← True;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>3</th>
<th>All marks AO1 (understanding)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>L ← M - 1;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Mark as follows:</strong> <strong>1 mark:</strong> L; <strong>1 mark:</strong> M ← L;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Maximum 1 mark:</strong> If not correct</td>
<td></td>
</tr>
</tbody>
</table>

<p>| 04 | 4 | Mark is for AO1 (understanding) | 1 |</p>
<table>
<thead>
<tr>
<th>04</th>
<th>5</th>
<th>Mark is for AO1 (knowledge)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$O(k^n)$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A \ k^n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>6</th>
<th>Mark is for AO1 (knowledge)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$O(\log n)$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A \ \log n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>7</th>
<th>Mark is for AO1 (knowledge)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$O(n)$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A \ n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>8</th>
<th>All marks AO1 (understanding)</th>
</tr>
</thead>
</table>
|    |   | 1 mark: As the size of the list increases the time taken to search for an item increases; at the same rate; //
|    |   | 1 mark: A linear search looks at each item in the list in turn (until it reaches the end of the list or the item being searched for is found); so if there are n items in the list the worst case would be n comparisons; |

<table>
<thead>
<tr>
<th>05</th>
<th>1</th>
<th>All marks AO2 (apply)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$3 \times 4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>05</th>
<th>2</th>
<th>All marks AO2 (apply)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$(12 + 8) \times 4;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>05</th>
<th>3</th>
<th>Mark for AO1 (understanding)</th>
</tr>
</thead>
</table>
|    |   | 1 mark: Simpler/easier for a machine/computer to evaluate // simpler/easier to code algorithm
|    |   | R Simpler/easier to understand
|    |   | Do not need brackets (to show correct order of evaluation/calculation);
|    |   | Operators appear in the order required for computation; |
No need for order of precedence of operators;  
No need to backtrack when evaluating;  
A RPN expressions cannot be ambiguous as Benefit Of Doubt (BOD)
### 4 marks for AO3 (design) and 8 marks for AO3 (programming)

#### Mark Scheme

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Mark Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets all of the requirements of <strong>Task 1</strong> and some of the requirements of <strong>Task 2</strong>. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements of both tasks must be met.</td>
<td>10-12</td>
</tr>
<tr>
<td>3</td>
<td>There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the decimal value and includes a loop, which might be a definite or indefinite loop. An attempt has been made to do the integer division, output the remainder within the loop and use the result of the division for the next iteration, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken. To award 9 marks, all of the requirements of <strong>Task 1</strong> must have been met.</td>
<td>7-9</td>
</tr>
<tr>
<td>2</td>
<td>A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality for either task, it can be seen that the response contains some of the statements that would be needed in a working solution to <strong>Task 1</strong>. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.</td>
<td>4-6</td>
</tr>
<tr>
<td>1</td>
<td>A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.</td>
<td>1-3</td>
</tr>
</tbody>
</table>
Guidance

Task 1:

Evidence of AO3 (design) - 3 points:

Evidence of design to look for in responses:

- Identifying that an indefinite loop must be used (as the length of the input is variable)
- Identifying the correct Boolean condition to terminate the loop
- Correct identification of which commands belong inside and outside the loop

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 6 points:

Evidence of programming to look for in responses:

- Prompt displayed
- Value input by user and stored into a variable with a suitable name
- Loop structure coded
- Remainder of integer division calculated
- Remainder of integer division output to screen
- Result of integer division calculated and assigned to variable so that it will be used in the division operation for the next iteration

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Task 2:

Evidence of AO3 (design) - 1 point:

Evidence of design to look for in responses:

- A sensible method adopted for reversing the output eg appending to a string or storing into an array
Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 2 points:

Evidence of programming to look for in responses:

- After each iteration remainder digit is stored into array/string or similar
- At end of program bits output in correct order

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Example Solution VB.Net

Task 1:

Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer

Console.WriteLine("Please enter decimal number to convert")
DecimalNumber = Console.ReadLine

Do
    ResultOfDivision = DecimalNumber \ 2
    BinaryDigit = DecimalNumber Mod 2
    Console.Write(BinaryDigit)
    DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0

Task 2:

Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer
Dim BinaryString As String

Console.WriteLine("Please enter decimal number to convert")
DecimalNumber = Console.ReadLine

BinaryString = ""

Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim DecimalNumber As Integer
        Dim ResultOfDivision As Integer
        Dim BinaryDigit As Integer
        Dim BinaryString As String

        Console.WriteLine("Please enter decimal number to convert")
        DecimalNumber = Console.ReadLine
        BinaryString = ""

        Do
            ResultOfDivision = DecimalNumber \ 2
            BinaryDigit = DecimalNumber Mod 2
            Console.Write(BinaryDigit)
            DecimalNumber = ResultOfDivision
        Loop Until ResultOfDivision = 0
    End Sub
End Class
Do
    ResultOfDivision = DecimalNumber \ 2
    BinaryDigit = DecimalNumber Mod 2
    
    BinaryString = BinaryDigit.ToString() +
    BinaryString
    DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0

Console.WriteLine(BinaryString)

**Example Solution Pascal**

**Task 1:**

Var    
DecimalNumber, ResultOfDivision, BinaryDigit : Integer;

Begin    
    Writeln('Please enter decimal number to convert');    
    Readln(DecimalNumber);
    Repeat
        ResultofDivision := DecimalNumber Div 2;
        BinaryDigit := DecimalNumber Mod 2;
        Write(BinaryDigit);
        DecimalNumber := ResultOfDivision;
    Until ResultOfDivison = 0;
    Readln;
End.

**Task 2:**

Var    
DecimalNumber, ResultOfDivision, BinaryDigit : Integer;
    
    BinaryString : String;

Begin    
    Writeln('Please enter decimal number to convert');    
    Readln(DecimalNumber);
    BinaryString := '';    
    Repeat
        ResultofDivision := DecimalNumber Div 2;
        BinaryDigit := DecimalNumber Mod 2;
        BinaryString := IntToStr(BinaryDigit) +
        BinaryString;    
        DecimalNumber := ResultOfDivision;
    Until ResultOfDivision = 0;
    Writeln(BinaryString);
    Readln;
Example Solution Python 3.x

Task 1:

```python
print("Input a decimal number to convert to binary: ", end = '')
decimal = int(input())

while decimal != 0:
    print(decimal % 2, end = '')
decimal //= 2

Task 2:

print("Input a decimal number to convert to binary: ", end = '')
decimal = int(input())

result = ""
while decimal != 0:
    result = str(decimal % 2) + result
decimal //= 2

print(result)
```

Alternative answers using break:

Task 1:

```python
print("Input a decimal number to convert to binary: ", end = '')
decimal = int(input())

while True:
    print(decimal % 2, end = '')
decimal //= 2
    if decimal == 0:
        break

Task 2:

print("Input a decimal number to convert to binary: ", end = '')
decimal = int(input())

result = ""
while True:
    result = str(decimal % 2) + result
decimal //= 2
    if decimal == 0:
        break

print(result)
```
**Example Solution Python 2.x**

**Task 1:**

print "Input a decimal number to convert to binary:\n",
decimal = int(input())

while decimal != 0:
    print decimal % 2,
    decimal /= 2

**Task 2:**

print "Input a decimal number to convert to binary:\n",
decimal = int(input())

result = ""
while decimal != 0:
    result = str(decimal % 2) + result
decimal /= 2

print(result)

**Alternative answers using break:**

**Task 1:**

print "Input a decimal number to convert to binary:\n",
decimal = int(input())

while True:
    print decimal % 2,
    decimal /= 2
    if decimal == 0:
        break

**Task 2:**

print "Input a decimal number to convert to binary:\n",
decimal = int(input())

result = ""
while True:
    result = str(decimal % 2) + result
decimal /= 2
    if decimal == 0:
        break

print result
### A. Use of // (floor division) or / (division) in Python 2.x. Python 3.x must use //

#### Example Solution C#

**Task 1:**

```csharp
int DecimalNumber;
int ResultOfDivision;
int BinaryDigit;

Console.WriteLine("Please enter decimal number to convert");
DecimalNumber = int.Parse(Console.ReadLine());

while (ResultOfDivision != 0) {
    ResultOfDivision = DecimalNumber / 2;
    BinaryDigit = DecimalNumber % 2;
    Console.Write(BinaryDigit);
    DecimalNumber = ResultOfDivision;
}

**Task 2:**

```csharp
int DecimalNumber;
int ResultOfDivision;
int BinaryDigit;

string BinaryString = "";

Console.WriteLine("Please enter decimal number to convert");
DecimalNumber = int.Parse(Console.ReadLine());

while (ResultOfDivision != 0) {
    ResultOfDivision = DecimalNumber / 2;
    BinaryDigit = DecimalNumber % 2;
    BinaryString = Convert.ToString(BinaryDigit) + BinaryString;
    DecimalNumber = ResultOfDivision;
}

Console.WriteLine(BinaryString);
```

#### Example Solution Java

**Task 1:**

```java
int decimalNumber;
int resultOfDivision;
int binaryDigit;
```
decimalNumber = console.readInteger("Please enter decimal number to convert");
   do {
      resultOfDivision = decimalNumber / 2;
      binaryDigit = decimalNumber % 2;
      console.print(binaryDigit);
      decimalNumber = resultOfDivision;
   } while (resultOfDivision != 0);

Task 2:

int decimalNumber;
int resultOfDivision;
int binaryDigit;
String binaryString;

decimalNumber = console.readInteger("Please enter decimal number to convert");
binaryString = "";
   do {
      resultOfDivision = decimalNumber / 2;
      binaryDigit = decimalNumber % 2;
      binaryString = Integer.toString(binaryDigit) + binaryString;
      decimalNumber = resultOfDivision;
   } while (resultOfDivision != 0);
calendar.println(binaryString);

<table>
<thead>
<tr>
<th>06</th>
<th>2</th>
<th>All marks AO3 (evaluate)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>SCREEN CAPTURE(S)</strong></strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info for examiner: Must match code from 06.1, including prompts on screen capture matching those in code. Code for 06.1 must be sensible.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 mark: Display of suitable prompt and user input of value 210;
1 mark: Display of correct bits in reverse (01001011) or forward (11010010) order;

A. Each bit value displayed on a separate line
A. Each bit value followed by a space
<table>
<thead>
<tr>
<th>07</th>
<th>1</th>
<th>All marks AO2 (analyse)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 mark: The arrow should be pointing towards the base class;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: There is no class called Monster // it should say Enemy, not</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monster;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>07</th>
<th>2</th>
<th>Mark is for AO2 (apply)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>VB.Net</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dim MyGame As New Game(False) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dim MyGame As New Game(True) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Player As New Character //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Cavern As New Grid(NSDistance, WEDistance) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Monster As New Enemy //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Flask As New Item //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Trap1 As New Trap //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Private Trap2 As New Trap;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pascal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MyGame := Game.Create(false); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MyGame := Game.Create(true); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Player := Character.Create; //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cavern := Grid.Create(NS,WE); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monster := Enemy.Create; //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flask := Item.Create; //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trap1 := Trap.Create; //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trap2 := Trap.Create;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MyGame = Game(False) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MyGame = Game(True) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Player = Character()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Cavern = Grid(NS, WE) //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Monster = Enemy()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Flask = Item()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Trap1 = Trap()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>self.Trap2 = Trap()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Position = CellReference()//</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>private Character Player = new Character(); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>private Grid Cavern = new Grid(); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>private Enemy Monster = new Enemy(); //</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>private Item Flask = new Item(); //</td>
<td></td>
</tr>
<tr>
<td>Mark</td>
<td>Q</td>
<td>Details</td>
<td>Total</td>
</tr>
<tr>
<td>------</td>
<td>---</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>07</td>
<td>3</td>
<td>Mark is for AO2 (apply)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VB.Net/Pascal/Python</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CavernState;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R If any additional code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R If spelt incorrectly</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I Case</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>4</td>
<td>Mark is for AO2 (apply)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trap // Character // Enemy;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A SleepyEnemy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R If any additional code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R If spelt incorrectly</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I Case</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>5</td>
<td>Mark is for AO2 (apply)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Choice // NoOfCellsEast // NoOfCellsSouth // Count // NS // WE // Count1 // Count2;</td>
<td></td>
</tr>
<tr>
<td>R If any additional code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R If spelt incorrectly</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I Case</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>6</td>
<td><strong>Mark is for AO2 (apply)</strong></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>----------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Game;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>R</strong> If any additional code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>R</strong> If spelt incorrectly</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Case</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>07</th>
<th>7</th>
<th><strong>Mark is for AO2 (analyse)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>So that a position of (0,0) is rejected // so that the item can't be in the player's starting position;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>07</th>
<th>8</th>
<th><strong>Marks are for AO1 (understanding)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Makes the program code easier to understand;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Makes it easier to update the program;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Makes it easier to change the size of the cavern (in the game);</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Max 2 points from the list above</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>07</th>
<th>9</th>
<th><strong>Marks are for AO2 (analyse)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 mark: Create a new object (Trap3) of class Trap;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Change the (3rd) If statement in the PlayGame subroutine by adding conditions to check if the player is in the same cell as Trap3 and that Trap3 has not been triggered already;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>08</th>
<th>1</th>
<th><strong>Marks are for AO3 (programming)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 mark: Selection structure with one correct condition;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Both conditions correct and correct logical operator(s);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 mark: Subroutine returns the correct True/False value under all conditions;</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>New conditions added to existing selection structure</strong></td>
</tr>
</tbody>
</table>

**VB.Net**

```vbnet
Public Function CheckValidMove(ByVal Direction As Char) As Boolean
    Dim ValidMove As Boolean
    ValidMove = True
    If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M") Then
        ValidMove = False
    End If
    Return ValidMove
End Function
```
Pascal
Function Game.CheckValidMove(Direction : char) : Boolean;
Var
  ValidMove : Boolean;
Begin
  ValidMove := True;
  If Not(Direction In ['N', 'S', 'W', 'E', 'M']) Then
    ValidMove := False;
  If (Direction = 'W') And
    (Player.GetPosition.NoOfCellsEast = 0) Then
    ValidMove := False;
  CheckValidMove := ValidMove;
End;

Python (2.x or 3.x)
def CheckValidMove(self, Direction):
    ValidMove = True
    if not(Direction in ['N', 'S', 'W', 'E', 'M']):
        ValidMove = False
    if Direction == 'W' and
       self.Player.GetPosition().NoOfCellsEast == 0:
        ValidMove = False
    return ValidMove

C#
public Boolean CheckValidMove(char Direction)
{
    Boolean ValidMove;
    ValidMove = true;
    if (!(Direction == 'N' || Direction == 'S' ||
          Direction == 'W' || Direction == 'E' || Direction == 'M'))
    {
        ValidMove = false;
    }
    if (Direction == 'W' &&
        Player.GetPosition().NoOfCellsEast == 0)
    {
        ValidMove = false;
    }
}
return ValidMove;
}

Java
public boolean checkValidMove(char direction) {
    boolean validMove;
    validMove = true;
    if (!(direction == 'N' || direction == 'S' ||
         direction == 'W' || direction == 'E' || direction ==
         'M')) {
        validMove = false;
    }
    if (direction == 'W' &&
        player.getPosition().noOfCellsEast == 0) {
        validMove = false;
    }
    return validMove;
}
Marks are for AO3 (programming)

1 mark: Selection structure with correct condition added in correct place in the code;
1 mark: Correct error message displayed which will be displayed when move is invalid, and only when the move is invalid;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

```vbnet
... 
ValidMove = CheckValidMove(MoveDirection) 
If Not ValidMove Then 
    Console.WriteLine("That is not a valid move, please try again") 
End If 
Loop Until ValidMove 
... 
```

Pascal

```pascal
... 
ValidMove := CheckValidMove(MoveDirection); 
If Not ValidMove Then 
    Writeln('That is not a valid move, please try again'); 
Until ValidMove; 
```

Python 3.x

```python
ValidMove = False 
while not ValidMove: 
    self.DisplayMoveOptions() 
    MoveDirection = self.GetMove() 
    ValidMove = self.CheckValidMove(MoveDirection) 
    if not ValidMove: 
        print("That is not a valid move, please try again") 
```

R. If indentation not correct (if not ValidMove: must be at same indent as rest of code inside while loop)

Python 2.x

```python
ValidMove = False 
while not ValidMove: 
    self.DisplayMoveOptions() 
    MoveDirection = self.GetMove() 
```
ValidMove = self.CheckValidMove(MoveDirection)
    if not ValidMove:
        print "That is not a valid move, please try again"

R. If indentation not correct (if not ValidMove: must be at same indent as rest of code inside while loop)

C#
...
    MoveDirection = GetMove();
    ValidMove = CheckValidMove(MoveDirection);
    if (!ValidMove)
        {
            Console.WriteLine("That is not a valid move, please try again");
        }
    } while (!ValidMove);
...

Java
...
    moveDirection = getMove();
    validMove = checkValidMove(moveDirection);
    if (!validMove) {
        console.println("That is not a valid move, please try again");
    }
    } while (!validMove);
...

Mark is for AO3 (evaluate)

****SCREEN CAPTURE(S)****
Must match code from 39 and 40, including prompts on screen capture matching those in code. Code for 39 and 40 must be sensible

Screen capture(s) showing the error message being displayed after the player tried to move to the west from a cell at the western end of the cavern;

A Alternative output messages if match code for 08.2
<table>
<thead>
<tr>
<th>Marks are for AO3 (programming)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mark: SleepyEnemy class created;</td>
</tr>
<tr>
<td>1 mark: Inheritance from Enemy class;</td>
</tr>
<tr>
<td>1 mark: MovesTillSleep property declared;</td>
</tr>
<tr>
<td>1 mark: Subroutine MakeMove that overrides the one in the base class;</td>
</tr>
<tr>
<td>1 mark: MovesTillSleep decremented in the MakeMove subroutine;</td>
</tr>
<tr>
<td>1 mark: Selection structure in MakeMove that calls ChangeSleepStatus if the value of MovesTillSleep is 0; A Changing Awake property instead of call to ChangeSleepStatus</td>
</tr>
<tr>
<td>1 mark: Subroutine ChangeSleepStatus that overrides the one in the base class;</td>
</tr>
<tr>
<td>1 mark: Value of MovesTillSleep set to 4 in the ChangeSleepStatus subroutine;</td>
</tr>
</tbody>
</table>

I Case of identifiers
A Minor typos in identifiers

**VB.Net**

Class SleepyEnemy
    Inherits Enemy
    Private MovesTillSleep As Integer

    Public Overrides Sub MakeMove(ByVal PlayerPosition As CellReference)
        MyBase.MakeMove(PlayerPosition)
        MovesTillSleep = MovesTillSleep - 1
        If MovesTillSleep = 0 Then
            ChangeSleepStatus()
        End If
    End Sub

    Public Overrides Sub ChangeSleepStatus()
        MyBase.ChangeSleepStatus()
        MovesTillSleep = 4
    End Sub
End Class

**Pascal**

SleepyEnemy = Class(Enemy)
    Strict Private
    MovesTillSleep : Integer;
    Public
    Procedure ChangeSleepStatus; Override;
    Procedure MakeMove(PlayerPosition: CellReference); Override;
End;

Procedure SleepyEnemy.ChangeSleepStatus;
Begin
  Inherited;
  MovesTillSleep := 4;
End;

Procedure SleepyEnemy.MakeMove(PlayerPosition: CellReference);
Begin
  Inherited;
  MovesTillSleep := MovesTillSleep - 1;
  If MovesTillSleep = 0 Then
    ChangeSleepStatus;
End;

Python 3.x/2.x

class SleepyEnemy(Enemy):
  def __init__(self):
    Enemy.__init__(self)
    self.MovesTillSleep = 4

  def ChangeSleepStatus(self):
    Enemy.ChangeSleepStatus(self)
    self.MovesTillSleep = 4

  def MakeMove(self, PlayerPosition):
    Enemy.MakeMove(self, PlayerPosition)
    self.MovesTillSleep -= 1
    if self.MovesTillSleep == 0:
      self.ChangeSleepStatus()

A No explicit initialisation of new instance, i.e., no
SleepyEnemy.__init__

C#

class SleepyEnemy : Enemy
{
  private int MovesTillSleep;

  public override void MakeMove(CellReference PlayerPosition)
  {
    base.MakeMove(PlayerPosition);
    MovesTillSleep = MovesTillSleep - 1;
  }
if (MovesTillSleep == 0)
{
    ChangeSleepStatus();
}
}

public override void ChangeSleepStatus()
{
    base.ChangeSleepStatus();
    MovesTillSleep = 4;
}

Java

class SleepyEnemy extends Enemy {
    private int movesTillSleep;

    public void makeMove(CellReference playerPosition)
    {
        super.makeMove(playerPosition);
        movesTillSleep = movesTillSleep - 1;
        if (movesTillSleep == 0) {
            changeSleepStatus();
        }
    }

    public void changeSleepStatus() {
        super.changeSleepStatus();
        movesTillSleep = 4;
    }
}
**Marks are for AO3 (evaluate)**

***SCREEN CAPTURE(S)***

**Info for examiner:** Must match code from 09.1, including prompts on screen capture matching those in code. Code for 09.1 must be sensible.

1 mark: Screen capture(s) showing the player moving east and then east again at the start of the training game. The monster then wakes up and moves two cells nearer to the player. The player then moves south;

1 mark: The monster moves two cells nearer to the player and then disappears from the cavern display;

---

**Mark is for AO3 (programming)**

Appropriate option added to menu;

**VB.Net**

Public Sub DisplayMoveOptions()
    Console.WriteLine()
    Console.WriteLine("Enter N to move NORTH")
    Console.WriteLine("Enter S to move SOUTH")
    Console.WriteLine("Enter E to move EAST")
    Console.WriteLine("Enter W to move WEST")
    Console.WriteLine("Enter A to shoot an arrow")
    Console.WriteLine("Enter M to return to the Main Menu")
    Console.WriteLine()
End Sub

**Pascal**

Procedure Game.DisplayMoveOptions;
Begin
    Writeln;
    Writeln('Enter N to move NORTH');
    Writeln('Enter E to move EAST');
    Writeln('Enter S to move SOUTH');
    Writeln('Enter W to move WEST');
    Writeln('Enter A to shoot an Arrow');
    Writeln('Enter M to return to the Main Menu');
    Writeln;
End;

**Python 3.x**

def DisplayMoveOptions(self):
    print()
    print("Enter N to move NORTH")

---
print("Enter S to move SOUTH")
print("Enter E to move EAST")
print("Enter W to move WEST")
print("Enter A to shoot an arrow")
print("Enter M to return to the Main Menu")
print()

Python 2.x
As for Python 3, but `print()` should be just `print`, and other
parentheses may be missing

C#
public void DisplayMoveOptions()
{
    Console.WriteLine();
    Console.WriteLine("Enter N to move NORTH");
    Console.WriteLine("Enter S to move SOUTH");
    Console.WriteLine("Enter E to move EAST");
    Console.WriteLine("Enter W to move WEST");
    Console.WriteLine("Enter A to shoot an arrow");
    Console.WriteLine("Enter M to return to the Main Menu");
    Console.WriteLine();
}

Java
public void displayMoveOptions() {
    console.println();
    console.println("Enter N to move NORTH");
    console.println("Enter S to move SOUTH");
    console.println("Enter E to move EAST");
    console.println("Enter W to move WEST");
    console.println("Enter A to shoot an arrow");
    console.println("Enter M to return to the Main Menu");
    console.println();
}

Marks are for AO3 (programming)

1 mark: Direction of A is allowed;
1 mark: Direction of A allowed only if player has got an arrow;

Maximum 1 mark: If any other invalid moves would be allowed or any valid moves not allowed
Public Function CheckValidMove(ByVal Direction As Char) As Boolean
    Dim ValidMove As Boolean
    ValidMove = True
    If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M" Or Direction = "A") Then
        ValidMove = False
    End If
    If Direction = "A" And Not Player.GetHasArrow Then
        ValidMove = False
    End If
    Return ValidMove
End Function

Function Game.CheckValidMove(Direction : Char) : Boolean;
Var
    ValidMove : Boolean;
Begin
    ValidMove := True;
    If Not(Direction In ['N', 'S', 'W', 'E', 'A', 'M']) Then
        ValidMove := False;
    If (Direction = 'A') And (Player.GetHasArrow = False) Then
        ValidMove := False;
    CheckValidMove := ValidMove;
End;

def CheckValidMove(self, Direction):
    ValidMove = True
    if not(Direction in ['N', 'S', 'W', 'E', 'A', 'M']):
        ValidMove = False
    if Direction == 'A' and self.Player.GetHasArrow() == False:
        ValidMove = False
    return ValidMove

A return instead of assignment to ValidMove

Alternative
def CheckValidMove(self, Direction):
ValidMove = True
if not(Direction in ['N', 'S', 'W', 'E', 'A', 'M']):
    ValidMove = False
    if Direction == 'A':
        ValidMove = self.Player.GetHasArrow()
        return ValidMove

A return instead of assignment to ValidMove

C#
public Boolean CheckValidMove(char Direction) {
    Boolean ValidMove;
    ValidMove = true;
    if (!(Direction == 'N' || Direction == 'S' || Direction == 'W' || Direction == 'E' || Direction == 'M' || Direction == 'A')) {
        ValidMove = false;
    }
    if (Direction == 'A' && !Player.GetHasArrow()) {
        ValidMove = false;
    }
    return ValidMove;
}

Java
public boolean checkValidMove(char direction) {
    boolean validMove;
    validMove = true;
    if (! (direction == 'N' || direction == 'S' || direction == 'W' || direction == 'E' || direction == 'M' || direction == 'A')) {
        validMove = false;
    }
    if (direction == 'A' && !player.getHasArrow()) {
        validMove = false;
    }
    return validMove;
}

Marks are for AO3 (programming)
1 mark: Property HasArrow created;
1 mark: HasArrow set to True when an object is instantiated;
1 mark: Subroutine GetHasArrow created;
1 mark: GetHasArrow returns the value of HasArrow;
1 mark: Subroutine `GetArrowDirection` created;
1 mark: `GetArrowDirection` has an appropriate output message and then gets a value entered by the user;
1 mark: In `GetArrowDirection`, value keeps being obtained from user until it is one of N, S, W or E;
1 mark: `HasArrow` is set to False in `GetArrowDirection`;

I Additional output messages
I Case of identifiers
I Minor typos in identifiers

**VB.Net**

```vbnet
Class Character
    Inherits Item
    Private HasArrow As Boolean
    Public Sub MakeMove(ByVal Direction As Char)
        Select Case Direction
            Case "N"
                NoOfCellsSouth = NoOfCellsSouth - 1
            Case "S"
                NoOfCellsSouth = NoOfCellsSouth + 1
            Case "W"
                NoOfCellsEast = NoOfCellsEast - 1
            Case "E"
                NoOfCellsEast = NoOfCellsEast + 1
        End Select
    End Sub

    Public Sub New()
        HasArrow = True
    End Sub

    Public Function GetHasArrow() As Boolean
        Return HasArrow
    End Function

    Public Function GetArrowDirection() As Char
        Dim Direction As Char
        Do
            Console.Write("What direction (E, W, S, N) would you like to shoot in?")
            Direction = Console.ReadLine
        Loop Until Direction = "E" Or Direction = "W" Or Direction = "S" Or Direction = "N"
        HasArrow = False
        Return Direction
    End Function
End Class
```
Pascal

Character = Class(Item)
  Strict Private
    HasArrow: Boolean;
  Public
    Constructor Create;
    Procedure MakeMove(Direction: Char);
    Function GetHasArrow: Boolean;
    Function GetArrowDirection: Char;
  End;

Constructor Character.Create;
Begin
  HasArrow := True;
End;

Function Character.GetArrowDirection: Char;
Var
  Direction: Char;
Begin
  Repeat
    Writeln('What direction (E,W,S,N) would you like to shoot in?');
    Readln(Direction);
  Until Direction In ['E', 'W', 'S', 'N'];
  HasArrow := False;
  GetArrowDirection := Direction;
End;

Function Character.GetHasArrow: Boolean;
Begin
  GetHasArrow := HasArrow;
End;

Python 3.x/2.x

class Character(Item):
  def __init__(self):
    Item.__init__(self)
    self.HasArrow = True

  def MakeMove(self, Direction):
    if Direction == 'N':
      self.NoOfCellsSouth = self.NoOfCellsSouth - 1
    elif Direction == 'S':
      self.NoOfCellsSouth = self.NoOfCellsSouth + 1
    elif Direction == 'W':
      self.NoOfCellsEast = self.NoOfCellsEast - 1
    elif Direction == 'E':
      self.NoOfCellsEast = self.NoOfCellsEast + 1
```python
def GetHasArrow(self):
    return self.HasArrow

def GetArrowDirection(self):
    print()
    print("Enter N to shoot NORTH")
    print("Enter S to shoot SOUTH")
    print("Enter E to shoot EAST")
    print("Enter W to shoot WEST")
    print()

    while True:
        Shoot = input()
        if Shoot != "" and Shoot[0] in ['N', 'S', 'E', 'W']:
            self.HasArrow = False
            return Shoot[0]
        else:
            print("Not a valid direction. Please enter N, S, E or W")
```

```csharp
class Character : Item {
    private Boolean HasArrow;

    public void MakeMove(char Direction) {
        switch(direction) {
            case 'N' : NoOfCellsSouth = NoOfCellsSouth - 1;
                        break;
            case 'S' : NoOfCellsSouth = NoOfCellsSouth + 1;
                        break;
            case 'W' : NoOfCellEast = NoOfCellsEast - 1;
                        break;
            case 'E' : NoOfCellsEast = NoOfCellsEast + 1;
                        break;
        }
    }

    public Character() {
        HasArrow = true;
    }

    public Boolean getHasArrow() {
        return HasArrow;
    }
```
public char GetArrowDirection() {
    char Direction;
    do {
        Console.Write("What direction (E, W, S, N) would you like to shoot in?");
        Direction = char.Parse(Console.ReadLine());
    } while (!(Direction == 'E' || Direction == 'W' || Direction == 'S' || Direction == 'N'));
    HasArrow = false;
    return Direction;
}

Java

class Character extends Item {
    private boolean hasArrow;
    public void makeMove(char direction) {
        switch (direction) {
            case 'N': noOfCellsSouth = noOfCellsSouth - 1;
                break;
            case 'S': noOfCellsSouth = noOfCellsSouth + 1;
                break;
            case 'W': noOfCellEast = noOfCellsEast - 1;
                break;
            case 'E': noOfCellsEast = noOfCellsEast + 1;
                break;
        }
    }

    public Character() {
        hasArrow = true;
    }

    public boolean getHasArrow() {
        return hasArrow;
    }

    public char getArrowDirection() {
        char direction;
        do {
            console.print("What direction (E, W, S, N) would you like to shoot in?");
            direction = console.readChar();
        } while (!(direction == 'E' || direction == 'W' || direction == 'S' || direction == 'N'));
        hasArrow = false;
        return direction;
    }
}
Marks are for AO3 (programming)

1 mark: Check for A having been entered – added in a sensible place in the code;
1 mark: If A was entered there is a call to GetArrowDirection;
1 mark: Selection structure that checks if the arrow direction is N;
1 mark: Detects if the monster is in any of the cells directly north of the player's current position;
1 mark: If the monster has been hit by an arrow then the correct output message is displayed and the value of FlaskFound is set to True;
1 mark: The code for moving the player and updating the cavern display is inside an else structure (or equivalent, e.g., correctly indented in Python) so that this code is not executed if the player chooses to shoot an arrow;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

If MoveDirection <> "M" Then
    If MoveDirection = "A" Then
        MoveDirection = Player.GetArrowDirection
        Select MoveDirection
            Case "N"
                    Console.WriteLine("You have shot the monster and it cannot stop you finding the flask")
                    FlaskFound = True
                End If
        End Select
    Else
        Cavern.PlaceItem(Player.GetPosition, " ")
        Player.MakeMove(MoveDirection)
        Cavern.PlaceItem(Player.GetPosition, "*")
        Cavern.Display(Monster.GetAwake)
        FlaskFound = Player.CheckIfSameCell(Flask.GetPosition)
    End If
End If
If FlaskFound Then
    ...

Pascal
If MoveDirection <> 'M' Then
Begin
  If MoveDirection = 'A' Then
    Case Player.GetArrowDirection Of
    'N':
      If (Monster.GetPosition.NoOfCellsSouth <
          Player.GetPosition.NoOfCellsSouth) And
        (Monster.GetPosition.NoOfCellsEast =
          Player.GetPosition.NoOfCellsEast) Then
        Begin
          Writeln('You have shot the monster and it
          cannot stop you finding the flask');
          FlaskFound := True;
        End;
    End;
  Else
    Begin
      Cavern.PlaceItem(Player.GetPosition, ' ');
      Player.MakeMove(MoveDirection);
      Cavern.PlaceItem(Player.GetPosition, '*');
      Cavern.Display(Monster.GetAwake);
      FlaskFound :=
        Player.CheckIfSameCell(Flask.GetPosition);
    End;
  End;
End;
If FlaskFound Then
...

Python 3.x/2.x
if MoveDirection != 'M':
  if MoveDirection == 'A':
    ShootDirection = self.Player.GetArrowDirection()
    if ShootDirection == 'N':
      if (self.Player.GetPosition().NoOfCellsEast ==
          self.Monster.GetPosition().NoOfCellsEast) and
        (self.Player.GetPosition().NoOfCellsSouth >
        print("You have shot the monster and it
cannot stop you finding the flask")
        FlaskFound = True
    else:
      self.Cavern.PlaceItem(self.Player ...

C#
if (MoveDirection != 'M') {
  if (MoveDirection == 'A') {
```csharp
MoveDirection = Player.GetArrowDirection();
switch (MoveDirection) {
    case 'N': if 
        (Monster.GetPosition().NoOfCellsSouth < 
        Player.GetPosition().NoOfCellsSouth && 
        Monster.GetPosition().NoOfCellsEast == 
        Player.GetPosition().NoOfCellsEast) {
            Console.WriteLine("You have shot the monster 
and it cannot stop you finding the flask");
            FlaskFound = true;
            break;
        }
    } else {
        Cavern.PlaceItem(Player.GetPosition(), " ");
        Player.MakeMove(MoveDirection);
        Cavern.PlaceItem(player.GetPosition(), "*");
        Cavern.Display(Monster.GetAwake());
        FlaskFound = 
        Player.CheckIfSameCell(Flask.GetPosition());
    }
if (FlaskFound) {
    ...

Java
if (moveDirection != 'M') {
    if (moveDirection == 'A') {
        moveDirection = player.getArrowDirection();
        switch (moveDirection) {
            case 'N': if 
                (monster.getPosition().noOfCellsSouth < 
                player.getPosition().noOfCellsSouth && 
                monster.getPosition().noOfCellsEast == 
                player.getPosition().noOfCellsEast) {
                    console.println("You have shot 
the monster and it cannot stop you finding the 
flask");
                    flaskFound = true;
                    break;
                }
        } else {
            cavern.placeItem(player.getPosition(), " ");
            player.makeMove(moveDirection);
            cavern.placeItem(player.getPosition(), "*");
            cavern.display(monster.getAwake());
            flaskFound = 
            player.checkIfSameCell(flask.getPosition());
        }
    }
```
if (flaskFound) { ...  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Mark is for AO3 (evaluate)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td><strong><strong>SCREEN CAPTURE(S)</strong></strong></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Info for examiner:</em> Must match code from 10.1, 10.2, 10.3 and 10.4, including prompts on screen capture matching those in code. Code for 10.1, 10.2, 10.3 and 10.4 must be sensible.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Screen capture(s) showing the user shooting an arrow northwards at the start of the training game and the message about the monster being shot is displayed;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A Alternative output messages if match code for 10.4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Mark is for AO3 (evaluate)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td><strong><strong>SCREEN CAPTURE(S)</strong></strong></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Info for examiner:</em> Must match code from 10.1, 10.2, 10.3 and 10.4, including prompts on screen capture matching those in code. Code for 10.1, 10.2, 10.3 and 10.4 must be sensible.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Screen capture(s) showing an arrow being shot, no message about the monster being hit is displayed and then the invalid move message is displayed when the player tries to shoot an arrow for a second time;</td>
<td></td>
</tr>
</tbody>
</table>