

INTERNATIONAL GCSE **COMPUTER SCIENCE**

9210/1

Paper 1 Programming

Mark scheme

November 2025

Version: 1.0 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 Examiner.

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 www.oxfordaqa.com

Copyright information

OxfordAQA retains the copyright on all its publications. However, registered schools/colleges for OxfordAQA are permitted to copy material from this booklet for their own internal use, with the following important exception: OxfordAQA cannot give permission to schools/colleges to photocopy any material that is acknowledged to a third party even for internal use within the centre.

Copyright © 2025 OxfordAQA International Examinations and its licensors. All rights reserved.

How to mark

Aims

When you are marking your allocation of scripts your main aims should be to:

- recognise and identify the achievements of students
- where relevant, place students in the appropriate mark band and in the appropriate part of that mark band (high, low, middle) for **each** assessment objective
- record your judgements with brief notes, annotations and comments that are relevant to the mark scheme and make it clear to other associates how you have arrived at the numerical mark awarded for each assessment objective
- ensure comparability of assessment for all students, regardless of question or examiner.

Approach

It is important to be **open-minded** and **positive** when marking scripts.

The specification recognises the variety of experiences and knowledge that students will have. It encourages them to study computer science in a way that is relevant to them. The questions have been designed to give them opportunities to discuss what they have found out about computer science. It is important to assess the quality of **what the student offers**.

Do not mark scripts based on the answer **you** would have written. The mark schemes have been composed to assess **quality of response** and not to identify expected items of knowledge.

Assessment Objectives

This component requires students to:

AO1: Demonstrate knowledge and understanding of the key concepts and principles of computer science.

AO2: Apply knowledge and understanding of key concepts and principles of computer science.

AO3: Analyse problems in computational terms in order to develop and test programmed solutions and demonstrate an understanding of programming concepts.

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 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 result in a candidate losing only 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.

Section A

Question	Part	Marking guidance	Total marks
01	1	<code>displayMenu // main;</code>	1 AO3=1

Question	Part	Marking guidance	Total marks
01	2	<code>drawStreet // playGame;</code>	1 AO3=1

Question	Part	Marking guidance	Total marks
01	3	Code can be (easily) reused; Easier to test/debug; Easier to update/maintain; Easier (for people) to understand; R. if implication is that it is easier for computer Code can be divided among a team; Lower likelihood of errors; Max 3	3 AO3=3

Question	Part	Marking guidance	Total marks
01	4	To allow play to continue indefinitely // to allow a game of any length to be played; We do not know how many times the code will (need to) repeat; FOR loops are (only) used when the number of iterations is known; Max 1	1 AO3=1

Question	Part	Marking guidance	Total marks
01	5	Check whether the last place is being processed; (and if not) draw the (vertical) line; Alternative answer To avoid printing a vertical line; after the last place;	2 AO3=2

Question	Part	Marking guidance	Total marks
01	6	While the place chosen is the (same as the) start place; choose a new (random) position; So that the pick-up and drop-off locations cannot be the same;; Max 2	2 AO3=2

Question	Part	Marking guidance	Total marks
02	1	Only lower bids / values can overwrite <code>winningBid</code> / result in <code>winningIndex</code> changing // the check is for less than;	1 AO3=1

Question	Part	Marking guidance	Total marks
02	2	Player 2/last player would win (in the event of a tie);	1 AO3=1

Question	Part	Marking guidance	Total marks
02	3	Using an IF statement to check for equal bids; Generate a random player/winner // alternate between players using a variable/Boolean/int to track which player's turn;	2 AO3=2

Question	Part	Marking guidance	Total marks						
03	1	<table><tr><td>C</td><td>A. iteration</td></tr><tr><td>B</td><td>A. declaration</td></tr><tr><td>A</td><td>A. constant</td></tr></table>	C	A. iteration	B	A. declaration	A	A. constant	2 AO3=2
		C	A. iteration						
		B	A. declaration						
A	A. constant								
1 mark one or two labels are correct or 2 marks all labels are correct									
R. a label used more than once									

Question	Part	Marking guidance	Total marks
03	2	<code>line = "" //</code> <code>selection = "";</code>	1 AO3=1

Question	Part	Marking guidance	Total marks
----------	------	------------------	-------------

03	3	A (variable/subroutine) name that identifies its purpose;	1 AO3=1
----	---	---	----------------

Question	Part	Marking guidance	Total marks
----------	------	------------------	-------------

04	1	<table><tr><th>winningBid</th><th>winningIndex</th><th>p</th></tr><tr><td>5</td><td>0</td><td>1</td></tr><tr><td>4</td><td>1</td><td>2</td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table> 1 mark winningBid column correct 1 mark winningIndex column correct 1 mark p column correct I. Different rows used so long as the order within columns is clear I. Unnecessary repetition of values Max 2 if any errors	winningBid	winningIndex	p	5	0	1	4	1	2							3 AO3=3
winningBid	winningIndex	p																
5	0	1																
4	1	2																

Question	Part	Marking guidance	Total marks
----------	------	------------------	-------------

04	2	Player 2 wins the job; I. minor spelling errors and incorrect case	1 AO3=1
----	---	---	----------------

Section B

Question	Part	Marking guidance	Total marks
05	1	1 mark outputting text The bid was I. minor errors 1 mark output includes the winning bid 1 mark new output between previous output and blank line Max 2 marks if any errors Python <pre>def processBids(places, players, pickUp, dropOff): winningBid = players[0].bid winningIndex = 0 for p in range(1, len(players)): if players[p].bid < winningBid: winningBid = players[p].bid winningIndex = p print(f"Player {winningIndex + 1} wins the job") print(f"The bid was {winningBid}") print() distanceToPickup = abs(players[winningIndex].position - places.index(pickUp)) distanceBetweenLocations = abs(places.index(pickUp) - places.index(dropOff)) cost = distanceToPickup + distanceBetweenLocations players[winningIndex].balance = players[winningIndex].balance + players[winningIndex].bid - cost players[winningIndex].position = places.index(dropOff)</pre> C# <pre>static void processBids(List<string> places, List<PlayerData> players, string pickUp, string dropOff) { int winningBid = players[0].Bid; int winningIndex = 0; for (int p = 1; p < players.Count; p++) { if (players[p].Bid < winningBid) { winningBid = players[p].Bid; winningIndex = p;</pre>	3 AO3=3

		<pre> } } Console.WriteLine(\$"Player {winningIndex + 1} wins the job"); Console.WriteLine(\$"The bid was {winningBid}"); Console.WriteLine(); int distanceToPickup = Math.Abs(players[winningIndex].Position - places.IndexOf(pickUp)); int distanceBetweenLocations = Math.Abs(places.IndexOf(pickUp) - places.IndexOf(dropOff)); int cost = distanceToPickup + distanceBetweenLocations; players[winningIndex].Balance = players[winningIndex].Balance + players[winningIndex].Bid - cost; players[winningIndex].Position = places.IndexOf(dropOff); } </pre> VB <pre> Sub processBids(places As List(Of String), players As List(Of PlayerData), pickUp As String, dropOff As String) Dim winningBid As Integer = players(0).Bid Dim winningIndex As Integer = 0 For p = 1 To players.Count - 1 If players(p).Bid < winningBid Then winningBid = players(p).Bid winningIndex = p End If Next Console.WriteLine(\$"Player {winningIndex + 1} wins the job") Console.WriteLine(\$"The bid was {winningBid}"); Console.WriteLine() Dim distanceToPickup As Integer = Math.Abs(players(winningIndex).Position - places.IndexOf(pickUp)) Dim distanceBetweenLocations As Integer = Math.Abs(places.IndexOf(pickUp) - places.IndexOf(dropOff)) Dim cost As Integer = distanceToPickup + distanceBetweenLocations players(winningIndex).Balance = players(winningIndex).Balance + players(winningIndex).Bid - cost </pre>	
--	--	---	--

		<pre> players(winningIndex).Position = places.IndexOf(dropOff) End Sub </pre>	
--	--	---	--

Question	Part	Marking guidance	Total marks
05	2	Screen capture(s) must match code from Q5.1 1 mark output contains the winning player followed by the winning bid, which should be 4 and should belong to whichever player was prompted first. Note that places can vary. Transport a package from the Cafe to the Gym Player 2's bid: 4 Player 1's bid: 5 Player 2 wins the job The bid was 4	1 AO3=1

Question	Part	Marking guidance	Total marks
06	1	1 mark variable to store building number // use of the index of places when generating an address 1 mark variable declared outside outer loop // index of places used to produce output next to all places 1 mark output string appended to at appropriate place in the code 1 mark output string correctly formatted, including spaces between name and opening bracket, and between building number and North Street I. minor spelling error, R. missing or wrongly-placed parenthesis 1 mark variable increased by 2 // index of places multiplied by 2 Max 4 marks if any errors Python <pre>def drawStreet(places, players): number = 1 for place in range(len(places)): line = "" for p in range(len(players)): if players[p].position == place: line = line + str(p + 1) + " " else: line += " " line = line + " " + places[place] + f" ({number} North Street)" number = number + 2 print(line) if place < len(places) - 1: print(" ") print()</pre> C# <pre>static void drawStreet(List<string> places, List<PlayerData> players) { int number = 1; for (int place = 0; place < places.Count; place++) { string line = ""; for (int p = 0; p < players.Count; p++) { if (players[p].position == place) { line = line + (p + 1).ToString() + " "; } } } }</pre>	5 AO3=5

		<pre> else { line = line + " "; } line = line + " " + places[place] + \$" ({number} North Street)"; number = number + 2; Console.WriteLine(line); if (place < places.Count - 1) { Console.WriteLine(" "); } Console.WriteLine(); } </pre> VB <pre> Sub drawStreet(places As List(Of String), players As List(Of PlayerData)) Dim number As Integer = 1 For place = 0 To places.Count - 1 Dim line As String = "" For p = 0 To players.Count - 1 If players(p).position = place Then line = line & (p + 1).ToString() & " " Else line = line & " " End If Next line = line & " " & places(place) & \$" ({number} North Street)" number = number + 2 Console.WriteLine(line) If place < places.Count - 1 Then Console.WriteLine(" ") End If Next Console.WriteLine() End Sub </pre>	
--	--	---	--

Question	Part	Marking guidance	Total marks
06	2	Screen capture(s) must match code from Q6.1 1 mark correct address displayed for each place. <pre> 1 Gym (1 North Street) Cafe (3 North Street) Office (5 North Street) Garage (7 North Street) Station (9 North Street) Market (11 North Street) 2 School (13 North Street) </pre>	1 AO3=1

Question	Part	Marking guidance	Total marks
07	1	1 mark indefinite iteration structure created 1 mark condition of iteration structure correct 1 mark correct message for an invalid bid A. incorrect condition, I. minor errors 1 mark error message does not display for valid input, I. minor errors, R. if no error message ever displayed 1 mark bid correctly stored if it is valid on first attempt 1 mark bid correctly stored if valid on subsequent attempts 1 mark code would work for any number of places Max 6 marks if any errors Python <pre>def collectBids(places, players): for p in range(len(players)): players[p].bid = -1 bidCount = 0 while bidCount < len(players): bidder = random.randint(0, len(players) - 1) if players[bidder].bid == -1: players[bidder].bid = int(input(f"Player {bidder + 1}'s bid: ")) while players[bidder].bid > len(places) * 2: print("Bid too high") players[bidder].bid = int(input(f"Player {bidder + 1}'s bid: ")) bidCount = bidCount + 1</pre> C# <pre>static void collectBids(List<string> places, List<PlayerData> players) { for (int p = 0; p < players.Count; p++) { players[p].bid = -1; } int bidCount = 0; Random rnd = new Random(); while (bidCount < players.Count) { int bidder = rnd.Next(0, players.Count); if (players[bidder].bid == -1) { Console.Write(\$"Player {bidder + 1}'s bid: ");</pre>	7 AO3=7

		<pre> players[bidder].bid = int.Parse(Console.ReadLine()); while (players[bidder].bid > places.Count * 2) { Console.WriteLine("Bid Too High"); Console.Write(\$"Player {bidder + 1}'s bid: "); players[bidder].bid = int.Parse(Console.ReadLine()); } bidCount = bidCount + 1; } } VB Sub collectBids(places As List(Of String), players As List(Of PlayerData)) For p = 0 To players.Count - 1 players(p).bid = -1 Next Dim bidCount As Integer = 0 Dim rnd As New Random() While bidCount < players.Count Dim bidder As Integer = rnd.Next(0, players.Count) If players(bidder).bid = -1 Then Console.Write(\$"Player {bidder + 1}'s bid: ") players(bidder).bid = Integer.Parse(Console.ReadLine()) While players(bidder).bid > places.Count * 2 Console.WriteLine("Bid Too High") Console.Write(\$"Player {bidder + 1}'s bid: ") players(bidder).bid = Integer.Parse(Console.ReadLine()) End While bidCount = bidCount + 1 End If End While End Sub </pre>	
--	--	--	--

Question	Part	Marking guidance	Total marks
07	2	Screen capture(s) must match code from Q7.1 1 mark prompt for one player is repeated for an input of 15, and prompt for the other player is given after 14 is entered. Prompt for the other player is also repeated for an input of 15 Note that places and player order can vary. <pre> Transport a package from the Cafe to the School Player 1's bid: 15 Bid too high Player 1's bid: 14 Player 2's bid: 15 Bid too high Player 2's bid: </pre>	1 AO3=1

Question	Part	Marking guidance	Total marks
08	1	1 mark fuel field within player data structure modified 1 mark fuel field modified correctly 1 mark check if fuel field greater than or equal to 1 // check if fuel field less than or equal to 5 1 mark combine both conditions correctly 1 mark check for fuel field being less than or equal to zero 1 mark out of fuel or low on fuel message displayed 1. minor errors 1 mark at least one message includes correct player number Max 6 marks if any errors Python <pre>def processBids(places, players, pickUp, dropOff): ... players[winningIndex].position = places.index(dropOff) players[winningIndex].fuel -= cost if players[winningIndex].fuel in range(1, 6): print(f"Player {winningIndex + 1} is low on fuel") if players[winningIndex].fuel <= 0: print(f"Player {winningIndex + 1} is out of fuel")</pre> C# <pre>static void processBids(List<string> places, List<PlayerData> players, string pickUp, string dropOff) { ... players[winningIndex].position = places.IndexOf(dropOff); players[winningIndex].fuel -= cost; if (players[winningIndex].fuel >= 1 && players[winningIndex].fuel <= 5) { Console.WriteLine(\$"Player {winningIndex + 1} is low on fuel"); } else if (players[winningIndex].fuel <= 0) { Console.WriteLine(\$"Player {winningIndex + 1} is out of fuel"); } }</pre>	7 AO3=7

		<pre>VB Sub processBids(places As List(Of String), players As List(Of PlayerData), pickup As String, dropOff As String) ... players(winningIndex).position = places.IndexOf(dropOff) players(winningIndex).fuel -= cost If players(winningIndex).fuel >= 1 And players(winningIndex).fuel <= 5 Then Console.WriteLine(\$"Player {winningIndex + 1} is low on fuel") ElseIf (players(winningIndex).fuel <= 0) Then Console.WriteLine(\$"Player {winningIndex + 1} is out of fuel") End If End Sub</pre>	
--	--	---	--

Question	Part	Marking guidance	Total marks
08	2	1 mark selection structure to check for a player's fuel being (less than or) equal to zero 1 mark iteration set to terminate within selection structure, i.e. incorrect selection structure provided that clear attempt made to check for no fuel. Max 1 marks if any errors Python <pre>def playGame(places, players): ... for p in range(len(players)): if players[p].fuel <= 0: playing = False if players[p].balance >= winningBalance: print(f"Player {p + 1} has won") playing = False</pre> C# <pre>static void playGame(List<string> places, List<PlayerData> players) { ... for (int p = 0; p < players.Count; p++) { if (players[p].fuel <= 0) { playing = false; } if (players[p].balance >= winningBalance) { Console.WriteLine(\$"Player {p + 1} has won"); playing = false; } } }</pre> VB <pre>Sub playGame(places As List(Of String), players As List(Of PlayerData)) ... For p = 0 To players.Count - 1 If players(p).fuel <= 0 Then playing = False End If</pre>	2 AO3=2

		<pre> If players(p).balance >= winningBalance Then Console.WriteLine(\$"Player {p + 1} has won") playing = False End If Next End While End Sub </pre>	
--	--	--	--

Question	Part	Marking guidance	Total marks
08	3	Screen capture(s) must match code from Q8.1 1 mark player 1 having a fuel value less than 13, then being out of fuel and the game ending. <pre> Player 1 - Balance: 56 - Fuel: 1 Player 2 - Balance: 50 - Fuel: 30 Gym 1 Cafe Office Garage Station Market 2 School Transport a package from the Garage to the School Player 1's bid: 5 Player 2's bid: 5 Player 1 wins the job Player 1 is out of fuel </pre>	1 AO3=1

Section C

Question	Part	Marking guidance	Total marks
09	1	1 mark new subroutine with correct name and parameters 1 mark loop that runs twice based on current code for (the first) two players 1 mark loop that would always run once per player, regardless of the number of players 1 mark prompt <code>move to?</code> displays for all players 1 mark input stored in a variable 1 mark check for empty/non-empty string 1 mark check for specified place existing in <code>places</code> 1 mark second check only occurs for a non-empty string 1 mark get index of place 1 mark only move player if place exists 1 mark change position of correct player 1 mark output <code>No such place</code> under correct circumstances, i. minor errors Max 11 marks if any errors Python <pre>def movePlayers(places, players): for player in range(len(players)): moveTo = input(f"Player {player + 1}, move to? ") if moveTo != "": found = False for i in range(len(places)): if places[i] == moveTo: position = i found = True if found: players[player].position = position else: print("No such place")</pre> Alternative answer <pre>def movePlayers(places, players): for player in range(len(players)): moveTo = input(f"Player {player + 1}, move to? ") if moveTo != "": if moveTo in places: players[player].position = places.index(moveTo) else: print("No such place")</pre> C#	12 AO3=12

		<pre> static void movePlayers(List<string> places, List<PlayerData> players) { for (int player = 0; player < players.Count; player++) { Console.Write(\$"Player {player + 1}, move to?"); string moveTo = Console.ReadLine(); if (moveTo != "") { int position = -1; bool found = false; for (int i = 0; i < places.Count; i++) { if (places[i] == moveTo) { position = i; found = true; } } if (found) { players[player].position = position; } else { Console.WriteLine("No such place"); } } } } </pre> VB <pre> Sub movePlayers(places As List(Of String), players As List(Of PlayerData)) For player = 0 To players.Count - 1 Console.Write(\$"Player {player + 1}, move to?") Dim moveTo As String = Console.ReadLine() If moveTo <> "" Then Dim found As Boolean = False Dim position As Integer = -1 </pre>	
--	--	---	--

		<pre> For place = 0 To places.Count - 1 If places(place) = moveTo Then position = place found = True End If Next If found Then players(player).position = position Else Console.WriteLine("No Such Place") End If End If Next End Sub </pre>	
--	--	---	--

Question	Part	Marking guidance	Total marks
09	2	1 mark valid call to drawStreet after processBids 1 mark valid call to movePlayers after drawStreet Max 1 mark if any errors Python processBids(places, players, pickUp, dropOff) drawStreet(places, players) movePlayers(places, players) C# processBids(places, players, pickUp, dropOff); drawStreet(places, players); movePlayers(places, players); VB processBids(places, players, pickUp, dropOff) drawStreet(places, players) movePlayers(places, players)	2 AO3=2

Question	Part	Marking guidance	Total marks
09	3	Screen capture(s) must match code from Q9.1 & Q9.2 1 mark inputs are blank and Cafe, with player 2 at the Cafe. Note that player 1's balance may vary. <pre> Player 2's bid: 5 Player 1's bid: 5 Player 1 wins the job The bid was 5 Gym (1 North Street) 1 Cafe (3 North Street) Office (5 North Street) Garage (7 North Street) Station (9 North Street) Market (11 North Street) 2 School (13 North Street) Player 1, move to? Player 2, move to? Cafe Player 1 - Balance: 50 - Fuel: 30 Player 2 - Balance: 50 - Fuel: 30 Gym (1 North Street) 1 2 Cafe (3 North Street) Office (5 North Street) Garage (7 North Street) Station (9 North Street) Market (11 North Street) School (13 North Street) </pre>	1 A03=1

Question	Part	Marking guidance	Total marks
10	1	1 mark subroutine declared with correct name and parameters 1 mark both <code>pickUp</code> and <code>dropOff</code> converted to indices 1 mark calculation includes either the difference between <code>current</code> and <code>pickUp</code> or <code>pickUp</code> and <code>dropOff</code> 1 mark calculation includes absolute value or equivalent 1 mark correct calculation of the whole distance 1 mark calculated value returned, 1 incorrect calculation as long as subtraction including at least two of <code>current</code>, index of <code>pickUp</code>, index of <code>dropOff</code>, is included Max 5 marks if any errors Python <pre>def getLowestBid(places, current, pickUp, dropOff): startIndex = places.index(pickUp) endIndex = places.index(dropOff) out = abs(current - startIndex) back = abs(startIndex - endIndex) lowestBid = out + back return lowestBid</pre> C# <pre>static int getLowestBid(List<string> places, int current, string pickUp, string dropOff) { int startIndex = places.IndexOf(pickUp); int endIndex = places.IndexOf(dropOff); int there = Math.Abs(current - startIndex); int back = Math.Abs(startIndex - endIndex); int lowestBid = there + back; return lowestBid; }</pre> VB <pre>Function getLowestBid(places As List(Of String), current As Integer, pickUp As String, dropOff As String) As Integer Dim startIndex As Integer = places.IndexOf(pickUp) Dim endIndex As Integer = places.IndexOf(dropOff) Dim out As Integer = Math.Abs(current - startIndex)</pre>	6 AO3=6

		<pre> Dim back As Integer = Math.Abs(startIndex - endIndex) Dim lowestBid As Integer = out + back Return lowestBid End Function </pre>	
--	--	--	--

Question	Part	Marking guidance	Total marks
----------	------	------------------	-------------

10	2	1 mark valid call to <code>getLowestBid</code> 1 mark second valid call to <code>getLowestBid</code> // loop that references all indices of <code>players</code> 1 mark returns from both/all calls stored in a list 1 mark list/array passed to <code>collectBids</code> Max 3 marks if any errors Python <pre> print(f"Transport a package from the {pickUp} to the {dropOff}") lowestBids = [] for p in range(len(players)): lowestBids.append(getLowestBid(places, players[p].position, pickUp, dropOff)) collectBids(places, players, lowestBids) processBids(places, players, pickUp, dropOff) </pre> C# <pre> Console.WriteLine(\$"Transport a package from the {pickUp} to the {dropOff}"); List<int> lowestBids = new List<int>(); for (int p = 0; p < players.Count; p++) { lowestBids.Add(getLowestBid(places, players[p].position, pickUp, dropOff)); } collectBids(places, players, lowestBids); processBids(places, players, pickUp, dropOff); </pre> VB <pre> Console.WriteLine(\$"Transport a package from the {pickUp} to the {dropOff}") Dim lowestBids As New List(Of Integer) For p = 0 To players.Count - 1 </pre>	4 AO3=4
----	---	---	-------------------------------------

		<pre>lowestBids.Add(getLowestBid(places, players(p).position, pickUp, dropOff)) Next collectBids(places, players, lowestBids) processBids(places, players, pickUp, dropOff)</pre>	
--	--	--	--

Question	Part	Marking guidance	Total marks
10	3	1 mark lowestBids included as a parameter in same position as in Q10.2 1 mark output including Minimum for player in appropriate place in code, I. minor errors 1 mark output includes correct player number 1 mark output includes correct minimum A. minimum from correct index in list/array Max 3 marks if any errors Python <pre>def collectBids(places, players, lowestBids): for p in range(len(players)): players[p].bid = -1 bidCount = 0 while bidCount < len(players): bidder = random.randint(0, len(players) - 1) if players[bidder].bid == -1: print(f"Minimum for player {bidder + 1}: {lowestBids[bidder]}") players[bidder].bid = int(input(f"Player {bidder + 1}'s bid: ")) bidCount = bidCount + 1</pre> C# <pre>static void collectBids(List<string> places, List<PlayerData> players, List<int> lowestBids) { for (int p = 0; p < players.Count; p++) { players[p].bid = -1; } int bidCount = 0; while (bidCount < players.Count) { int bidder = rnd.Next(0, players.Count); if (players[bidder].bid == -1) { Console.WriteLine(\$"Minimum for Player {bidder + 1}: {lowestBids[bidder]}"); Console.Write(\$"Player {bidder + 1}'s bid: "); players[bidder].bid = Convert.ToInt32(Console.ReadLine()); bidCount = bidCount + 1; } } }</pre>	4 AO3=4

		<pre> } VB Sub collectBids(places As List(Of String), players As List(Of PlayerData), lowestBids As List(Of Integer)) For p = 0 To players.Count - 1 players(p).bid = -1 Next Dim bidCount As Integer = 0 While bidCount < players.Count Dim bidder As Integer = rnd.Next(0, players.Count) If players(bidder).bid = -1 Then Console.WriteLine(\$"Minimum for Player {bidder + 1}: {lowestBids(bidder)}") Console.Write(\$"Player {bidder + 1}'s bid: ") players(bidder).bid = Convert.ToInt32(Console.ReadLine()) bidCount = bidCount + 1 End If End While End Sub </pre>	
--	--	---	--

Question	Part	Marking guidance	Total marks
10	4	Screen capture(s) must match code from Q10.1–Q10.3 1 mark minimum values for both players correctly output the distance from <code>current</code> to <code>pickUp</code> then <code>dropOff</code> locations. Note that values will vary due to random generation of locations. <pre> 1 Gym Cafe Office Garage Station Market 2 School Transport a package from the Station to the Garage Minimum for player 1: 5 Player 1's bid: 5 Minimum for player 2: 3 Player 2's bid: </pre>	1 AO3=1