A-Level

COMPUTING

COMP1 – Problem Solving, Programming, Data Representation and Practical Exercise

Mark scheme

2510
June 2014

Version/Stage: 1.1 Final

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.

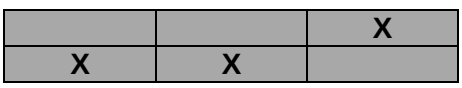
It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment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

Further copies of this Mark Scheme are available from aqa.org.uk

The following annotation is used in the mark scheme:

; - means a single mark
// - means alternative response
/ - means an alternative word or sub-phrase
A. - means acceptable creditworthy answer
R. - means reject answer as not creditworthy
NE - means not enough
I. - means ignore
DPT - means "Don't penalise twice". In some questions a specific error made by a candidate, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should only result in a candidate losing one mark, on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated'.

No marks will be awarded for answers to testing questions where there is no evidence of programming code for the question(s) asked or where the screen captures provided by the candidate do not match what would be produced by the programming code.

Qu	Part	Marking Guidance	Marks
1	1	182;	1
1	2	-;74;	2
1	3	-128; to (+)127; **Mark as follows:** Lowest value identified correctly; Highest value identified correctly;	2
1	4	5 11/16 // 5.6875;; **A.** 91 ÷ 16;; **Mark as follows:** Correct whole number part (5); Correct fractional/decimal part (11/16 or 0.6875);	2
1	5	B;6;	2
1	6	Easier for people to read/understand; **R.** If implication is it easier for a computer to read/understand Can be displayed using fewer digits; More compact when printed/displayed; **NE.** Takes up less space **NE.** More compact	MAX 1
1	7	Shift all the bits one place to the left; and add a zero // Add an extra 0; to the RHS of the bit pattern; // **A.** Arithmetic left shift applied once / by one place;;	2
2	8	A (step-by-step) description of how to complete a task / a description of a process that achieves some task / a sequence of steps that solve a problem / A sequence of unambiguous instructions for solving a problem; Independent of any programming language; That can be completed in finite time;	MAX 2
2	9	 **Marks as follows:** 1 mark for any two correct columns; 2 marks for all three columns correct; **A.** Other, sensible, indicators instead of X	2

<table>
<tr><td>2</td><td>10</td><td>
<table>
<tr><th>x</th><th>c</th><th>b</th><th>a</th><th>Printed output</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>000</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>001</td></tr>
<tr><td>2</td><td>0</td><td>1</td><td>1</td><td>011</td></tr>
<tr><td>3</td><td>0</td><td>1</td><td>0</td><td>010</td></tr>
<tr><td>4</td><td>1</td><td>1</td><td>0</td><td>110</td></tr>
<tr><td>5</td><td>1</td><td>1</td><td>1</td><td>111</td></tr>
<tr><td>6</td><td>1</td><td>0</td><td>1</td><td>101</td></tr>
<tr><td>7</td><td>1</td><td>0</td><td>0</td><td>100</td></tr>
</table>
Mark as follows:
Any one row containing the correct values for c, b and a;
Any three rows containing the correct values for c, b and a;
All rows contain the correct values for c, b and a;
X column correct;
Printed output column correct; A. printed output column incorrect – but matches the (incorrect) values provided for c, b and a, as long as a minimum of 3 rows have been completed

I. Extra row at start of table containing the values 0, 0, 0, 0, 000
</td><td>5</td></tr>
<tr><td>2</td><td>11</td><td>Print the (first 8) Gray code numbers; //
(3 bit) Gray code counter;

NE Convert to Gray code</td><td>1</td></tr>
<tr><td>3</td><td>12</td><td>Sort the list of numbers // Sort L;</td><td>1</td></tr>
<tr><td>3</td><td>13</td><td>The initial situation;</td><td>1</td></tr>
<tr><td>3</td><td>14</td><td>Ownership;
Resources;
Constraints;</td><td>MAX 2</td></tr>
<tr><td>3</td><td>15</td><td>FOR Count2← 1 TO (MAX – 1);

A. Any answer where meaning is clear</td><td>1</td></tr>
<tr><td>3</td><td>16</td><td>L[Count2] ← L[Count2 + 1];

A. Any answer where meaning is clear</td><td>1</td></tr>
<tr><td>3</td><td>17</td><td>L[Count2 + 1] ← Temp;

A. Any answer where meaning is clear</td><td>1</td></tr>
</table>

3	18	63;	1
3	19	Set `SwapMade` to have a value of `False` before the inner loop starts; If a swap is made then set `SwapMade` to `True`; Change the outer loop so that it keeps on repeating until `SwapMade` equals `False`; **Note:** if neither of the first two mark points have been awarded 1 mark should be awarded for the idea of creating a flag / Boolean variable **Alternative answer** Set `NoMoreSwaps` to have a value of `True` before the inner loop starts; If a swap is made then set `NoMoreSwaps` to `False`; Change the outer loop so that it keeps on repeating until `NoMoreSwaps` equals `True`; **Note:** if neither of the first two mark points have been awarded 1 mark should be awarded for the idea of creating a flag / Boolean variable **Alternative answer** Set `NoOfSwaps` to have a value of `0` before the inner loop starts; If a swap is made then increment `NoMoreSwaps`; Change the outer loop so that it keeps on repeating until `NoMoreSwaps` equals `0`; **Note:** if neither of the first two mark points have been awarded 1 mark should be awarded for the idea of creating a counter variable **A.** any sensible identifier **A.** no identifier specified **A.** alternative sensible data type **A.** pseudo-code answers	3
4	20	Correct variable declarations for `ISBN`, `CalculatedDigit` and `Count`; `For` loop, with syntax allowed by the programming language, set up to repeat the correct number of times; Correct prompt "`Please enter next digit of ISBN: `"; Followed by `ISBN[Count]` assigned value entered by the user – must be inside the 1st iterative structure; `CalculatedDigit` and `Count` initialised correctly (must be after 1st iterative structure and before 2nd iterative structure); 2nd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A.** alternative correct logic for condition	15

		`CalculatedDigit` assigned the value of its original value added to `ISBN[Count]` followed by incrementing `Count` – both inside the loop; `CalculatedDigit` assigned the value of its original value added to `ISBN[Count] * 3` followed by incrementing `Count` – must be in the loop and after the 1st two assignment statements in the loop; 3rd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A.** alternative correct logic for the condition 10 subtracted from value of`CalculatedDigit` and result assigned to `CalculatedDigit` – must be the only statement inside an iterative structure; Assignment statement `CalculatedDigit ← 10 – CalculatedDigit` - must not be in an iteration or selection structure; 1st `IF` statement with correct condition – must not be in an iterative structure – with `CalculatedDigit` being assigned the value 0 inside the selection structure; 2nd `IF` statement with correct condition – must not be in an iterative structure or inside the 1st selection structure; Correct output message (`Valid ISBN`) in `THEN` part of selection structure; Correct output message (`Invalid ISBN`) in `ELSE` part of selection structure; **I.** Case of variable names and output messages **A.** Minor typos in variable names and output messages **I.** spacing in prompts **A.** initialisation of variables at declaration stage **A.** Arrays using positions 0 to 12 instead of 1 to 13	

4	21	****SCREEN CAPTURE**** *Must match code from 20, including prompts on screen capture matching those in code. Code for 20 must be sensible.* **Mark as follows:** 'Please enter next digit of ISBN: ' + user input of 9 'Please enter next digit of ISBN: ' + user input of 7 'Please enter next digit of ISBN: ' + user input of 8 'Please enter next digit of ISBN: ' + user input of 0 'Please enter next digit of ISBN: ' + user input of 0 'Please enter next digit of ISBN: ' + user input of 9 'Please enter next digit of ISBN: ' + user input of 9 'Please enter next digit of ISBN: ' + user input of 4 'Please enter next digit of ISBN: ' + user input of 1 'Please enter next digit of ISBN: ' + user input of 0 'Please enter next digit of ISBN: ' + user input of 6 'Please enter next digit of ISBN: ' + user input of 7 'Please enter next digit of ISBN: ' + user input of 6; 'Valid ISBN ' message shown; **A.** alternative output messages if match code for 20 **A.** if can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (6, 7, 6)	2
4	22	****SCREEN CAPTURE**** *Must match code from 20, including prompts on screen capture matching those in code. Code for 20 must be sensible.* **Mark as follows:** 'Please enter next digit of ISBN: ' + user input of 9 'Please enter next digit of ISBN: ' + user input of 7 'Please enter next digit of ISBN: ' + user input of 8 'Please enter next digit of ISBN: ' + user input of 1 'Please enter next digit of ISBN: ' + user input of 8 'Please enter next digit of ISBN: ' + user input of 5 'Please enter next digit of ISBN: ' + user input of 7 'Please enter next digit of ISBN: ' + user input of 0 'Please enter next digit of ISBN: ' + user input of 2 'Please enter next digit of ISBN: ' + user input of 8 'Please enter next digit of ISBN: ' + user input of 8 'Please enter next digit of ISBN: ' + user input of 9 'Please enter next digit of ISBN: ' + user input of 4 'Invalid ISBN ' message shown; **A.** alternative output messages if match code for 20 **A.** if can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (8, 9, 4)	1

5	23	`TCard` // `TRecentScore` // `TDeck` (Pascal only) // `TRecentScores` (Pascal only); **R.** if any additional code **R.** if spelt incorrectly **I.** case	1

5	24	`CInt` (VB.Net / VB6 only) //
		`Val` (Pascal only) //
		`StrToInt` (Delphi only) //
		`parseInt` (Java only) //
		`Integer.parseInt` (Java only) //
		`int` (Python only);
		
		R. if any additional code
		R. if spelt incorrectly
		I. case
			1

5	25	`Deck`//`RecentScores`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
5	26	Temporary;	1
5	27	Most recent holder;	1
5	28	Stepper;	1
5	29	When the name in the variable `PlayerX` is not in the array `RecentScores`; **A.** answer that does not use identifiers but clearly suggests that the name is not in the array	1

5	30	`WHILE Found = False AND Position <= NoOfRecentScores;;` **A.** Alternative loop conditions that would provide correct functionality eg `Position <4` , `Position <= 3` **A.** Description of how to alter code instead of altered code **Mark as follows:** 1 mark for identifying that an additional termination condition is needed // identifying that code is needed to prevent the attempt to access an array element that does not exist; 1 mark for correct additional condition and correct logic of entire compound condition; **Note:** alternative valid methods that solve the problem should be referred to team leader	2
5	31	Linear search; **NE** Search	1
6	32	Repetition structure in the correct place in the code with correct termination condition; Correct error message displayed; Error message will be displayed every time an invalid name has been entered and will only be displayed when an invalid name has been entered; Getting name from user is inside the repetition structure; **A.** Minor typos in error message **I.** Capitalisation and spacing in error message	4
6	33	****SCREEN CAPTURE**** *Must match code from 32, including prompts on screen capture matching those in code. Code for 32 must be sensible.* **Mark as follows:** No name entered and either error message displayed or asked to enter name; **R.** If does not match code for 32 Name of `Emily` entered and no error message displayed;	2

7	34	Selection structure that checks if the current and last card have the same rank; **A.** equivalent logic Selection structure that checks if the suit of the next card is higher than the suit of the last card; **A.** equivalent logic **A.** one selection structure with two conditions **Note:** if overall logic for the first two mark points is not correct only one of the two marks is to be given `Higher` assigned value of `True` if the two cards have the same rank and the suit of the next card is higher; **R.** If `Higher` always assigned the value of `True` Value of `Higher` is returned to the calling routine; **R.** if no evidence of code used to calculate value of `Higher` when the two cards have the same rank **MAX 3** if any existing functionality is incorrectly changed or if an incorrect value is returned under any circumstances	4
7	35	****SCREEN CAPTURE**** *Must match code from 34, including prompts on screen capture matching those in code. Code for 34 must be sensible.* **Mark as follows:** y entered by user results in message `Well done! You guessed correctly.` Followed by n entered by user resulting in message `Well done! You guessed correctly.`; **I.** if first y entered and first message not shown on screen capture **A.** if code for 34 has been attempted and screen capture matches what would be produced by code for 34 Followed by y entered by user resulting in message `Well done! You guessed correctly.`; **R.** if test is for a random (shuffled deck) game **R.** If answer for 34 has no code for checking if the ranks of the two cards are equal / not equal	2
8	36	Modified message in sensible place in code `Do you think the next card will be higher than the last card (enter y or n or j to play a joker)?`; **A.** any sensible message **A.** two messages instead of a modified message **A.** no evidence in 36 of this modified message being in the correct place in the code if there is supporting evidence from screen capture(s) for 38 that it is in the correct place	1

8	37	Appropriately named variable (eg `NoOfJokers`), of sensible data type, given initial value of 2; Modify loop condition so that `y`, `n` and `j` are all allowed; Additional condition so that `j` is only allowed if `NoOfJokers` is greater than 0; correct logic used; **A.** player loses game if they try to play a 3rd joker as long as correct final score is displayed – note that using this method it is possible that a selection structure is being used instead of a modified loop;; **A.** equivalent logic Value of `NoOfJokers` decremented by 1 inside a selection structure; which has correct condition to check if `j` was option chosen by user; Modify selection structure so that correct guess is called if either the user has guessed correctly or the player used a Joker; **R.** If code will not allow the player to always use two jokers **Alternative answer** Appropriately named variable (e.g.`NoOfJokers`) of sensible data type, given initial value of 0; **A.** value of 0 not explicitly given if code would work without this Modify loop condition so that `y`, `n` and `j` are all allowed; Additional condition so that `j` is only allowed if `NoOfJokers` is less than 2; correct logic used; **A.** player loses game if they try to play a 3rd joker as long as correct final score is displayed – note that using this method it is possible that a selection structure is being used instead of a modified loop;; **A.** equivalent logic Value of `NoOfJokers` incremented by 1 inside a selection structure; which has correct condition to check if `j` was option chosen by user; Modify selection structure so that correct guess is called if either the user has guessed correctly or the player used a Joker; **R.** If code will not allow the player to always use two jokers **MAX 5** if, when the game continues, there are unwanted side-effects (e.g. 3rd joker allowed, `Deck` changed when it shouldn't be, score goes up when `j` entered for a third time, etc...)	7

8	38	****SCREEN CAPTURE**** *Must match code from 36 and 37, including prompts on screen capture matching those in code. Code for 37 must be sensible.* **Mark as follows:** `j` entered by user results in message `Well done! You guessed correctly.`; **R.** if this aspect of test is for a random (shuffled deck) game 2nd `j` entered by user results in message `Well done! You guessed correctly.`; **R.** if this aspect of test is for a random (shuffled deck) game 3rd `j` entered by user results in message `Do you think the next card will be higher than the last card (enter y or n)?`; **A.** if test for 3rd joker being played is for a random (shuffled deck) game **A.** message not being displayed and game ends (only if matches code for 37) **I.** additional error messages being displayed after j entered and before the message `Do you think the next card will be higher than the last card (enter y or n)?` as long as error messages match code for 37 **R.** if player's score is increased when they play a 3rd joker	3
9	39	**A.** any sensibly named identifiers for variables/parameters instead of those used in this mark scheme **There are 5 marks available for setting up a new subroutine and the routine interface:** Created a new subroutine named `CalculateProbability`; Correct routine interface with parameters of `LastCard` and `Deck` of correct data type; All data needed by new subroutine is passed to the subroutine via the routine interface (ie no data values obtained from global variables); Mechanism to return a numeric value to the calling routine set up; **R.** use of global variable Value calculated by subroutine is returned to calling routine; **I.** additional parameters	11

		There are then 6 marks available for calculating the probability: Repetition structure set up to look at each card in `Deck` that has not yet been used in the game; Selection structure, inside repetition structure, that checks if `LastCard` is higher than a card in `Deck`; Inside the selection structure `NoOfCardsHigher` incremented if the condition in the selection structure is for a comparison of two cards; **R.** If `NoOfCardsHigher` always incremented Inside the selection structure `NoOfCardsLower` incremented **R.** If `NoOfCardsLower` is always incremented; // Inside the selection structure `NoOfCardsHigher` incremented if the condition in the selection structure is for a comparison of two cards; **R.** If `NoOfCardsHigher` always incremented Correct calculation for `NoOfCardsInDeck` (does not matter if inside or outside repetition structure); // Inside the selection structure `NoOfCardsLower` incremented if the condition in the selection structure is for a comparison of two cards; **R.** If `NoOfCardsLower` always incremented Correct calculation for `NoOfCardsInDeck` (does not matter if inside or outside repetition structure); Correctly calculates the number of cards, that have not been used in the game so far, that are higher/lower than `LastCard` in `Deck`; Dividing `NoOfCardsHigher` by `NoOfCardsInDeck` // Dividing `NoOfCardsHigher` by the sum of `NoOfCardsHigher` and `NoOfCardsLower`; **A.** any equivalent calculation **A.** correct expression using incorrectly calculated values for `NoOfCardsHigher` / `NoOfCardsLower` **Note:** alternative methods that calculate the probability correctly should be referred to team leader.	

9	40	Call to `CalculateProbability` subroutine in correct place; **R.** if parameter list does not match answer to 39 Displays "`The probability of the next card being higher is` " in correct place; **A.** minor typos in prompt **I.** capitalisation Displays the calculated probability; **R.** if probability not returned by `CalculateProbability` subroutine **A.** use of temporary variable to store the value returned by `CalculateProbability` with contents of temporary variable then displayed using output message **A.** incorrect probability as long as value displayed is the value returned by `CalculateProbability` subroutine **I.** Case of identifiers and output messages **A.** Minor typos in output messages **I.** spacing in output messages	3
9	41	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 39 and/or 40* `The probability of the next card being higher is 1`; User enters y followed by `The probability of the next card being higher is 0.9` ; **A.** probabilities expressed as percentages (100, 90) **A.** probabilities expressed as fractions (51 / 51, 45 / 50) **A.** probabilities expressed in scientific form (1.00E+00, 0.90E+00) **A.** 0.9411765 and 0.88 instead of 1 and 0.9 - if question 7 not completed / completed after question 9 **A.** other values for probabilities that are correct based on incorrect answer for question 7 only if code for question 9 is correct **R.** if test is for a random (shuffled deck) game **R.** probability of 0	2

Pascal

Qu	Part	Marking Guidance	Marks

4	20	Program Question4;
		Var
		  CalculatedDigit : Integer;
		  ISBN : Array[1..13] Of Integer;
		  Count : Integer;
		Begin
		  For Count := 1 To 13
		  Do
		  Begin
		  Writeln('Please enter next digit of ISBN: ');
		  Readln(ISBN[Count]);
		  End;
		  CalculatedDigit := 0;
		  Count := 1;
		  While Count < 13
		  Do
		  Begin
		  CalculatedDigit := CalculatedDigit + ISBN[Count];
		  Count := Count + 1;
		  CalculatedDigit := CalculatedDigit + ISBN[Count] * 3;
		  Count := Count + 1;
		  End;
		  While CalculatedDigit >= 10
		  Do CalculatedDigit := CalculatedDigit - 10;
		  CalculatedDigit := 10 - CalculatedDigit;
		  If CalculatedDigit = 10
		  Then CalculatedDigit := 0;
		  If CalculatedDigit = ISBN[13]
		  Then Writeln('Valid ISBN')
		  Else Writeln('Invalid ISBN');
		  Readln;
		End.
			15

6	32	...
		Writeln;
		Repeat
		  Write('Please enter your name: ');
		  Readln(PlayerName);
		  If Length(PlayerName) = 0
		  Then Writeln('You must enter a name');
		Until Length(PlayerName) > 0;
		Writeln;
		...
			4

		Alternative answer
		
		`...`
		`Writeln;`
		`Repeat`
		`  Write('Please enter your name: ');`
		`  Readln(PlayerName);`
		`  If PlayerName = ''`
		`  Then Writeln('You must enter a name');`
		`Until PlayerName <> '';`
		`Writeln;`
		`...`
		
		Alternative answer
		
		`...`
		`Writeln;`
		`PlayerName := '';`
		`While PlayerName = ''`
		`  Do`
		`  Begin`
		`  Write('Please enter your name: ');`
		`  Readln(PlayerName);`
		`  If Length(PlayerName) = 0`
		`  Then Writeln('You must enter a name');`
		`  End;`
		`Writeln;`
		`...`	

7	34	`Function IsNextCardHigher(LastCard, NextCard : TCard)`
		`: Boolean;`
		`  Var`
		`  Higher : Boolean;`
		`  Begin`
		`  Higher := False;`
		`  If NextCard.Rank > LastCard.Rank`
		`  Then Higher := True;`
		`  If NextCard.Rank = LastCard.Rank`
		`  Then`
		`  If NextCard.Suit > LastCard.Suit`
		`  Then Higher := True;`
		`  IsNextCardHigher := Higher;`
		`  End;`
		
		Alternative answer
		
		`Function IsNextCardHigher(LastCard, NextCard : TCard)`
		`: Boolean;`
		`  Var`
		`  Higher : Boolean;`
		`  Begin`
			4

		If NextCard.Rank > LastCard.Rank
		  Then Higher := True
		Else
		  If NextCard.Rank < LastCard.Rank
		  Then Higher := False
		  Else
		  If NextCard.Suit > LastCard.Suit
		  Then Higher := True
		  Else Higher := False;
		IsNextCardHigher := Higher;
		End;
		
		Alternative answer
		
		Function IsNextCardHigher(LastCard, NextCard : TCard) : Boolean;
		Var
		  Higher : Boolean;
		Begin
		  If NextCard.Rank > LastCard.Rank
		  Then Higher := True
		  Else Higher := (NextCard.Rank = LastCard.Rank) AND (NextCard.Suit > LastCard.Suit);
		IsNextCardHigher := Higher;
		End;	

8	36	... Var Choice : Char; Begin Write('Do you think the next card will be higher than the last card (enter y or n **or j to play a joker)**? '); Readln(Choice); ...	1

8	37	...
		Choice : Char;
		NoOfJokers : Integer;
		Begin
		NoOfJokers := 2;
		GameOver := False;
		...
		While (NoOfCardsTurnedOver < 52) And Not GameOver
		...
		Repeat
		Choice := GetChoiceFromUser;
		Until (Choice = 'y') Or (Choice = 'n'**) Or (Choice = 'j') And (NoOfJokers > 0)**;
		If Choice = 'j'
		Then NoOfJokers := NoOfJokers - 1;
			7

		DisplayCard(NextCard);
		NoOfCardsTurnedOver := CardsTurnedOver + 1;
		Higher := IsNextCardHigher(LastCard, NextCard);
		If Higher And(Choice='y') Or Not Higher And (Choice = 'n') **Or (Choice = 'j')**
		Then
		Begin
		DisplayCorrectGuessMessage(NoOfCardsTurnedOver);
		...
		
		A. equivalent logic for condition (eg NoOfJokers >=1)
		
		Alternative Answer
		...
		Choice : Char;
		NoOfJokers : Integer;
		Begin
		NoOfJokers := 0;
		GameOver := False;
		...
		While (NoOfCardsTurnedOver< 52) And Not GameOver
		...
		Repeat
		Choice := GetChoiceFromUser;
		Until (Choice = 'y') Or (Choice = 'n'**) Or (Choice = 'j') And (NoOfJokers <=1)**;
		If Choice = 'j'
		Then NoOfJokers := NoOfJokers + 1;
		DisplayCard(NextCard);
		NoOfCardsTurnedOver := CardsTurnedOver + 1;
		Higher := IsNextCardHigher(LastCard, NextCard);
		If Higher And(Choice='y') Or Not Higher And (Choice = 'n') **Or (Choice = 'j')**
		Then
		Begin
		DisplayCorrectGuessMessage(NoOfCardsTurnedOver);
		...
		
		A. equivalent logic for condition (eg NoOfJokers < 2)	

9	39	Function CalculateProbability(Deck : TDeck; NoOfCardsTurnedOver : Integer; LastCard : TCard) : Real; Var Probability : Real; Count : Integer; NoOfCardsHigher : Integer; NoOfCardsLower : Integer; Begin NoOfCardsHigher := 0; NoOfCardsLower := 0;	11

```
    For Count := 1 To (52 – NoOfCardsTurnedOver)
      Do
        If IsNextCardHigher(LastCard, Deck[Count])
          Then NoOfCardsHigher := NoOfCardsHigher + 1
          Else NoOfCardsLower := NoOfCardsLower + 1;
    Probability := NoOfCardsHigher / (NoOfCardsHigher
+ NoOfCardsLower);
    CalculateProbability := Probability;
  End;
```

Alternative answer

```
...
For Count := 1 To (52 – NoOfCardsTurnedOver)
  Do
    If IsNextCardHigher(LastCard, Deck[Count])
      Then NoOfCardsHigher := NoOfCardsHigher + 1;
Probability := NoOfCardsHigher / (52 -
NoOfCardsTurnedOver);
CalculateProbability := Probability;
...
```

Alternative answer

```
Function CalculateProbability(Deck : TDeck; LastCard :
TCard) : Real;
Var
  Probability :Real;
  Count : Integer;
  NoOfCardsHigher : Integer;
  Begin
    NoOfCardsHigher := 0;
    Count := 1;
    While (Count < 52) And (Deck[Count].Suit <> 0)
      Do
        Begin
          If IsNextCardHigher(LastCard, Deck[Count])
            Then NoOfCardsHigher := NoOfCardsHigher +
1;
            Count := Count + 1;
        End;
    Probability := NoOfCardsHigher / (Count - 1);
    CalculateProbability:= Probability;
  End;
```

A. `Deck[Count].Rank` instead of `Deck[Count].Suit`

9	40	... **Writeln('The probability of the next card being higher is ', CalculateProbability(Deck, NoOfCardsTurnedOver,**	3

		LastCard):3:2); GetCard(NextCard, Deck, NoOfCardsTurnedOver); Repeat Choice := GetChoiceFromUser; ...	

VB.Net

Qu	Part	Marking Guidance	Marks

4	20	Module Module1
		  Sub Main()
		  Dim CalculatedDigit As Integer
		  Dim ISBN(13) As Integer
		  Dim Count As Integer
		  For Count = 1 To 13
		  Console.Write("Please enter next digit of ISBN: ")
		  ISBN(Count) = Console.ReadLine()
		  Next
		  CalculatedDigit = 0
		  Count = 1
		  While Count < 13
		  CalculatedDigit = CalculatedDigit + ISBN(Count)
		  Count = Count + 1
		  CalculatedDigit = CalculatedDigit + ISBN(Count) * 3
		  Count = Count + 1
		  End While
		  While CalculatedDigit >= 10
		  CalculatedDigit = CalculatedDigit - 10
		  End While
		  CalculatedDigit = 10 - CalculatedDigit
		  If CalculatedDigit = 10 Then
		  CalculatedDigit = 0
		  End If
		  If CalculatedDigit = ISBN(13) Then
		  Console.WriteLine("Valid ISBN")
		  Else
		  Console.WriteLine("Invalid ISBN")
		  End If
		  Console.ReadLine()
		  End Sub
		End Module
			15

6	32	...
		Console.WriteLine()
		Do
		  Console.Write("Please enter your name: ")
		  PlayerName = Console.ReadLine
		  If PlayerName.Length = 0 Then
		  Console.WriteLine("You must enter a name")
		  End If
		Loop Until PlayerName.Length > 0
		Console.WriteLine()
		...
			4

		Alternative answer
		
		`...`
		`Console.WriteLine()`
		`Do`
		`  Console.Write("Please enter your name: ")`
		`  PlayerName = Console.ReadLine`
		`  If PlayerName = "" Then`
		`  Console.WriteLine("You must enter a name")`
		`  End If`
		`Loop Until PlayerName <> ""`
		`Console.WriteLine()`
		`...`
		
		Alternative answer
		
		`...`
		`Console.WriteLine()`
		`PlayerName = ""`
		`While PlayerName = ""`
		`  Console.Write("Please enter your name: ")`
		`  PlayerName = Console.ReadLine`
		`  If PlayerName = "" Then`
		`  Console.WriteLine("You must enter a name")`
		`  End If`
		`End While`
		`Console.WriteLine()`
		`...`	

7	34	`Function IsNextCardHigher(ByVal LastCard As TCard, ByVal NextCard As TCard) As Boolean`
		`  Dim Higher As Boolean`
		`  Higher = False`
		`  If NextCard.Rank > LastCard.Rank Then`
		`  Higher = True`
		`  End If`
		`  If NextCard.Rank = LastCard.Rank Then`
		`  If NextCard.Suit > LastCard.Suit Then`
		`  Higher = True`
		`  End If`
		`  End If`
		`  Return Higher`
		`End Function`
		
		Alternative answer
		
		`Function IsNextCardHigher(ByVal LastCard As TCard, ByVal NextCard As TCard) As Boolean`
		`  Dim Higher As Boolean`
		`  If NextCard.Rank > LastCard.Rank Then`
		`  Higher = True`
			4

		ElseIf NextCard.Rank < LastCard.Rank Then
		  Higher = False
		ElseIf NextCard.Suit > LastCard.Suit Then
		  Higher = True
		Else
		  Higher = False
		End If
		Return Higher
		End Function
		
		Alternative answer
		
		Function IsNextCardHigher(ByVal LastCard As TCard, ByVal NextCard As TCard) As Boolean
		Dim Higher As Boolean
		Higher = False
		If NextCard.Rank > LastCard.Rank Then
		  Higher = True
		ElseIf (NextCard.Rank = LastCard.Rank) And (NextCard.Suit > LastCard.Suit) Then
		  Higher = True
		End If
		Return Higher
		End Function	

8	36	... Dim Choice As Char Console.Write("Do you think the next card will be higher than the last card (enter y or n **or j to play a joker)**? ") Choice = Console.ReadLine ...	1

8	37	...
		Dim Choice As Char
		Dim NoOfJokers As Integer
		NoOfJokers = 2
		GameOver = False
		...
		While NoOfCardsTurnedOver < 52 And Not GameOver
		...
		Do
		  Choice = GetChoiceFromUser()
		Loop Until Choice = "y" Or Choice = "n**" Or Choice = "j" And NoOfJokers > 0**
		If Choice = "j" Then
		  NoOfJokers = NoOfJokers - 1
		End If
		DisplayCard(NextCard)
		CardsTurnedOver = CardsTurnedOver + 1
		Higher = IsNextCardHigher(LastCard, NextCard)
			7

		If Higher And Choice = "y" Or Not Higher And Choice = "n" **Or Choice = "j"** Then
		  DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
		...
		
		A. equivalent logic for condition (eg `NoOfJokers >= 1`)
		
		Alternative Answer
		
		...
		Dim Choice As Char
		Dim NoOfJokers As Integer
		NoOfJokers = 0
		GameOver = False
		...
		While NoOfCardsTurnedOver < 52 And Not GameOver
		  ...
		  Do
		  Choice = GetChoiceFromUser()
		  Loop Until Choice = "y" Or Choice = "n" **Or Choice = "j" And NoOfJokers <= 1**
		  If Choice = "j" Then
		  NoOfJokers = NoOfJokers + 1
		  End If
		  DisplayCard(NextCard)
		  CardsTurnedOver = CardsTurnedOver + 1
		  Higher = IsNextCardHigher(LastCard, NextCard)
		  If Higher And Choice = "y" Or Not Higher And Choice = "n"**Or Choice = "j"** Then
		  DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
		...
		
		A. equivalent logic for condition (eg `NoOfJokers < 2`)	

9	39	Function CalculateProbability(ByVal Deck() As TCard, ByVal NoOfCardsTurnedOver As Integer, ByVal LastCard As TCard) As Single
		  Dim Probability As Single
		  Dim Count As Integer
		  Dim NoOfCardsHigher As Integer = 0
		  Dim NoOfCardsLower As Integer = 0
		  For Count = 1 To(52 – NoOfCardsTurnedOver)
		  If IsNextCardHigher(LastCard, Deck(Count)) Then
		  NoOfCardsHigher = NoOfCardsHigher + 1
		  Else
		  NoOfCardsLower = NoOfCardsLower + 1
		  End If
		  Next
		  Probability = NoOfCardsHigher / (NoOfCardsHigher +
			11

		NoOfCardsLower)
		  Return Probability
		End Function
		
		Alternative answer
		
		...
		For Count = 1 To (52 – NoOfCardsTurnedOver)
		  If IsNextCardHigher(LastCard, Deck(Count)) Then
		  NoOfCardsHigher = NoOfCardsHigher + 1
		  End If
		Next
		Probability = NoOfCardsHigher / (52 -
		NoOfCardsTurnedOver)
		CalculateProbability = Probability
		...
		
		Alternative answer
		
		Function CalculateProbability(ByVal Deck() As TCard,
		ByVal LastCard As TCard) As Single
		  Dim Probability As Single
		  Dim Count As Integer
		  Dim NoOfCardsHigher As Integer = 0
		  Count = 1
		  While Count < 52 And Deck(Count).Suit <> 0
		  If IsNextCardHigher(LastCard, Deck(Count)) Then
		  NoOfCardsHigher = NoOfCardsHigher + 1
		  End If
		  Count = Count + 1
		  End While
		  Probability = NoOfCardsHigher / (Count - 1)
		  Return Probability
		End Function
		
		A. `Deck(Count).Rank` instead of `Deck(Count).Suit`
		Note: return mechanism does not need to be explicitly set up in routine interface	

9	40	... **Console.WriteLine("The probability of the next card being higher is " & CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard))** GetCard(NextCard, Deck, NoOfCardsTurnedOver) Do Choice = GetChoiceFromUser() ...	3

VB6

Qu	Part	Marking Guidance	Marks

4	20	`Private Sub Form_Load()`
		`  Dim CalculatedDigit As Integer`
		`  Dim ISBN(13) As Integer`
		`  Dim Count As Integer`
		`  For Count = 1 To 13`
		`  ISBN(Count) = ReadLine("Please enter next digit of ISBN: ")`
		`  Next`
		`  CalculatedDigit = 0`
		`  Count = 1`
		`  While Count < 13`
		`  CalculatedDigit = CalculatedDigit + ISBN(Count)`
		`  Count = Count + 1`
		`  CalculatedDigit = CalculatedDigit + (ISBN(Count) * 3)`
		`  Count = Count + 1`
		`  Wend`
		`  While CalculatedDigit >= 10`
		`  CalculatedDigit = CalculatedDigit - 10`
		`  Wend`
		`  CalculatedDigit = 10 - CalculatedDigit`
		`  If CalculatedDigit = 10 Then`
		`  CalculatedDigit = 0`
		`  End If`
		`  If CalculatedDigit = ISBN(13) Then`
		`  WriteLine("Valid ISBN")`
		`  Else`
		`  WriteLine("Invalid ISBN")`
		`  End If`
		`End Sub`
		
		Alternative answers could use some of the following instead of WriteLine / ReadLine:
		`Console.Text = Console.Text & ...`
		`WriteLineWithMsg`
		`WriteWithMsg`
		`Msgbox`
		`InputBox`
		`WriteNoLine`
			15

6	32	`...`
		`WriteLine("")`
		`Do`
		`PlayerName = ReadLine("Please enter your name: ")`
		`  If Len(PlayerName) = 0 Then`
		`  WriteLineWithMsg ("You must enter a name")`
		`  End If`
			4

		Loop Until Len(PlayerName) > 0
		WriteLine("")
		...
		
		Alternative answer
		
		...
		WriteLine("")
		Do
		  PlayerName = ReadLine("Please enter your name: ")
		  If PlayerName = "" Then
		  WriteLineWithMsg ("You must enter a name")
		  End If
		Loop Until PlayerName <>""
		WriteLine("")
		...
		
		Alternative answer
		
		...
		WriteLine("")
		PlayerName = ""
		While PlayerName = ""
		PlayerName = ReadLine("Please enter your name: ")
		  If PlayerName = "" Then
		  WriteLineWithMsg ("You must enter a name")
		  End If
		Wend
		WriteLine("")
		...
		
		Alternative answers could use some of the following instead of WriteLineWithMsg:
		Console.Text = Console.Text & ...
		WriteLine
		WriteWithMsg
		Msgbox
		WriteNoLine	

7	34	Private Function IsNextCardHigher(ByRef LastCard As TCard, ByRef NextCard As TCard) As Boolean
		  Dim Higher As Boolean
		  Higher = False
		  If NextCard.Rank > LastCard.Rank Then
		  Higher = True
		  End If
		  If NextCard.Rank = LastCard.Rank Then
		  If NextCard.Suit > LastCard.Suit Then
		  Higher = True
		  End If
		  End If
			4

		IsNextCardHigher = Higher
		End Function
		
		Alternative answer
		
		Private Function IsNextCardHigher(ByRef LastCard As TCard, ByRefNextCard As TCard) As Boolean
		Dim Higher As Boolean
		If NextCard.Rank > LastCard.Rank Then
		Higher = True
		ElseIf NextCard.Rank < LastCard.Rank Then
		Higher = False
		ElseIf NextCard.Suit > LastCard.Suit Then
		Higher = True
		Else
		Higher = False
		End If
		IsNextCardHigher = Higher
		End Function
		
		Alternative answer
		
		Private Function IsNextCardHigher(ByRef LastCard As TCard, ByRefNextCard As TCard) As Boolean
		Dim Higher As Boolean
		Higher = False
		If NextCard.Rank > LastCard.Rank Then
		Higher = True
		ElseIf (NextCard.Rank = LastCard.Rank) And (NextCard.Suit > LastCard.Suit) Then
		Higher = True
		End If
		IsNextCardHigher = Higher
		End Function	

8	36	... Dim Choice As String Choice = ReadLine("Do you think the next card will be higher than the last card (enter y or n **or j to play a joker**)? ") GetChoiceFromUser = Choice...	1
8	37	... Dim Choice As String **Dim NoOfJokers As Integer** **NoOfJokers = 2** GameOver = False ... While NoOfCardsTurnedOver< 52 And Not GameOver	7

		...
		Do
		Choice = GetChoiceFromUser()
		Loop Until Choice = "y" Or Choice = "n" **Or Choice = "j" And NoOfJokers > 0**
		If Choice = "j" Then
		NoOfJokers = NoOfJokers - 1
		End If
		Call DisplayCard(NextCard)
		NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
		Higher = IsNextCardHigher(LastCard, NextCard)
		If Higher And Choice = "y" Or Not Higher And Choice = "n"**Or Choice = "j"** Then
		DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
		...
		
		A. equivalent logic for condition (eg NoOfJokers >= 1)
		
		Alternative Answer
		
		Dim Choice As String
		Dim NoOfJokers As Integer
		NoOfJokers = 0
		GameOver = False
		...
		While NoOfCardsTurnedOver < 52 And Not GameOver
		...
		Do
		Choice = GetChoiceFromUser()
		Loop Until Choice = "y" Or Choice = "n" **Or Choice = "j" And NoOfJokers <=1**
		If Choice = "j" Then
		NoOfJokers = NoOfJokers + 1
		End If
		Call DisplayCard(NextCard)
		NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
		Higher = IsNextCardHigher(LastCard, NextCard)
		If Higher And Choice = "y" Or Not Higher And Choice = "n"**Or Choice = "j"** Then
		DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
		...
		
		A. equivalent logic for condition (eg NoOfJokers < 2)	

9	39	Private Function CalculateProbability(ByRef Deck() As TCard, ByVal NoOfCardsTurnedOver As Integer, ByRef LastCard As TCard) As Single Dim Probability As Single	11

```
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer
  Dim NoOfCardsLower As Integer
  NoOfCardsHiger = 0
  NoOfCardsLower = 0
  For Count = 1 To (52 – NoOfCardsTurnedOver)
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    Else
      NoOfCardsLower = NoOfCardsLower + 1
    End If
  Next
  Probability = NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower)
  CalculateProbability = Probability
End Function
```

Alternative answer

```
...
For Count = 1 To (52 – NoOfCardsTurnedOver)
  If IsNextCardHigher(LastCard, Deck(Count)) Then
    NoOfCardsHigher = NoOfCardsHigher + 1
  End If
Next
Probability = NoOfCardsHigher / (52 -
NoOfCardsTurnedOver)
CalculateProbability = Probability
...
```

Alternative answer

```
Private Function CalculateProbability(ByRef Deck() As
TCard, ByRef LastCard As TCard) As Single
  Dim Probability As Single
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer
  NoOfCardsHigher = 0
  Count = 1
  While Count < 52 And Deck(Count).Suit <> 0
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    End If
    Count = Count + 1
  Wend
  Probability = NoOfCardsHigher / (Count - 1)
  CalculateProbability = Probability
End Function
```

A. `Deck(Count).Rank` instead of `Deck(Count).Suit`

9	40	... **WriteLine("The probability of the next card being higher is " & CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard))** Call GetCard(NextCard, Deck, NoOfCardsTurnedOver) Do Choice = GetChoiceFromUser() ... **Alternative answers could use some of the following instead of WriteLine:** Console.Text = Console.Text & ... WriteLineWithMsg WriteWithMsg Msgbox WriteNoLine	3

Python 2

Qu	Part	Marking Guidance	Marks

4	20	`# Question 4`
		`if __name__ == "__main__":`
		`  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
		`  for Count in range(1, 14):`
		`  print 'Please enter next digit of ISBN: ',`
		`  ISBN[Count] = int(raw_input())`
		`  CalculatedDigit = 0`
		`  Count = 1`
		`  while Count < 13:`
		`  CalculatedDigit = CalculatedDigit + ISBN[Count]`
		`  Count = Count + 1`
		`  CalculatedDigit = CalculatedDigit + ISBN[Count] * 3`
		`  Count = Count + 1`
		`  while CalculatedDigit >= 10:`
		`  CalculatedDigit = CalculatedDigit - 10`
		`  CalculatedDigit = 10 - CalculatedDigit`
		`  if CalculatedDigit == 10:`
		`  CalculatedDigit = 0`
		`  if CalculatedDigit == ISBN[13]:`
		`  print 'Valid ISBN'`
		`  else:`
		`  print 'Invalid ISBN'`
		
		Alternative print/input combination:
		
		`ISBN[Count] = int(raw_input('Please enter next digit of ISBN: ',))`
			15

6	32	`...`
		`print`
		`Playername = ''`
		`while len(PlayerName) == 0:`
		`  print 'Please enter your name: '`
		`  PlayerName = raw_input()`
		`  if len(PlayerName) == 0:`
		`  print 'You must enter a name'`
		`print`
		
		Alternative answer:
		
		`...`
		`print`
		`PlayerName =` `''`
		`while PlayerName == '':`
		`  print 'Please enter your name: ',`
		`  PlayerName = raw_input()`
			4

		`  if PlayerName == '':`
		`    print 'You must enter a name'`
		`print`	

7	34	`def IsNextCardHigher(LastCard, NextCard):`
		`  Higher = False`
		`  if NextCard.Rank > LastCard.Rank:`
		`    Higher = True`
		`  **if NextCard.Rank == LastCard.Rank:**`
		`  **  if NextCard.Suit > LastCard.Suit:**`
		`  **    Higher = True**`
		`  return Higher`
		
		Alternative answer
		
		`def IsNextCardHigher(LastCard, NextCard):`
		`  Higher = False`
		`  if NextCard.Rank > LastCard.Rank:`
		`    Higher = True`
		`  **else:**`
		`  **  if NextCard.Rank < LastCard.Rank:**`
		`  **    Higher = False**`
		`  **  else:**`
		`  **    if NextCard.Suit > LastCard.Suit:**`
		`  **      Higher = True**`
		`  **    else:**`
		`  **      Higher = False**`
		`  return Higher`
		
		Alternative answer
		
		`def IsNextCardHigher(LastCard, NextCard):`
		`  Higher = False`
		`  if NextCard.Rank > LastCard.Rank:`
		`    Higher = True`
		`  **elif NextCard.Rank == LastCard.Rank and NextCard.Suit > LastCard.Suit:**`
		`  **  Higher = True**`
		`  return Higher`
			4

8	36	`...`
		`print 'Do you think the next card will be higher than the last card (enter y or n **or j toplay a joker**)? '`
		`Choice = raw_input()`
		`...`
			1

8	37	`...`
		`**NoOfJokers = 2**`
		`GameOver = False`
			7

		...
		while (NoOfCardsTurnedOver < 52) and (not GameOver):
		...
		  Choice = ''
		  while (Choice != 'y') and (Choice != 'n') **and ((Choice != 'j') or (NoOfJokers == 0))**:
		  Choice = GetChoiceFromUser()
		  if Choice == 'j':
		  NoOfJokers = NoOfJokers - 1
		  DisplayCard(NextCard)
		  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
		  Higher = IsNextCardHigher(LastCard, NextCard)
		  if (Higher and Choice == 'y') or (not Higher and Choice == 'n') **or (Choice == 'j')**:
		...
		
		A. Equivalent logic for condition (eg NoOfJokers < 1)
		
		Alternative answer:
		
		...
		NoOfJokers = 0
		GameOver = False
		...
		while (NoOfCardsTurnedOver < 52) and (not GameOver):
		...
		  Choice = ''
		  while (Choice != 'y') and (Choice != 'n') **and ((Choice != 'j') or (NoOfJokers == 2))**:
		  Choice = GetChoiceFromUser()
		  if Choice == 'j':
		  NoOfJokers = NoOfJokers + 1
		  DisplayCard(NextCard)
		  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
		  Higher = IsNextCardHigher(LastCard, NextCard)
		  if (Higher and Choice == 'y') or (not Higher and Choice == 'n') **or (Choice == 'j')**:
		...
		
		A. Equivalent logic for condition (eg NoOfJokers > 1)	

9	39	def CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard):
		  NoOfCardsHigher = 0
		  NoOfCardsLower = 0
		  for Count in range(1, 53 – NoOfCardsTurnedOver):
		  if (IsNextCardHigher(LastCard, Deck[Count]):
			11

		NoOfCardsHigher = NoOfCardsHigher + 1
		else:
		NoOfCardsLower = NoOfCardsLower + 1
		Probability = NoOfCardsHigher / (NoOfCardsHigher + NoOfCardsLower)
		return Probability
		
		Alternative answer:
		
		...
		for Count in range(1, 53 – NoOfCardsTurnedOver):
		if (IsNextCardHigher(LastCard, Deck[Count]):
		NoOfCardsHigher = NoOfCardsHigher + 1
		Probability = NoOfCardsHigher / (52 – NoOfCardsTurnedOver)
		return Probability
		...
		
		Alternative answer:
		
		def CalculateProbability(Deck,LastCard):
		NoOfCardsHigher = 0
		Count = 1
		while (Count < 52) and (Deck[Count].Suit != 0):
		if (IsNextCardHigher(LastCard, Deck[Count]):
		NoOfCardsHigher = NoOfCardsHigher + 1
		Count = Count + 1
		Probability = NoOfCardsHigher / (Count - 1)
		return Probability
		
		A. Deck[Count].Rank **instead of** Deck[Count].Suit	

9	40	... **print 'The probability of the next card being higher is %3.2f' %CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard)** GetCard(NextCard, Deck, NoOfCardsTurnedOver) Choice = '' while (Choice != 'y') and (Choice != 'n'): Choice = GetChoiceFromUser() ...	3

Python 3

Qu	Part	Marking Guidance	Marks

4	20	`# Question 4`
		`if __name__ == "__main__":`
		`  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
		`for Count in range(1, 14):`
		`  print('Please enter next digit of ISBN: '),`
		`  ISBN[Count] = int(input())`
		`  CalculatedDigit = 0`
		`  Count = 1`
		`  while Count < 13:`
		`  CalculatedDigit = CalculatedDigit + ISBN[Count]`
		`  Count = Count + 1`
		`  CalculatedDigit = CalculatedDigit + ISBN[Count] * 3`
		`  Count = Count + 1`
		`  while CalculatedDigit >= 10:`
		`  CalculatedDigit = CalculatedDigit - 10`
		`  CalculatedDigit = 10 - CalculatedDigit`
		`  if CalculatedDigit == 10 :`
		`  CalculatedDigit = 0`
		`  if CalculatedDigit == ISBN[13]:`
		`  print('Valid ISBN')`
		`  else:`
		`  print('Invalid ISBN')`
		
		Alternative print/input combination:
		
		`ISBN[Count] = int(input('Please enter next digit of ISBN: ',))`
			15

6	32	`...`
		`print()`
		`PlayerName = ''`
		`while len(PlayerName) == 0:`
		`  print('Please enter your name: '),`
		`  PlayerName = input()`
		`  if len(PlayerName) == 0:`
		`  print('You must enter a name')`
		`print()`
		
		Alternative Answer:
		
		`...`
		`print()`
		`PlayerName = ''`
		`while PlayerName == '':`
		`  print('Please enter your name: '),`
		`  PlayerName = input()`
		`  if PlayerName == '':`
			4

		print('You must enter a name') print() ...	

7	34	def IsNextCardHigher(LastCard, NextCard):
		  Higher = False
		  if NextCard.Rank > LastCard.Rank:
		  Higher = True
		  if NextCard.Rank == LastCard.Rank:
		  if NextCard.Suit > LastCard.Suit:
		  Higher = True
		  return Higher
		
		Alternative answer:
		
		def IsNextCardHigher(LastCard, NextCard):
		  Higher = False
		  if NextCard.Rank > LastCard.Rank:
		  Higher = True
		  else:
		  if NextCard.Rank < LastCard.Rank:
		  Higher = False
		  else:
		  if NextCard.Suit > LastCard.Suit:
		  Higher = True
		  else:
		  Higher = False
		  return Higher
		
		Alternative answer
		
		def IsNextCardHigher(LastCard, NextCard):
		  Higher = False
		  if NextCard.Rank > LastCard.Rank:
		  Higher = True
		  elif NextCard.Rank == LastCard.Rank and NextCard.Suit > LastCard.Suit:
		  Higher = True
		  return Higher
			4

8	36	...
		  print('Do you think the next card will be higher than the last card (enter y or n **or j to play a joker**)? ')
		  Choice = input()
		...
			1

8	37	... **NoOfJokers = 2** GameOver = False	7

```
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and
((Choice != 'j') or (NoOfJokers == 0)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers - 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and
Choice == 'n') or (Choice == 'j'):
...
```

A. Equivalent logic for condition (eg `NoOfJokers < 1`)

Alternative answer:

```
...
NoOfJokers = 0
GameOver = False
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and
((Choice != 'j') or (NoOfJokers == 2)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers + 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and
Choice == 'n') or (Choice == 'j'):
...
```

A. Equivalent logic for condition (eg `NoOfJokers > 1`)

9	39	`def CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard):`
		`  NoOfCardsHigher = 0`
		`  NoOfCardsLower = 0`
		`  for Count in range(1, 53 – NoOfCardsTurnedOver):`
		`  if (IsNextCardHigher(LastCard, Deck[Count]):`
		`  NoOfCardsHigher = NoOfCardsHigher + 1`
		`  else:`
			11

		NoOfCardsLower = NoOfCardsLower + 1
		Probability = NoOfCardsHigher / (NoOfCardsHigher + NoOfCardsLower)
		return Probability
		
		Alternative answer:
		
		...
		for Count in range(1, 53 – NoOfCardsTurnedOver):
		if (IsNextCardHigher(LastCard, Deck[Count]):
		NoOfCardsHigher = NoOfCardsHigher + 1
		Probability = NoOfCardsHigher / (52 – NoOfCardsTurnedOver)
		return Probability
		...
		
		Alternative answer:
		
		def CalculateProbability(Deck, LastCard):
		NoOfCardsHigher = 0
		Count = 1
		while (Count < 52) and (Deck[Count].Suit != 0):
		if (IsNextCardHigher(LastCard, Deck[Count]):
		NoOfCardsHigher = NoOfCardsHigher + 1
		Count = Count + 1
		Probability = NoOfCardsHigher / (Count - 1)
		return Probability
		
		A. Deck[Count].Rank **instead of** Deck[Count].Suit	

9	40	... **print('The probability of the next card being higher is %3.2f' %CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard))** GetCard(NextCard, Deck, NoOfCardsTurnedOver) Choice = '' while (Choice != 'y') and (Choice != 'n'): Choice = GetChoiceFromUser() ...	3

Java

Qu	Part	Marking Guidance	Marks

4	20	`public class Question4 {`
		`  AQAConsole2014 console = new AQAConsole2014();`
		
		`  public Question4() {`
		`  int ISBN[] = new int[14];`
		`  int count;`
		`  int calculatedDigit;`
		`  for (count = 1; count <= 13; count++) {`
		`  ISBN[count] = console.readInteger("Please enter next digit of ISBN: ");`
		`  }`
		`  calculatedDigit = 0;`
		`  count = 1;`
		`  while (count < 13) {`
		`  calculatedDigit = calculatedDigit + ISBN[count];`
		`  count++;`
		`  calculatedDigit = calculatedDigit + ISBN[count] * 3;`
		`  count++;`
		`  }`
		`  while (calculatedDigit >= 10) {`
		`  calculatedDigit = calculatedDigit - 10;`
		`  }`
		`  calculatedDigit = 10 - calculatedDigit;`
		`  if (calculatedDigit == 10) {`
		`  calculatedDigit = 0;`
		`  }`
		`  if (calculatedDigit == ISBN[13]) {`
		`  console.println("Valid ISBN");`
		`  } else {`
		`  console.println("Invalid ISBN");`
		`  }`
		`  }`
		
		`  public static void main(String[] args) {`
		`  new Question4();`
		`  }`
		`}`
			15

6	32	`...`
		`console.println();`
		`do {`
		`  playerName = console.readLine("Please enter your name: ");`
		`  if (playerName.length() == 0) {`
		`  console.println("You must enter a name");`
		`  }`
		`} while (playerName.length() == 0);`
			4

		console.println();
		...
		
		Alternative answer:
		
		...
		console.println();
		do {
		  playerName = console.readLine("Please enter your name: ");
		  if (playerName.equals("") {
		  console.println("You must enter a name");
		  }
		} while (playerName.equal(""));
		console.println();
		...	

7	34	boolean isNextCardHigher(TCard lastCard, TCard nextCard){
		  boolean higher;
		  higher = false;
		  console.println(lastCard.rank);
		  console.println(nextCard.rank);
		  if (nextCard.rank > lastCard.rank){
		  higher = true;
		  }
		  if (nextCard.rank == lastCard.rank) {
		  if (nextCard.suit > lastCard.suit) {
		  higher = true;
		  }
		  }
		  return higher;
		}
		
		Alternative answer
		
		boolean isNextCardHigher(TCard lastCard, TCard nextCard){
		  boolean higher;
		  higher = false;
		  console.println(lastCard.rank);
		  console.println(nextCard.rank);
		  if (nextCard.rank > lastCard.rank){
		  higher = true;
		  } **else if (lastCard.rank == nextCard.rank) {**
		  if (nextCard.suit > lastCard.suit) {
		  higher = true;
		  }
		  }
		  return higher;
		}
			4

		Alternative answer
		`boolean isNextCardHigher(TCard lastCard, TCard nextCard){`
		`  boolean higher;`
		`  higher = false;`
		`  console.println(lastCard.rank);`
		`  console.println(nextCard.rank);`
		`  if (nextCard.rank > lastCard.rank){`
		`  higher = true;`
		`  }`
		`  **if (nextCard.rank == lastCard.rank) && (nextCard.suit > lastCard.suit) {**`
		`  **higher = true;**`
		`  **}**`
		`  return higher;`
		`}`	

8	36	`...`
		`choice = console.readChar("Do you think the next card will be higher than the last card (enter y or n **or j to play a joker**)? ");`
		`...`
			1

8	37	`...`
		`char choice;`
		`**int noOfJokers;**`
		`...`
		`**noOfJokers = 2;**`
		`gameOver = false;`
		`...`
		`while (noOfCardsTurnedOver < 52 && !gameOver) {`
		`  getCard(nextCard, deck, noOfCardsTurnedOver);`
		`  do {`
		`  choice = getChoiceFromUser();`
		`  } while (!(choice == 'y' || choice == 'n' **|| choice == 'j' && noOfJokers != 0**));`
		`**if (choice == 'j') {**`
		`  **noOfJokers = noOfJokers - 1;**`
		`  **}**`
		`  displayCard(nextCard);`
		`  noOfCardsTurnedOver = noOfCardsTurnedOver + 1;`
		`  higher = isNextCardHigher(lastCard, nextCard);`
		`  if (higher && choice =='y' || !higher && choice=='n' **|| choice == 'j'**) {`
		`...`
		
		A. Equivalent logic for condition (eg `NoOfJokers > 0`)
		
		Alternative answer:
			7

		...
		char choice;
		int noOfJokers;
		...
		noOfJokers = 0;
		gameOver = false;
		...
		while (noOfCardsTurnedOver < 52 && !gameOver) {
		  getCard(nextCard, deck, noOfCardsTurnedOver);
		  do {
		  choice = getChoiceFromUser();
		  } while (!(choice == 'y' || choice == **'n' || choice == 'j' && noOfJokers != 2**));
		  if (choice == 'j') {
		  noOfJokers = noOfJokers + 1;
		  }
		  displayCard(nextCard);
		  noOfCardsTurnedOver = noOfCardsTurnedOver + 1;
		  higher = isNextCardHigher(lastCard, nextCard);
		  if (higher && choice =='y' || !higher && choice=='n' **|| choice == 'j'**) {
		...
		
		A. Equivalent logic for condition (eg `NoOfJokers < 2`)	

9	39	float calculateProbability(TCard[] deck, int noOfCardsTurnedOver, TCard lastCard) {
		  int noOfCardsHigher;
		  int noOfCardsLower;
		  float probability;
		  noOfCardsHigher = 0;
		  noOfCardsLower = 0;
		  for (int count = 1; count <= 52 - noOfCardsTurnedOver; count++) {
		  if (isNextCardHigher(lastCard, deck[count])) {
		  noOfCardsHigher = noOfCardsHigher + 1;
		  } else {
		  noOfCardsLower = noOfCardsLower + 1;
		  }
		  }
		  probability = (float) noOfCardsHigher / (noOfCardsHigher + noOfCardsLower);
		  return probability;
		}
		
		Alternative answer:
		
		float calculateProbability(TCard[] deck, int noOfCardsTurnedOver, TCard lastCard) {
		  int noOfCardsHigher;
			11

		float probability;
		noOfCardsHigher = 0;
		for (int count = 1; count <= 52 - noOfCardsTurnedOver; count++) {
		if (isNextCardHigher(lastCard, deck[count])) {
		noOfCardsHigher = noOfCardsHigher + 1;
		}
		}
		probability = (float) noOfCardsHigher / (52 - noOfCardsTurnedOver);
		return probability;
		}
		
		Alternative answer:
		
		float calculateProbability(TCard[] deck, int noOfCardsTurnedOver, TCard lastCard) {
		int noOfCardsHigher;
		float probability;
		int count;
		count = 1;
		noOfCardsHigher = 0;
		while (count < 52 && deck[count].suit != 0) {
		if (isNextCardHigher(lastCard, deck[count])) {
		noOfCardsHigher = noOfCardsHigher + 1;
		}
		count = count + 1;
		}
		probability = (float) noOfCardsHigher / (count - 1);
		return probability;
		}
		
		A. deck[count].rank **instead of** deck[count].suit	

9	40	... **console.print("The probability of the next card being higher is ");** **console.println(calculateProbability(deck, noOfCardsTurnedOver, lastCard));** getCard(nextCard, deck, noOfCardsTurnedOver); ...	3