

A-LEVEL COMPUTER SCIENCE

MARK SCHEME

Practice Paper 1

Maximum marks: 100

View detailed guidance on the conclusions you can draw from your students' performance in these papers on the MERiT welcome page. Understand how your students compare with others and target revision effectively by entering marks into MERiT.

Section A

1 (a) **Marks are AO2 (apply)**

Position	Value	Order Examined In
8	Philip	1
10	Ravi	3
11	Richard	4
12	Timothy	2

1 mark for row 8 correct
 1 mark for row 11 correct
 1 mark for both rows 10 and 12 correct
 Do not award mark for a particular number if same number is written more than once

3

(c) **Mark is AO2 (analysis)**

Order of complexity	Tick one box
$O(\log_2 n)$	<input checked="" type="checkbox"/>
$O(n)$	<input type="checkbox"/>
$O(n^2)$	<input type="checkbox"/>

Do not award mark if more than one box ticked

1

(b) **Mark is AO2 (analysis)**

8

1
[5 marks]

2 (a) **Mark is AO1 (understanding)**

Values/cards need to be taken out of the data structure from the opposite end that they are put in // cards removed from top/front and added at end/bottom/rear;

Values/cards need to be removed in the same order that they are added;

A. It is First In First Out // It is FIFO;

A. It is Last In Last Out // It is LIFO;

Max 1

1

(b) (i) **Mark is AO2 (apply)**

FrontPointer = 13

RearPointer = 52

QueueSize = 40

1 mark for all three values correct

1

(ii) **Mark is AO2 (apply)**

FrontPointer = 13

RearPointer = 3

QueueSize = 43

1 mark for all three values correct

A. Incorrect value for `FrontPointer` if it matches the value given in part (i) and incorrect value for `QueueSize` if it is equal to the value given for `QueueSize` in part (i) incremented by three (follow through of errors previously made). However, `RearPointer` must be 3.

1

(c) **Marks are AO2 (apply)**

```
If DeckQueue is empty THEN
  Report error
ELSE
  Output DeckQueue[FrontPointer]
  Decrement QueueSize
  Increment FrontPointer
  IF FrontPointer > 52 THEN
    FrontPointer ← 1
  ENDIF
ENDIF
```

1 mark for IF statement to check if queue is empty – alternative for test is `QueueSize = 0`

1 mark for reporting an error message if the queue is empty // dealing with the error in another sensible way – this mark can still be awarded if there is an error in the logic of the IF statement, as long as there is an IF statement with a clear purpose

1 mark for only completing the rest of the algorithm if the queue is not empty – this mark can still be awarded if there is an error in the logic of the IF statement, as long as there is an IF statement with a clear purpose

1 mark for outputting the card at the correct position

1 mark for incrementing `FrontPointer` and decrementing `QueueSize`

1 mark for IF statement testing if the end of the queue has been reached

1 mark for setting `FrontPointer` back to 1 if this is the case – this mark can still be awarded if minor error in logic of IF statement, e.g. `>=` instead of `=`

A. `FrontPointer = (FrontPointer MOD 52) + 1` for **3 marks** or

`FrontPointer = (FrontPointer MOD 52)` for **2 marks**, both as alternatives to incrementing and using and the second IF statement –

deduct 1 mark from either of the above if `QueueSize` has not been decremented

A. Any type of brackets for array indexing

I. Additional reasonable `ENDIF` Statements

MAX 5 unless all of the steps listed above are carried out and algorithm fully working

6
[9 marks]

3 (a) **Marks are for AO2 (analysis)**

Connected // There is a path between each pair of vertices;

Undirected // No direction is associated with each edge;

Has no cycles // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;

A no loops

Alternative definitions:

A simple cycle is formed if any edge is added to graph;

Any two vertices can be connected by a unique simple path;

Max 2

2

(b) **Marks is for AO2 (analysis)**

No route from entrance to exit / through maze;

Maze contains a loop / circuit;

A more than one route through maze;

Part of the maze is inaccessible / enclosed;

R Responses that clearly relate to a graph rather than the maze

Max 1

1

(c) **Marks are AO2 (apply)**

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1
6	0	0	0	0	1	0	0
7	0	0	0	0	1	0	0

(allow some symbol in the central diagonal to indicate unused)

or

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2		0	1	1	0	0	0
3			0	0	0	0	0
4				0	1	0	0
5					0	1	1
6						0	0
7							0

(with the shaded portion in either half – some indication must be made that half of the matrix is not being used. This could just be leaving it blank, unless the candidate has also represented absence of an edge by leaving cells blank)

1 mark for drawing a 7x7 matrix, labelled with indices on both axis and filled only with 0s and 1s, or some other symbol to indicate presence/absence of edge e.g. T/F. Absence can be represented by an empty cell.

1 mark for correct values entered into matrix, as shown above;

2

- (d) (i) **Mark is AO1 (knowledge)**
Routine defined in terms of itself // Routine that calls itself;
A alternative names for routine e.g. procedure, algorithm
NE repeats itself

1

- (ii) **Marks are AO1 (understanding)**
Stores return addresses;
Stores parameters;
Stores local variables; **NE** temporary variables
Stores contents of registers;
A To keep track of calls to subroutines/methods etc.

Max 1

Procedures / invocations / calls must be returned to in reverse order
(of being called);

As it is a LIFO structure;

A FILO

As more than one / many return addresses / sets of values may need
to be stored (at same time) // As the routine calls itself and for each
call/invocation a new return address / new values must be stored;

Max 1

2

(e) Marks are AO2 (apply)

				Discovered							Completely Explored							
Call	V	U	EndV	1	2	3	4	5	6	7	1	2	3	4	5	6	7	F
	-	-	7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(2,7)	2	1	7	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
		3	7	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(3,7)	3	2	7	T	T	T	F	F	F	F	F	F	T	F	F	F	F	F
DFS(2,7)	2	4	7	T	T	T	F	F	F	F	F	F	T	F	F	F	F	F
DFS(4,7)	4	2	7	T	T	T	T	F	F	F	F	F	T	F	F	F	F	F
		5	7	T	T	T	T	F	F	F	F	F	T	F	F	F	F	F
DFS(5,7)	5	4	7	T	T	T	T	T	F	F	F	F	T	F	F	F	F	F
		6	7	T	T	T	T	T	F	F	F	F	T	F	F	F	F	F
DFS(6,7)	6	5	7	T	T	T	T	T	T	F	F	F	T	F	F	T	F	F
DFS(5,7)	5	7	7	T	T	T	T	T	T	F	F	F	T	F	F	T	F	F
DFS(7,7)	7	5	7	T	T	T	T	T	T	T	F	F	T	F	F	T	T	T
DFS(5,7)	5	-	7	T	T	T	T	T	T	T	F	F	T	F	T	T	T	T
DFS(4,7)	4	-	7	T	T	T	T	T	T	T	F	F	T	T	T	T	T	T
DFS(2,7)	2	-	7	T	T	T	T	T	T	T	F	T	T	T	T	T	T	T
DFS(1,7)	1	-	7	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T

1 mark for having the correct values changes in each region highlighted by a rectangle and no incorrect changes in the region. Ignore the contents of any cells that are not changed.

A alternative indicators that clearly mean True and False.

A it is not necessary to repeat values that are already set (shown lighter in table)

5
[13 marks]

4 **Marks are AO1 (understanding)**

Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time whereas size of dynamic structure can change at run-time;

Static structures can waste storage space / memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;

Dynamic data structures (typically) require memory to store pointer(s) to the next item(s) which static structures (typically) do not need // Static structures (typically) store data in consecutive memory locations, which dynamic data structures (typically) do not;

Max 2

A just one side of points, other side is by implication

NE Dynamic data structures use pointers

[2 marks]

5 (a) **Marks are AO2 (apply)**

18, 23, 21, 36, 40, 45, 58, 59

Mark as follows:

18 in the first place;

23 and 21 in correct order and in the second and third places;

21 and 36 in the correct order and in the third and fourth places;

40, 45, 58 and 59 in the correct order and in the last four places;

A Table 3 instead of Table 2 as long as the bottom cell of each of the scores column is correct (**I.** any working out)

4

(b) **Mark is AO1 (knowledge)**

Bubble sort;

NE sort

1

(c) **Mark is AO2 (analysis)**

Order of Time Complexity	Tick one box
$O(n)$	
$O(n^2)$	✓
$O(2^n)$	

A alternative indicators instead of a tick e.g. a cross, Y, Yes

R responses in which more than one box is ticked

1

[6 marks]

6 (a) **Marks are AO1 (knowledge)**

Is it possible in general to *write a program / algorithm*; that can tell, given any program and its inputs and without *running / executing the program*;, whether the given program with its given inputs will halt?

A "it" in second reference to program

A "create a Turing machine" for "write an algorithm"

2

(b) **Mark is AO1 (understanding)**

Shows that some problems are non-computable / undecidable // shows that some problems cannot be solved by a computer / algorithm;

In general, inspection alone cannot always determine whether any given algorithm will halt for its given inputs // a program cannot be written that can determine whether any given algorithm will halt for its given inputs;

A it is not computable

Max 1

1
[3 marks]

Section B

7 (a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

Level	Description	Mark range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met.	10-12
3	There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the two words and includes one iterative structure and two selection structures. An attempt has been made to check if the words are anagrams of each other, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made.	7-9
2	A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4-6
1	A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.	1-3

Guidance

Evidence of AO3 design – 4 points:

Evidence of design to look for in responses:

1. Identifying that a selection structure is needed after all letter counts have been compared to output a message saying the words are anagrams or are not.
2. Identifying that a loop is needed that repeats a number of times based on the length of the first or second word // identifying that a loop is needed that repeats 26 times // identifying that a loop is needed that repeats a number of times determined by the number of unique characters in the first or second word
3. Identifying that the number of times a letter occurs in the first string needs to be equal to the number of times it occurs in the second string
4. Boolean (or equivalent) variable used to indicate if the first word is an anagram of the second word // array of suitable size to store the count of each letter

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

Evidence for AO3 programming – 8 points:

Evidence of programming to look for in response:

5. (Suitable prompts asking user to enter the two words followed by) user inputs being assigned to appropriate variables (R. if inside or after iterative structure), two variables with appropriate data types created to store the two words entered by the user
6. Iterative structure to look at each letter in first or second word has correct syntax and start/end conditions // iterative structure to look at each letter in the alphabet has correct syntax and start/end conditions
7. Correctly counts the number of times that a letter occurs in one of the words
8. Selection structure that compares the count of a letter in the first word with the count of that letter in the second word
A. incorrect counts
A. incorrect comparison operator
9. Correctly counts the number of times each letter occurs in one of the two words
10. Program works correctly if the two words entered are the same word
11. Program works correctly if the two words entered are not anagrams
12. Program works correctly for all word pairs consisting of just uppercase letters

Alternative mark scheme

(based on removing a single instance of a letter from the 2nd word each time it appears in the 1st word or vice-versa)

1. Identifying that a selection structure is needed after all the letters that appear in both words have been removed from the first/second word to output a message saying the words are or are not anagrams
3. Identifying that a letter can be removed from the second/first word if it appears in the first/second word
7. Selection structure that checks if letter in first/second word appears in the second/first word
8. Removes a letter from the second/first word if it appears in the first/second word
9. Sets indicator to false if there are any letters that occur in one word but not the other

12

VB.NET

```
Dim Word1, Word2 As String
Dim IsAnagram As Boolean = True
Dim Counts(25, 1) As Integer
Console.Write("Enter the first word: ")
Word1 = Console.ReadLine
Console.Write("Enter the second word: ")
Word2 = Console.ReadLine
For Each ch In Word1
    Counts(Asc(ch) - 65, 0) += 1
Next
For Each ch In Word2
    Counts(Asc(ch) - 65, 1) += 1
Next
For Pos As Integer = 0 To 25
    If Counts(Pos, 0) <> Counts(Pos, 1) Then
        IsAnagram = False
    End If
Next
If IsAnagram Then
    Console.WriteLine("Yes")
Else
    Console.WriteLine("No")
End If
```

PYTHON 2

```
Word1 = ""
Word2 = ""
IsAnagram = True
Word1 = raw_input("Enter the first word: ")
Word2 = raw_input("Enter the second word: ")
if len(Word1) <> len(Word2):
    print "No"
else:
    for Pos in range(0, len(Word1)):
        if Word1.count(Word1[Pos]) <>
Word2.count(Word1[Pos]):
            IsAnagram = False
    if IsAnagram:
        print "Yes"
    else:
        print "No"
```

PYTHON 3

```
Word1 = ""
Word2 = ""
IsAnagram = True
Word1 = input("Enter the first word: ")
Word2 = input("Enter the second word: ")
if len(Word1) != len(Word2):
    print("No")
else:
    for Pos in range(0, len(Word1)):
        if Word1.count(Word1[Pos]) !=
Word2.count(Word1[Pos]):
            IsAnagram = False
    if IsAnagram:
        print("Yes")
    else:
        print("No")
```

C#

```
string word1, word2;
bool IsAnagram = true;
int[,] counts = new int[26, 2];
Console.Write("Enter the first word: ");
word1 = Console.ReadLine();
Console.Write("Enter the second word: ");
word2 = Console.ReadLine();
foreach (var ch in word1) counts[ch - 65, 0]++;
foreach (var ch in word2) counts[ch - 65, 1]++;
int pos = 0;
while (pos <= 25)
{
    if (counts[pos, 0] != counts[pos, 1])
    {
        IsAnagram = false;
    }
    pos++;
}
if (IsAnagram) Console.WriteLine("Yes");
else Console.WriteLine("No");
```

PASCAL

```
var
    word1, word2: string;
    characterCount: array['A' .. 'Z'] of integer;
    character: char;
    canBeMade: boolean;

begin
    for character := 'A' to 'Z' do
        characterCount[character] := 0;
    write('First word: ');
    readln(word1);
    write('Second word: ');
    readln(word2);
    for character in word2 do
        inc(characterCount[character]);
    canBeMade := true;
    for character in word1 do
        dec(characterCount[character]);
    canBeMade := True;
    for character := 'A' to 'Z' do
        if characterCount[character]<>0 then
            canBeMade := False;
    if canBeMade then
        writeln('Yes')
    else
        writeln('No');
    readln;
end.
```

JAVA

```
String word1, word2;
boolean IsAnagram = true;
int pos;
int[][] counts = new int[26][2];
Console.write("Enter the first word: ");
word1 = Console.readLine();
Console.write("Enter the second word: ");
word2 = Console.readLine();
for (pos = 0; pos < word1.length(); pos++) {
    counts[(int)word1.charAt(pos) - 65][0]++;
}
for (pos = 0; pos < word2.length(); pos++) {
    counts[(int)word2.charAt(pos) - 65][1]++;
}
pos = 0;
while (pos <= 25) {
    if (counts[pos][0] != counts[pos][1]) {
        IsAnagram = false;
    }
    pos++;
}
if (IsAnagram) {
    Console.WriteLine("Yes");
} else {
    Console.WriteLine("No");
}
Console.readLine();
```

(b) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******

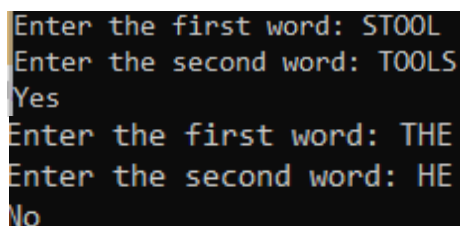
Must match code from part (a), including prompts on screen capture matching those in code.

Code for part (a) must be sensible.

Screen captures showing:

- the string STOOL being entered followed by the string TOOLS and then a message displayed saying that the words are anagrams.
- the string THE being entered followed by the string HE and then a message displayed saying that the words are not anagrams.

1



```
Enter the first word: STOOL
Enter the second word: TOOLS
Yes
Enter the first word: THE
Enter the second word: HE
No
```

[13 marks]

Section C

8 (a) **Marks are for AO2 (analyse)**

Feature	Is present in Figure 11? (Yes/No)
Composition	Yes
Private method	Yes
Protected attribute	No

A. alternative indicators instead of Yes/No e.g. Y/N.

Mark as follows:

One mark per correct row

3

(b) **Mark is for AO2 (analyse)**

`Animal;`

R. if spelt incorrectly

R. if any additional code

I. case

1

(c) **Marks are for AO1 (understanding)**

A protected attribute can be accessed (within its class and) by derived class instances / subclasses;

A public attribute can be accessed in any class; **A.** anywhere

2

(d) **1 mark for AO1 (understanding)**

The way `IsAlive` is represented can be modified without having to change any other classes that interact with `Animal` // this allows

`IsAlive/data/properties` to be modified in a controlled way;

1

(e) **Mark is for AO2 (analyse)**

To ensure that there is an equal probability that a rabbit will be male or female;

A. to try to achieve equal numbers of male and female rabbits

1

(f) **Marks are for AO1 (knowledge) and AO2 (analyse)**

AO1 (knowledge): A method shared (up and down the inheritance hierarchy chain) but with a subclass implementing it differently to the base class;

AO2 (analyse): So that the `Rabbit` class can display additional information about a rabbit that is not displayed by the `Animal` class; **A.** “different” for “additional”

2

(g) **Marks is for AO2 (analyse)**

If all the rabbits were killed by the call to `KillByOtherFactors` there would be no need to call `AgeRabbits` // there would be no rabbits to age;

R. the code would crash / not work

1

(h) **Marks are for AO2 (analyse)**

To select a rabbit for the rabbit to mate with;

That isn't another female rabbit (or the rabbit itself) // a different rabbit will be selected if the selected rabbit is a female (or the rabbit itself);

2

(i) **Marks are for AO2 (analyse)**

The check that the rabbit is not going to mate with itself could be removed; as if the rabbit chosen to mate with was itself then it would be a female rabbit, so the other condition would always (**A.** also) be true;

2

[15 marks]

Section D

9 (a) (i) **Marks are for AO3 (programming)**

1 mark: 1. Calculation of the square of the landscape size // calculation of the square root of the number of warrens

1 mark: 2. Comparison that initial warren count is greater than / less than or equal to square of landscape size // comparison that square root of initial warren count is greater than / less than or equal to landscape size

1 mark: 3. Error message displayed in correct circumstances

1 mark: 4. 1-3 happen repeatedly until valid input and forces re-entry of data each time

Max 3 if not fully working

4

VB.NET

Do

```
Console.Write("Initial number of warrens: ")
InitialWarrenCount = CInt(Console.ReadLine())
If InitialWarrenCount > Math.Pow(LandscapeSize, 2) Then
    Console.WriteLine("Error: Too many warrens to fit on
landscape")
End If
Loop Until InitialWarrenCount <= Math.Pow(LandscapeSize, 2)
```

PYTHON 2

```
InitialWarrenCount = int(raw_input("Initial number of
warrens: "))
while InitialWarrenCount>math.pow(LandscapeSize,2):
    print "Error: Too many warrens to fit on landscape"
    InitialWarrenCount = int(raw_input("Initial number of
warrens: "))
```

PYTHON 3

```
InitialWarrenCount = int(input("Initial number of
warrens: "))
while InitialWarrenCount>math.pow(LandscapeSize,2):
    print("Error: Too many warrens to fit on landscape")
    InitialWarrenCount = int(input("Initial number of
warrens: "))
```

C#

do

```
{
    Console.Write("Initial number of warrens: ");
    InitialWarrenCount =
Convert.ToInt32(Console.ReadLine());
    if (InitialWarrenCount > Math.Pow(LandscapeSize, 2))
        Console.WriteLine("Error: Too many warrens to fit on
landscape");
} while (InitialWarrenCount > Math.Pow(LandscapeSize,
2));
```

PASCAL

```
repeat
  write('Initial number of warrens: ');
  readln(InitialWarrenCount);
  if InitialWarrenCount > Math.Power(LandScapeSize,2)
  then
    writeln('Error: Too many warrens to fit on
landscape');
until InitialWarrenCount <= Math.Power(LandScapeSize,2);
```

JAVA

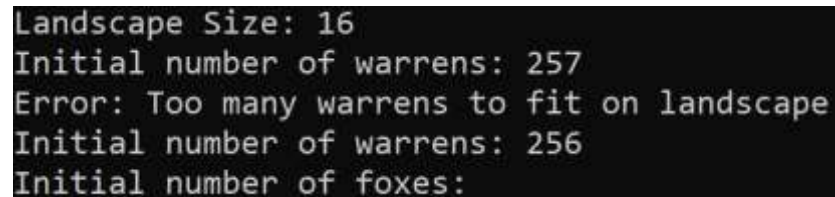
```
do
{
  InitialWarrenCount = Console.readInteger("Initial
number of warrens: ");
  if (InitialWarrenCount > Math.pow(LandscapeSize,2))
    Console.println("Error: Too many warrens to fit on
landscape");
} while (InitialWarrenCount >
Math.pow(LandscapeSize,2));
```

(ii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Must match code from part (a)(i), including error message. Code for part (a)(i) must be sensible.

1 mark: Screen capture(s) showing the required sequence of inputs (16, 257, 256), an error message being displayed for 257, and that 256 has been accepted as the program has displayed next prompt.



```
Landscape Size: 16
Initial number of warrens: 257
Error: Too many warrens to fit on landscape
Initial number of warrens: 256
Initial number of foxes:
```

A. prompt for initial number of foxes not shown if clear from code that 256 would be accepted

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: 1. New subroutine created, with correct name, that overrides the subroutine in the `Animal` class

I. private, protected, public modifiers

1 mark: 2. `CalculateNewAge` subroutine in `Animal` class is always called

1 mark: 3. Check made on gender of rabbit, and calculations done differently for each gender

I. incorrect calculations

1 mark: 4. Probability of death by other causes calculated correctly for male rabbits

1 mark: 5. Probability of death by other causes calculated correctly for female rabbits

5

VB.NET

```
Public Overrides Sub CalculateNewAge()  
    MyBase.CalculateNewAge()  
    If Gender = Genders.Male Then  
        ProbabilityOfDeathOtherCauses =  
ProbabilityOfDeathOtherCauses * 1.4  
    Else  
        If Age >= 2 Then  
            ProbabilityOfDeathOtherCauses =  
ProbabilityOfDeathOtherCauses + 0.1  
        End If  
    End If  
End Sub
```

PYTHON 2

```
def CalculateNewAge(self):  
    super(Rabbit, self).CalculateNewAge()  
    if self.__Gender == Genders.Male:  
        self._ProbabilityOfDeathOtherCauses =  
self._ProbabilityOfDeathOtherCauses * 1.4  
    else:  
        if self._Age >= 2:  
            self._ProbabilityOfDeathOtherCauses =  
self._ProbabilityOfDeathOtherCauses + 0.1
```

PYTHON 3

```
def CalculateNewAge(self):  
    super(Rabbit, self).CalculateNewAge()  
    if self.__Gender == Genders.Male:  
        self._ProbabilityOfDeathOtherCauses =  
self._ProbabilityOfDeathOtherCauses * 1.4  
    else:  
        if self._Age >= 2:  
            self._ProbabilityOfDeathOtherCauses =  
self._ProbabilityOfDeathOtherCauses + 0.1
```

C#

```
public override void CalculateNewAge()  
{  
    base.CalculateNewAge();  
    if (Gender == Genders.Male)  
    {  
        ProbabilityOfDeathOtherCauses =  
ProbabilityOfDeathOtherCauses * 1.4;  
    }  
    else  
    {  
        if (Age >= 2)  
        {  
            ProbabilityOfDeathOtherCauses =  
ProbabilityOfDeathOtherCauses + 0.1;  
        }  
    }  
}
```

PASCAL

```
procedure Rabbit.CalculateNewAge();
begin
    inherited;
    if Gender = Male then
        ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses * 1.4
    else
        if Age >= 2 then
            ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses + 0.1;
        end;
end;
```

JAVA

```
@Override
public void CalculateNewAge()
{
    super.CalculateNewAge();
    if (Gender == Genders.Male)
    {
        ProbabilityOfDeathOtherCauses *= 1.4;
    }
    else if (Age >= 2)
    {
        ProbabilityOfDeathOtherCauses += 0.1;
    }
}
```

(ii) Mark is for AO3 (evaluate)

****SCREEN CAPTURE(S)****

Must match code from part (b)(i). Code for part (b)(i) must be sensible.

1 mark: Any screen capture(s) showing the correct probability of death by other causes for a male rabbit (0.10 to 2dp) and a female rabbit (0.15);

Example:

```
ID 265 Age 2 LS 4 Pr dth 0.1 Rep rate 1.2 Gender Male
ID 266 Age 2 LS 4 Pr dth 0.15 Rep rate 1.2 Gender Female
```

1

(c) (i) Marks are for AO3 (programming)

1 mark: Property created to indicate whether a location contains a trap or not

1 mark: Trap indicator passed to constructor as parameter

1 mark: Trap indicator stored into property by constructor

3

VB.NET

```
Class Location
    Public Fox As Fox
    Public Warren As Warren
    Public Trap As Boolean

    Public Sub New(ByVal T As Boolean)
        Fox = Nothing
        Warren = Nothing
        Trap = T
    End Sub
End Class
```

PYTHON 2

```
class Location:
    def __init__(self, T):
        self.Fox = None
        self.Warren = None
        self.Trap = T
```

PYTHON 3

```
class Location:
    def __init__(self, T):
        self.Fox = None
        self.Warren = None
        self.Trap = T
```

C#

```
class Location
{
    public Fox Fox;
    public Warren Warren;
    public bool Trap;

    public Location(bool T)
    {
        Fox = null;
        Warren = null;
        Trap = T;
    }
}
```

PASCAL

```
type
    Location = class
        Fox : Fox;
        Warren : Warren;
        Trap : boolean;
        constructor New(T : boolean);
    end;

constructor Location.New(T : boolean);
begin
    Fox := nil;
    Warren := nil;
    Trap := T;
end;
```

JAVA

```
class Location
{
    public Fox Fox;
    public Warren Warren;
    public Boolean Trap;

    public Location(Boolean T)
    {
        Fox = null;
        Warren = null;
        Trap = T;
    }
}
```

(ii) **Marks are for AO3 (programming)**

1 mark: 1. An indicator of whether a trap is present or not will be stored for every location

I. wrong value stored in a location

1 mark: 2. Trap created at (4,10)

1 mark: 3. Trap created at (12,14)

MAX 2 if creates any traps in incorrect locations

3

VB.NET

```
For x = 0 To LandscapeSize - 1
  For y = 0 To LandscapeSize - 1
    If x = 4 And y = 10 Or x = 12 And y = 14 Then
      Landscape(x, y) = New Location(True)
    Else
      Landscape(x, y) = New Location(False)
    End If
  Next
Next
```

PYTHON 2

```
for x in range (0, self.__LandscapeSize):
  for y in range (0, self.__LandscapeSize):
    if x == 4 and y == 10 or x == 12 and y == 14:
      self.__Landscape[x][y] = Location(True)
    else:
      self.__Landscape[x][y] = Location(False)
```

PYTHON 3

```
for x in range (0, self.__LandscapeSize):
  for y in range (0, self.__LandscapeSize):
    if x == 4 and y == 10 or x == 12 and y == 14:
      self.__Landscape[x][y] = Location(True)
    else:
      self.__Landscape[x][y] = Location(False)
```

C#

```
for (int x = 0; x < LandscapeSize; x++)
{
  for (int y = 0; y < LandscapeSize; y++)
  {
    if ((x == 4 && y == 10) || (x == 12 && y == 14))
      Landscape[x, y] = new Location(true);
    else
      Landscape[x, y] = new Location(false);
  }
}
```

PASCAL

```
for x := 0 to LandscapeSize - 1 do
  for y := 0 to LandscapeSize - 1 do
    if (x = 4) and (y = 10) or (x = 12) and (y = 14)
  then
    Landscape[x][y] := Location.New(true)
  else
    Landscape[x][y] := Location.New(false);
```

JAVA

```
for(int x = 0 ; x < LandscapeSize; x++)
{
  for(int y = 0; y < LandscapeSize; y++)
  {
    if ((x == 4 && y == 10) || (x == 12 && y == 14))
      Landscape[x][y] = new Location(true);
    else
      Landscape[x][y] = new Location(false);
  }
}
```

(iii) **Marks are for AO3 (programming)**

1 mark: T or other indicator displayed in each location where there is a trap (could be different letters, use of different colours)

1 mark: Row containing column indices matches new display of landscape **I.** number of dashes not adjusted to match new width **R.** if trap indicators not displayed **A.** no adjustment made if indicators for traps used mean no adjustment to width of display for trap was needed

2

VB.NET

```
Private Sub DrawLandscape()  
    Console.WriteLine()  
    Console.WriteLine("TIME PERIOD: " & TimePeriod)  
    Console.WriteLine()  
    Console.Write("  ")  
    For x = 0 To LandscapeSize - 1  
        Console.Write(" ")  
        If x < 10 Then  
            Console.Write(" ")  
        End If  
        Console.Write(x & " |")  
    Next  
    Console.WriteLine()  
    For x = 0 To LandscapeSize * 5 + 3  
        Console.Write("-")  
    Next  
    Console.WriteLine()  
    For y = 0 To LandscapeSize - 1  
        If y < 10 Then  
            Console.Write(" ")  
        End If  
        Console.Write(" " & y & "|")  
        For x = 0 To LandscapeSize - 1  
            If Not Landscape(x, y).Warren Is Nothing Then  
                If Landscape(x, y).Warren.GetRabbitCount() <  
10 Then  
                    Console.Write(" ")  
                End If  
                Console.Write(Landscape(x,  
y).Warren.GetRabbitCount())  
            Else  
                Console.Write(" ")  
            End If  
            If Not Landscape(x, y).Fox Is Nothing Then  
                Console.Write("F")  
            Else  
                Console.Write(" ")  
            End If  
            If Landscape(x, y).Trap Then  
                Console.Write("T")  
            Else  
                Console.Write(" ")  
            End If  
            Console.Write("|")  
        Next  
        Console.WriteLine()  
    Next  
End Sub
```

PYTHON 2

```
def __DrawLandscape(self):
    print
    print "TIME PERIOD:", str(self.__TimePeriod)
    print
    sys.stdout.write(" ")
    for x in range (0, self.__LandscapeSize):
        sys.stdout.write(" ")
        if x < 10:
            sys.stdout.write(" ")
            sys.stdout.write(str(x) + " |")
    print
    for x in range (0, self.__LandscapeSize * 5 + 3):
        sys.stdout.write("-")
    print
    for y in range (0, self.__LandscapeSize):
        if y < 10:
            sys.stdout.write(" ")
            sys.stdout.write(str(y) + "|")
            for x in range (0, self.__LandscapeSize):
                if not self.__Landscape[x][y].Warren is None:
                    if
self.__Landscape[x][y].Warren.GetRabbitCount() < 10:
                        sys.stdout.write(" ")
                        sys.stdout.write(self.__Landscape[x][y].
Warren.GetRabbitCount())
                    else:
                        sys.stdout.write(" ")
                    if not self.__Landscape[x][y].Fox is None:
                        sys.stdout.write("F")
                    else:
                        sys.stdout.write(" ")
                    if self.__Landscape[x][y].Trap:
                        sys.stdout.write("T")
                    else:
                        sys.stdout.write(" ")
                        sys.stdout.write("|")
    print
```

PYTHON 3

```
def __DrawLandscape(self):
    print()
    print("TIME PERIOD:", self.__TimePeriod)
    print()
    print(" ", end = "")
    for x in range (0, self.__LandscapeSize):
        print(" ", end = "")
        if x < 10:
            print(" ", end = "")
            print(x, "|", end = "")
            print()
        for x in range (0, self.__LandscapeSize * 5 + 3):
#CHANGE
            print("-", end = "")
        print()
        for y in range (0, self.__LandscapeSize):
            if y < 10:
                print(" ", end = "")
                print("", y, "|", sep = "", end = "")
                for x in range (0, self.__LandscapeSize):
                    if not self.__Landscape[x][y].Warren is None:
                        if
self.__Landscape[x][y].Warren.GetRabbitCount() < 10:
                            print(" ", end = "")
                            print(self.__Landscape[x][y].Warren.GetRabbitC
ount(), end = "")
                        else:
                            print(" ", end = "")
                            if not self.__Landscape[x][y].Fox is None:
                                print("F", end = "")
                            else:
                                print(" ", end = "")
                            if self.__Landscape[x][y].Trap:
                                print("T", end = "")
                            else:
                                print(" ", end = "")
                                print("|", end = "")
                print()
```

C#

```
private void DrawLandscape()
{
    Console.WriteLine();
    Console.WriteLine("TIME PERIOD: " + TimePeriod);
    Console.WriteLine();
    Console.Write(" ");
    for (int x = 0; x < LandscapeSize; x++)
    {
        if (x < 10)
        {
            Console.Write(" ");
        }
        Console.Write(x + " |");
    }
    Console.WriteLine();
    for (int x = 0; x <= LandscapeSize * 5 + 3; x++)
    {
        Console.Write("-");
    }
    Console.WriteLine();
    for (int y = 0; y < LandscapeSize; y++)
    {
        if (y < 10) { Console.Write(" "); }
        Console.Write(" " + y + "|");
        for (int x = 0; x < LandscapeSize; x++)
        {
            if (Landscape[x, y].Warren != null)
            {
                if (Landscape[x, y].Warren.GetRabbitCount() < 10)
                {
                    Console.Write(" ");
                }
                Console.Write(Landscape[x, y].Warren.GetRabbitCount());
            }
            else
            {
                Console.Write(" ");
            }
            if (Landscape[x, y].Trap) Console.Write("T");
            else Console.Write(" ");
            if (Landscape[x, y].Fox != null)
            {
                Console.Write("F");
            }
            else
            {
                Console.Write(" ");
            }
            Console.Write("|");
        }
        Console.WriteLine();
    }
}
```


PASCAL

```
procedure Simulation.DrawLandscape();
var
  x : integer;
  y : integer;
begin
  writeln;
  writeln('TIME PERIOD: ', TimePeriod);
  writeln;
  write(' ');
  for x := 0 to LandscapeSize - 1 do
    begin
      write(' ');
      if x < 10 then
        write(' ');
      write(x, ' |');
    end;
  writeln;
  for x:=0 to LandscapeSize * 5 + 3 do
    write('-');
  writeln;
  for y := 0 to LandscapeSize - 1 do
    begin
      if y < 10 then
        write(' ');
      write(' ', y, '|');
      for x:= 0 to LandscapeSize - 1 do
        begin
          if not(self.Landscape[x][y].Warren =
nil) then
            begin
              if
self.Landscape[x][y].Warren.GetRabbitCount() < 10 then
                write(' ');
                write(Landscape[x][y].Warren.GetRab
bitCount());
            end
          else
            write(' ');
            if not(self.Landscape[x][y].fox = nil)
then
              write('F')
            else
              write(' ');
              if Landscape[x][y].Trap then
                write('T')
              else
                write(' ');
              write('||');
            end;
          writeln;
        end;
      end;
    end;
  end;
```

JAVA

```
private void DrawLandscape()
{
    Console.println();
    Console.println("TIME PERIOD: " + TimePeriod);
    Console.println();
    Console.print(" ");
    for(int x = 0; x < LandscapeSize; x++)
    {
        Console.print(" ");
        if (x < 10)
        {
            Console.print(" ");
        }
        Console.print(x + " |");
    }
    Console.println();
    for(int x = 0; x < LandscapeSize * 5 + 4; x++)
    {
        Console.print("-");
    }
    Console.println();
    for(int y = 0; y < LandscapeSize; y++)
    {
        if(y < 10 )
        {
            Console.print(" ");
        }
        Console.print(" " + y + "|");
        for(int x = 0; x < LandscapeSize; x++)
        {
            if ( Landscape[x][y].Warren != null )
            {
                if ( Landscape[x][y].Warren.GetRabbitCount() <
10)
                {
                    Console.print(" ");
                }
            }
            Console.print(Landscape[x][y].Warren.GetRabbitCount());
        }
        else
        {
            Console.print(" ");
        }
        if ( Landscape[x][y].Fox != null)
        {
            Console.print("F");
        }
        else
        {
            Console.print(" ");
        }
        if (Landscape[x][y].Trap)
            Console.print("T");
        else
            Console.print(" ");
        Console.print("|");
    }
    Console.println();
}
}
```

(iv) **Marks are for AO3 (programming)**

1 mark: Fox will not be placed in a location that contains a trap

1 mark: Fox will not be placed where there is already a fox

2

VB.NET

```
Private Sub CreateNewFox()  
    Dim x As Integer  
    Dim y As Integer  
    Do  
        x = Rnd.Next(0, LandscapeSize)  
        y = Rnd.Next(0, LandscapeSize)  
        Loop While Not Landscape(x, y).Fox Is Nothing Or  
Landscape(x, y).Trap  
        If ShowDetail Then  
            Console.WriteLine(" New Fox at (" & x & ", " & y &  
")")  
        End If  
        Landscape(x, y).Fox = New Fox(Variability)  
        FoxCount += 1  
    End Sub
```

PYTHON 2

```
def __CreateNewFox(self):  
    x = random.randint(0, self.__LandscapeSize - 1)  
    y = random.randint(0, self.__LandscapeSize - 1)  
    while not self.__Landscape[x][y].Fox is None or  
self.__Landscape[x][y].Trap:  
        x = random.randint(0, self.__LandscapeSize - 1)  
        y = random.randint(0, self.__LandscapeSize - 1)  
        if self.__ShowDetail:  
            sys.stdout.write(" New Fox at (" + str(x) + ", " +  
str(y) + ")")  
        self.__Landscape[x][y].Fox = Fox(self.__Variability)  
        self.__FoxCount += 1
```

PYTHON 3

```
def __CreateNewFox(self):  
    x = random.randint(0, self.__LandscapeSize - 1)  
    y = random.randint(0, self.__LandscapeSize - 1)  
    while not self.__Landscape[x][y].Fox is None or  
self.__Landscape[x][y].Trap:  
        x = random.randint(0, self.__LandscapeSize - 1)  
        y = random.randint(0, self.__LandscapeSize - 1)  
        if self.__ShowDetail:  
            print(" New Fox at (" + str(x) + ", " + str(y) + ")", sep = "")  
        self.__Landscape[x][y].Fox = Fox(self.__Variability)  
        self.__FoxCount += 1
```

C#

```
private void CreateNewFox()
{
    int x, y;
    do
    {
        x = Rnd.Next(0, LandscapeSize);
        y = Rnd.Next(0, LandscapeSize);
    } while ((Landscape[x, y].Fox != null) ||
(Landscape[x, y].Trap));
    if (ShowDetail) { Console.WriteLine(" New Fox at (" +
x + "," + y + ")"); }
    Landscape[x, y].Fox = new Fox(Variability);
    FoxCount++;
}
```

PASCAL

```
procedure Simulation.CreateNewFox();
var
    x : integer;
    y : integer;
begin
    randomize();
    repeat
        x := Random(LandscapeSize);
        y := Random(LandscapeSize);
    until (Landscape[x][y].fox = Nil) and
(Landscape[x][y].Trap = False);
    if ShowDetail then
        writeln(' New Fox at (' ,x, ', ',y, ')');
    Landscape[x][y].Fox := Fox.New(Variability);
    inc(FoxCount);
end;
```

JAVA

```
private void CreateNewFox()
{
    int x;
    int y;
    do
    {
        x = Rnd.nextInt( LandscapeSize);
        y = Rnd.nextInt( LandscapeSize);
    }while (Landscape[x][y].Fox != null ||
Landscape[x][y].Trap) ;
    if (ShowDetail)
    {
        Console.println(" New Fox at (" + x + "," + y +
")");
    }
    Landscape[x][y].Fox = new Fox(Variability);
    FoxCount += 1;
}
```

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Must match code from part (c)(i) to (c)(iv). Code for these parts must be sensible

1 mark: Screen capture(s) indicating which locations contain traps

A. incorrect location of traps if these match those set in parts (c)(ii)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
1		38					F								
2															
3											52				
4													F	67	
5															
6								F							
7										20					
8			80												
9															
10			F		T										
11															
12															
13												F			
14													T		

1

(d) (i) **Marks are for AO3 (programming)**

Structure of subroutine:

- 1 mark:** Subroutine created with correct name
`CheckIfTrapNearFox` **I.** private/public/protected modifiers
- 1 mark:** Subroutine has two parameters of appropriate data type, which are the coordinates of a fox to check for a trap nearby **I.** `self` parameter in Python answers **I.** additional parameters
- 1 mark:** Subroutine returns a Boolean value
- 1 mark:** One or more locations that a trap that would catch the fox could be in are checked
- 1 mark:** All locations that a trap that would catch the fox could be in are checked
- 1 mark:** Checks would work for foxes that are near to the edge of the landscape (ie distance from fox to edge of landscape is less than two)
- 1 mark:** True or False correctly returned depending on whether trap found or not **A.** if only some locations are checked, the correct value is returned based on the locations checked

MAX 6 if not fully working

7

VB.NET

```
Private Function CheckIfTrapNearFox(ByVal FoxX As Integer, ByVal
FoxY As Integer) As Integer
    Dim LeftBound As Integer = Math.Max(0, FoxX - 2)
    Dim RightBound As Integer = Math.Min(LandscapeSize - 1, FoxX +
2)
    Dim TopBound As Integer = Math.Max(0, FoxY - 2)
    Dim BottomBound As Integer = Math.Min(LandscapeSize - 1, FoxY +
2)
    For X As Integer = LeftBound To RightBound
        For Y As Integer = TopBound To BottomBound
            If Landscape(X, Y).Trap Then
                Return True
            End If
        Next
    Next
    Return False
End Function
```

PYTHON 2

```
def __CheckIfTrapNearFox(self, FoxX, FoxY):
    LeftBound = max(0, FoxX - 2)
    RightBound = min(self.__LandscapeSize - 1, FoxX + 2)
    TopBound = max(0, FoxY - 2)
    BottomBound = min(self.__LandscapeSize - 1, FoxY + 2)
    for x in range (LeftBound, RightBound + 1):
        for y in range (TopBound, BottomBound + 1):
            if self.__Landscape[x][y].Trap:
                return True
```

PYTHON 3

```
def __CheckIfTrapNearFox(self, FoxX, FoxY):
    LeftBound = max(0, FoxX - 2)
    RightBound = min(self.__LandscapeSize - 1, FoxX + 2)
    TopBound = max(0, FoxY - 2)
    BottomBound = min(self.__LandscapeSize - 1, FoxY + 2)
    for x in range (LeftBound, RightBound + 1):
        for y in range (TopBound, BottomBound + 1):
            if self.__Landscape[x][y].Trap:
                return True
    return False
```

C#

```
private bool CheckIfTrapNearFox(int foxx, int foxy)
{
    int leftbound = Math.Max(0, foxx - 2);
    int rightbound = Math.Min(LandscapeSize - 1, foxx + 2);
    int topbound = Math.Max(0, foxy - 2);
    int bottombound = Math.Min(LandscapeSize - 1, foxy + 2);
    for (int x=leftbound;x<=rightbound;x++)
    {
        for (int y=topbound;y<=bottombound;y++)
        {
            if (Landscape[x,y].Trap) return true;
        }
    }
    return false;
}
```

PASCAL

```
function Simulation.CheckIfTrapNearFox(FoxX : integer;
FoxY : integer) : boolean;

var
  LeftBound : integer;
  RightBound : integer;
  TopBound : integer;
  BottomBound : integer;
  x : integer;
  y : integer;
  Trapped : boolean;

begin
  Trapped := false;
  LeftBound := Math.Max(0, FoxX - 2);
  RightBound := Math.Min(LandscapeSize - 1, FoxX + 2);
  TopBound := Math.Max(0, FoxY - 2);
  BottomBound := Math.Min(LandscapeSize - 1, FoxY +
2);
  for x := LeftBound to RightBound do
    for y := TopBound to BottomBound do
      if Landscape[x][y].Trap then Trapped := true;
      CheckIfTrapNearFox := Trapped;
    end;
```

JAVA

```
private boolean CheckIfTrapNearFox(int FoxX, int FoxY)
{
  int LeftBound = Math.max(0, FoxX - 2);
  int RightBound = Math.min(LandscapeSize - 1, FoxX +
2);
  int TopBound = Math.max(0, FoxY - 2);
  int BottomBound = Math.min(LandscapeSize - 1, FoxY +
2);
  for (int x=LeftBound;x<=RightBound;x++)
  {
    for (int y=TopBound;y<=BottomBound;y++)
    {
      if (Landscape[x][y].Trap) return true;
    }
  }
  return false;
}
```

(ii) **Marks are for AO3 (programming)**

1 mark: The value of `IsAlive` can be changed from outside the class

1 mark: Updating `IsAlive` is via a setter function (any reasonable name acceptable)

MAX 1 if not fully working

2

VB.NET

```
Public Sub SetDead()  
    IsAlive = False  
End Sub
```

PYTHON 2

```
def SetDead(self):  
    self._IsAlive = False
```

PYTHON 3

```
def SetDead(self):  
    self._IsAlive = False
```

C#

```
public void SetDead()  
{  
    IsAlive = false;  
}
```

PASCAL

```
procedure Animal.SetDead();  
  
begin  
    IsAlive := false;  
end;
```

JAVA

```
public void SetDead()  
{  
    IsAlive = false;  
}
```


(iii) **Marks are for AO3 (programming)**

1 mark: CheckIfTrapNearFox subroutine is called for each fox on the landscape

1 mark: If the subroutine returns true, the fox's IsAlive property will be set to false

MAX 1 if not fully working

2

VB.NET

```
For x = 0 To LandscapeSize - 1
  For y = 0 To LandscapeSize - 1
    If Not Landscape(x, y).Fox Is Nothing Then
      If ShowDetail Then
        Console.WriteLine("Fox at (" & x & ", " & y & "): ")
      End If
      If CheckIfTrapNearFox(x, y) Then
        Landscape(x, y).Fox.SetDead()
      End If
      Landscape(x, y).Fox.AdvanceGeneration(ShowDetail)
      If Landscape(x, y).Fox.CheckIfDead() Then
        Landscape(x, y).Fox = Nothing
        FoxCount -= 1
      Else
        If Landscape(x, y).Fox.ReproduceThisPeriod() Then
          If ShowDetail Then
            Console.WriteLine(" Fox has reproduced. ")
          End If
          NewFoxCount += 1
        End If
        If ShowDetail Then
          Landscape(x, y).Fox.Inspect()
        End If
        Landscape(x, y).Fox.ResetFoodConsumed()
      End If
    End If
  Next
Next
```

PYTHON 2

```
for x in range (0, self.__LandscapeSize):
  for y in range (0, self.__LandscapeSize):
    if not self.__Landscape[x][y].Fox is None:
      if self.__ShowDetail:
        sys.stdout.write("Fox at (" + str(x) + ", " +
str(y) + "): " + "\n")
      if self.__CheckIfTrapNearFox(x, y):
        self.__Landscape[x][y].Fox.SetDead()
      self.__Landscape[x][y].Fox.AdvanceGeneration
(self.__ShowDetail)
      if self.__Landscape[x][y].Fox.CheckIfDead():
        self.__Landscape[x][y].Fox = None
        self.__FoxCount -= 1
      else:
        if self.__Landscape[x][y].Fox.Reproduce
ThisPeriod():
          if self.__ShowDetail:
            print " Fox has reproduced. "
          NewFoxCount += 1
        if self.__ShowDetail:
          self.__Landscape[x][y].Fox.Inspect()
        self.__Landscape[x][y].Fox.ResetFoodConsumed()
```

PYTHON 3

```
for x in range (0, self.__LandscapeSize):
    for y in range (0, self.__LandscapeSize):
        if not self.__Landscape[x][y].Fox is None:
            if self.__ShowDetail:
                print("Fox at (" + x + "," + y + "): ", sep = "")
            if self.__CheckIfTrapNearFox(x, y):
                self.__Landscape[x][y].Fox.SetDead()
            self.__Landscape[x][y].Fox.AdvanceGeneration
            (self.__ShowDetail)
            if self.__Landscape[x][y].Fox.CheckIfDead():
                self.__Landscape[x][y].Fox = None
                self.__FoxCount -= 1
            else:
                if self.__Landscape[x][y].Fox.Reproduce
                ThisPeriod():
                    if self.__ShowDetail:
                        print(" Fox has reproduced. ")
                        NewFoxCount += 1
                    if self.__ShowDetail:
                        self.__Landscape[x][y].Fox.Inspect()
                        self.__Landscape[x][y].Fox.ResetFoodConsumed()
```

C#

```
for (int x = 0; x < LandscapeSize; x++)
{
    for (int y = 0; y < LandscapeSize; y++)
    {
        if (Landscape[x, y].Fox != null)
        {
            if (ShowDetail)
            {
                Console.WriteLine("Fox at (" + x + "," + y + "): ");
            }
            if (CheckIfTrapNearFox(x, y))
            {
                Landscape[x, y].Fox.SetDead();
            }
            Landscape[x, y].Fox.AdvanceGeneration(ShowDetail);
            if (Landscape[x, y].Fox.CheckIfDead())
            {
                Landscape[x, y].Fox = null;
                FoxCount--;
            }
            else
            {
                if (Landscape[x, y].Fox.ReproduceThisPeriod())
                {
                    if (ShowDetail) { Console.WriteLine(" Fox has
reproduced. "); }
                    NewFoxCount++;
                }
                if (ShowDetail) { Landscape[x, y].Fox.Inspect(); }
                Landscape[x, y].Fox.ResetFoodConsumed();
            }
        }
    }
}
```

PASCAL

```
for x:= 0 to LandscapeSize - 1 do
  for y:= 0 to LandscapeSize - 1 do
    if not(Landscape[x][y].Fox = nil) then
      begin
        if ShowDetail then
          writeln('Fox at (', x, ', ', y, '): ');
          if CheckIfTrapNearFox(x,y) then
            Landscape[x][y].Fox.SetDead();
        Landscape[x][y].Fox.AdvanceGeneration(ShowDetail);
        if Landscape[x][y].Fox.CheckIfDead() then
          begin
            Landscape[x][y].Fox := nil;
            dec(FoxCount);
          end
        else
          begin
            if Landscape[x][y].Fox.ReproduceThisPeriod()
then
              begin
                if ShowDetail then
                  writeln(' Fox has reproduced. ');
                  inc(NewFoxCount);
                end;
                if ShowDetail then
                  Landscape[x][y].Fox.Inspect();
                  Landscape[x][y].Fox.ResetFoodConsumed();
                end;
              end;
            end;
          end;
        end;
```

JAVA

```
for (int x = 0; x < LandscapeSize; x++)
{
    for (int y = 0; y < LandscapeSize; y++)
    {
        if (Landscape[x][y].Fox != null)
        {
            if (ShowDetail)
            {
                Console.println("Fox at (" + x + ", " + y + "):
");
            }
            if (CheckIfTrapNearFox(x,y))
            {
                Landscape[x][y].Fox.SetDead();
            }
            Landscape[x][y].Fox.AdvanceGeneration(ShowDetail);
            if (Landscape[x][y].Fox.CheckIfDead())
            {
                Landscape[x][y].Fox = null;
                FoxCount -= 1;
            }
            else
            {
                if (Landscape[x][y].Fox.ReproduceThisPeriod())
                {
                    if (ShowDetail)
                    {
                        Console.println(" Fox has reproduced. ");
                    }
                    NewFoxCount += 1;
                }
                if (ShowDetail)
                {
                    Landscape[x][y].Fox.Inspect();
                }
                Landscape[x][y].Fox.ResetFoodConsumed();
            }
        }
    }
}
```

(iv) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Must match code from part (d)(i) to (d)(iii). Code for these parts must be sensible

1 mark: Screen capture(s) show that the foxes at (2,10) and (11,13) have died.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
1		47					F								
2											45				
3													F	62	
4									F	11					
5			82												
6															
7															
8															
9															
10					T										
11															
12															
13															
14													T		

1
[34 marks]