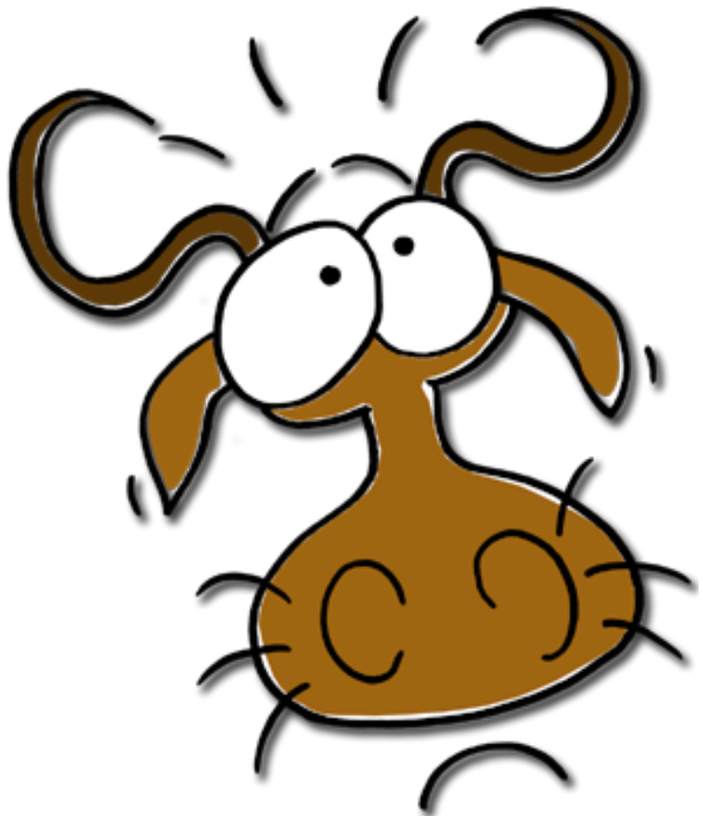
How to THINK like a Programmer

Problem Solving for the Bewildered

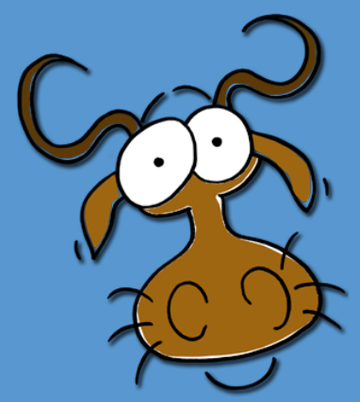
paul vickers



chapter 1

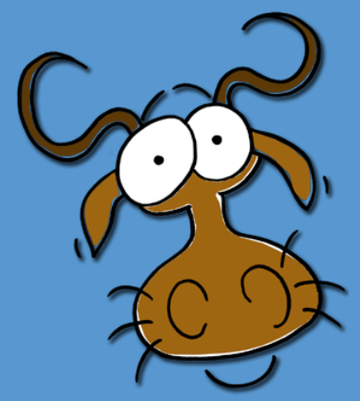
starting to think like a programmer

Purpose

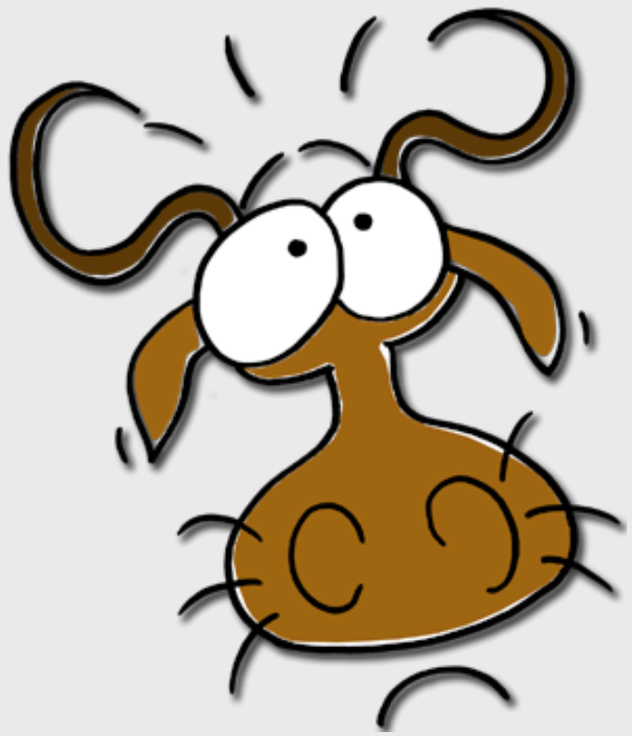


- This chapter presents an overview of the course:
  - The problem with programming
  - How to use the book
  - Programs & algorithms
  - Thinking like a programmer
  - Abstraction
  - Pseudo-code

Overview

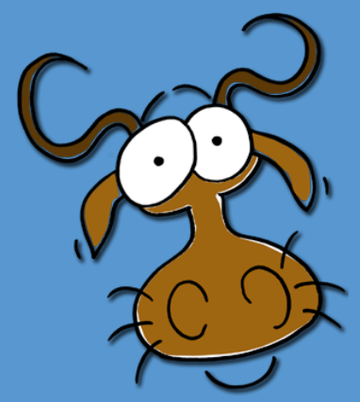


- Too often, learning to program is like this:



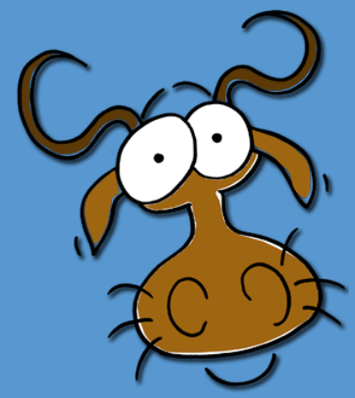
- The blame is placed on a ‘difficult’ programming language
- The trick is to separate coding from problem-solving

Overview



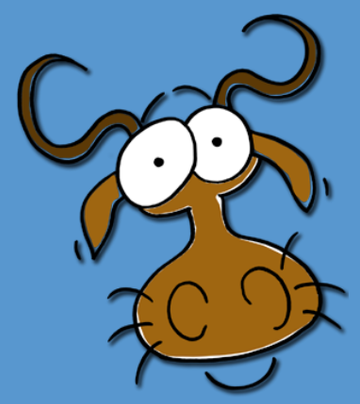
- Learning to program is about learning to solve problems
- Learning how to write programming language code should come **after** we have learnt how to solve problems
- This course focuses on analyzing, understanding, and solving problems in an **algorithmic** way
- The course uses **no** programming language
- Programming does not need to leave you feeling bewildered

How to use the book



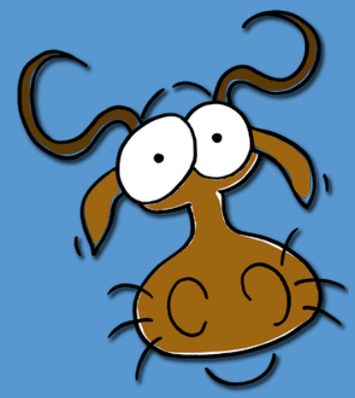
- Thinkspots
  - The Think Spot is a point in the text where a question (or a number of questions) is raised for you to think about. To get the most benefit you should take a little time to think about the questions rather than just reading them and moving straight on
- In-text exercises
  - A picture of a pencil in the margins alongside some bold text in a shaded box denotes a short exercise. You should not proceed beyond an exercise until you have had a reasonable attempt at solving it
  - If an in-text exercise has a key in the margin, then a solution is also available in Appendix C

How to use the book



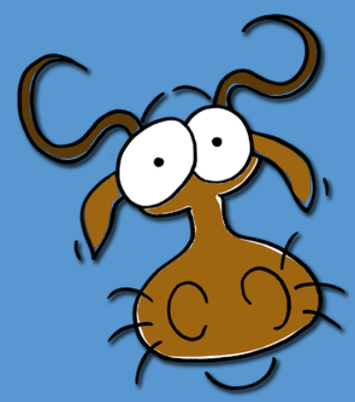
- Key terms and important points
  - Key terms and important points appear in the margin next to the paragraph or section in which they are introduced or defined. Together with the index this feature should make it easier for you to find what you are looking for
- Brian Wildebeest FAQ
  - Throughout the book you will see pictures of Brian looking bewildered in the margin. He appears at points where beginning programmers often have trouble understanding the point being made. Further down the page (or possibly on the next page) you will find a box like this
  - In the box is a question Brian is asking about the material. Brian's queries are the FAQ I have been asked over the years. The answer to the FAQ usually appears in the box below the question, though sometimes you are required to try and answer Brian's FAQ yourself

How to use the book



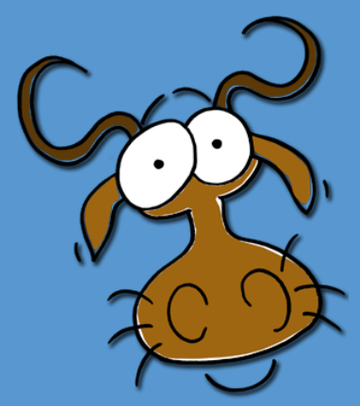
- End of chapter exercises
  - The exercises are designed to help you reflect on what has been covered in that chapter. Any time you cannot complete an exercise suggests that you would benefit from going over the material again. Some exercises may be based on a single section (identified by the heading number), others may require ideas from several sections, and a few bring together the whole chapter. Solutions to selected exercises are given in Appendix C
- Projects
  - After the normal end-of-chapter exercises you will find some longer project-style exercises. These longer exercises are themed and will give you practice in incrementally building larger and larger solutions to more complex problems

How to use the book



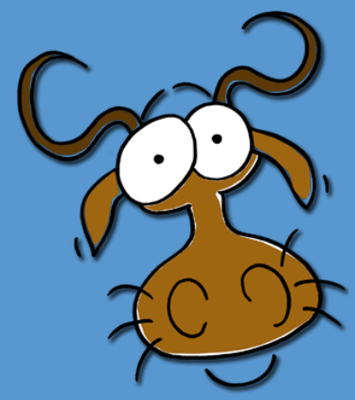
- Reflections
  - Sometimes you will see a word or phrase set in SMALL CAPITAL LETTERS. Such words and phrases are the titles of short reflective opinion pieces which appear in the Reflections chapter (immediately following Chapter 8). These Reflections are designed to introduce some more complicated ideas that the interested reader can use to deepen their understanding of some of the problems and issues faced by programmers today.
- Online resources
  - Visit http://www.cengage.co.uk/vickers for online resources including a dynamic FAQ section, solutions to **all** exercises, etc.

Why program?



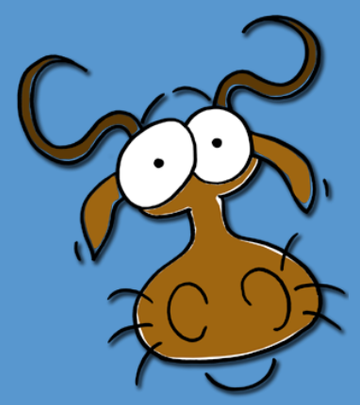
- We use computers to
  - Calculate answers to complex mathematical equations
  - Manage the download of and payment for digital music
  - Schedule classes in a university
  - Store and edit documents/books/assignments
  - Edit home videos
  - Make music
  - Browse the world wide web
  - etc. etc.
- The computer needs to be told how to do these tasks
- Telling the computer what to do and how to do it is called **programming**

Programs



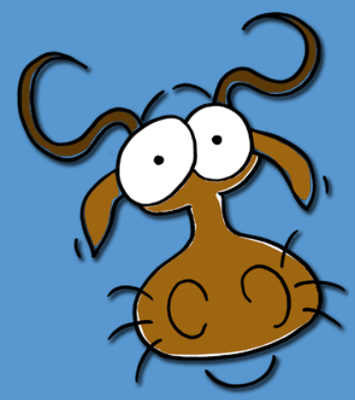
- The instructions computers follow are called programs
  - An ATM has a program to tell it how to dispense cash
  - A word processor is a program that tells the computer how to store documents and allow you to edit them
  - The computer (microchip) in your iPod has a program to give you access to your music and video files
  - Your mobile phone has programs to enable it communicate with the cellular network and place & receive phone calls, SMS messages, voicemail etc

Programming



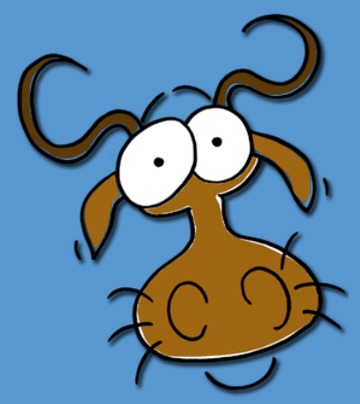
- Before the computer can do what we want we must tell it **how** to perform its task
  - We do this by **first** solving the problem of deciding how the task can be carried out
  - then expressing that solution in a form that can be turned into something the computer can interpret.
- The first stage (deciding how the task can be carried out) is the really important part as without doing this well we cannot progress to feeding the information into the computer
- Many beginners have a sense of awe, as if a program is some mystical artefact that only those initiated into the dark arts of computer science can produce

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

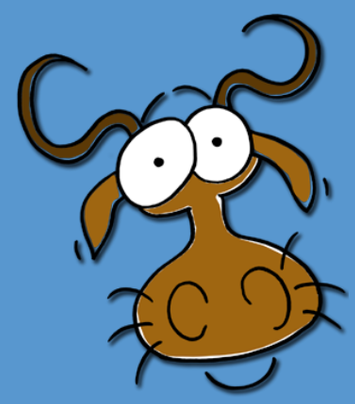


- Computer program is like a recipe. A good recipe tells you:
  - what ingredients (and their quantities) are needed
  - how to prepare the ingredients
  - in what order the ingredients must be added
  - what temperature the oven should be set to
  - how long to cook everything
- A recipe has a clear 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

ACTIVITY

If you have never done so, go and find a cookery book and look at some of the recipes. You will see there all the things you have to do in order to prepare various meals.

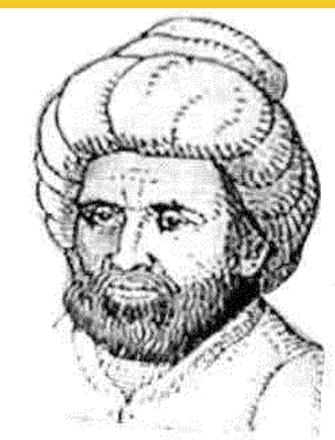
Thinking like a programmer



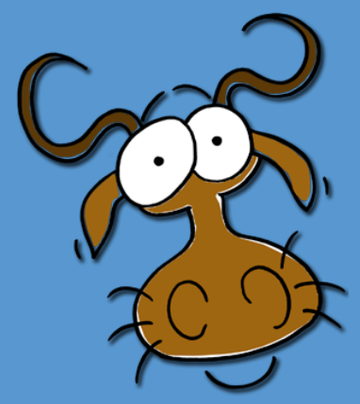
- To be a programmer you need:
  1. To understand how to examine and analyze problem statements
  2. To understand the component parts of a problem
  3. To understand what is required (what the outcome should be)
  4. To solve the problem
  5. Check the solution for mistakes
  6. Write down your solution in a way that it is clear and easily followed (like a good recipe)
- Only after doing 1–6 above should you consider translating your solution into programming language code (C, Java, C#, C++, etc)
- This course deals with steps 1–6

Algorithms

- A rule, or a finite set of steps, for solving a mathematical problem
- In computing it means a set of procedures for solving a problem or computing a result
- The word ‘algorithm’ is a derivation of Al-Khwarizmi (native of Khwarizm), the name given to the ninth century mathematician Abu Ja’far Mohammed ben Musa who came from Khwarizm (modern day Khiva in the south of Uzbekistan)
- This course is about learning how to understand problems and design algorithms that are solutions to those problems

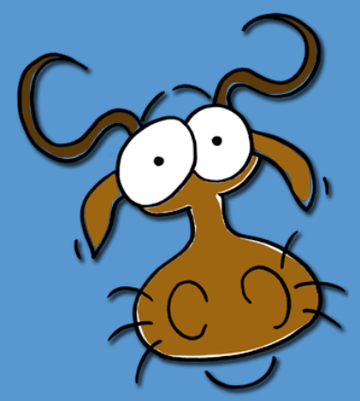


Key idea: ABSTRACTION



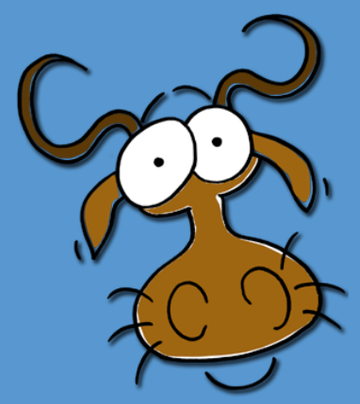
- **abstraction** is a cornerstone of programming
- Programmers do it all the time: it's all about the level of view we take
- 'I drove my **car**' this morning:
  - 'car' is an abstraction for the particular car I own
  - It's a way of referring to something without revealing all the details
- A map is a high-level abstraction of a piece of land:
  - Not every tree is shown
  - Probably houses are not shown
  - The animals and insect life almost certainly are not shown

Abstraction



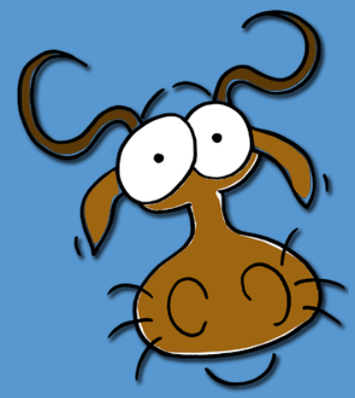
- We use abstraction in everyday life as a way of coping with the infinite detail of reality, e.g.:
- ‘The money in my pocket’
  - how much, what currency, how many coins, what year were they minted, how many notes, their serial numbers, etc.
- ‘The people in the shop’
  - how many men, women, boys, girls, what are their names, nationalities, ethnic groups, ages, heights, educational qualifications, first languages, etc.
- ‘The stars in the sky’
  - where to begin???
- These are abstract ways of managing an otherwise unmanageable amount of information

Abstraction in programming



- Programmers use abstraction to simplify or hide detail (often we **defer** dealing with the detail until later)
- **Control abstractions:**
  - Carrying out tasks and operations
- **Data abstractions:**
  - Describing the data and items of information needed to solve a problem
- This course starts with fairly general abstractions and refines them as it progresses (taking us nearer and nearer to a real programming language)

Pseudo-code



- This course uses a special notation called **pseudo-code** for writing down solutions to problems
- Pseudo-code is a form of structured English:
  - It has features that resemble real programming language code
  - But it retains sufficient natural language to allow solutions to be expressed without needing to understand the precise details of a programming language
- The course starts with a fairly loose (highly abstract) pseudo-code and progressively formalizes the notation with each chapter
- This allows us to progress seamlessly from acquiring general problem-solving skills to using quite precise language which is very close to real code

end of chapter 1