Informatics 1 - Computation & Logic: Tutorial 0

Introduction

Week 2: 24–28 September 2018

Before your first tutorial please take an hour to watch Google's unconscious bias video[1], and be prepared to discuss the issues it raises in the tutorial.

In your tutorials, and more generally during your time at Edinburgh, you will develop many skills. In particular, you will learn to work with other people. Working effectively with others is not as easy as you might think. We have to **learn** to work with others. Unconscious bias describes situations where our background, personal experiences, societal stereotypes and cultural context can impact our interactions with others. We all have unconscious biases. In order to work effectively with others, we must learn to overcome them.

Informatics is the science of systems that store, process and communicate information. The INF1 course provides a foundation for the study of informatics. It has two major strands. In one strand you will learn to program in Haskell — processing information.

The other strand introduces you to computational thinking. In this strand you will learn to talk about computational systems, and reason logically about their behaviour.

We will use the language of set theory to talk about computational systems. This first tutorial concerns finite sets, and properties of their elements — statements that may be true or false.

In advance of the tutorial session in week 2, **you should read Chapter 1 of Mathematical Methods in Linguistics** (MML), and **answer** questions 1–15 below.[2]

Please spend at most one hour on each of sections I–III. Record the time taken for each section. Let us know if one hour was insufficient.

[1] Text highlighted like this is a clickable link.
[2] Exercises 1, 2, 3, 6 and 8 are adapted from MML.

I Some basic ideas

The natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ correspond to the sizes of finite sets. For a finite set, the answer to the question, *How many?*, will be a number.
Since $\emptyset = \{\}$ is a finite set, 0 is a natural number.[3]

Counting and arithmetic If we want to go beyond yes/no questions, it is natural to ask, *How many ...?* We are interested in sets, so we will ask how many elements there are in a set. We will focus on finite sets. We write $|A|$ or $\#A$ for the number of elements in A.

1. If $n \in \mathbb{N}$, what is $|\{x \mid x \in \mathbb{N},\ x < n\}|$?

The familiar arithmetic operations correspond to operations on sets.
Let A and B be disjoint finite sets, with $|A| = a$, and $|B| = b$.
Suppose B has at least one element, $y \in B$.

2. What can you say about b?
3. Is $y \in A$?
4. Use arithmetic operators to give expressions for the following numbers:

   (a) $|\{\}| =$
   (b) $|A \cup \{y\}| =$
   (c) $|\{\langle x, y\rangle \mid x \in A\}| =$
   (d) $|A \cup B| =$
   (e) $|A \times B| =$
   (f) $|\wp A| =$

n-ary notation Numbers are abstract objects; we can represent them in many different ways. In most living languages it is possible to name an arbitrary natural number. so, we can use natural language to give the answer. However, these names soon become unwieldy.
In our everyday lives, we usually use decimal notation for natural numbers.[4]
A finite sequence of k digits $x_i \in \{0, \ldots, 9\}$ represents a number.

$$\langle x_{k-1}, \ldots, x_0\rangle \text{ represents } \sum_{i<k} 10^i x_i$$

Binary notation is similar. A finite sequence of k digits $x_i \in \{0, 1\}$ represents a number.

$$\langle x_{k-1}, \ldots, x_0\rangle \text{ represents } \sum_{i<k} 2^i x_i$$

In general, for n-ary notation ($n > 1$), a finite sequence of k digits $x_i \in \{0, \ldots, n-1\}$, for $i < k$, represents a number.

$$\langle x_{k-1}, \ldots, x_0\rangle \text{ represents } \sum_{i<k} n^i x_i$$

[3] Computer scientists should remember that numbers start at zero – as pointed out by Dijkstra.
[4] Haskell uses decimal notation (as the default) for literals of type `Int`.

For n-ary notation with $n \leq 10$ we use the normal digits $0, ...n-1$. We then move on to use letters of the alphabet as digits > 10. So the *octal* (8-ary) digits are $0, 1, 2, 3, 4, 5, 6, 7$ and the *hexadecimal* (16-ary) digits are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$[5]. When we represent a number in base n, we use digits $0 \ldots (n-1)$. Just as the places in decimal notation count units, tens, hundreds, thousands, etc., the places in n-ary notation represent units, ns, n^2s, n^3s, etc.

Just as with decimal arithmetic, when we add multiply, subtract, or take powers of numbers in base n, the value in the units position of the result depends only on the value(s) in the units position of the argument(s).

The arithmetic of the units position is called arithmetic $\mod n$, (arithmetic modulo n). We write $x \mod n$ for the value of the units digit in the n-ary expansion of x. It is just the remainder of the integer division of x by n.

A very practical application of arithmetic $\mod n$ is to calculate checksums within serial number identifiers. For example, International Bank Account Numbers (IBANs) make use of arithmetic $\mod 97$ to spot user input errors in bank account numbers. In cryptography, modular arithmetic directly underpins public key systems, such as RSA and Diffie–Hellman, which use exponentiation $\mod p$, where p is a large prime.

In this tutorial you will focus on simpler examples. Both $(x \mod n)$, and the result, $(x \operatorname{div} n)$, of the integer division, can be defined by the following properties:

$$0 \leq x \mod n < n \quad x = n \times (x \operatorname{div} n) + (x \mod n)$$

Haskell provides three relevant functions `div, mod, and divMod`. You can use these to write conversion functions. For example[6]

```
digits = "0123456789abcdef"
base11 :: Int -> [Char]
base11 0 = ""
base11 n = base11 d ++ [digits !! m] -- (digits !! m) selects mth digit
  where (d,m) = n `divMod` 11
```

The work required to compute `d = n `div` x` is essentially the same as the work reuired to compute `m = n `mod` x`; the function `divMod` does the work once and returns both answers as a pair, `(d, m) = n `divMod` x`.

5. Generalise `base11` to a function taking the base (11 in the example code) as a parameter.

```
base :: Int -> Int -> [Char] -- so that, e.g. base 11 n = base11 n
```

A small amount of coding should save some time for the following question – and you will need a related function in cl-tutorial 3.

[5] Lower case letters may also be used, $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f$.

[6] This code is not robust. It only works for non-negative numbers.

6. Each row of the table below should show the same number represented in the various bases. Complete the table.

Base Name	2 binary	3 ternary	5	7	8 octal	10 decimal	16 hexadecimal
	1111	120	30	21	17	15	F
	1000						
		200					
			42				
				666			
					700		
						666	
							AB

You will encounter numbers in various bases — the bases will often (but not always) be powers of 2. A MAC (media access control) address is a unique 48-bit or 64-bit identifier assigned to a network interface controller in a device – each of your networked devices will have at least one MAC address. An Internet Protocol address (IP address) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. Internet Protocol version 4 (IPv4) defines an IP address as a 32-bit number. However, because there are now more devices than IPv4 addresses, a new version, IPv6, developed in 1995, uses 128 bits for the IP address.

Ethernet or MAC addresses have 48 or 64 bits usually written as 12- or 16-digit hexadecimal numbers. These hex digits are often grouped in pairs to give a more-readable 6- or 8-digit representation, base $16 \times 16 = 256$; e.g. `b0:70:2d:7a:29:14`. IPv4 addresses are also usually written as four digits base 256. However these digits are usually given in decimal notation; for example, `129.215.90.19`. IPv6 addresses are written as eight-digit numbers base 2^{16}, each of the eight components is written in hexadecimal, using up to four hex digits, e.g. `fe80::8f9:2ade:bf20:d8b6`. Leading zeros are omitted. The empty string represents the number 0 (in any base), so `fe80::8f9:2ade::d8b6` is a valid IPv6 address.[7]

[7] This example is correct, but the rule for omitting zeros is actually a bit more complicated than it appears from this example.

7. Complete the addition and multiplication tables for arithmetic mod $3, 5,$ and 7. Remember, this is just the arithmetic of the units column, so each square should contain just one digit in the range 0–$n-1$.

$+$	0	1	2
0			
1			
2			

$\times$	0	1	2
0			
1			
2			

$+$	0	1	2	3	4
0					
1					
2					
3					
4					

$\times$	0	1	2	3	4
0					
1					
2					
3					
4					

$+$	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

$\times$	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

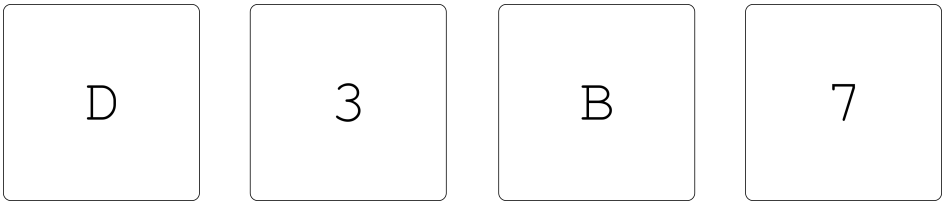
II Thinking logically

When we process information, context is often very important. We can reason about familiar situations, but have difficulty reasoning in unfamiliar contexts. To start you thinking about thinking logically, try the following experiment, designed in the the 1960s by Peter Wason, to examine humans' ability to reason logically. It has since become most researched in reasoning psychology and is referred to as the 'Wason task'.

Just for fun, try the experiment yourself. You will be first presented with the original Wason task, and then with its modern variation. Try to answer fairly quickly rather than deliberate for hours.

But relax! Rather than giving a particular solution to the experiments' questions, your job is to experiment, experience, and judge afterwards which one was easier for you to answer.

8. Consider the four cards below. Each has a letter on one side, and a number on its other side.



Consider **Statement 1** about them:

"If there is a 3 on one side of any card, then there is a D on its other side."

What is the smallest number of cards above you have to turn around to make sure that none of them falsifies this statement?

Cards to turn:

Number:

Now try another version of the Wason task:

9. Consider **Statement 2**:

   "If you are over 18, you cannot buy alcohol here."

   Consider the four cards below. Each card represents a person, with age on one side, and the item they want to buy on the other (let's assume each person buys only one item).

   What is the smallest number of cards above you have to turn around to make sure that no one violates the rule?

   Cards to turn:

   Number:

Which question was easier for you to think about? Did you answer both correctly?[8]

Can you say how the two statements and questions are similar? What is the nature of their differences? Generally people find the second version much easier to think about. While most people answer the second version 'correctly', most people give answers to the first task that do not correspond to the rules of formal logic.

We will see that the two versions are logically identical. Logic is what remains when we take away context and just depend on the structure of the problem.

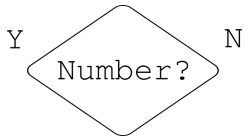
Intricacies of why this is will be explored later in the course. For now, let's see how the two versions are the same while practising drawing flow charts that will model the rules above.

Imagine a program (or, more precisely, a function) that looks at one card at a time and decides whether it has to be turned or not. Keep in mind that we will also only turn cards if it is strictly necessary, aiming to keep the number of cards turned to a minimum. Our goal is to reach one of the two possible final states:

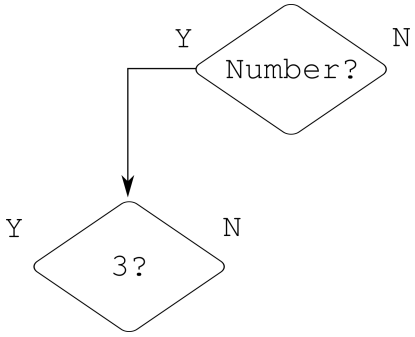
[8] According to formal logic, the correct answers would be:
Version1 - second card, with '3' (check if indeed there is D on its other side), and third card, 'B' (check if the other side is not '3' - a card having 3 on one side and B on the other would violate the rule;
Version2 - second card, the person who drinks (check whether they are over 18), and third card, the kid who is 12 (check if he/she drinks).

The program will begin by looking at the card.

It will be helpful to know which side of the card we are looking at the, 'first' (with a number) or the 'other' (with a letter).[9]

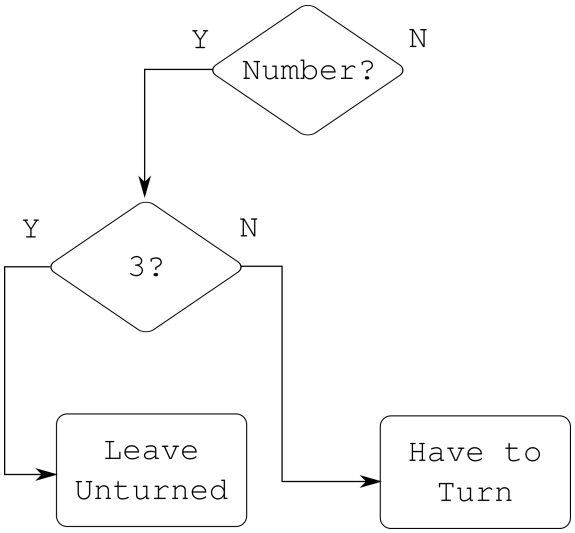


If it is a number, it will be good to check if it's 3, as it is mentioned in the 'if' part of **Statement 1**.



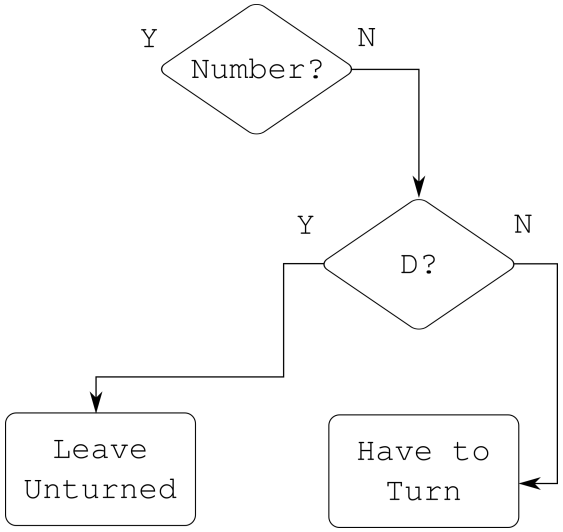
We have to turn every visible '3' card to make sure that there is a 'D' on the other side.

If it is a number different than 3, we don't care what is on the other side. Whatever it is, cards with numbers $\neq 3$ cannot falsify the rule whose pre-condition regarding the number side is being '3'. So, for example, cards (1,E), (441,D), or (-4,Y) do not falsify **Statement 1**.

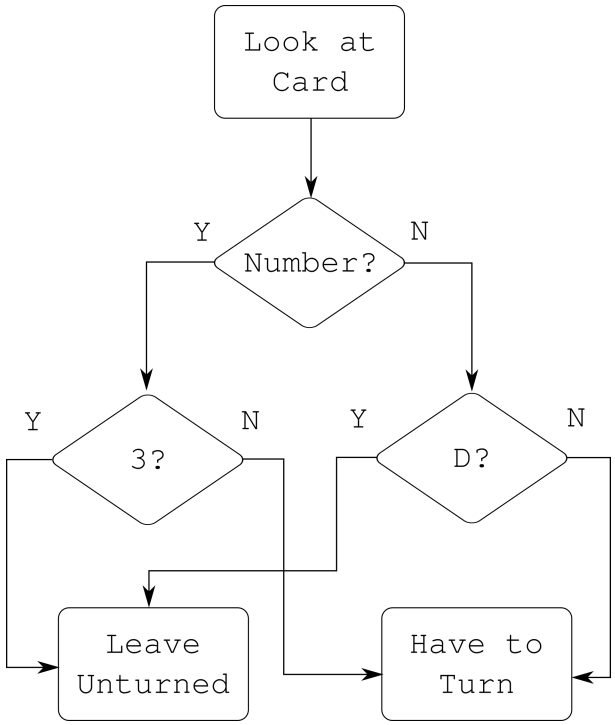


[9]For simplicity, let us assume here that this function will always be given cards with numbers on one side and letters on the other, and will not have to handle unexpected input.

On the other side, logic is sort of 'opposite': if the visible letter is 'D', then the condition from **Statement 1** is fulfilled, so the card cannot violate the statement. In contrast, we have to turn around every card with a visible letter different than 'D' - because a card with a 3 on one side and $\neq$ D on the other side would falsify the rule.



Combined, we get a full model of the function's execution:



10. Write a flowchart for a program (function) of a self-checkout machine that has to decide whether to ask the customer for an ID, abiding to the rule from **Statement 2**.

    Let's be unrealistic but keep things simple and say that **input** will be either **alcohol presence**, or **age** estimate. An (admittedly weak) reason for this could for example be that checking database for items' ingredients is somehow costly and when there is a lot of items it is preferable to consult the camera for an age estimate.

    Feel free to get creative, keep in mind though that it will give you less chance of checking your solution against the model answer.

III Sets

Notation

$\emptyset$: the Empty Set $\{\}$	$x \bmod 3 == 0$: x is divisible by 3 (remainder of dividing x by 3 is 0)
$\{a, b, c\}$: the set with three elements, a, b, c	$a \in B$: a is a member of the set B
$\mathbb{N}$: the Natural Numbers (non-negative integers)	$a \notin B$: a is not a member of B
	$A \subseteq B$: A is a subset of B
$\mathbb{Z}$: the Integers	$A \subset B$: A is a proper subset of B

Section 1.3 of MML features three ways of denoting the set of positive integers that are less than 7: by listing (1-12), by predicate notation, which is also called a comprehension (1-13) and by (recursive) rules (1-14):

(1–12) $\{1, 2, 3, 4, 5, 6\}$

and

(1–13) $\{x \mid x \text{ is a positive integer less than } 7\}$

and

(1–14) a) $1 \in A$
b) if $x \in A$ and x is less than 6, then $x + 1 \in A$
c) nothing else is in A

11. Write a specification by rules and as a comprehension for each of the following sets of integers.

    (a) $\{300, 301, 302, \ldots, 399, 400\}$
    (b) $\{5, 10, 15, 20, \ldots\}$
    (c) $\{7, 17, 27, 37, \ldots\}$
    (d) $\{1, 1/2, 1/4, 1/8, 1/16, \ldots\}$
    (e) $\{0, 2, -2, 4, -4, 6, -6, \ldots\}$
    (f) $\{3, 4, 7, 8, 11, 12, 15, 16, 19, 20, \ldots\}$

Use the notation given earlier, recall also that:

- a recursive rule can contain many statements
- the left hand side of a statement can contain many conditions (see (1-14) (b))
- the right hand side of a comprehension also can contain many conditions, e.g.: $\{x \mid x \in \mathbb{N}, P(x), R(x)\}$, for some predicates (properties), P, R.

You may not yet know some functions that are necessary to complete the next exercise - so here are some you might find helpful.

Some Useful Haskell

List Comprehension for a List of Pairs

```
[ (x, y) | x <- xs, y <- ys]
```

Concatenating Lists The **concat** function creates a single list from a list of lists:

```
Prelude> concat [ [ 5, 4 ],[ 3, 2 ],[ 1, 0 ] ]
[5,4,3,2,1,0]
```

Division The standard **‘div‘** function computes **Integer division**. In practice, this is useful if you want to obtain a whole number remainder while dividing. For example:

```
Prelude> 7 ‘div‘ 3
2
```

Integer 3 'fits' 2 times in Integer 7, leaving a reminder of 1 (which, in turn, can be obtained if you use the **‘mod‘** function: 7 ‘mod‘ 3 returns the result 1. Indeed, these two are very handy when you want to work with **modular arithmetic**. It's a very different case (in Haskell as well as in programming in general) if you want to work with real **division**. When you're looking to obtain a result like $\frac{7}{3}$, use Haskell's `/` function:

```
Prelude> 7 / 3
```

12. Define Haskell list comprehensions that list the elements of the sets from question 11. Test them using `take`.[10]

13. Given the following sets:

$$A = \{a, b, c, 2, 3, 4\} \qquad D = \{b, c\} \qquad G = \{\{a, b\}, \{c, 2\}\}$$
$$B = \{a, b\} \qquad E = \{a, b, \{c\}\}$$
$$C = \{c, 2\} \qquad F = \emptyset$$

say whether each of the following statements is true or false:

(a) $c \in A$
(b) $c \in F$
(c) $c \in E$
(d) $\{c\} \in E$
(e) $\{c\} \in C$
(f) $B \subseteq A$
(g) $D \subset A$
(h) $A \subseteq C$
(i) $D \subseteq E$
(j) $F \subseteq A$
(k) $E \subseteq F$
(l) $B \in G$
(m) $B \subseteq G$
(n) $\{B\} \subseteq G$
(o) $D \subseteq G$
(p) $\{D\} \subseteq G$
(q) $G \subseteq A$
(r) $\{\{c\}\} \subseteq E$

14. How big is each of the following sets? Specify each set by listing its members:

(a) $\wp\{a, b, c\}$
(b) $\wp\{a\}$
(c) $\wp\,\emptyset$
(d) $\wp\{\emptyset\}$
(e) $\wp\,\wp\{a, b\}$
(f) $\wp\,\wp\{a, b, c\}$

[10] See section IV below for solutions to (a) and (b) if you need help.

IV Haskell recap

The following section is a recap of various ways of defining lists in Haskell, using example set 3(a) from the previous exercise. If you feel you already know everything about the topic, feel free to skip to the next question.

Each of the sets from the previous exercises can be translated into Haskell. Sets can be defined in the interactive GHC environment, or in an .hs file; standalone, or generated with a use of defined functions.

For a quick example let's start by opening GHCi. Creating a sequence in Haskell is as simple as putting the first and last number in brackets with two dots between them, for example [1..10]:

```
> ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude> [1..10]
```

will generate numbers from 1 to 10. By default Haskell assumes the intervals between numbers are meant to be 1. Example 3.(a) has such interval, starts with 300 and ends with 400, which looks like this when denoted as a list comprehension:

```
Prelude> [ x | x <- [300..400]]
```

Or we can just write a simpler expression for the same list:

```
Prelude> [300..400]
```

What's even more exciting, Haskell is intelligent enough to guess the sequence if given the first two numbers with any interval, e.g. using example 3(b):

```
Prelude> [5,10..100]
```

For simplicity we generated them up to 100 here. The sets from question 12 are all infinite, which you can create in Haskell if you leave out the last number:

```
Prelude> [5,10..]
```

The infinite generation can be interrupted by pressing ctrl-c.[11] We will use Haskell's built-in function `take` to generate only the few first numbers of the sequence, for example to take the first 20 items we write:

```
Prelude> take 20 [5,10..]
```

[11] Also, if you're like me and you like an empty terminal window, ctrl-d can be used to exit GHCi and the command 'clear' will wipe out the current output.

Collecting information

15. In tutorials you will be in groups of five or six students, sharing a table. We want each group to try to find three (or, if necessary, more) yes/no questions that, taken together serve to distinguish every member of the group – which means that no two people will give the same answers to all three questions.

    To **prepare for this task**, you should each think up six yes-no questions, that you will ask the other members of the group.
    For example, you might ask, *Do you have red hair?* or *Were you born in Scotland?*

    You may be more imaginative, but have to take some care choosing your questions – they should be questions that everyone should be able *and willing* to answer truthfully with a *Yes* or *No*.

    You should come to the tutorial with a print-out of this page, on which you have written your questions, and filled in the first answer column. Write your name in the space provided and make a mark in the box for each question for which your answer is *Yes*; if your answer is *No* leave the box blank.

    You will be given further instructions at your tutorial.

Write questions below and names to the right.						
	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	☐

V Tutorial Activities

You should have five or six students at each table. If there are more than six of you, one must move; if you are fewer than five, you must steal someone from a group with six.

1. Start with the questions you prepared in answer to question 15, to help you to get to know one another.

   10m First get the other students at your table to answer your questions and fill in their names and answers on your question sheet. Take turns to do this so you all hear all the questions and answers.

   If anyone is uncomfortable or unable to answer any question then circle the box to show that they have not answered it.

   Discuss any such questions with your group.

   The answers to your six questions give a five-or-six-digit binary code for each student. Are all these codes different? If so we say that the questions discriminate the set of students at your table.

   10m As a group can you think of a few inappropriate questions – ones that it would not have been good to ask?

   Work together as a group to find the smallest subset of the union of all your question sets that discriminates the set of students at your table.

   What makes a question good as a member of a discriminating set?

2. This is the technical part ... (3×15m)
   For each of sections I–III

   5m Split your group 2-2-2 or 2-3 (you can use a different split for each question) and, working in 2s or 3s, compare your answers to the questions.

   5m As a single group, check whether you can all agree on your final answers to all the questions in this section.

   Discuss any differences and see if you can agree on the correct answer. If you need help raise a hand.

   5m We will briefly run through any common problems from the podium.

3. Discuss the Google unconscious bias video. Ensure that everyone's views are heard.

   5m Have you any personal experience of bias?
   How might unconscious bias affect your activities as a group?

   5m Are the four methods used to combat implicit bias at Google relevant to your interactions with students and staff in Informatics?
   Are there any other ways you might suggest?

   15m Select one member of your group to give a (up to) one minute summary of your discussion.

This tutorial exercise sheet was written by Dagmara Niklasiewicz and Michael Fourman. Please send comments to `michael.fourman@ed.ac.uk`