

Markscheme

Mock examination — Set 1

Computer science

Standard level

Paper 2

This markscheme is **confidential** and for the exclusive use of the teachers marking this mock examination.

It must **not** be shared with candidates before the mock examination has been sat.

General marking instructions

1. Follow the markscheme provided, award only whole marks and mark only in **RED**.
2. Where a mark is awarded, a tick/check (✓) **must** be placed in the text at the **precise point** where it becomes clear that the candidate deserves the mark. **One tick to be shown for each mark awarded.**
3. Sometimes, careful consideration is required to decide whether or not to award a mark. In these cases write a brief comment in the margin to support your decision, especially for moderation purposes. It should be remembered that the script may be returned to the candidate.
4. Personal codes/notations are unacceptable.
5. Where an answer to a part question is worth no marks but the candidate has attempted the part question, write a zero next to the answer. Where the candidate has not attempted the part question, write “NR”.
6. Ensure that you have viewed **every** page including any additional sheets.
7. A mark should not be awarded where there is contradiction within an answer. Write “CON” next to the contradiction.
8. **If a candidate has attempted more than one Option** within a paper, mark all the candidate’s work and award the marks for the higher scoring Option.

Subject details: Computer Science SL Paper 2 markscheme (Mock examination — Set 1)

Mark allocation

Candidates are required to answer **all** questions in **one** Option. Total 50 marks.

General

A markscheme often has more specific points worthy of a mark than the total allows. This is intentional. Do not award more than the maximum marks allowed for that part of a question.

When deciding upon alternative answers by candidates to those given in the markscheme, consider the following points:

- Each statement worth one point has a separate line and the end is signified by means of a semi-colon (;).
- An alternative answer or wording is indicated in the markscheme by a “/”; either wording can be accepted.
- Words in (...) in the markscheme are not necessary to gain the mark.
- If the candidate’s answer has the same meaning or can be clearly interpreted as being the same as that in the markscheme then award the mark.
- Mark positively. Give candidates credit for what they have achieved and for what they have got correct, rather than penalizing them for what they have not achieved or what they have got wrong.
- Remember that many candidates are writing in a second language; be forgiving of minor linguistic slips. In this subject effective communication is more important than grammatical accuracy.
- Occasionally, a part of a question may require a calculation whose answer is required for subsequent parts. If an error is made in the first part then it should be penalized. However, if the incorrect answer is used correctly in subsequent parts then **follow through** marks should be awarded. Indicate this with “FT”.

General guidance

Issue	Guidance
Answering more than the quantity of responses prescribed in the questions	• In the case of an “identify” question, read all answers and mark positively up to the maximum marks. Disregard incorrect answers. • In the case of a “describe” question, which asks for a certain number of facts eg “describe two kinds”, mark the first two correct answers. This could include two descriptions, one description and one identification, or two identifications. • In the case of an “explain” question, which asks for a specified number of explanations eg “explain two reasons ...”, mark the first two correct answers. This could include two full explanations, one explanation, one partial explanation etc.

Option A — Java

1. (a) Award [1 max]

N must be a whole number / an integer;

N must not be negative / N must be greater than or equal to 0 / N must be positive;

Note: Do not accept answers that only say that N must be a number.

(b) Award [3] as follows:

correct values of N;

correct TRUE/FALSE values in both decision columns;

correct values of count and correct output;

N	N > 0 ?	N is odd ?	count	output
13			0	
	TRUE	TRUE	1	
6	TRUE	FALSE		
3	TRUE	TRUE	2	
1	TRUE	TRUE	3	
0	FALSE			3

Note: Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.

(c) Award [2 max]

$O(\log N)$ / logarithmic time;

...because the value of N is (about) halved on each pass through the loop;

...so the loop is repeated once for each binary digit of N / about $\log_2 N$ times;

...so doubling N adds only one more pass through the loop / the time grows very slowly as N increases;

Note: Award [1] for $O(\log N)$ and [1] for a valid description linked to the halving of N. Do not accept $O(N)$ or $O(N^2)$ for the first mark; a correct description of the halving can still gain [1].

(d) Award [4 max]

the search starts with the whole list / low = 0 and high = 14;

the middle item, 47 (index 7), is compared with 29; 29 is smaller so indices 7 to 14 are discarded / high = 6;

the middle of indices 0 to 6, 23 (index 3), is compared with 29; 29 is larger so indices 0 to 3 are discarded / low = 4;

the middle of indices 4 to 6, 34 (index 5), is compared with 29; 29 is smaller so only index 4 remains / high = 4;

the item at index 4 is 29, which matches, so the search stops and the value is found (at index 4);

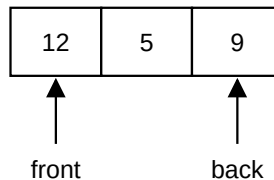
Comparison	low	high	mid	Value at mid	Result
1	0	14	7	47	$29 < 47$, so high = 6
2	0	6	3	23	$29 > 23$, so low = 4
3	4	6	5	34	$29 < 34$, so high = 4
4	4	4	4	29	found at index 4

*Note: Accept positions numbered from 1 (middle positions 8, 4, 6 and 5). Allow FT if a different but consistent choice of middle item is made. A general description of a binary search that is not applied to this list gains **[2]** max. Award **[3 max]** if the search is not shown to end by finding 29 at index 4.*

(e) Award **[2]**.

correct elements in the correct order (12, 5, 9) [1];

correct indication of the front (12) and the back (9) [1];



Note: Accept the queue drawn horizontally in either direction or vertically, provided that the front and the back are clearly labelled.

(f) Award **[3]** as follows:

Award **[1]** for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false;

 dequeue an element and enqueue it onto a second (empty) queue only if it is not negative / is 0 or more;

then dequeue each element of the second queue and enqueue it back onto the original queue (until the second queue is empty) / use the second queue in place of the original queue;

Alternative answer

enqueue a marker value (that cannot be in the queue) at the back of the queue;

repeatedly dequeue an element until the marker is dequeued;

 if the element is not negative, enqueue it again at the back of the queue (otherwise do not enqueue it);

Note: Answer must demonstrate an algorithmic approach.

2. (a) **Award [2 max]**

the memory is allocated once, when the structure is created / no time is spent resizing the structure;

...as the maximum number of passengers (150) is known in advance, a fixed size is sufficient;

the amount of memory used is fixed/predictable;

...so the system will not need to request more memory while tickets are being sold / there is no overhead for pointers (unlike a linked list);

the elements are stored in consecutive memory locations / can be accessed directly by their index (unlike a linked list);

...so a passenger name can be found/accessed quickly;

Note: Award [1] for an advantage and [1] for an expansion. The expansion does not need to refer to the ferry for the mark.

(b) **Award [2] as follows:**

Award [1] for each correct pass.

first pass: 48, 35, 90, 61, 27, 112;

second pass: 35, 48, 61, 27, 90, 112;

Note: Award FT for the second pass if it follows correctly from an incorrect first pass.

(c) **Award [2 max]**

Integer.parseInt("3x") cannot convert the text to an integer, so a NumberFormatException is thrown / an exception occurs (and tickets is not changed);

the catch block handles the exception and outputs "Enter a whole number" (instead of the program crashing);

the finally block is always executed, so "Input checked" is also output;

the method returns 0 / the initial value of tickets;

(d) **Award [2 max]**

lowest is initialized to 0 / no fare is less than 0, so lowest is never updated (and "Lowest fare: 0.0" is output);

initialize lowest to the first fare, fares[0] / to a value larger than any possible fare (eg 1000);

Note: Award [1] for identifying the error and [1] for the correction. Changing < to > is not a correct correction (it would output the highest fare). Replacing the loop with a library call (eg Arrays.stream(fares).min()) does not gain the correction mark, because the given code must be corrected.

(e) **Award [3 max] as follows:**

Award [1] for a method header with a String parameter that returns a String;

Award [1] for extracting the hours (characters at index 8 and 9);

Award [1] for extracting the minutes (characters at index 10 and 11);

Award [1] for joining the hours, ":" and the minutes and returning the result;

Example answer

```
public static String departureTime(String code)
{
    String hours = code.substring(8, 10);
    String minutes = code.substring(10, 12);    // allow substring(10)
    return hours + ":" + minutes;
}
```

*Note: Accept any method that correctly isolates the time, eg `split("-")`, `lastIndexOf("-")`, `code.substring(code.length() - 4)`, or a character-by-character answer such as `"" + code.charAt(8) + code.charAt(9) + ":" + code.charAt(10) + code.charAt(11)`. Award **[2 max]** if the result is not in the form `hh:mm`. Ignore minor syntax errors; penalize a repeated error once only.*

(f) Award **[4 max]** as follows:

Award [1] for opening `tickets.txt` for reading;
Award [1] for a loop that reads the file line by line until the end of the file;
Award [1] for comparing the first three characters of each line with the parameter (using `equals()`);
Award [1] for initializing and incrementing a counter and outputting it after the loop;
Award [1] for closing the file;

Example answer

```
public static void countFrom(String port) throws IOException
{
    BufferedReader reader = new BufferedReader(new FileReader("tickets.txt"));
    int count = 0;
    String line = reader.readLine();
    while (line != null)
    {
        if (line.substring(0, 3).equals(port)) // allow line.startsWith(port)
            count = count + 1;
        line = reader.readLine();
    }
    reader.close();
    System.out.println ("Tickets sold from " + port + ": " + count);
}
```

Alternative answer

```
public static void countFrom(String port) throws IOException
{
    Scanner reader = new Scanner(new File("tickets.txt"));
    int count = 0;
    while (reader.hasNextLine())
    {
        String line = reader.nextLine();
        if (line.substring(0, 3).equals(port))
            count = count + 1;
    }
    reader.close();
    System.out.println ("Tickets sold from " + port + ": " + count);
}
```

Note: Exception handling (throws IOException or try/catch) is not required for any mark. Comparing the Strings with == does not gain the comparison mark. contains(port) / indexOf(port) >= 0 does not gain the comparison mark (it would also count arrivals, eg CAL-ARD-1030). Accept line.split("-")[0].equals(port), line.startsWith(port) or line.indexOf(port) == 0. Ignore minor syntax errors; penalize a repeated error once only.

3. (a) **Award [3 max]**

Award [1] for a box with three compartments and the class name `Performer` at the top;
Award [1] for all five attributes with their data types;
Award [1] for all four methods with their parameters and return types;
Award [1] for correct visibility symbols (- for private, + for public);

Performer
- <u>maxRating</u> : int - name: String - stage: String - ratings: int[] - numRatings: int
+ Performer(name: String, stage: String) + getName(): String + getStage(): String + addRating(r: int): void + averageRating(): double

Note: Accept other clear notations for data types (eg `int maxRating`). The constructor does not need to be shown. `maxRating` must be shown as private (-) for the visibility mark; underlining it to show that it is static is not required.

(b) **Award [2 max]**

Award [1] for a valid point about `maxRating` and **[1]** for a contrasting point about `numRatings`.

`maxRating` belongs to the class / there is one copy shared by all `Performer` objects;
`numRatings` belongs to each object / each `Performer` object has its own copy of `numRatings`;

`maxRating` can be used without creating any `Performer` object / is referred to through the class (`Performer.maxRating`);

`numRatings` can only be accessed through a particular `Performer` object;

a change to `maxRating` applies to every act;

a change to `numRatings` for one act does not affect any other act;

`maxRating` is created when the program starts / the class is loaded;

`numRatings` is created each time a new `Performer` object is instantiated;

(c) **Award [2 max]**

`ratings/numRatings` cannot be accessed or changed directly by code outside the class;

...so new ratings can only be added through the public method `addRating()`;

...which only stores values from 1 to `maxRating` / rejects invalid values such as 0 or 7;

...and `maxRating` is also private, so code outside the class cannot change the limit (eg to 10);

...so `numRatings` always matches the number of ratings stored / the average rating cannot be corrupted / the object's state stays valid;

Note: Award [1 max] for a general answer that does not refer to `addRating()` or to the ratings of a `Performer` object.

(d) Award **[3]** as follows:

Award [1] for a constructor header named `Performer` with no return type and two `String` parameters;

Award [1] for assigning both parameters to the attributes `name` and `stage`;

Award [1] for creating the ratings array with 2000 elements;

Example answer

```
public Performer(String name, String stage)
{
    this.name = name;
    this.stage = stage;
    this.ratings = new int[2000];
    this.numRatings = 0;
}
```

Note: Setting `numRatings` to 0 is not required (an `int` attribute is 0 by default). this. is not required if the parameter names are different from the attribute names. Ignore minor syntax errors; penalize a repeated error once only.

(e) Award **[5]** as follows:

Award [1] for returning 0 if no ratings have been stored (`numRatings == 0`);

Award [1] for initializing and updating a total;

Award [1] for a loop through the stored ratings;

Award [1] for dividing the total by `numRatings` so that the result is not truncated (eg casting to `double`);

Award [1] for returning the average;

Example answer

```
public double averageRating()
{
    if (numRatings == 0)
        return 0;
    int total = 0; // allow double total = 0;
    for (int i = 0; i < numRatings; i++)
    {
        total = total + ratings[i];
    }
    return (double) total / numRatings;
}
```

*Note: A loop to `ratings.length` is acceptable, as the unused elements hold 0. Integer division (`int` total divided by `numRatings` without a cast) loses only the division mark. Using `Arrays.stream()/IntStream` (not permitted by the question) gains **[3]** max.*

(f) *Award [5 max] as follows:*

Award [1] for a method header with the array and the name as parameters;
Award [1] for a loop through the elements of lineup;
Award [1] for comparing getName() of each object with the name (using equals());
Award [1] for outputting the stage and the average rating using getStage() and averageRating();
Award [1] for stopping the search when the act is found / using a Boolean flag;
Award [1] for outputting "Not on the line-up" only when the act is not found;

Award [4 max] if Not on the line-up is never output, or is output for each act that does not match.

Example answer

```
public static void showPerformer(Performer[] lineup, String name)
{
    boolean found = false;
    int i = 0;
    while (i < 40 && found == false)           // allow i < lineup.length
    {
        if (lineup[i].getName().equals(name))
        {
            found = true;
            System.out.println ("Stage: " + lineup[i].getStage());
            System.out.println ("Average rating: " + lineup[i].averageRating());
        }
        i = i + 1;
    }
    if (found == false)
        System.out.println ("Not on the line-up");
}
```

Note: Accept a for loop that breaks or returns after the output, or a loop through all 40 elements that uses a Boolean flag. Comparing the Strings with == does not gain the comparison mark. Ignore minor syntax errors; penalize a repeated error once only.

Option B — Python

4. (a) Award [1 max]

N must be a whole number / an integer;

N must not be negative / N must be greater than or equal to 0 / N must be positive;

Note: Do not accept answers that only say that N must be a number.

(b) Award [3] as follows:

correct values of N;

correct TRUE/FALSE values in both decision columns;

correct values of count and correct output;

N	N > 0 ?	N is odd ?	count	output
13			0	
	TRUE	TRUE	1	
6	TRUE	FALSE		
3	TRUE	TRUE	2	
1	TRUE	TRUE	3	
0	FALSE			3

Note: Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.

(c) Award [2 max]

$O(\log N)$ / logarithmic time;

...because the value of N is (about) halved on each pass through the loop;

...so the loop is repeated once for each binary digit of N / about $\log_2 N$ times;

...so doubling N adds only one more pass through the loop / the time grows very slowly as N increases;

Note: Award [1] for $O(\log N)$ and [1] for a valid description linked to the halving of N. Do not accept $O(N)$ or $O(N^2)$ for the first mark; a correct description of the halving can still gain [1].

(d) Award [4 max]

the search starts with the whole list / low = 0 and high = 14;

the middle item, 47 (index 7), is compared with 29; 29 is smaller so indices 7 to 14 are discarded / high = 6;

the middle of indices 0 to 6, 23 (index 3), is compared with 29; 29 is larger so indices 0 to 3 are discarded / low = 4;

the middle of indices 4 to 6, 34 (index 5), is compared with 29; 29 is smaller so only index 4 remains / high = 4;

the item at index 4 is 29, which matches, so the search stops and the value is found (at index 4);

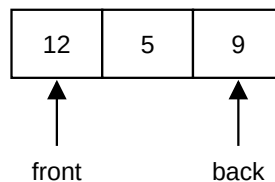
Comparison	low	high	mid	Value at mid	Result
1	0	14	7	47	$29 < 47$, so high = 6
2	0	6	3	23	$29 > 23$, so low = 4
3	4	6	5	34	$29 < 34$, so high = 4
4	4	4	4	29	found at index 4

*Note: Accept positions numbered from 1 (middle positions 8, 4, 6 and 5). Allow FT if a different but consistent choice of middle item is made. A general description of a binary search that is not applied to this list gains **[2]** max. Award **[3 max]** if the search is not shown to end by finding 29 at index 4.*

(e) Award **[2]**.

correct elements in the correct order (12, 5, 9) [1];

correct indication of the front (12) and the back (9) [1];



Note: Accept the queue drawn horizontally in either direction or vertically, provided that the front and the back are clearly labelled.

(f) Award **[3]** as follows:

Award **[1]** for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false;

 dequeue an element and enqueue it onto a second (empty) queue only if it is not negative / is 0 or more;

then dequeue each element of the second queue and enqueue it back onto the original queue (until the second queue is empty) / use the second queue in place of the original queue;

Alternative answer

enqueue a marker value (that cannot be in the queue) at the back of the queue;

repeatedly dequeue an element until the marker is dequeued;

 if the element is not negative, enqueue it again at the back of the queue (otherwise do not enqueue it);

Note: Answer must demonstrate an algorithmic approach.

5. (a) **Award [2 max]**

the memory is allocated once, when the structure is created / no time is spent resizing the structure;

...as the maximum number of passengers (150) is known in advance, a fixed size is sufficient;

the amount of memory used is fixed/predictable;

...so the system will not need to request more memory while tickets are being sold / there is no overhead for pointers (unlike a linked list);

the elements are stored in consecutive memory locations / can be accessed directly by their index (unlike a linked list);

...so a passenger name can be found/accessed quickly;

Note: Award [1] for an advantage and [1] for an expansion. The expansion does not need to refer to the ferry for the mark.

(b) **Award [2] as follows:**

Award [1] for each correct pass.

first pass: [48, 35, 90, 61, 27, 112];

second pass: [35, 48, 61, 27, 90, 112];

Note: Award FT for the second pass if it follows correctly from an incorrect first pass.

(c) **Award [2 max]**

int("3x") cannot convert the text to an integer, so a ValueError is raised / an exception occurs (and tickets is not changed);

the except block handles the exception and outputs "Enter a whole number" (instead of the program crashing);

the finally block is always executed, so "Input checked" is also output;

the function returns 0 / the initial value of tickets;

(d) **Award [2 max]**

lowest is initialized to 0 / no fare is less than 0, so lowest is never updated (and "Lowest fare: 0" is output);

initialize lowest to the first fare, fares[0] / to a value larger than any possible fare (eg 1000);

Note: Award [1] for identifying the error and [1] for the correction. Changing < to > is not a correct correction (it would output the highest fare). Replacing the loop with lowest = min(fares) does not gain the correction mark, because the given code must be corrected.

(e) **Award [3 max] as follows:**

Award [1] for a function definition with one parameter that returns a string;

Award [1] for extracting the hours (characters at index 8 and 9);

Award [1] for extracting the minutes (characters at index 10 and 11);

Award [1] for joining the hours, ":" and the minutes and returning the result;

Example answer

```
def departure_time(code):
    hours = code[8:10]
    minutes = code[10:12]          # allow code[10:]
    return hours + ":" + minutes
```

*Note: Accept any method that correctly isolates the time, eg `split("-")`, `rfind("-")`, `code[-4:]`, or a character-by-character answer such as `code[8] + code[9] + ":" + code[10] + code[11]`. Award **[2 max]** if the result is not in the form `hh:mm`. Ignore minor syntax errors; penalize a repeated error once only.*

(f) Award **[4 max]** as follows:

Award [1] for opening `tickets.txt` for reading;
Award [1] for a loop that reads the file line by line until the end of the file;
Award [1] for comparing the first three characters of each line with the parameter;
Award [1] for initializing and incrementing a counter and outputting it after the loop;
Award [1] for closing the file;

Example answer

```
def count_from(port):
    file = open("tickets.txt", "r")
    count = 0
    line = file.readline()
    while line != "":
        if line[0:3] == port:          # allow line.startswith(port)
            count = count + 1
        line = file.readline()
    file.close()
    print("Tickets sold from", port, ":", count)
```

Alternative answer

```
def count_from(port):
    file = open("tickets.txt", "r")
    count = 0
    for line in file:
        if line[0:3] == port:
            count = count + 1
    file.close()
    print("Tickets sold from", port, ":", count)
```

Note: Accept with `open("tickets.txt", "r")` as `file`: for both the opening and the closing marks. `port in line / line.find(port) >= 0` does not gain the comparison mark (it would also count arrivals, eg `CAL-ARD-1030`). Accept `line.split("-")[0] == port`, `line.startswith(port)`, and a loop over `file.readlines()`. Exception handling is not required for any mark. Ignore minor syntax errors; penalize a repeated error once only.

6. (a) Award [3 max]

Award [1] for a box with three compartments and the class name `Performer` at the top;
Award [1] for all five attributes with their data types;
Award [1] for all four methods with their parameters and return types;
Award [1] for correct visibility symbols (- for private, + for public);

Performer
- <u>max_rating</u> : int - _name: str - _stage: str - _ratings: list - _num_ratings: int
+ __init__(name: str, stage: str) + get_name(): str + get_stage(): str + add_rating(r: int) + average_rating(): float

Note: Accept other clear notations for data types and names without the underscore (eg - name: str). The constructor does not need to be shown. `_max_rating` must be shown as private (-) for the visibility mark; underlining it to show that it is static is not required. Accept no return type for `add_rating()`.

(b) Award [2 max]

Award [1] for a valid point about `_max_rating` and [1] for a contrasting point about `self._num_ratings`.

`_max_rating` belongs to the class / there is one copy shared by all `Performer` objects;
`self._num_ratings` belongs to each object / each `Performer` object has its own copy of `_num_ratings`;

`_max_rating` can be used without creating any `Performer` object / is referred to through the class (`Performer._max_rating`);

`self._num_ratings` can only be accessed through a particular `Performer` object;

a change to `_max_rating` applies to every act;

a change to `self._num_ratings` for one act does not affect any other act;

`_max_rating` is declared inside the class but outside any method / is created when the class is defined;

`self._num_ratings` is created inside the `__init__()` method, each time a new `Performer` object is instantiated;

(c) **Award [2 max]**

`_ratings/_num_ratings` should not be accessed or changed directly by code outside the class;
...so new ratings can only be added through the public method `add_rating()`;
...which only stores values from 1 to `_max_rating` / rejects invalid values such as 0 or 7;
...and `_max_rating` is also private, so code outside the class should not change the limit (eg to 10);
...so `_num_ratings` always matches the number of ratings stored / the average rating cannot be corrupted / the object's state stays valid;

Note: Award [1 max] for a general answer that does not refer to `add_rating()` or to the ratings of a `Performer` object.

(d) **Award [3] as follows:**

Award [1] for a header `def __init__(self, name, stage):` with `self` and two parameters;
Award [1] for assigning both parameters to the attributes `self._name` and `self._stage`;
Award [1] for creating an empty list `self._ratings` **and** setting `self._num_ratings` to 0;

Example answer

```
def __init__(self, name, stage):
    self._name = name
    self._stage = stage
    self._ratings = []
    self._num_ratings = 0
```

Note: Do not accept `self._ratings = [0] * 2000` for the list mark, because `add_rating()` appends to the list. Ignore minor syntax errors; penalize a repeated error once only.

(e) **Award [5] as follows:**

Award [1] for returning 0 if no ratings have been stored (`self._num_ratings == 0`);
Award [1] for initializing and updating a total;
Award [1] for a loop through the stored ratings;
Award [1] for dividing the total by the number of ratings;
Award [1] for returning the average;

Example answer

```
def average_rating(self):
    if self._num_ratings == 0:
        return 0
    total = 0
    for i in range(0, self._num_ratings):    # allow range(len(self._ratings))
        total = total + self._ratings[i]
    return total / self._num_ratings
```

Alternative answer

```
def average_rating(self):
    if len(self._ratings) == 0:
        return 0
    total = 0
    for r in self._ratings:
        total = total + r
    return total / len(self._ratings)
```

*Note: A solution that uses `sum()` or the `statistics` module gains **[3]** max. Using integer division (`//`) loses the division mark.*

(f) Award **[5 max]** as follows:

Award [1] for a function header with the list and the name as parameters;
Award [1] for a loop through the elements of `lineup`;
Award [1] for comparing `get_name()` of each object with the name;
Award [1] for outputting the stage and the average rating using `get_stage()` and `average_rating()`;
Award [1] for stopping the search when the act is found / using a Boolean flag;
Award [1] for outputting "Not on the line-up" only when the act is not found;

*Award **[4 max]** if Not on the line-up is never output, or is output for each act that does not match.*

Example answer

```
def show_performer(lineup, name):
    found = False
    i = 0
    while i < 40 and found == False:          # allow i < len(lineup)
        if lineup[i].get_name() == name:
            found = True
            print ("Stage: ", lineup[i].get_stage())
            print ("Average rating: ", lineup[i].average_rating())
            i = i + 1
    if found == False:
        print ("Not on the line-up")
```

Alternative answer

```
def show_performer(lineup, name):
    found = False
    for act in lineup:
        if act.get_name() == name:
            found = True
            print ("Stage: ", act.get_stage())
            print ("Average rating: ", act.average_rating())
            break
    if found == False:
        print ("Not on the line-up")
```

Note: Accept a for loop that breaks or returns after the output, or a loop through all 40 elements that uses a Boolean flag. Ignore minor syntax errors; penalize a repeated error once only.
