Universitas Esa Unggul

Smart, Creative and Entrepreneurial

www.esaunggul.ac.id

CMC 101 TOPIK DALAM PEMROGRAMAN

PERTEMUAN 1

PROGRAM STUDI MAGISTER ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER

TOPIK DALAM PEMROGRAMAN

Pertemuan 1

PERKENALAN

- Nama : Dr. Gerry Firmansyah S.T M.Kom
- Kontak : gerryfirmansyahs2@gmail.com
- Mobile : 0811-8111-610 (WA,SMS,LINE,KAKAO)
- S3 - Computer Science , Universitas Indonesia
- S2 - Magister TI, Universitas Indonesia
- S1 - Informatika, ITB
- D3 - Teknik Komputer, Politeknik ITB

Topik Research

- E-Government : GEAF
- E-Governance : IT Audit, COBIT, IT Risk
- Smart Mobility : VR, Smart device
- Geographical Information System

Cakupan Perkuliahan

1. berpikir komputasional
2. paradigma pemrograman
3. paradigma fungsional (bahasa Haskell)
4. paradigma deklaratif (bahasa Prolog)
5. paradigma berorientasi objek (bahasa C++)
6. paradigma berbasis *event/ reactive* (java)
7. Problem solver dgn paradigma yang sesuai
8. paradigma prosedural (bahasa Pascal)
9. Problem solver dgn struktur data yang sesuai
10. analisis terhadap efisiensi suatu algoritma
11. Notasi O
12. Teknik Greedy dan Dynamic Programming

Buku Acuan

- Computational Thinking for the Modern Problem Solver (Chapman & Hall/CRC Textbooks in Computing) 1st Edition, David D. Riley &Kenny A. Hunt, Chapman and Hall/CRC; 1 edition (March 27, 2014)
- Programming Languages: Principles and Paradigms (Undergraduate Topics in Computer Science), Maurizio Gabbrielli, Simone Martini, Springer; 2010 edition (April 15, 2010)
- Learn You a Haskell for Great Good!: A Beginner's Guide 1st Edition, MiranLipovaca, No Starch Press; 1 edition (April 21, 2011)
- Prolog Programming Success in a Day: Beginners Guide to Fast, Easy and Efficient Learning of Prolog Programming, Sam Key, CreateSpace Independent Publishing Platform (August 12, 2015)
- The C++ Programming Language 3rd Edition, Stroustrup, Addison-Wesley (1997)
- Learning Reactive Programming with Java 8, NickolayTsvetinov, Packt Publishing (June 24, 2015)
- Pascal: An Introduction to the Art and Science of Programming (4th Edition), Walter Savitch, Addison Wesley; 4 edition (December 31, 1994)
- Introduction to the Design and Analysis of Algorithms (2nd Edition), AnanyLevitin, Addison Wesley; 2 edition (February 24, 2006)

Perkenalan Peserta

- Nama
- Program sarjana
  - Dimana
  - Jurusan apa
  - Mengambil tugas akhir apa

Mekanisme Perkuliahan

- Penilaian
  - 35% Ujian Tengah Semester
  - 20% Tugas kelompok
  - 35% Ujian akhir semester
  - 10% Absen

BERPIKIR KOMPUTASIONAL

Computational Thinking in CS
(Jeannette Wing)

- ***Conceptualizing, not programming***
- ***Fundamental, not rote skill***
- ***A way that humans, not computers think***
- ***Complements and combines mathematical and engineering thinking***
- ***Ideas, not artifacts***
- ***For everyone, everywhere***

- **Thinking Recursively**
- **Thinking Abstractly**
- **Thinking Ahead**
- **Thinking Algorithmically**
- **Thinking Logically**
- **Thinking Concurrently**

Sumber : BATEC

Computational Thinking in IT

Criteria	Definition	Measures
Logical Thinking	Creatively develop, select and test relevant hypotheses	Asks probing questions to uncover details of the problem Clearly defines the problem Defines clear success criteria for the project including measurable objectives
Strategizing	Ability to anticipate and evaluate potential outcomes	Anticipates and evaluates the effects of various design options Makes design decisions based on rational criteria
Abstract Thinking	Ability to find appropriate level of detail to define and solve a problem	Decomposes a problem into component parts Understands the relationships between components
Procedural Thinking	Ability to select and execute appropriate steps to solve a problem	Identifies the steps required to solve a problem Identifies the sequence of steps including possible decisions and branching Identifies normal and exceptional behavior of a solution
Optimizing	Ability to analyze processes for optimal efficiency and use of resources	Understands available resources Develops a solution that uses only available resources Measures and adapts the solution to optimize resource utilization
Iterative Refinement	Process refinement with the goal of improving quality or precision.	Measures and evaluates solutions against the success criteria Adjusts the design and implementation as needed

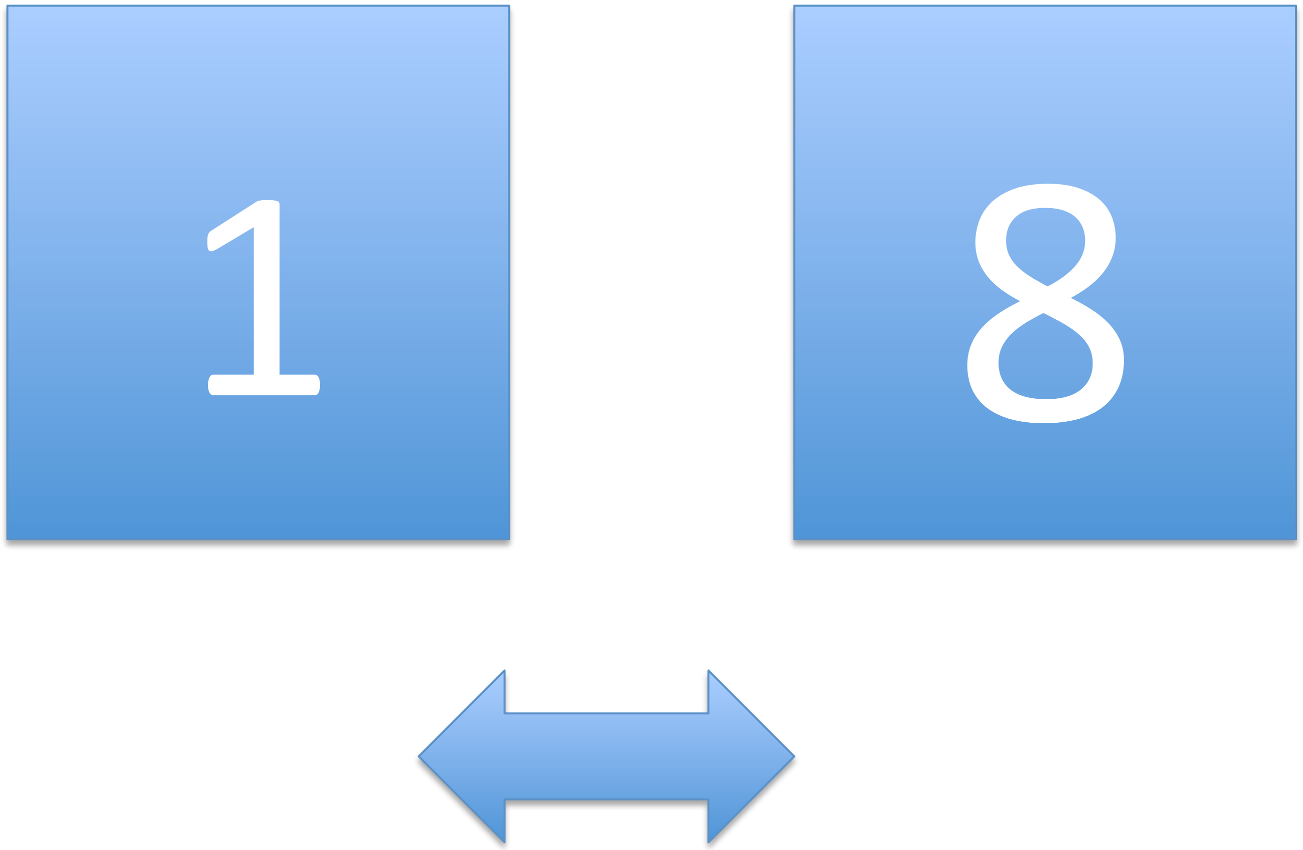
Sumber : BATEC

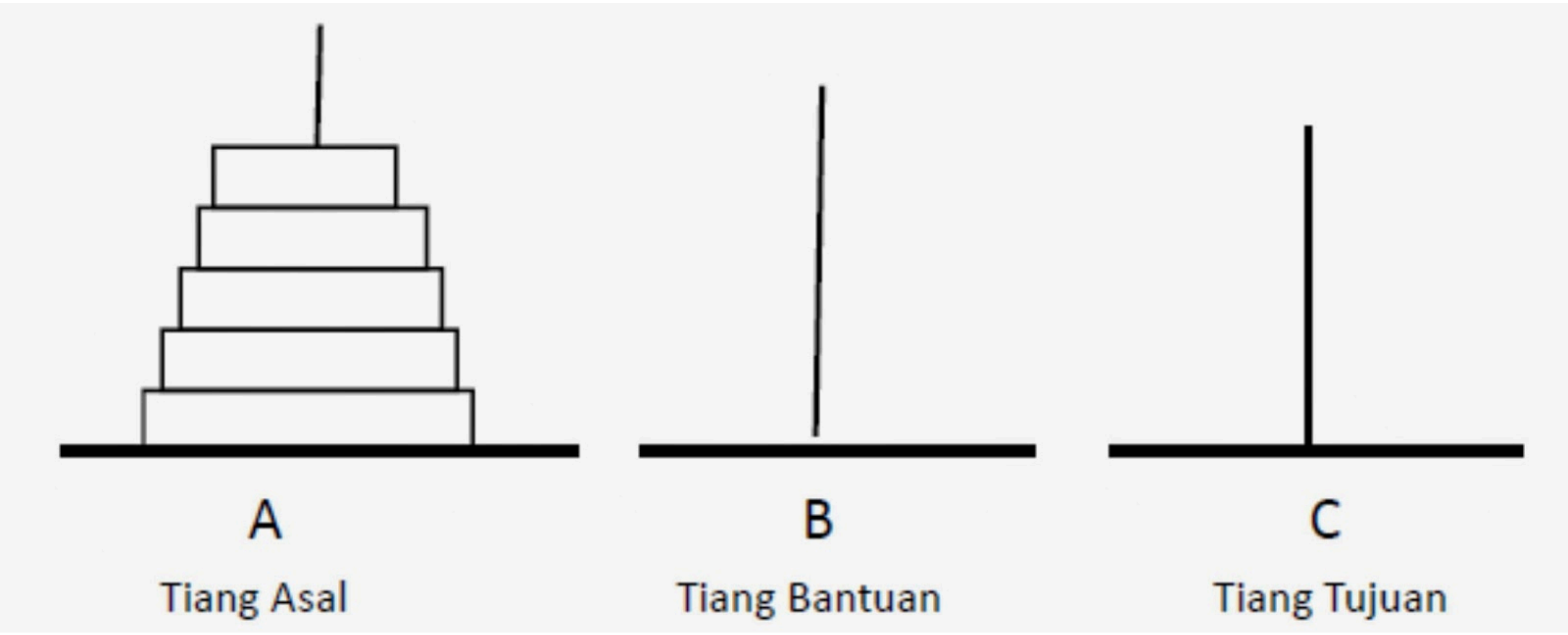
Computational thinking

How do we think about problems so that computers can help?

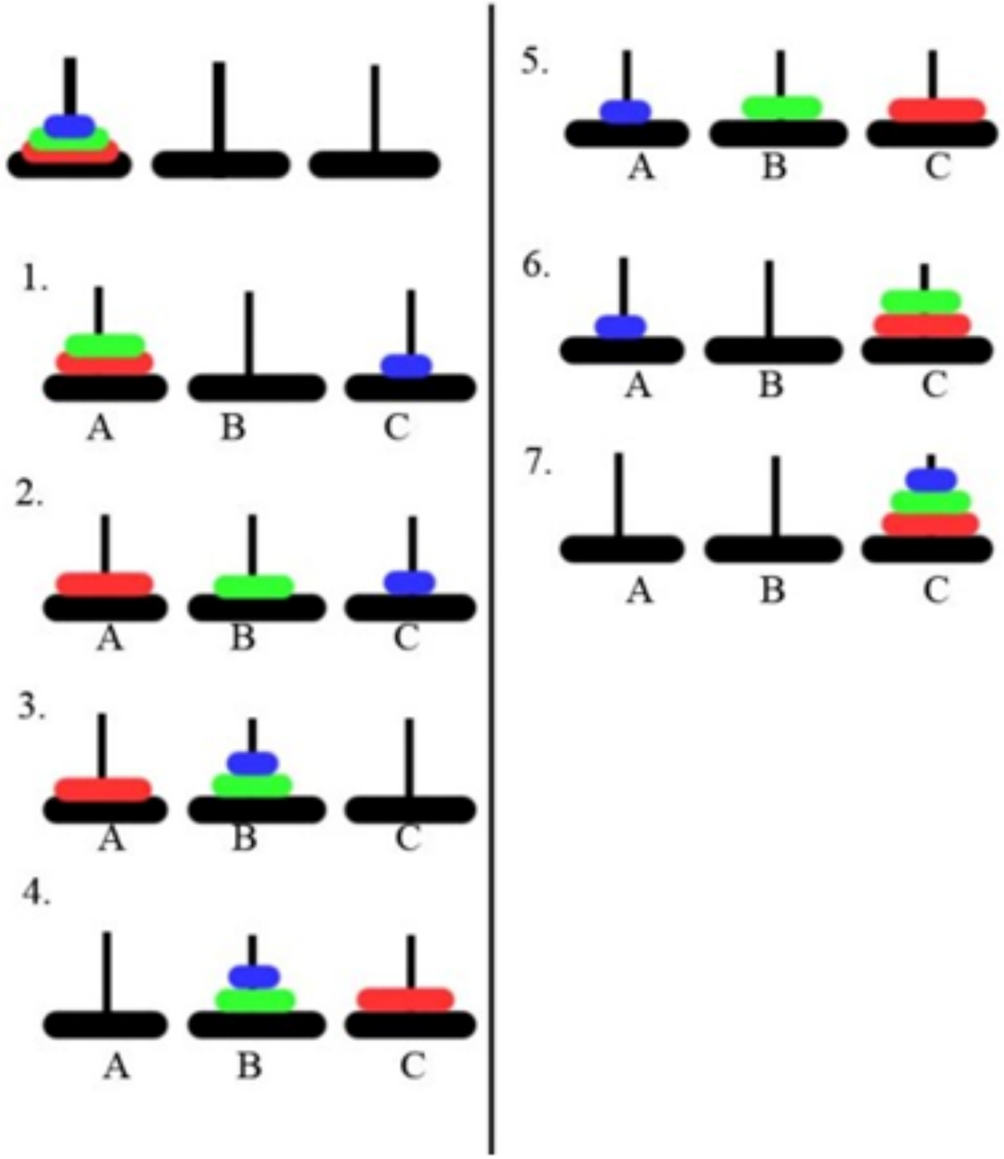
Computers are incredible devices: they extend what we can do with our brains. With them, we can do things faster, keep track of vast amounts of information and share our ideas with other people.

dopod


1
8


A
Tiang Asal
B
Tiang Bantuan
C
Tiang Tujuan

Menara Hanoi dengan 3 piringan



What is computational thinking?

Getting computers to help us to solve problems is a two-step process:

1. First, we think about the steps needed to solve a problem.
2. Then, we use our technical skills to get the computer working on the problem.

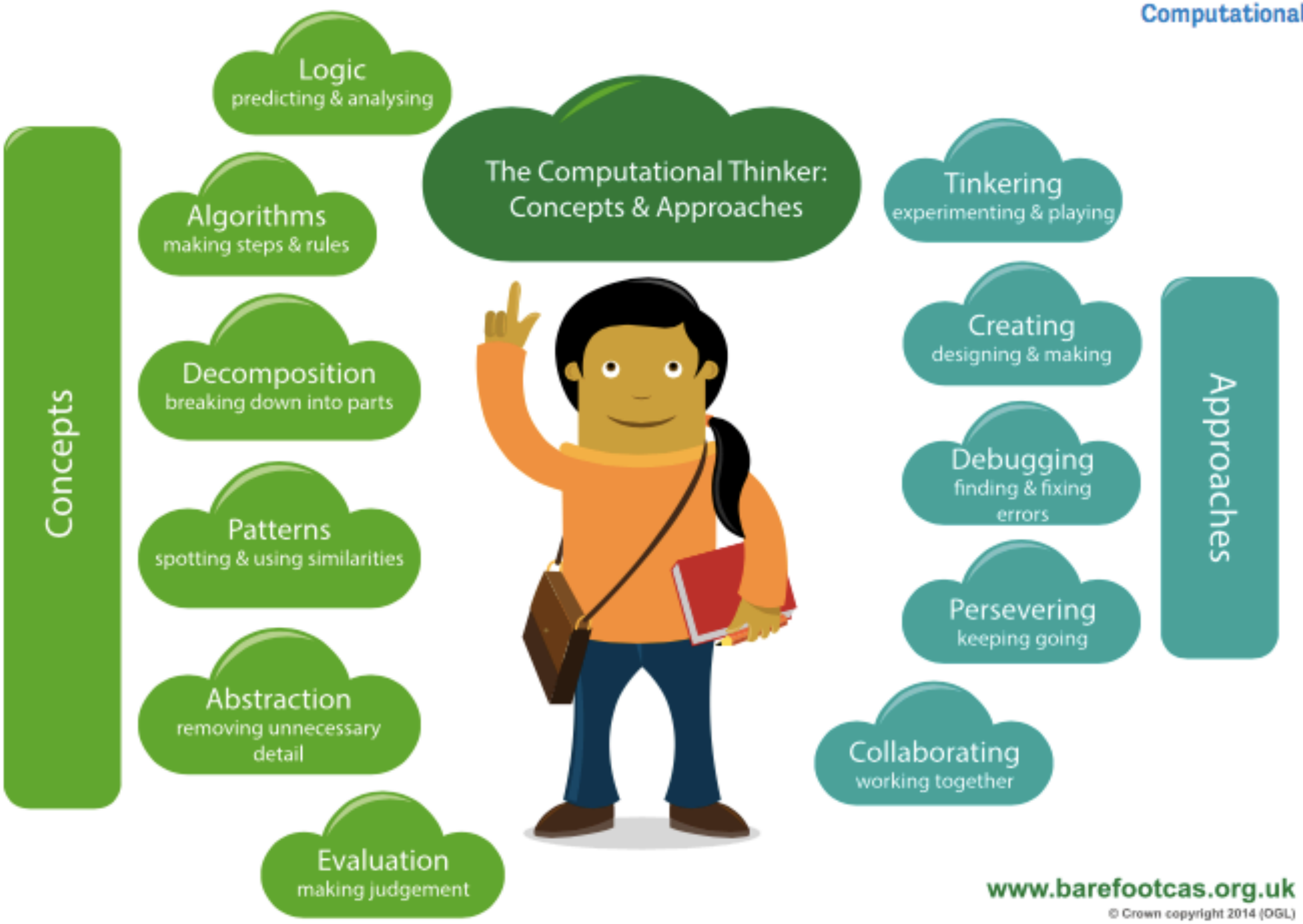
- Take something as simple as using a calculator to solve a word problem in maths. First, you have to understand and interpret the problem *before* the calculator can help out with the arithmetic bit.
- Similarly, if you're going to make an animation, you need to start by planning the story and how you'll shoot it *before* you can use computer **hardware** and software to help you get the work done.

- Computational thinking describes the processes and approaches we draw on when thinking about problems or systems in such a way that a computer can help us with these.
- Computational thinking is *not* thinking about computers or like computers. Computers don’t think for themselves. Not yet, at least!
- Computational thinking is about looking at a problem in a way that a computer can help us to solve it.

When we do computational thinking, we use the following processes to tackle a problem:

- Logical reasoning: predicting and analysing
- Algorithms: making steps and rules
- Decomposition: breaking down into parts
- Abstraction: removing unnecessary detail
- Patterns and generalisation: spotting and using similarities
- Evaluation: making judgements

Computational thinking


The Computational Thinker: Concepts & Approaches
Concepts
Logic predicting & analysing
Algorithms making steps & rules
Decomposition breaking down into parts
Patterns spotting & using similarities
Abstraction removing unnecessary detail
Evaluation making judgement
Approaches
Tinkering experimenting & playing
Creating designing & making
Debugging finding & fixing errors
Persevering keeping going
Collaborating working together
www.barefootcas.org.uk
© Crown copyright 2014 (OGL)

Logical reasoning

Can you explain why something happens?

- If you set up two computers in the same way, give them the same instructions (the program) and the same **input**, you can pretty much guarantee the same **output**.
- Computers don’t make things up as they go along or work differently depending on how they happen to be feeling at the time. This means that they are predictable. Because of this we can use *logical reasoning* to work out exactly what a program or computer system will do.

How is logical reasoning used in computing?

- Logic is fundamental to how computers work: deep inside the computer’s central processing unit (CPU), every operation the computer performs is reduced to logical operations carried out using electrical signals.
- It’s because everything a computer does is controlled by logic that we can use logic to reason about program behaviour.

Algorithms

What's the best way to solve a problem?

- An algorithm is a sequence of instructions or a set of rules to get something done.
- You probably know the fastest route from school to home, for example, turn left, drive for five miles, turn right. You can think of this as an 'algorithm' – as a sequence of instructions to get you to your chosen destination. There are plenty of algorithms (i.e. routes) that will accomplish the same goal; in this case, there are even algorithms (such as in your satnav) for working out the shortest or fastest route

How are algorithms used in the real world?

- Search engines such as Bing or Google use algorithms to put a set of search results into order, so that more often than not, the result we're looking for is at the top of the front page.
- Your Facebook news feed is derived from your friends' status updates and other activity, but it only shows that activity which the algorithm (EdgeRank) thinks you'll be most interested in seeing. The recommendations you get from Amazon, Netflix and eBay are algorithmically generated, based in part on what other people are interested in.

Decomposition

How do I solve a problem by breaking it into smaller parts?

- The process of breaking down a problem into smaller manageable parts is known as decomposition. Decomposition helps us solve complex problems and manage large projects.
- This approach has many advantages. It makes the process a manageable and achievable one – large problems are daunting, but a set of smaller, related tasks are much easier to take on. It also means that the task can be tackled by a team working together, each bringing their own insights, experience and skills to the task.

How is decomposition used in the real world?

- Decomposing problems into their smaller parts is not unique to computing: it’s pretty standard in engineering, design and project management.
- Software development is a complex process, and so being able to break down a large project into its component parts is essential – think of all the different elements that need to be combined to produce a program, like PowerPoint.

Abstraction

How do you manage complexity?

- *The abstraction process – deciding what details we need to highlight and what details we can ignore – underlies computational thinking.*4

- Abstraction is about simplifying things; identifying what is important without worrying too much about the detail. Abstraction allows us to manage complexity.

- We use abstractions to manage the complexity of
  life in schools. For example, the school timetable is
  an abstraction of what happens in a typical week: it captures key information such as who is taught what subject where and by whom, but leaves to one side further layers of complexity, such as the learning objectives and activities planned in any individual lesson.

Patterns and generalisation

How can you make things easier for yourself?

In computing, the method of looking for a general approach to a class of problems is called generalisation. By identifying patterns we can make predictions, create rules and solve more general problems. For example,
in learning about area, pupils *could* find the area of a particular rectangle by counting the centimetre squares on the grid on which it's drawn. But a better solution would be to multiply the length by the width: not only
is this quicker, it's also a method that will work on *all* rectangles, including really small ones and really large ones. Although it takes a while for pupils to understand this formula, once they do it's so much faster than counting squares.

HOW DOES SOFTWARE GET WRITTEN?

How does software get written?

- As well as the above processes, there are also a number of approaches that characterise computational thinking. If pupils are to start thinking computationally, then it’s worth helping them to develop these approaches to their work, so they can be more effective in putting their thoughts into action.

Tinkering

- There is often a willingness to experiment and explore in computer scientists' work. Some elements of learning a new programming language or exploring a new system look quite similar to the sort of purposeful play that's seen as such an effective approach to learning in the best nursery and reception classrooms.

Creating

- Programming *is* a creative process. Creative work involves both originality and making something of value: typically something that is useful or at least fit for the purpose intended.
- Encourage pupils to approach tasks with a creative spirit, and look for programming tasks that allow some scope for creative expression rather than merely arriving at the right answer.

Debugging

- Because of its complexity, the code programmers write often doesn’t work as it’s intended.
- Getting pupils to take responsibility for thinking through their algorithms and code, to identify and
fix errors is an important part of learning to think, and work, like a programmer. It’s also something to encourage across the curriculum: get pupils to check through their working in maths, or to proofread their stories in English. Ask pupils to debug one another’s code (or indeed proofread one another’s work), looking for mistakes and suggesting improvements. There’s evidence that learning from mistakes is a particularly effective approach, and the process of pupils debugging their own or others’ code is one way to do this.

Collaborating

Software is developed by teams of programmers and others working together on a shared project. Look for ways to provide pupils with this experience in computing lessons too. Collaborative group work has long had a place in primary education, and computing should be no different.

TERIMA KASIH