Implementing Digital Technologies Masterclass

James Curran (james@groklearning.com)
Associate Professor, School of IT, University of Sydney
Director, National Computer Science School
CEO and Co-founder, Grok Learning

Vivian Li (viv@groklearning.com)
Engagement Engineer, Grok Learning
Today’s program

8:30  Lessons from Microsoft WeSpeakCode
     Introducing 7,000 students to coding in one week
     Andrew Coates, Developer Evangelist, Microsoft

9:20  Digital Technologies
     An overview of where we are at, where we are heading & global connections
     Julie King, Senior Project Officer Technologies, ACARA

10:00 Computational Thinking
     Transforming learning spaces, engaging and empowering teachers & students through practical ideas, tools & strategies
     Nikos Bogiannidis, Dean of Learning Technologies, Haileybury School

10:45  Morning Tea & Networking
Today’s program

11:15  Digital Technology Curriculum Masterclass
      How to implement the new Digital Technologies Curriculum in your school
      *James Curran and Vivian Li*

1:00  Networking Lunch

2:00  Digital Technology Curriculum Masterclass (cont.)
      How to implement the new Digital Technologies Curriculum in your school
      *James Curran and Vivian Li*

4:30  End of Masterclass
Programming in Digital Technologies

- What you need to cover
- Platforms and languages
- Grok Learning / NCSS Challenge
Some good reasons to teach $x$

- $x$ is a different way of thinking or making
- $x$ is tells us something about the world
- $x$ is challenging, engaging and creative
- $x$ is fundamental for understanding other disciplines
- $x$ is a skill that takes years to master, so we should expose kids to it early
Some bad reasons to teach $x$

- industry wants $x$
- there are jobs in $x$ now
- the money is good in $x$
- $x$ gives you a high ATAR
- $x$ is in the exam
- ...

What you need to cover
Aspects of programming languages

- visual or text programming
- object-oriented programming
- teaching (specialised) or real-world (general purpose)
Visual programming

- *mandatory* Year 3-6, maybe Year 7/8
- drag and drop elements to create code
- avoids syntax (and some semantic) errors

- unplugged: *flowcharts*, decision trees
- teaching: *Scratch*, Alice, Blockly, Lego NXT
- real-world: *Labview*
Text programming

- mandatory by Year 8, maybe Year 6-7
- write code by hand
- more efficient and succinct than visual

- unplugged: pseudocode
- teaching: Actionscript, *Basic, Python, ...
- real-world: Actionscript, C++, C#, Java, Javascript, Python, Visual Basic
Object-oriented (OO) programming

- mandatory by Year 10, maybe Year 8-9
- data and code organised as objects
  - objects model the real world
  - support classes, inheritance, and polymorphism
- unplugged: the world is all objects
- teaching: C#, Java, Python, VB(ish)
- real-world: C++, C#, Java, Javascript(ish), Python
- Scratch has objects, but isn’t OO
Specialised versus general purpose

- **Some languages are domain specific:**
  - teaching (Scratch, Alice, Blockly, ABC, Blue)
  - game development (Game Maker, Unrealscript)
  - mathematics (Mathematica, MATLAB)
  - databases (SQL) and spreadsheets (Excel formula)
  - often lack common language features

- **Most languages are general purpose:**
  - solve problems in a wide range of domains
  - solve specific domain problems less elegantly
Specialised versus general purpose

- Language may be general, but platform not
- Migrate specialised → general purpose
  - Javascript (now used on the backend as node.js)
  - Scratch (used for control on Raspberry Pi)
- General purpose → specialised libraries
  - Python: Django and others to hide SQL
  - Python: numpy to replace MATLAB

- Digital Technologies mandates general purpose by Year 8 to provide students with a practical programming tool
Implementation

4. Implement simple digital solutions as **visual programs** with algorithms involving **branching** (decisions) and **user input**

6. Implement digital solutions as simple visual programs involving branching, **iteration (repetition)**, and user input
Implementation

8. Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language.

10. Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language.
## Implementation

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Programming</th>
<th>Test and debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching (decisions) and user input</td>
<td>Visual programming</td>
<td></td>
</tr>
<tr>
<td>Iteration (repetition)</td>
<td>Visual programming</td>
<td></td>
</tr>
<tr>
<td>User interfaces and functions</td>
<td>General purpose text programming</td>
<td>In algorithms content descriptor</td>
</tr>
<tr>
<td>Modularity, algorithms and data structures</td>
<td>Object-oriented programming</td>
<td>In algorithms content descriptor</td>
</tr>
</tbody>
</table>
Follow, describe and represent a sequence of steps and decisions (algorithms) needed to solve simple problems.

Define simple problems, and describe and follow a sequence of steps and decisions (algorithms) needed to solve them.

Define problems in terms of data and functional requirements drawing on previously solved problems.

Design a user interface for a digital system.

Design, modify and follow simple algorithms involving sequences of steps, branching, and iteration.
Define and decompose real-world problems taking into account functional requirements and economic, environmental, social, technical and usability constraints.

Design the user experience of a digital system, generating, evaluating and communicating alternative designs.

Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors.
Specification and Algorithms

1. Define and decompose real-world problems precisely, taking into account functional and non-functional requirements and including interviewing stakeholders to identify needs.

2. Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics.

3. Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases.
# Specification and Algorithms

<table>
<thead>
<tr>
<th>Describe problems</th>
<th>Follow/design algorithms</th>
<th>Design user interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follow, describe and represent a sequence of steps and decisions</td>
<td>Describe/follow a sequence of steps and decisions (algorithms) needed to solve problems</td>
<td>Design a user interface</td>
</tr>
<tr>
<td>Define simple problems</td>
<td>Design/modify simple algorithms (also) involving iteration</td>
<td></td>
</tr>
<tr>
<td>Define problems using data and functional requirements</td>
<td>Represent algorithms using diagrams and English; trace algorithms</td>
<td>Generate and evaluate alternative designs</td>
</tr>
<tr>
<td>Decompose real-world problems; Consider economic, environmental, social, technical and usability constraints</td>
<td>Validate algorithms and programs through tracing and test cases</td>
<td>Evaluate designs against criteria: functionality, accessibility, usability, and aesthetics</td>
</tr>
<tr>
<td>Interviewing stakeholders to identify needs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Platforms and languages
Bee Bots
Bee bot!
Cambodian Children's Trust
Lightbot (HoC activity)
lightbot.com/hocflash2014.html
You can use the P1 command inside PROC1 to make a loop!
Code.org and Hour of Code

code.org/learn
move forward by 100 pixels
turn right by 90 degrees
Scratch
scratch.mit.edu
Grok Learning
https://groklearning.com
Representation

2. Recognise and explore patterns in data and represent data as pictures, symbols and diagrams

4. Recognise different types of data and explore how the same data can be represented in different ways

6. Examine how whole numbers are used to represent all data in digital systems

8. Investigate how digital systems represent text, image and audio data in binary

10. Analyse simple compression of data and how content data are separated from presentation
# Representation

<table>
<thead>
<tr>
<th>Representation</th>
<th>Types of data</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represent data as pictures, symbols and diagrams</td>
<td>Different types of data</td>
<td></td>
</tr>
<tr>
<td>The same data can be represented in different ways</td>
<td>Different types of data</td>
<td></td>
</tr>
<tr>
<td>How whole numbers are used to represent all data</td>
<td>All (simple) data: types should be more complex</td>
<td></td>
</tr>
<tr>
<td>Represent data in binary</td>
<td>Text, image and audio</td>
<td></td>
</tr>
<tr>
<td>Content vs. presentation: documents are represented</td>
<td>All data: structured data</td>
<td>Simple compression of data</td>
</tr>
</tbody>
</table>
Collection and interpretation

2. Collect, explore and sort data, and use digital systems to present the data creatively.

4. Collect, access and present different types of data using simple software to create information and solve problems.

6. Acquire, store and validate different types of data, and use a range of software to interpret and visualise data to create information.
Collection and interpretation

- Acquire data from a range of sources and evaluate authenticity, accuracy and timeliness.
- Analyse and visualise data using a range of software to create information, and use structured data to model objects or events.
- Develop techniques for acquiring, storing and validating quantitative/qualitative data from a range of sources, considering privacy and security requirements.
- Analyse and visualise data to create information and address complex problems, and model processes, entities and their relationships using structured data.
## Collection and interpretation

<table>
<thead>
<tr>
<th>Collect</th>
<th>Organise / create</th>
<th>Visualise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect and explore data</td>
<td>Sort data</td>
<td>Present the data</td>
</tr>
<tr>
<td>Collect and access different types of data</td>
<td>Create information and solve problems</td>
<td></td>
</tr>
<tr>
<td>Acquire, store and validate different types of data</td>
<td>Interpret data to create information</td>
<td>Visualise data to create information</td>
</tr>
<tr>
<td>Evaluate authenticity, accuracy and timeliness</td>
<td>Use structured data to model objects or events</td>
<td>Visualise data using a range of software</td>
</tr>
<tr>
<td>Validating quantitative and qualitative data; considering privacy and security</td>
<td>Model processes, entities and their relationships using structured data</td>
<td>Visualise data to create information and address complex problems</td>
</tr>
</tbody>
</table>
Follow, describe and represent a **sequence of steps and decisions** (algorithms) needed to solve simple problems.

**Define** simple problems, and describe and follow a **sequence of steps and decisions** (algorithms) needed to solve them.

Define problems in terms of **data and functional requirements** drawing on previously solved problems.

Design a **user interface** for a digital system.

Design, modify and follow simple algorithms involving **sequences of steps, branching, and iteration**.
Define and decompose real-world problems taking into account functional requirements and economic, environmental, social, technical and usability constraints.

Design the user experience of a digital system, generating, evaluating and communicating alternative designs.

Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors.
Specification and Algorithms

1. Define and decompose real-world problems precisely, taking into account functional and non-functional requirements and including interviewing stakeholders to identify needs.

2. Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics.

3. Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases.
# Specification and Algorithms

<table>
<thead>
<tr>
<th>Describe problems</th>
<th>Follow/design algorithms</th>
<th>Design user interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define simple problems</td>
<td>Follow, describe and represent a sequence of steps and decisions</td>
<td></td>
</tr>
<tr>
<td>Define problems using data and functional requirements</td>
<td>Describe/follow a sequence of steps and decisions (algorithms) needed to solve problems</td>
<td></td>
</tr>
<tr>
<td>Decompose real-world problems; Consider economic, environmental, social, technical and usability constraints</td>
<td>Design/modify simple algorithms (also) involving iteration</td>
<td>Design a user interface</td>
</tr>
<tr>
<td>Interviewing stakeholders to identify needs</td>
<td>Represent algorithms using diagrams and English; trace algorithms</td>
<td>Generate and evaluate alternative designs</td>
</tr>
<tr>
<td></td>
<td>Validate algorithms and programs through tracing and test cases</td>
<td>Evaluate designs against criteria: functionality, accessibility, usability, and aesthetics</td>
</tr>
</tbody>
</table>
Implementation

4. Implement simple digital solutions as **visual programs** with algorithms involving **branching** (decisions) and **user input**

6. Implement digital solutions as simple visual programs involving branching, **iteration (repetition)**, and user input
Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language.

Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language.
## Implementation

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Programming</th>
<th>Test and debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching (decisions) and user input</td>
<td>Visual programming</td>
<td></td>
</tr>
<tr>
<td>Iteration (repetition)</td>
<td>Visual programming</td>
<td></td>
</tr>
<tr>
<td>User interfaces and functions</td>
<td>General purpose text programming</td>
<td>In algorithms content descriptor</td>
</tr>
<tr>
<td>Modularity, algorithms and data structures</td>
<td>Object-oriented programming</td>
<td>In algorithms content descriptor</td>
</tr>
</tbody>
</table>
Questions?

Find me at:
@drjamescurran / @groklearning
james@groklearning.com