

Programming 1 / Object Oriented Programming

Thinking like a programmer
Lecture #1: Introduction

Who is this short bloke then?

- ◉ Paul Neve
 - ◉ Module Leader for Programming 1 (CI4100)
 - ◉ Lecturer on
 - ◉ "Thinking like a programmer"
 - ◉ Introduction to Java (CI4100A and CI4520)
 - ◉ Introduction to Web Programming (CI4100B)
 - ◉ Client-side web programming in Javascript (CI4100 and CI4520)
 - ◉ Object Oriented Programming with Java (CI4100A)
 - ◉ Web Applications with PHP and Javascript (CI4100B)
 - ◉ Workshop MC



- ◉ paul@kingston.ac.uk

What you can expect from the module team

- ⦿ to arrive on time for lectures and workshops
- ⦿ to keep you informed about everything you need to know as the module goes on (e.g. dates, times, procedures)
- ⦿ to provide good lecture slides and workshop materials that will help you to learn
- ⦿ to be available to support you during tutorials, office hours and via email
- ⦿ to do our best to answer any questions in lectures
- ⦿ to make the module engaging, interesting, and hopefully fun!

What we expect from you

- ⦿ To show consideration for your fellow students by:
 - ⦿ Arriving on time for lectures and workshops
 - ⦿ If you are late, please enter the room quietly
 - ⦿ If you are *really* late, you will be refused entry!
 - ⦿ Attendance is taken at the start of a lecture
 - ⦿ If you're late, officially you were never there...
 - ⦿ Switching off mobile phones (which means no texting 😊)
 - ⦿ Asking questions - if you are wondering something, chances are someone else is wondering too!

What we expect from you

- ⦿ To show consideration for *yourself* by:
 - ⦿ Turning up to *all* lectures and workshops
 - ⦿ Making use of the materials we make available online
 - ⦿ Bring pen and paper and **MAKE NOTES!**
 - ⦿ Not everything we say in lectures or workshops will be in the Powerpoint slides or available on the online learning environment...
 - ⦿ ...but anything we do say in lectures or workshops can and will be used against you in a ~~court of law~~ exam or coursework! 😊
- ⦿ Using the lectures, workshops and online materials as a *starting* point for study - not the end point of your learning!
 - ⦿ We will NOT spoon feed you! You will not ever be given step by step instructions i.e. “do this, then do this, then do this”! You will need to *think* for yourself!

Attendance

- ⦿ Academic Regulations document GR1:

58: “To remain re-enrolled on a course leading to a University award, you must

- ⦿ comply with any specific attendance requirements for your course

61: “You are expected to attend all-course-related activities unless you have a good reason for absence”

- ⦿ If you do not attend, you risk your enrolment at the university!

- ⦿ If applicable...

- ⦿ You also risk the Student Loan Company being notified

- ⦿ They will want their money back - all of it 😊

- ⦿ You also risk Immigration being notified

- ⦿ They might turf you out of the country!

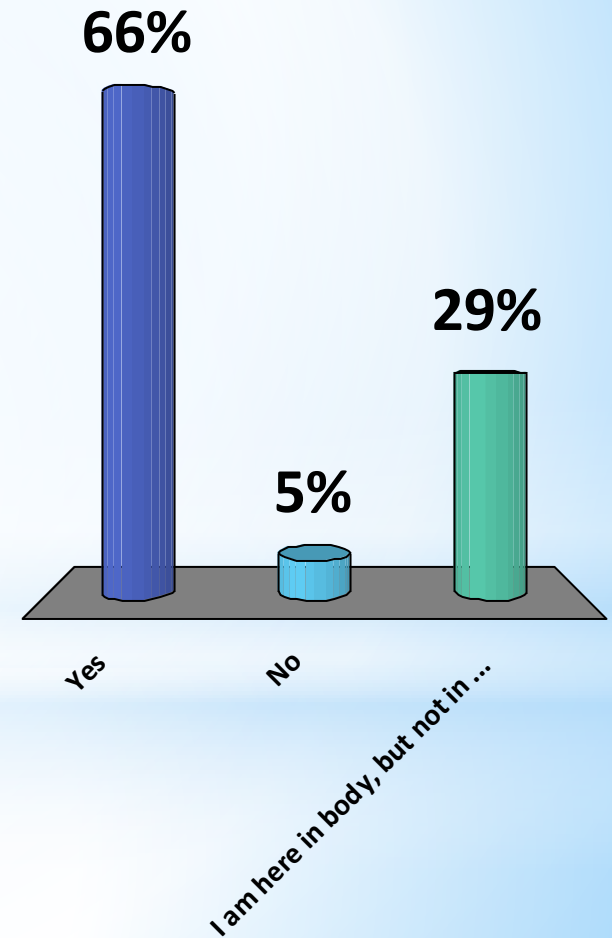
- ⦿ Turn up to your lecturers and workshops - not just for this module but for ALL modules! It's a no-brainer really...

Clickers or ARSes

- ⦿ We'll use clickers throughout the module during lectures
- ⦿ We'll use clickers to make the lectures a bit more interesting, but also (as a side effect) to monitor engagement and attendance
- ⦿ Little known fact: the correct term for “clicker” is *audience response system* or *ARS*
- ⦿ **NEVER FORGET YOUR ARS! 😊**
- ⦿ **BRING YOUR ARS TO EVERY CLASS!**
- ⦿ **IF YOU FAIL TO GET YOUR ARS TO CLASS, THEN OFFICIALLY YOU FAILED TO ATTEND!**
 - ⦿ (this could have repercussions for your student loan or lead to grief from immigration people)

Clicker test: Are you here?

- A. Yes
- B. No
- C. I am here in body, but not in spirit



What we expect from you

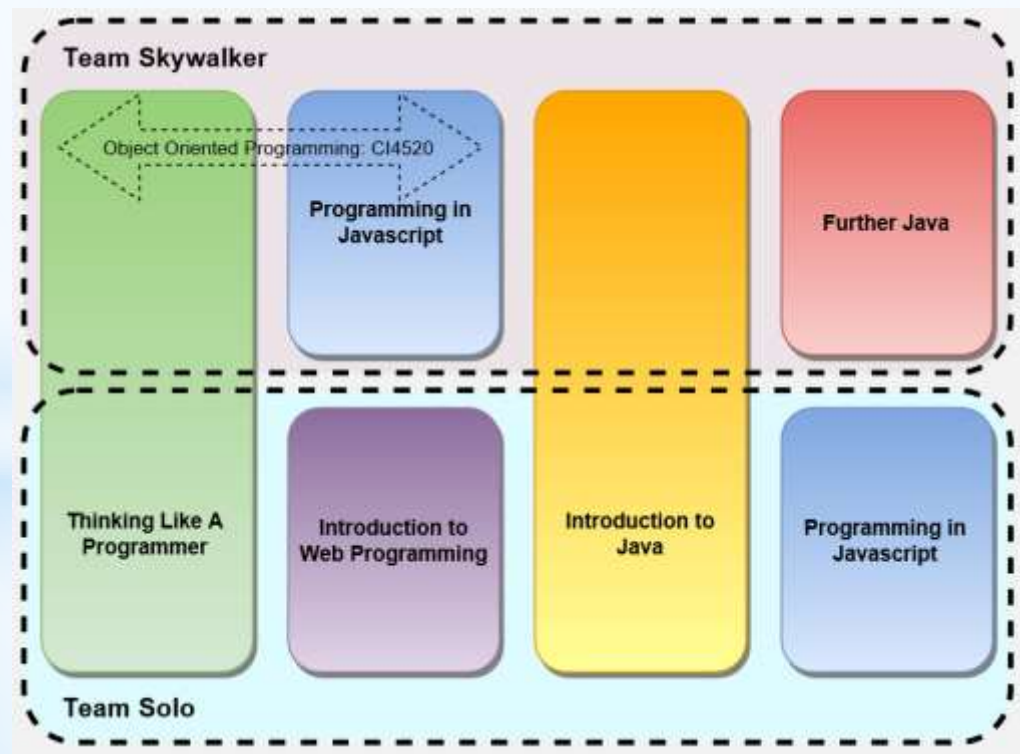
- ⦿ That you will do your OWN work!
 - ⦿ The assignments in this module are NOT group work, they are individual work
 - ✓ "I asked my friend for help, and then used their advice to create my own solution"
 - ✗ "My friend and I worked on the solution together and then both handed in the same solution we created together"
 - ⦿ This is *collusion*, and considered the same as plagiarism!
 - ⦿ Don't be tempted to take shortcuts
 - ⦿ If you get a solution from a friend and then submit it as your own, you will be *heavily* penalised!
- ⦿ IT IS BETTER TO GET ZERO FOR SOMETHING THAN TO CHEAT, GET FOUND OUT, AND GET ZERO PLUS AN ACADEMIC MISCONDUCT CHARGE!
 - ⦿ ...and you WILL be found out!

An interlude on note-taking

- ⦿ "...participants who had taken notes with laptops performed worse on tests of both factual content and conceptual understanding"
(Mueller and Oppenheimer, 2014)
- ⦿ "...those who took organized notes showed much less forgetting over a 24h delay than those who used a transcription strategy"
(Bui and Myerson, 2014)
- ⦿ NB:
 - ⦿ "organized notes" = handwritten
 - ⦿ "transcription" = typing what the lecturer says verbatim on a computer
- ⦿ So, bring **pen and paper** and MAKE NOTES!

How the module will operate

- ⦿ You will study four submodules
- ⦿ Each submodule lasts for approximately 5 weeks (half a teaching block)
- ⦿ The schedule for Programming 1 students is as follows:



How the module will operate

- ⦿ CI4100 Programming 1 students:
 - ⦿ On the Monday of *Enrichment Activity* week (EAW; this happens in week 6) you'll be assigned to one of the two teams, either *Team Skywalker* or *Team Solo*
 - ⦿ (no, there is no *Team Chewbacca*, you can't be in that!)
 - ⦿ You may change your assignment during that week and that week alone
 - ⦿ After this your team membership is fixed
- ⦿ CI4520 Object Oriented Programming students:
 - ⦿ For the duration of the time you spend with me in Teaching Block 1, consider yourselves honorary members of *Team Skywalker*...
 - ⦿ In Teaching Block 2, you go off to do C++ with Ahmed

How the module will operate

- ⦿ Most of the practical activities during the module will be done using our *NoobLab* environment
- ⦿ NoobLab lets you view workshop details, write your program code, and see the results right there in your web browser in a single place
- ⦿ We will see NoobLab a bit later...

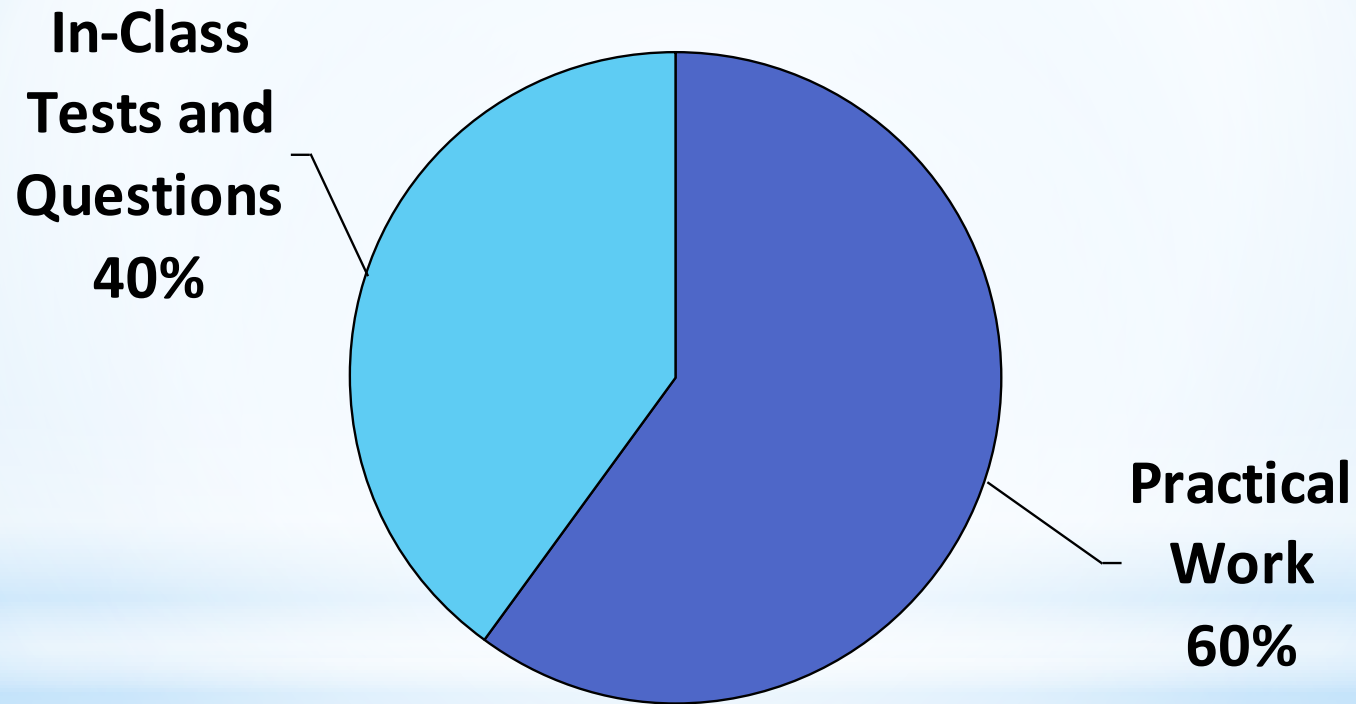
How you will be assessed

- ⦿ There will be practical workshop activities for you to do
 - ⦿ These can be done either during lab sessions or at home
 - ⦿ Each activity will result in you being awarded a "medal"
 - ⦿ There are three types of medals: gold, silver and bronze
 - ⦿ A gold is worth more than a silver, which in turn is worth more than a bronze
 - ⦿ Some activities have only a single medal possibility attached - others offer the opportunity to get a varying grade of medal depending on how good your solution is
 - ⦿ You can get help from tutors, workshop assistants or friends in class - but your solution to a medal activity **MUST** be your own work
 - ⦿ **IF YOU ARE CAUGHT USING SOMEONE ELSE'S WORK TO GET A MEDAL, YOU MAY BE GIVEN ZERO FOR THE ENTIRE MODULE!**
 - ⦿ NoobLab **KNOWS** when you are doing this...
 - ⦿ **...YOU WILL BE FOUND OUT!**

How you will be assessed

- ⦿ There will also be in-class questions which you will use your clicker to answer
- ⦿ These slides will stand out because they will be in orange
- ⦿ **THESE COUNT TOWARDS YOUR MODULE MARK**
- ⦿ If you forget your clicker - tough! No marks!
 - ⦿ (It's like turning up to a written exam without a pen. No-one is under any obligation to lend you one!)
- ⦿ **So, again, BRING YOUR ARS TO EVERY CLASS!**
- ⦿ Every lecture will probably have a couple of questions sprinkled in there for you to answer - so make sure you are paying attention throughout
- ⦿ At the end of each unit, we'll also have a final in-class test to check what you've learned over the last five weeks

How you will be assessed



Contacting the module team

- ⦿ In the first instance, email the specific lecturer or workshop leader that is dealing with the work you need help with
- ⦿ If you don't hear back from them, email the module leader
 - ⦿ For CI4100 it's Paul - paul@kingston.ac.uk
 - ⦿ For CI4520 it's joint leaders
 - ⦿ Ahmed Shihab - a.shihab@kingston.ac.uk
 - ⦿ Dave Livingstone - d.livingstone@kingston.ac.uk

Contacting the module team

- ⦿ If you need to come and see us, email first to request an appointment
 - ⦿ Always use the module code in the subject line
 - ⦿ CI4100 for Programming 1
 - ⦿ CI4520 for Object Oriented Programming
 - ⦿ (For Programming 1) Later, when you're in your groups, make sure you specify if you're *Team Skywalker* or *Team Solo*
 - ⦿ Give us as much information as you can about you and the issue you're emailing about - there are a *lot* of you 😊
 - ⦿ Look on the (many!) resources and information sources available to you first - chances are any questions have been answered already
 - ⦿ If I get an email that has been answered on Studyspace, or in class, you will simply receive a response saying "This question has already been answered"
 - ⦿ Remember we receive hundreds of spam mails that we delete immediately - don't make your email look like one of them!

One last thing...

- ⦿ YOU are responsible for YOUR learning!
- ⦿ You will NOT be spoon-fed!
- ⦿ You will never be told what to do, step-by-step.
- ⦿ You are going to have to think for yourself!
- ⦿ We will expect that you can read Studyspace, that you are paying attention in class when I tell you about upcoming dates and events, and will take responsibility for being in the right place at the right time
- ⦿ If you do not attend a lecture, or if you weren't paying attention - TOUGH. YOU are the one who will lose out because of it.
- ⦿ Last word: **THIS IS NOT SCHOOL**

programming: the basics

Things to know about programming

- ⦿ Learning to program is about learning how to solve problems
- ⦿ Learning how to write programming language code should come **after** we have learnt how to solve problems
 - ⦿ I will often refer to this skill as "Thinking Like A Programmer"
- ⦿ So, the first part of this module focuses on analyzing, understanding, and solving problems

Everyday programming

- ⦿ A lot of everyday activities use some of the skills needed to program a computer:
 - ⦿ Cooking a meal for guests: you need to know how many guests so that you can work out what quantities of ingredients are needed; you need a method for making sure everything is cooked and ready at the right time
 - ⦿ Driving somewhere new: you need to plan a route, estimate the duration of the journey, decide whether the car needs more fuel
 - ⦿ Getting dressed: where are you going today? What clothes are appropriate? - you don't wear the same clothes to graduation as you would to a beach party; what order do you put your clothes on

Programs as recipes

- ⦿ A computer program is like a recipe. A good recipe tells you:
 - ⦿ what ingredients (and their quantities) are needed
 - ⦿ how to prepare the ingredients
 - ⦿ in what order the ingredients must be added
 - ⦿ what temperature the oven should be set to
 - ⦿ how long to cook everything
- ⦿ A recipe has a clear method. An ordered series of steps that must be followed exactly to get the right result. Recipes are **repeatable: do it the same each time, get the same results**
- ⦿ A program tells a computer what steps to carry out and in what order in order to get the correct result

Things to know about programming

- ⦿ You may hear the word **algorithm** used when people talk about programs or programming

algorithm

noun

Word used by programmers when they do not want to explain what they did

Things to know about programming

- ⦿ You may hear the word **algorithm** used when people talk about programs or programming

algorithm

noun

A step-by-step procedure for calculations, or to solve a specific problem

- ⦿ i.e. our "recipe"!

A simple "program"

- ⦿ "Walk across the room"

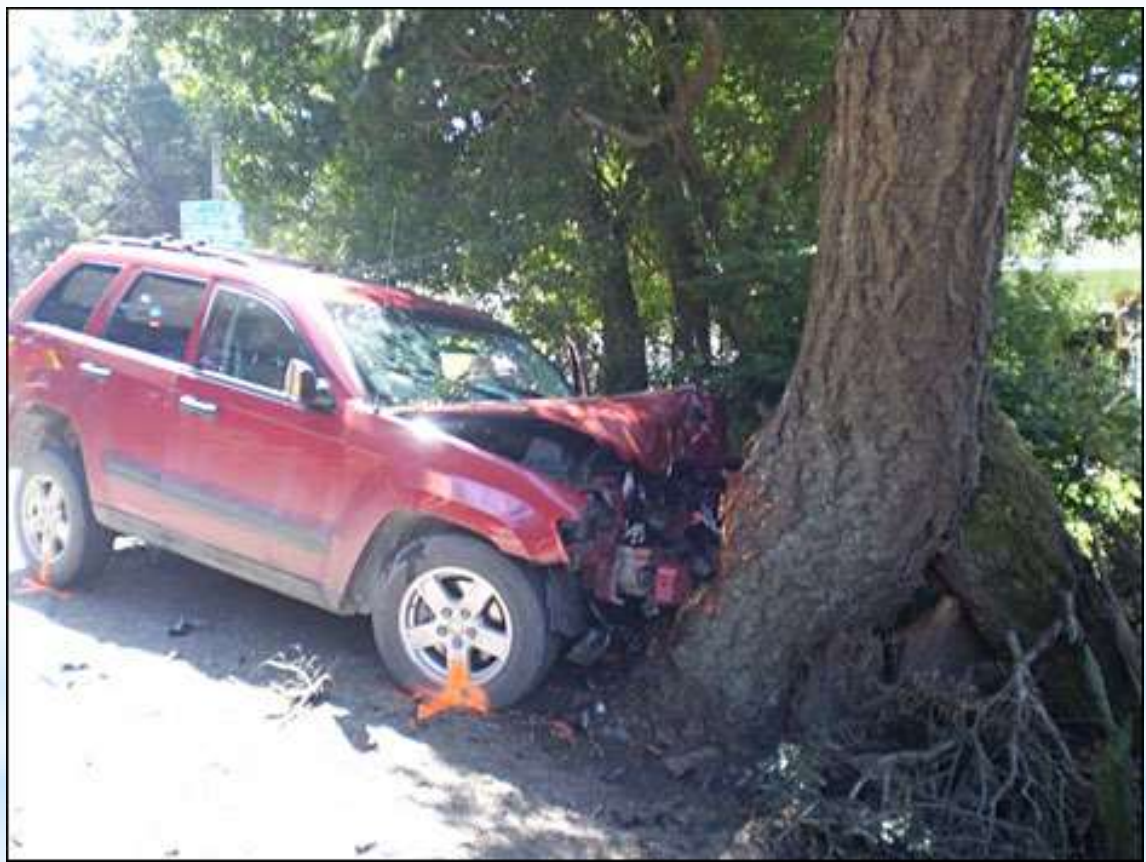
Implicit or "hidden" statements in natural language

- ⊙ When we give a set of instructions in natural languages the average human will make many assumptions about what is meant
- ⊙ "Walk across the room"
 - ⊙ What if there is an obstacle in the way?
 - ⊙ Most people would assume that the obstacle should be avoided - but a computer would not!
 - ⊙ If you were giving the same command to a computer, you would have to explicitly state that the obstacle should be avoided

Another example problem



- ⦿ How would you give instructions to drive from the restaurant to "home"?
- ⦿ One example might start off by saying
 - ⦿ "Start from the restaurant, drive down the road and turn right at the first lights..."



The second law of computers*

Programming is like trying to give directions to a really, really, *really* stupid person

* A prize goes to who can tell me the *first* law of computers...!

Ambiguity

- ⊙ Natural language can be ambiguous (ask a lawyer), and one idea can be expressed in multiple ways:
 - ⊙ “empty the bin when it is full”
 - ⊙ " “if the bin is full then empty it”
 - ⊙ “should the bin run out of room then you must empty it”
- ⊙ Programmers must express themselves concisely and clearly:
 - ⊙ computers will not interpret what we say or try to infer what we mean
 - ⊙ e.g. did we mean "traffic lights", or "street lights", or "lights in the sky that human beings call stars" when we said "turn right at the lights"
 - ⊙ " we must tell them **precisely** what we want them to do
 - ⊙ " computers will follow your instructions to the letter **even if the instructions are obviously wrong!**
 - ⊙ computers will follow your instructions to the letter **even if obvious things are missing!**

Planning out a program

- ⦿ When we write a program, we should **first solve the problem of deciding how the task** may be carried out
- ⦿ Only when we've got a plan for the "recipe" or program and understand the steps and "ingredients" involved do we turn the solution into something the computer can interpret.
- ⦿ This planning before any programming or creation of code takes place is **crucial**
- ⦿ If we don't know how the task may be accomplished in the first place, how can we write program code to achieve it?!

Programming is about communicating processes

- ⦿ A program is basically a concise statement that communicates a process
 - ⦿ Some say that a *good* program is the most *concise* statement possible!
 - ⦿ At beginner's level, a "good" program is probably one that works 😊
- ⦿ The ability to think like a programmer is an important skill in the real world...
 - ⦿ ...to communicate efficiently, without ambiguity, in a clear way in as few words as possible
 - ⦿ ...to work with and manage computer scientists!

Programming is also about eliminating assumptions and ambiguity

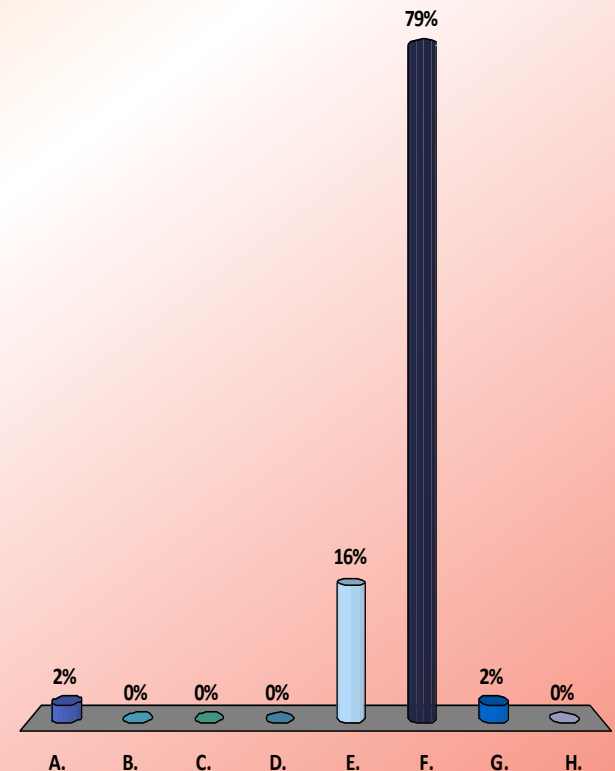
- ⦿ When you create your solution to a problem, you must identify any assumptions you are making and remove them
 - ⦿ "turn right at the lights"
 - ⦿ ...assuming "lights" will always mean "traffic lights"
- ⦿ Assumptions or ambiguities might also come from the person asking for the program (the "specification")
 - ⦿ "Write a program to calculate interest on a bank account"
 - ⦿ What formula should be used?
 - ⦿ When is interest paid?
 - ⦿ Is there tax payable?
- ⦿ If the specification has too many assumptions or ambiguities
 - ⦿ Option 1: go back to the person who is asking for the program (the "client") and ask them to clarify
 - ⦿ Option 2 (when client unavailable): clearly choose one possible scenario and stick to it - but document what you have chosen

Making a sandwich:

First, get the bread. Then, get the filling. Butter both sides of the bread. Put the filling between the two slices. Then enjoy your sandwich!

What is wrong with the above sandwich “program”?

- A. It doesn't specify where to get the bread and filling from
- B. It doesn't specify what is meant by “butter”
- C. It doesn't specify what should be done if there any ingredients are missing
- D. It doesn't say that you should join the two slices of bread together at any point
- E. All of the above
- F. All of the above plus other problems too!
- G. None of the above
- H. There's nothing wrong with these instructions - what are you on about, Neve?!



Programming is about starting small and building up

- ⦿ Programming is all about breaking a (large) problem down into smaller problems
- ⦿ Solve the smaller problems individually and you'll end up with a solution to the larger problem
- ⦿ If you can acquire this skill, programming will become easy - even fun!
- ⦿ Without this skill, programming will be a mystery!
- ⦿ The first part of this module is all about developing this skill

The structure of a program is like lego

- ⦿ Computer programs are like lego
 - ⦿ There are different types of blocks
 - ⦿ The blocks fit together in certain ways
 - ⦿ Some combinations fit together, some don't
 - ⦿ The combination of blocks fitted together ends up as a sum greater than the whole of the parts
 - ⦿ The skill of programming is knowing how to fit and arrange the blocks so that the combination solves your problem
 - ⦿ Different blocks and combinations of blocks map onto the different aspects of your programming problem

Programming languages

- ⊙ Programming languages are very logically precise and make it harder* to issue statements that are ambiguous or rely on implicit assumptions
- ⊙ However, most programming languages are very strict about structure, grammar and punctuation
- ⊙ Consider in English
 - ⊙ "Much to learn, you still have"
 - ⊙ "you, Stlll; HAVE much. To learN"
- ⊙ Despite mangling grammar in the first example and punctuation in the second, an English speaker should still be able to decipher the intent behind these statements

Programming languages

- ⦿ But programming languages are not so forgiving:
- ⦿ **System.out.println("Hello there");**
- ⦿ system.out.println("Hello there");
- ⦿ SYSTEM OUT PRINTLN "Hello there"
- ⦿ System.out.println(Hello there);
- ⦿ System.Out.Println("Hello there");
- ⦿ System.out.println("Hello there);
- ⦿ System.out.println("Hello there")

Programming languages and novices

- ⦿ I think the worst thing for a novice to use when learning how to think like a programmer is a programming language!
- ⦿ In the beginning, the important thing is for you to learn how to break problems down, create solutions, and think logically
- ⦿ This is difficult enough without a programming language complaining about missing semi-colons or brackets in your code in a bizarre inscrutable way
 - ⦿ "Parse error: Syntax error, unexpected T_ECHO"
- ⦿ Many novices get frustrated and turned off because of this before ever acquiring the skill of thinking like a programmer
- ⦿ We don't want this to happen to you!


Introducing NoobLab

- ⦿ KU has its own online learning environment for programming called NoobLab
- ⦿ NoobLab lets you do programming activities in your web browser from any computer, either in the uni or elsewhere
- ⦿ NoobLab has a variety of features that are designed to make learning programming easier and more fun

NoobLab quick reference

Arrays

We saw in the lecture that if a variable is like a box, an array is like a box with compartments:



Compose pseudocode that will create an array that has five elements. The variable name for the array should be `numbers`, and you should place the numbers into the array as follows:



- The first element should contain 10
- The second element should contain 20
- The third element should contain 30
- The fourth element should contain 40
- The fifth element should contain 50


Note that the specification doesn't specify that you should print the values of the array out to the screen (although you can if you want to). It is sufficient to write the pseudocode to set up the array correctly as described.

There are two awards available for this exercise. You will get the bronze medal for correctly creating and populating the array with the correct data. You will, however, get a gold medal if you can populate the array *without* having 5 individual statements that assign the 5 elements' values one by one, by using a `for` loop.

[>>> Click here to test your code for a bronze medal <<<](#)

[>>> Click here to test your code for a gold medal <<<](#)

 Lesson Navigation: 

 Options

Thinking Like a Programmer 3

nooblab.kingston.ac.uk/NoobLab/contents/paulneve.com/programming1/tlap3#1

Run Stop Load file Save file Clear editor

`1 declare x[5]
2 for i = 1 to 5
3 set x[i-1] = i*10
4 display i*10
5 endfor`

`10
20
30
40
50`

Program successfully completed its initial run.

NoobLab quick reference

The screenshot shows the NoobLab web interface. On the left, a sidebar contains a lesson titled 'Arrays' with a diagram of an array named 'box' containing five compartments labeled 'box[0]' through 'box[4]'. The main area displays a programming exercise with pseudocode instructions and a code editor. The code editor contains a BASIC-style program that declares an array, loops from 1 to 5, sets each element to its index multiplied by 10, and displays the value. Below the code editor is an output window showing the results: 10, 20, 30, 40, 50, and a success message. At the bottom, there are buttons for testing code for medals and a lesson navigation bar.

The general content and/or instructions for a workshop appear here

Use these buttons to run, save and load your programs

Type your program code here

See what output your program produces here

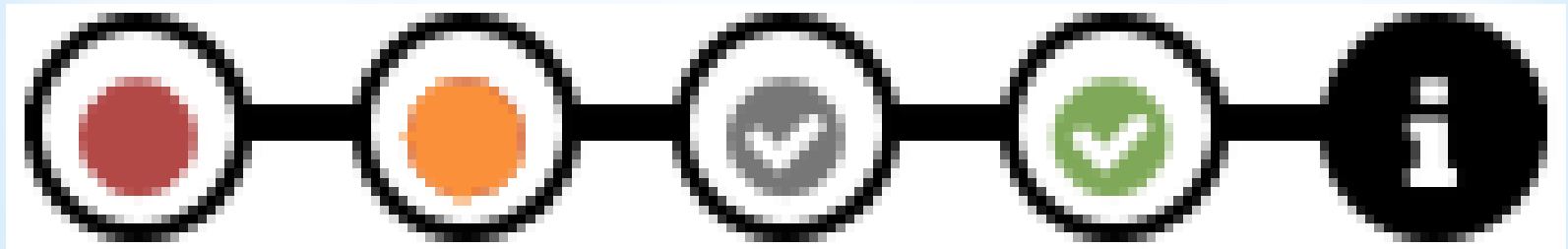
Click links like these to try for a medal

See your current medals and the lecture slides here

Navigate to the different parts of a workshop here

NoobLab navigation icons

- ◉ The current page is highlighted in black
- ◉ There are different icons which tell you activities are still left to do on a given page of a workshop:



**RED
CIRCLE**

No medal
activities
attempted
yet

**ORANGE
CIRCLE**

Some medals
completed
but still
some left
to do...

**GREY
TICK**

All of the
activities
have been
done, but
not to the
gold medal
standard...

**GREEN
TICK**

All of the
activities
have been
done to the
gold medal
standard...

**INFORMATION
PAGE**

These pages
have no
medal
activities
(but you still
need to read
them!)

Introducing Carol

- ⦿ Meet Carol - the brain dead virtual robot with a shopping trolley
- ⦿ Carol lives in a grid of squares
- ⦿ Each square can contain either a wall, a number or "item", or the "goal"
- ⦿ Carol can move one square forward, turn left, pick up an item and put down an item
- ⦿ She can also detect whether her path directly ahead is blocked, whether she is at the goal, whether there is an item currently in sight and how far away such an item is
- ⦿ Your job will be to compose the commands required to navigate Carol through mazes and get her to the goal
- ⦿ Later, you will also have to re-arrange the list of items into a new order before going to the goal



Carol control blocks

- ⦿ To begin with, we will compose programs to solve Carol challenges using a drag-and-drop block based system
- ⦿ Demo and run through this week's workshop exercises...

About medals

- ⦿ Most exercises in NoobLab will result in you winning a "medal"
- ⦿ Medals count towards your final mark!
- ⦿ There are three grades of medal: gold, silver and bronze
 - ⦿ The level of medal is dependent on how hard an activity is or how good your solution is
- ⦿ Some activities will give you the choice between (up to) three possible challenges - an easy one, a medium one and a hard one
 - ⦿ Beat the easy one, you get bronze
 - ⦿ Beat the medium one, you get silver
 - ⦿ Beat the hard one, you get gold
- ⦿ In these activities, you can only win ONE of the medals
 - ⦿ e.g. if you do the bronze and then go on and win the silver, you only get to keep the silver
 - ⦿ (Usain Bolt doesn't get bronze, silver AND gold after all!)

About medals

- ⦿ Other activities are based around a single challenge
- ⦿ Depending on how hard the single challenge is, you'll get a one-shot bronze, silver or gold medal to add to your tally

On working together and "assists"

- ⦿ Notwithstanding the issues of collusion, we don't want you to work in complete isolation, never talking to any of your fellow students
- ⦿ Bouncing ideas off of other programmers and discussing your work is an important part of learning about programming
- ⦿ So - we have put into place a means where you can acknowledge the help other students gives you
- ⦿ These are called "assists"

How Assists Work

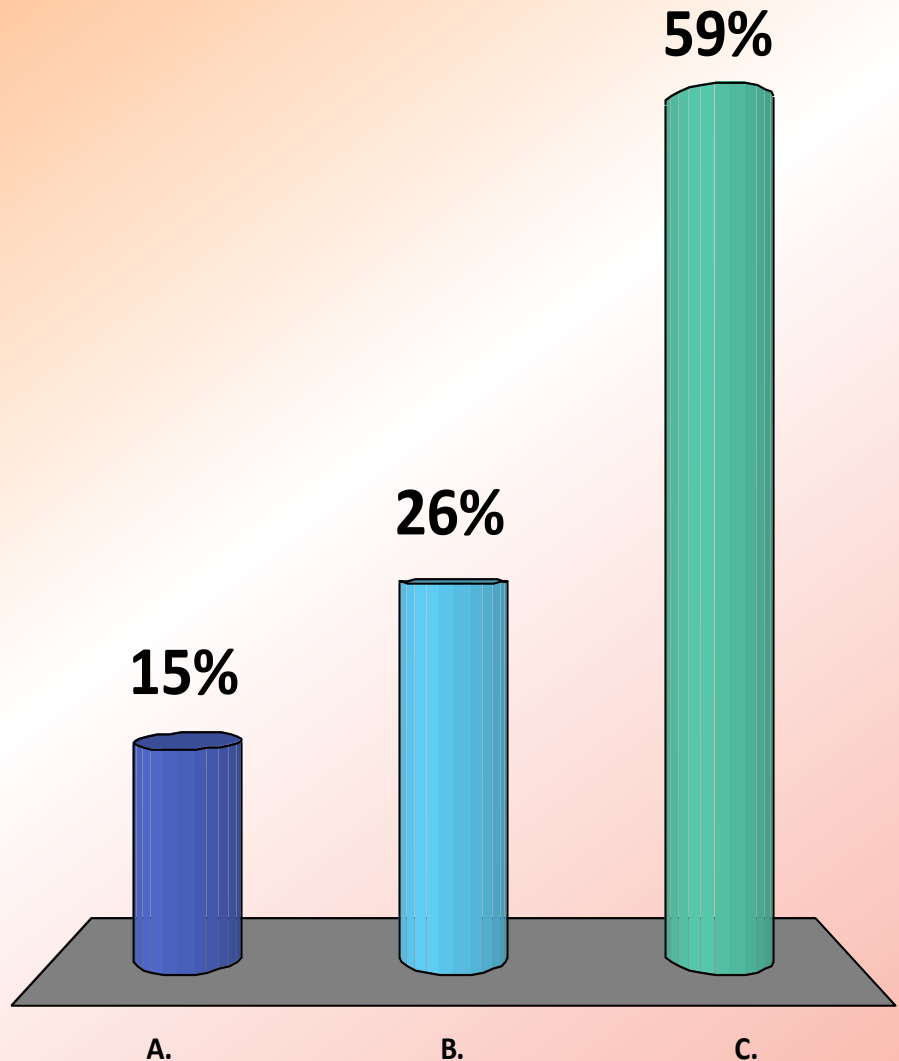
- ⦿ When you win a medal, you have the opportunity to award some of the points you gain to another student - this is called an "assist"
- ⦿ Depending on the medal, you'll be able to award up to three points to the person(s) who assisted you
 - ⦿ You can award assist points to more than one person - if the medal has enough points to go around, of course...
- ⦿ If you are the recipient of those points - the "assistant", i.e. the person who helped your colleague get the medal - then these count towards your final mark!

Why should I award assist points?

- ⦿ People are more likely to help you out if you're stuck!
 - ⦿ If someone helps you win a gold medal that you couldn't have won without the help, even if you give them some of your points you'll still be better off than if you'd got nothing for the exercise!
- ⦿ People won't help you if you get a reputation for being stingy with your assist points

Assists: Which is permissible?

- A. “Can I please have your finished solution to workshop 3, exercise 2? I'll give you some assist points.”
- B. “Let's all work on this activity together and create one single solution. Then I'll submit it for a medal on my screen and give the rest of you assist points.”
- C. “Hey, you've got the medal for workshop 3, exercise 2! How did you manage to get Carol to figure out whether she'd reached the goal once she got through the gap in the wall? If you help me out I'll give you some assist points.”



Assists: The rules

- ⦿ Plagiarism is STILL plagiarism!
 - ⦿ "Can I please have your finished solution to workshop 3, exercise 2? I'll give you some assist points"
 - ⦿ This is academic misconduct!
 - ⦿ best case scenario you'll both be capped for the unit at 40%
 - ⦿ worst case scenario you get expelled from the university...!!!!
- ⦿ Collusion is STILL collusion!
 - ⦿ "Let's all work on this activity together and create one single solution. Then I'll submit it for a medal on my screen and give the rest of you assist points"
 - ⦿ This is academic misconduct!
 - ⦿ best case scenario you will be capped for the unit at 40%
 - ⦿ worst case scenario you all get expelled from the university...!!!!

Assists: The Rules

- ⦿ We want you to learn from each other!
- ⦿ We don't want you to do someone else's work for them!
- ⦿ By all means help your colleagues - give them pointers, discuss how you did things with them, brainstorm with them
- ⦿ BUT...
- ⦿ When all's said and done you **MUST** do your own work! You can get help, but what you submit for medals has to be your own work.

Assists: one final caveat

- ⦿ The whole assist system is a new thing for this year
- ⦿ We want to encourage collaboration, a lively workshop environment where people learn from each other
- ⦿ We want to see no plagiarism or collusion, though!
- ⦿ If the assist system is abused we reserve the right to either deduct the points awarded or, if it comes to that, to turn it off for the entire module

Summary

- ⦿ This module will be very much a practical, hands-on affair
- ⦿ Most activities will use NoobLab, and you'll do practical programming activities
- ⦿ These activities win you "medals", which count towards your final grade
- ⦿ You will be assessed through these practical programming activities and in-class questions
- ⦿ Questions will happen during in-class tests but also there will be a few each week during lectures.
- ⦿ Attendance is key! Every time you don't turn up you are throwing marks away!
- ⦿ Do your own work! Get help if you need it, but your solutions must be your own.

Summary

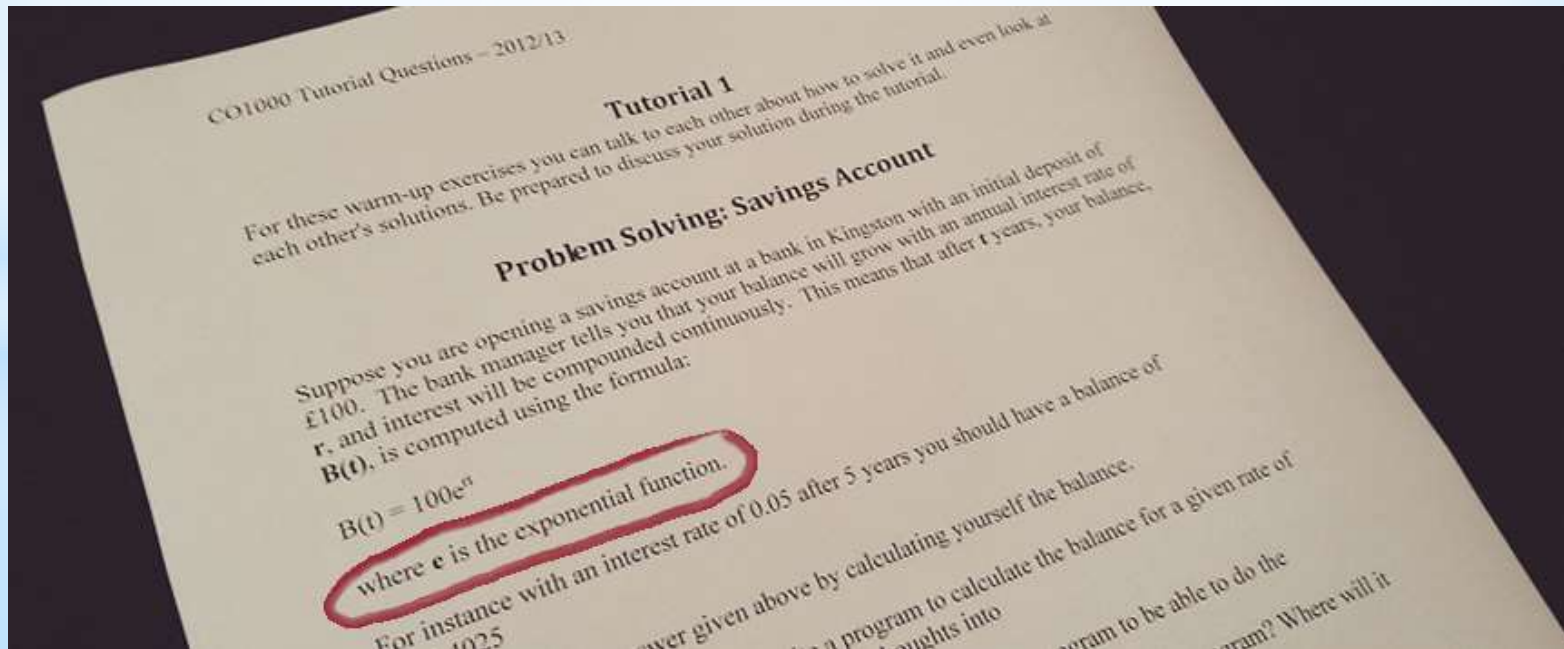
- ⦿ Programs are like recipes
 - ⦿ They describe a series of ingredients, what you do to those ingredients, the way in which the ingredients fit together and the order in which things need to happen
 - ⦿ Because computers are dumb, we have to explain *everything*
 - ⦿ We cannot make ANY assumptions at all
 - ⦿ If there are any implicit assumptions in our "recipe", we must eliminate them first - either by consulting our "client" (the person asking for the program) or by making a decision ourselves

Summary

- ⦿ The first thing you need to do is to learn to "think like a programmer"
- ⦿ This is a very specific, logical way of thinking (if not particularly *human*!)
- ⦿ It involves
 - ⦿ Being able to break a problem down into smaller parts
 - ⦿ Being able to create a set of instructions that has no ambiguity or assumptions

Epilogue: Why "control blocks" and Carol?

- ⊙ Programming courses often start off throwing you in using an actual programming language, with a complex syntax
- ⊙ Programming courses often involve rather abstract, sometimes mathematical problems...
- ⊙ For example...



Epilogue: Why "control blocks" and Carol?

- ⦿ In the early stages we are trying to teach the skill of "thinking like a programmer" - this is key
 - ⦿ It is important* that the problems you solve as you practice and learn this skill are not so abstract or mathematical that you can't easily visualise them
 - ⦿ It is important* that the tools you use to practice and learn this skill do not have other complexities or conventions to learn that will get in the way of your learning
 - ⦿ e.g. *Unexpected T_ECHO* style error messages...!
 - ⦿ It is important* that you realise that programming does not have to be mystifying!
 - ⦿ It is important* that you have fun while you learn!
- * well, *I* think it's important anyway.

Epilogue: Why "control blocks" and Carol?

- ⦿ "But I already know how to program in Java/C/Visual Basic/Asgardian Machine Language - this block/Carol rubbish is going to do my head in"
- ⦿ Maybe - but being able to visualise what can go where and what can *fit* where is a crucial part of being a *good* programmer
- ⦿ We don't just want you to be programmers - we want you to be *good* programmers
- ⦿ The blocks represent the constituent parts of a program
- ⦿ Being able to visualise and know instinctively how a program structure fits together, how the bits of code link into place and what fits where is a crucial skill
- ⦿ Using the blocks will let you learn and/or expand this skill without the drag factor of your existing assumptions and/or knowledge