Mark scheme

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td>1</td><td></td><td></td><td>1 mark per bullet to max 5:

• Suitable logic for class dog declaration
• Suitable logic to define the 4 (private) attributes: name, breed, height, weight
• Suitable logic to declare a <u>public</u> method for constructor (e.g. new or class name)...
• ....taking <u>only</u> 4 different parameters in any order.
• Suitable logic to set the values of each attribute

MP4 - if it is obvious the candidate has used Pythonic syntax: allow the additional 5th parameter representing the object e.g. <code>self, turtle</code> <u>but this must be the first parameter listed.</u>

MP2 - If they have done the above BP2 can be considered implicitly met (Python doesn't require attributes to be declared outside the constructor)</td><td>5</td><td>Mark in a vertical line against each MP.

Ignore data types in attribute names

Allow colon/empty brackets at the end of class def

Example Solutions
<u>Pseudocode style</u>
<pre>class Dog
private name
private breed
private height
private weight
public procedure new
(nameIn,
breedIn, heightIn, weightIn)
name = nameIn
breed = breedIn
height = heightIn
weight = weightIn
endclass</pre>
<u>Java / C# Style</u>
Class Dog

<pre>{
private String name
private String breed
private float height
private float weight

public void Dog(String name,
String
breed, float height, float
weight)
{
this.name = name
this.breed = breed
this.height = height
this.weight = weight
}
}</pre>
<u>Python Style</u>
<pre>class Dog:</pre></td></tr>
</table>

					`def __init__(self, name, breed, height, weight):`
					`self.name = name`
					`self.breed = breed`
					`self.height = height`
					`self.weight = weight`
					
					Examiner's Comments
					
					There were a range of responses to this question with the most common responses being written in pseudocode or python. There was a clear division between candidates who were comfortable writing Object Oriented Programming (OOP) code and class constructors and those that were not.
					
					Less successful candidates with little OOP experience often confused the class declaration with the constructor declaration. A few input the values rather than setting the attributes to the values passed in the parameters.

			Total	**5**	
2			**Mark Band 3-High Level (7-9 marks)** The candidate demonstrates a thorough knowledge and understanding of each of the cultural issues. The material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. *A mark band 3 answer will cover all three points relating to layout, colour and character sets and the cultural impacts of these on the design of the program. The examples will be relevant to the pitch-side program and the needs of the athletes and will go on to evaluate why it is important to*	9	Points may include but **aren't limited to**: **AO1 Knowledge and Understanding**Accessibility, colour blindness etc are not relevant to context. Cultural issues are about how different groups of people with particular beliefs, practices or languages may be affected by something. The layout is an important factor to consider. This will determine where items will be placed on the screen. Where items are placed on the screen can massively impact how easily people can use them. The layout which is suitable for one country may not be suitable for another. Colour should be carefully considered. Humans have their own personal perception of different colours and will use this to determine what it means.

			ensure these considerations are made. **Mark Band 2-Mid Level (4-6 marks)** The candidate demonstrates reasonable knowledge and understanding of most cultural issues; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence. *A mark band 2 answer will cover at least two of layout, colour and character sets and will expand the points, relating these to cultural considerations, the pitch-side program and/or the needs of the athletes although these may not be balanced.* **Mark Band 1-Low Level (1-3 marks)** The candidate demonstrates a basic knowledge of some cultural issues; the material is basic and contains some inaccuracies. The candidate makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides nothing more than an unsupported assertion. The information is basic and communicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear. *A mark band 1 answer will contain some basic, relevant points related to layout, colour and/or character sets. It may not be linked to cultural considerations or the context of the pitch-side program or athlete's needs.*		Colour can be used to confirm messages to the audience, e.g. if our interactions with the software has been successful or not. However, colours can mean different things in different cultures. A character set is a list of characters that can be recognised by the hardware and software. There are various different character sets such as ASCII and UNICODE. **AO2 Application** Western audiences will often read from left to right and top to bottom. However, other countries around the world read from right to left. Athletes are from around the world and this needs to be considered. Traditionally, green often indicates that something is positive or that an interaction with the computer has been successful. Red often indicates something negative or that an interaction with the computer has not been successful. However, these colours may have different meanings and could be seen as offensive in different cultures. Designers therefore need to ensure they don't offend any particular culture as the app is for international athletes. There are many different character sets. The ASCII character set can only represent 128/256 characters which would be suitable for English and European languages. However, this would not be a suitable character set for other world-wide languages and if used, it would mean that characters could not be displayed which would affect the athletes' use of the app. **AO3 Evaluation** As the software is being used by different cultures, it is important that the user interface can be used and understood by all audiences otherwise people may choose not to use the fitness trackers e.g. using icons rather

			0 mark No attempt to answer the question or response is not worthy of credit.		than words could be more universally understood by the athletes. The programmer needs to consider the typical layout, colours and characters used in different cultures and ensure these are taken into consideration to ensure that groups of people are not offended and will still understand the words, data and charts. Designers could release different versions of the software for different parts of the world or make sure the program has the ability for settings to be changed to overcome any potential issues and allow the athletes to customise their experience. **Examiner's Comments** There were a good range of Level 2 responses with good discussions for at least two of the bullet points and related to different cultures. Some candidates were unable to apply their knowledge to different cultures and instead discussed accessibility concerns. The candidates with Level 3 responses gave good discussions of all three bullet points and related it to different cultures and the athletes. They were also able to discuss how the programmer could go about this. **Assessment for learning** Candidates should be aware that in level of response questions application of knowledge to the scenario is needed to get into the Level 2 and Level 3 mark bands.
			Total	**9**	
3	a	i	1 mark per bullet to max 3: • 0 • 1, 2, 4	3	CAO **Examiner's Comments** Most candidates were able to gain at

			• 8 <u>with no numbers after it</u>		least one mark for the first output on this question with many gaining full marks. A small minority of candidates simply copied the outputs given in the question stem.

		ii	1 mark for each correct line.
			
			`START LDA MAX`
			`BRZ END`
			`LDA A`
			`OUT`
			<u>LDA B</u>
			`STA TEMP`
			<u>LDA A</u>
			`ADD B`
			`STA B`
			<u>LDA TEMP</u>
			`STA A`
			`LDA MAX`
			`SUB ONE`
			`STA MAX`
			`BRA START`
			`END HLT`
			`A DAT 0`
			`B DAT 1`
			<u>TEMP</u> `DAT 0`
			`MAX DAT 5`
			`ONE DAT 1`
				4	CAO
					
					Case for mnemonics can be ignored.
					
					Case for A, B, TEMP must be all caps. Penalise first error and allow FT.
					
					<u>Examiner's Comments</u>
					
					Most candidates were able to get the mark for the final answer gap and there was a good range of marks across the cohort.

	b		1 mark per bullet to max 3: • Immediate • Indirect • Indexed	3	DNA direct BOD 'index' for MP3 **<u>Examiner's Comments</u>** Most candidates were able to give indirect as a mode of memory addressing and there were many candidates who gained full marks on this question. A small number of candidates gave responses like by reference or value, and some gave direct which was mentioned in the question stem.
	c		1 mark per bullet to max 2: • **Quicker/more efficient** to <u>translate</u>	2	MP1 must be a comparison e.g. faster/quicker not just fast/quick **<u>Examiner's Comments</u>**

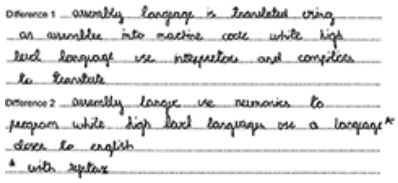
			• Makes more efficient use of the CPU / memory / system resources / where a system may have limited resources • The programmer wants direct control over hardware/memory / to access machine specific functionality • Code might be written for a specific architecture • Compilers/interpreters may not be available		More successful candidates were able to give two valid reasons. Less successful candidates tended to give responses like it is easier to read or easier for the computer to understand. Some candidates were confused between an assembler and compilers/interpreters. The most common correct responses were the programmer wanting direct control over the hardware and it needing to be written for specific hardware.
			Total	**12**	

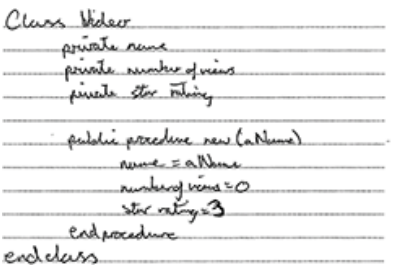
4	a	i	1 mark each
			
			• Defining class Treasure
			• Defining the private attributes value and level
			• Defining a new public procedure…
			• …Taking two parameters (integer and string)
			• Correctly assigning both parameters to the attributes
			
			e.g.
			`class Treasure`
			
			`    private value`
			`    private level`
			
			`    public procedure new(valueP, levelP)`
			`      value = valueP`
			`      level = levelP`
			`    endprocedure`
			`endclass`
				5	Allow use of this/self or equivalent dependent on language
					
					`public procedure new(value, level)`
					`  this.value = value`
					`  this.level = level`
					`endprocedure`
					
					Python answers must either use comments to indicate private attributes or use the double underscore private attribute convention to be credited.
					self.__level
					self.level # private
					
					<u>Examiner's Comments</u>
					
					This question either tended to be very poorly answered or very well answered. There was a clear division between candidates who were comfortable writing OOP code and class constructors and those that were not.
					
					Weaker candidates with little OOP experience often confused the class declaration with the instantiation method or offered no response. Many candidates often assigned the parameters to the private attribute values rather than setting the attributes to the parameters being passed in.
					
					Where candidates gave responses in programming languages and the intent

					of the response was clear, marks were given. However, sometimes Python programmers did not make it clear (by convention/comment) that class attributes were private.

		ii	1 mark each
			• get level method header with no parameter
			• Returning `level` attribute
			
			e.g.
			`public function getLevel()`
			`  return level`
			`endfunction`
				2	Note Python *self* will appear, but no other parameters
					`def getLevel(self):`
					
					Examiner's Comments
					
					While there were many good responses it was also noted that a significant number of candidates erroneously tried to send a parameter to a getter() function and then return the same parameter. Some candidates did not return a value or simply tried to print the value. Other errors included writing the getter method as a function call by omitting any indication that a function was being defined.

		iii	1 mark each • Encapsulation • Allowing an attribute to only be accessed/changed via a method	2	**Examiner's Comments** Some candidates randomly picked an erroneous OOP term like polymorphism. Those candidates who could correctly identify the use of encapsulation did not always go on to specifically state that this meant access was controlled by a public getter() method.
	b		1 mark each to max 4 e.g. • Code can easily be reused… • …classes can be used in other programs • …inheritance can be to extend upon existing classes • …as a class can be based on an existing class • Easier to maintain…. • ….as classes can be modified or extended • …debugging can be easier as encapsulation limits how attibutes are changed. • Code can be more secure… • … as access to attributes can be restricted to being via methods.	4	1 mark per benefit identified and 1 mark per expansion. Max 2 benefits and 1 expansion per benefit. **Examiner's Comments** Many candidates were more successful on this part of Section B as it was an AO1 book learnt topic rather than the preceding AO2 OOP programming application elements. However, benefits were often poorly described, or OOP components were just named without a relevant expansion as to how they could be used.

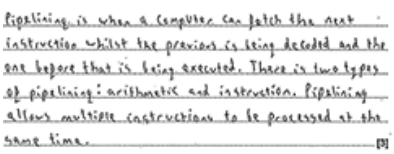
			• Better for coding as part of a team… • …as classes can be distributed between team members.		
			Total	**13**	
5			• Assembly language uses mnemonics • HLL uses English-like words • Assembly language uses an assembler toconvert to machine code • HLL uses a translator (compiler/interpreter) toconvert to machine code • Assembly language is a one-to-one conversionto machine code • HLL may produce multiple lines of machine codeper line of code / one-to-many • Assembly language requires more knowledge ofthe processor / allows direct control of theprocessor • HLL provides more abstraction / requires lessknowledge of the processor • Assembly language is likely to be specific to theprocessor type used / is machine dependent • HLL is portable / can be used for multipleprocessor types / programmer can pick from anumber of HLLs and paradigms / is machineindependent	4	Mark in pairs. Allow examples (e.g. JMP, print) for MP1 and 2 **<u>Examiner's Comments</u>** There was a range of marks candidates could access for this question. The question asked for two differences and those candidates with a good understanding of language paradigms were able to gain full marks by giving the difference for both. Candidates tended to lose marks on this question by only giving one side of the difference, e.g. saying a high level language is translated using an interpreter or compiler but then not giving the other side that assembly language is translated using an assembler. Exemplar 2  Difference 1 assembly language is translated using an assembler into machine code while high level language use interpreters and compilers to translate Difference 2 assembly langage use memonics to program while high level language use a language* closer to english * with syntax In this response the candidate has given two clear differences and has explained what the difference is for both.
			Total	**4**	
6		i	• Class definition with identifier <u>video</u> • name, number of views and star rating attributes defined… • …As private • Constructor method definition <u>inside class definition</u>…	7	Accept implementations in high-level languages (e.g. __ for private, class name used for constructor, no need for end of class definition in Python) BP1 - allow empty brackets. Do not allow anything in the brackets BP5 - ignore self if included as

			• …that accepts only one parameter
			• …Name attribute set to parameter passed in
			• Views set to 0 and rating set to 3 **either** when initialised or in constructor.
					parameter
					
					`class video`
					`  private name`
					`  private views`
					`  private starrating`
					
					`  public procedure new(newname)`
					`  name = NewName`
					`  views = 0`
					`  starrating = 3`
					`  end procedure`
					`end class`
					
					Examiner's Comments
					This question was well answered by some candidates. The question asks for pseudocode or program code and candidates should be encouraged to do one or the other if given a choice, rather than a combination of the two. Many candidates did not use the information in the question stem to help them structure their answer and gave more than one parameter in the constructor.
					
					Exemplar 3
					 
					Class Video
					private name
					private number of views
					private star rating
					
					public procedure new (aName)
					name = aName
					number of views = 0
					star rating = 3
					end procedure
					end class
					
					This was a good clear example of an answer given in pseudocode. The candidate has declared the 3 given attributes as private, shown a constructor with one parameter and set name to the parameter and views and rating to 0 and 3. The candidate gained 7 marks.

		ii	• Method definition that is public
			• View attribute incremented by one
				2	`public procedure updateviews()`
					`  views = views + 1`
					`end procedure`
					
					View attribute must have the same name as part i

					Examiner's Comments Most candidates were able to gain at least 1 mark for this question. Those who were not given marks used pseudocode but did not state that the procedure was public, or they did not use the same attribute they had declared in Question 2 (g) (i).
			Total	**9**	
7		i	• 10 • 60 • 200	3	1 mark per number **Examiner's Comments** This was generally well answered by candidates who had a good understanding of LMC. Candidates should be encouraged to trace through LMC programs with different values as well as writing them.

		ii	• Loads a value into the accumulator
			• Establishes a zero value (by use of DAT / SUB)
			• Stores a <u>zero value</u> into total
			• Program stops
				4	Example 1
					`LDA zero`
					`STA total`
					`HLT`
					`zero DAT 0`
					
					Example 2
					`LDA total`
					`SUB total`
					`STA total`
					`HLT`
					
					BP1 can be given for any value being loaded into the accumulator e.g. INP
					
					If candidate writes LDA donation/total (case sensitive) they can get BP2 as they've used the labels from the question
					
					BP3 - total is case sensitive as given in the question
					
					BP4 - must not be given if the zero value will be attempted to be fetched e.g. HLT is placed after DAT
					
					Examiner's Comments
					This was generally answered well, and the majority of students were able to gain marks with most gaining 3 or 4 marks. Less successful responses over complicated the program leading to them making mistakes. A small

					number of candidates attempted to answer in pseudocode rather than LMC. Candidates should be encouraged to use the commands in Appendix 5d of the specification.
		iii	• One **instruction** can be fetched while another is being decoded… • …and another is executed • The output of one <u>process/instruction</u> is the input of the next. • Concurrent processing of multiple instructions / completing multiple FDE cycles at once	3	For BP1, allow any 2 of the 3 parts of the FDE cycle For BP2, must give the other part of the FDE cycle not given in BP1 Do not award if explaining multiple cores working on different parts of FDE cycle **<u>Examiner's Comments</u>** Many candidates were able to gain at least 2 marks on this question. Some candidates were not awarded marks as they wrote about multiple cores or programs being fetched instead, of instructions. **Exemplar 1**  Pipelining is when a computer can fetch the next instruction whilst the previous is being decoded and the one before that is being executed. There is two types of pipelining: arithmetic and instruction. Pipelining allows multiple instructions to be processed at the same time. [3] The candidate has clearly described pipelining with correct terminology. They gained full marks for the description of one <u>instruction</u> being decoded while another is fetched and another is executed, as well as describing that it allows <u>multiple instructions</u> to be processed <u>at the same time</u>.
		iv	• More <u>instructions</u> can be carried out in a **set amount of time** / **less time** to execute the same number of <u>instructions</u> • Increasing the speed/performance/efficiency of the computer/program / quicker for the program to complete	2	Do not allow "each instruction is quicker to execute". BP2 has to be specific to the charity e.g. processing more donations **<u>Examiner's Comments</u>** Many candidates gained 1 mark for giving a benefit to the charity, but they did not go on to say why pipelining enabled that. Some candidates did not apply their answer to the charity, so were not awarded the mark for the benefit.

			Total	**12**	
8		i	• Object – instantiated from class • Method – action object performs / link to procedure/functions • Attribute – value held by object / link to variable	3 AO1.1	

		ii	• Class definition statement
			• Defining name and attendance attributes
			• Appropriate get methods for name **and** attendance that return a value and have no parameter
			• Appropriate set methods for name **and** attendance that take a parameter
			• …that restricts attendance to be 0 to 100.
				5 AO3.2	Example answer
					`class Worker`
					`  private name`
					`  private attendance`
					
					`  public function getName()`
					`  return name`
					`  end function`
					
					`  public function`
					`  getAttendance()`
					`  return attendance`
					`  end function`
					
					`  public procedure`
					`  setName(newName)`
					`  name = newName`
					`  end procedure`
					
					`  public procedure`
					`  setAttendance(newAttend)`
					`  if newAttend >=0 and`
					`  newAttend <=100 then`
					`  attendance =`
					`  newAttend`
					`  end if`
					`  end procedure`
					`end class`

			Total	**8**	
9	a		**Mark Band 3–High Level (9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of different modes of addressing. The material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation.	12 AO1.1 (2), AO1.2 (2), AO2.1 (3), AO3.3. (5)	AO1Immediate addressing is where the operand holds the actual data to be used; no address referenced Direct addressing is where the operand holds the address that holds the data to be used. Indirect addressing is where the operand holds an address which is where the data to be used is stored Indexed addressing is where the operand holds an address which is offset using the Index Register to find the true address of the data to be used

			The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well-considered. There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. **Mark Band 2-Mid Level (5-8 marks)** The candidate demonstrates reasonable knowledge and understanding of different modes of addressing; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a sound discussion, the majority of which is focused. Evaluative comments are for the most part appropriate, although one or two opportunities for development are missed. There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence. **Mark Band 1-Low Level (1-4 marks)** The candidate demonstrates a basic knowledge of different modes of addressing; the material is basic and contains some inaccuracies. The candidate makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides a limited discussion which is narrow in focus. Judgments if made are weak and unsubstantiated. The information is basic and communicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be		AO2 Immediate addressing; operand of 27 would refer to the value 27 Direct addressing; operand of 27 would tell the processor to fetch the data held at address 27. Indirect addressing; operand of 27 would tell the processor to fetch the data held at address 27, which itself would then be used an address to fetch data from. Indexed addressing; operand of 27 would be added to the value of the Index register and this would then be used as the address to fetch data from. AO3 Immediate addressing allows simple access to data with no fetch required, but limited by the data size of the operand. Direct addressing allows data to be fetched from memory. Data can potentially be larger in size that with immediate addressing but address range limited by size of operand. Indirect addressing allows a larger range of addresses to be accessed as address fetched. However, multiple fetches required to access data. Indexed addressing allows the Index register to be manipulated to access data stored sequentially e.g. in an array.

			clear. **0 marks** No attempt to answer the question or response is not worthy of credit.		
	b		• 2, 2 • 9,0 • 4,0 • 0,3	4 AO2.2	Code finds integer division and remainder.
			Total	**16**	
10	a		1 mark per bullet up to a maximum of 2 marks, e.g: • When the child/derived/subclass class office/house takes on attributes/methods… • … from building / parent/base/superclass/ class	9AO1.1 (2)AO1.2 (2)AO2.1 (2)AO3.3 (3)	

	b		1 mark for each completed statement up to a maximum of 5 marks:
			
			`private numberFloors`
			`private width`
			`private` **`height`**
			`public procedure new(pFloors, pWidth, pHeight)`
			`numberFloors =` **`pFloors`**
			`width = pWidth`
			`height = pHeight`
			`endprocedure`
			`public function getNumberFloors()`
			`return` **`numberFloors`**
			`endfunction`
			`public function setNumberFloors(pFloors)`
			`if pFloors >= 1 then`
			`numberFloors =` **`pFloors`**
			`return true`
			`else`
			`return` **`false`**
			`endif`
			`endfunction`
			`endclass`
				5AO2.2
				(3)AO3.2
				(2)
					Accept other specific language conventions that would correctly achieve the same outcomes.

	c		1 mark per bullet up to a maximum of 6 marks:	6AO2.1 (1)AO2.2	

			• Class declaration for house with inherits building
			• Declaring bedrooms and bathrooms as private
			• New declaration …
			• … with all **five** parameters
			• Calling super constructor / equivalent with floors, width and height set
			• Setting bedrooms and bathrooms
			
			e.g.
			`class house inherits building`
			`  private bedrooms, bathrooms`
			`  public procedure new(pFloors,`
			`  pWidth, pHeight, pBedrooms,`
			`  pBathrooms)`
			`  super.new(pFloors, pWidth,`
			`  pHeight)`
			`  bedrooms = pBedrooms`
			`  bathrooms = pBathrooms`
			`  endprocedure`
			`endclass`
				(2)AO3.2
				(3)	

	d		1 mark per bullet up to a maximum of 4 marks:
			
			• Procedure header taking parameter
			• Adding parameter to array
			• …at position `numberBuildings`
			• Incrementing `numberBuilding`
			
			e.g.
			`procedure newbuilding(pBuilding)`
			`  buildings[numberBuildings] =`
			`  pBuilding`
			`  numberBuildings = numberBuildings + 1`
			`endprocedure`
				4AO2.1
				(1)AO3.2
				(3)	

	e		1 mark per bullet up to a maximum of 4 marks:	4AO2.1 (1)AO3.2 (3)	

			• Creating a new instance of house with identifier `houseOne`
			• …with the correct parameters
			• Creating a new instance of houseRoad named `limeAvenue`
			• …sending `houseOne` as parameter to the constructor
			
			e.g.
			`houseOne = new house(2, 8, 10, 3, 2)`
			`limeAvenue = new houseRoad(houseOne)`		

	f		To create an instance of an object from a class	1AO2.1 (1)	
			Total	**22**	

11		i	**1 mark for each number/statement up to a maximum of 6 marks:**
			
			`function binarySearch(dataArray:byref, upperbound, lowerbound, ***searchValue***)`
			`  while true`
			`  middle = lowerbound +)`
			`  ((upperbound - lowerbound))`
			`  ***DIV 2***)`
			`  if upperbound < lowerbound`
			`  then`
			`  return **-1**`
			`  else`
			`  if dataArray[middle] < searchValue then`
			`  lowerbound = ***middle + 1***`
			`  elseif dataArray[middle] > searchValue then`
			`  upperbound = ***middle - 1***`
			`  else`
			`  return ***middle***`
			`  endif`
			`  endif`
			`  endwhile`
			`endfunction`
				6AO1.2 (2)AO3.3 (4)	

		ii	Do…until / repeat…until / post condition	1AO1.2 (1)	
			Total	**7**	

12	a	i	• Constructor method definition (.e.g `new`)
			• itemname, price passed in as parameters (must use different identifier names to the ones in the question)
			• …and assigned to attributes
			• `discount` attribute assigned to 0.
				4 AO3.2	Example answer
					`public procedure new(nItemName, nPrice)`
					`  itemname = nItemname`
					`  price = nPrice`
					`  discount = 0`
					`endprocedure`
					
					Look out for alternative notation e.g.
					
					this.itemname = itemName

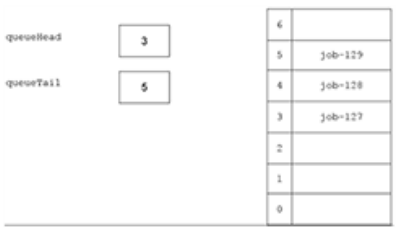
		ii	• 1 mark for creating object with identifier `mushypeas =`
			• 1 mark for creating object as type `ItemForSale`
			• mark for parameters passed in as needed
				3 AO3.2	Example answers
					`mushypeas=new ItemForSale("mushy peas", 0.89)`
					
					`ItemForSale mushypeas = ItemForSale("mushy peas",0.89);`
					
					`mushypeas=ItemForSale(("mushy peas",0.89);`
					Do not penalise for use of self parameter as used by languages such as Python.
					Must be correct case and spelling

		iii	• method definition for `calculatePrice()`
			• applies percentage discount
			• …returns calculated value
				3 AO3.2	Percentage discount must be applied correctly.
					
					Example answer
					`function calculatePrice()`
					`  newPrice = price – (price * discount/100)`
					`  return newPrice`
					`end function`

	b		• `discount` attribute made private • Set method created • …that restricts value to maximum 50%	3 AO2.2	
	c		• Create new class • …inheriting from / subclass of **ItemsForSale** • new attribute for stock location	4 AO2.2	

			• `calculatePrice()` method overridden / new version of `calculatePrice()` implemented in subclass.		
			Total	**17**	
13		i	• Iteration	1 AO2.2 (1)	
		ii	• 5	1 AO2.2 (1)	

		iii	1 mark per bullet up to a maximum of 4 marks, e.g:
			• Initialise Y and Z AND set X
			• Correct use of IF
			• Correct condition (e.g. if X >= Y then)
			• Assignment of Z in correct places
				4 AO3.1 (2) AO3.3 (2)	X Y Z variable alternatives are acceptable
					Solution:
					`X = input()`
					`Y = 5`
					`Z = 0`
					`if X >= Y then`
					`  Z = Y`
					`else`
					`  Z = X`
					`Endif`

			Total	**6**	
14	a		1 mark per pointer • queueHead: Point to the first element in the queue / next element to remove • queueTail: Point to the last element in the queue	2 AO1.2 (2)	
	b		1 mark per bullet up to max 5 • first 3 jobs removed • 128 and 129 added in positions 4 and 5 respectively • no additional jobs • queueHead being 3 (FT errors) • queueTail being 5 (FT errors) 	5 AO2.1 (2) AO2.2 (3)	The underlying implementation of the queue has not been specified, so allow alternative valid answers. e.g. queueHead = 0 queueTail = 2 Location 2: 129 Location 1: 128 Location 0: 127

	c	i	1 mark per bullet to max 5
			
			• Function declaration
			• Checking if queue is empty
			• …returning `null`
			• (Otherwise) incrementing `queueHead`
			• …returning `buffer[queueHead-1]`
			
			e.g.
			`function dequeue()`
			`  if queueHead > queueTail then`
			`  return null`
			`  else`
			`  queueHead = queueHead + 1`
			`  return buffer[queueHead-1]`
			`  endif`
			`endfunction`
				5
				AO2.2
				(2)
				AO3.3
				(3)
					Note: Accept alternative valid underlying implementation answers e.g. Shifting all elements in queue forward.

		ii	1 mark per bullet to max 6
			
			• Function declaration taking parameter
			• Checking if queue is full
			• …returning -1
			• (Otherwise) incrementing queueTail
			• Adding newJob to buffer(queueTail)
			• Returning 1
			
			e.g.
			`function enqueue(newJob)`
			`  if queueTail == 99 then`
			`  return -1`
			`  else`
			`  queueTail = queueTail + 1`
			`  buffer[queueTail] = newJob`
			`  return 1`
			`  endif`
			`endfunction`
				6
				AO2.2
				(3)
				AO3.3
				(3)	

		iii	1 mark per bullet to max 8 • Inputting user choice • If enqueue chosen input job name • …call enqueue with input value as parameter	8 AO2.2 (2) AO3.3 (6)	

			• …check if return value is -1 and output full
			• …otherwise output message that item is added
			• If dequeue chosen
			• …call dequeue **and** save returned value
			• …output returned value (jobname) if not null
			• …or output queue is empty
			
			e.g.
			`main()`
			`  choice = input("Add or remove?")`
			`  if choice == "ADD" then`
			`  jobname = input("Enter job name")`
			`  returnValue = enqueue(jobname)`
			`  if returnValue == -1 then`
			`  print("Queue full")`
			`  else`
			`  print("Job added")`
			`  endif`
			`  else`
			`  returnValue = dequeue()`
			`  if returnValue == null then`
			`  print("Queue empty")`
			`  else`
			`  output returnValue`
			`  endif`
			`  endif`
			`endmain`
					Allow equivalent checks / logic

	d		1 mark per bullet to 3 • Check if either head or tail are incremented to above 99 • … set to be 0 instead • When checking if array is full check if (queueTail == queueHead – 1) OR (queueTail==99 AND queueHead==0)	3 AO2.1 (1) AO2.2 (2)	Credit equivalent modulo arithmetic solution
	e		1 mark per bullet to max 3, e.g. • Use a different structure e.g. a linked list • …items can be added at different points in the linked list depending on priority • …by changing the pointers to items needing priority • Have different queues for different priorities	3 AO2.1 (2) AO2.1 (1)	Allow other suitable descriptions that show how the program could be amended.

			• …add the job to the queue relevant to its priority • …print all the jobs in the highest priority queue first		
			Total	**32**	

15			• Stores current player
			• …and alternates between player on each go
			• Allows a number to be input
			• …and validates that this is between 1 and 3
			• Outputs numbers chosen (e.g. 4, 5, 6)
			• Checks if number 15 has been reached
			• …and displays winning message
			• …and stops
				8
				
				AO3.2
					Example answer
					
					`num = 1`
					`turn = "player 1"`
					`while num <= 15`
					`  print(turn + "'s turn")`
					`  choice = input("how many numbers?")`
					`  if choice >= 1 and choice <= 3 then`
					`  for y = 1 to choice`
					`  print(num)`
					`  num = num + 1`
					`  next y`
					`  //swap turn`
					`  if turn == "player 1" then`
					`  turn = "player 2"`
					`  else`
					`  turn = "player 1"`
					`  end if`
					`  end if`
					`endwhile`
					`print(turn + " wins!")`

			Total	**8**	
16			**Mark Band 3–High Level (9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of procedural and object oriented programming. The material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well-considered. There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. **Mark Band 2-Mid Level (5-8 marks)** The candidate demonstrates reasonable knowledge and understanding of procedural and object	12 AO1.1 (2) AO1.2 (2) AO2.1 (23 AO3.3 (5)	AO1 Object oriented programming makes use of classes (templates) from which objects are made. Classes have attributes and methods Classes can be encapsulated by making attributes private and providing public access methods Object oriented programming supports inheritance which allows classes to use attributes and methods of parent classes. Object oriented programming supports polymorphism meaning that class attributes and methods can take on many different forms if required. AO2 OO approach would call on classes per type of player or enemy object Objects made from these classes, so one enemy class may generate many enemy objects, each with different values for their attributes (e.g. speed, energy) Special types of Player or Enemy objects could be instantiated from classes that inherit from the original Player/Enemy classes, but have attributes/methods of their own.

			oriented programming; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a sound discussion, the majority of which is focused. Evaluative comments are for the most part appropriate, although one or two opportunities for development are missed. There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence. **Mark Band 1-Low Level (1-4 marks)** The candidate demonstrates a basic knowledge of procedural or object oriented programming; the material is basic and contains some inaccuracies. The candidate makes a limited attempt to apply acquired knowledge and understanding to the context provided.		Polymorphism would allow for the attributes/methods of Player/Enemy objects to behave differently from normal if required. AO3 OO promotes a modular approach (procedural through use of subroutines, OO through classes). OO is an abstraction of a real world problem, with classes for each type of things to be modelled and objects for each instance of these. OO has advantages in data security in that encapsulation forces developers/users to use methods (with their built in validation) to access/amend data stored in attributes. OO has advantages in efficiency of design where classes can be reused and can inherit from one another. Procedural programming struggles to support this. OO also offers flexibility through polymorphism.
			Total	**12**	