Supporting the transition from block to text based programming languages

By Andrew Csizmadia, Mark Dorling & Steve Bunce

CONSTRUIT 2017
Warwick University
What do we have to teach?
Programmes of Study
(Concepts and key constructs)
At Key Stage 4
Supporting the transition from: Block to Text
**IDLE: Script mode**

- Interactive mode is great but it’s not designed to create programs that you can save and run later.
- The Script mode enables you to write, save, open, and edit programs.
- File > New File
IDLE: Script mode

• Run > Run Module

• Save the file.
IDLE: Script mode

• See the results!
Using comments

- It is good practice to use comments at the end of a program to give:
  - Explanation of what it does
  - The author
  - The version:
  - The date:
  - Any copyright information

```python
######
#
# Explanation of what the program does
#
# Author:
# Version:
# Date:
#####
```
Wait in Python

• The is no print for ‘n’ seconds in Python like in Scratch and Snap!

• How do we get around this?
• Python's time module has a handy function called sleep()
Wait (sleep) in Python

• The syntax is this:  `time.sleep(secs)`
• We need to remember to import the time module.

```python
import time

for i in range(1,10):
    print("Hello World!")
    time.sleep(1)
```
Introducing: Data Types

```
str1 = str("Hello")
str2 = str("World!")
print(str1 + str2)
```
Introducing: Concatenation

print "red" + 3

Traceback (most recent call last): File "", line 1, in
TypeError: cannot concatenate 'str' and 'int' objects

• Python doesn't know how to add a word and a number, so it says "cannot concatenate 'str' and 'int' objects.

• You can't add a str and a int together. But you can turn an integer into a string if you use the str() function.

print "red" + str(3)
Bottom/post-tested loops

What if the count variable is set to 1 at the start of the program and not 100?
Introducing: Top & bottom tested loops

```
Repeat
    print("Enter name of item")
    input(item)
    print("Another item?")
    input(response)
Until response = 'N' or "No"
```

Repeat While response = 'N' or "No"
```
print("Enter name of item")
input(item)
print("Another item?")
input(response)
```
Range of languages

@APCsizmadia, @MarkDorling
Range of problems

• Range of recursive and iterative activities
  • 10 green bottles
  • One man went to Mow
  • Fibonacci and Lucas series
• Rock Paper Scissors
• Hang man style games
• Searching Algorithms
  • Linear
  • Binary
• Sorting algorithms
  • Bubble
  • Insert
  • Merge
  • Selection
Transition...

KS3/4

KS3

KS2

@APCszmadia, @MarkDorling
Exploring programming pedagogy
Why programming?

• We want to make models of the world to:
  • Understand it
  • Ask “what if” questions and predict the way it will change

• How do we make models?
  • Solving problems
    • Asking good questions
    • By characterising a problem
      • Looking for similar problems you already know how to solve
    • Think about what makes a problem similar to another

(Professor Greg Michaelson, Heriot-Watt University)
Abstractions...

- How do we turn models into programs?
- Writing programs based on algorithms
- Programming bridges gaps between thinking and computers?
- Choosing appropriate technology
Our aim for learners is to...

- The McCracken group focused on problem solving in programming and suggested a 5-step process that students should learn:
  1. Abstract the problem from its description (Abstraction)
  2. Generate sub-problems (Decomposition)
  3. Transform sub-problems into sub-solutions (Generalisation and Algorithmic Thinking)
  4. Re-compose, and
  5. Evaluate and iterate (Evaluation)

McCracken et al. (2001)
Challenges for novice programmers

• Students might lack skills that are a precursor to problem-solving.

  Lister et al. (2004)

• Being able to read and trace code is really important pre-cursor to the problem-solving needed to write code.

  Lister et al. (2008)

• Novice programmers need to be able to trace code with greater than 50% accuracy before they can independently begin to write programs.

  Lister (2011)
## Essential Programming Skills

<table>
<thead>
<tr>
<th>Create</th>
<th>Design: Devise a solution structure</th>
<th>Model: Illustrate or implement an abstraction of a problem</th>
<th>Refactor: Redesign a solution for optimisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Implement: put a completed design into code</td>
<td>Adapt: modify a solution for other domains</td>
<td>Debug: Both detect and correct flaws in design</td>
</tr>
<tr>
<td></td>
<td>Recognise: Base knowledge and vocabulary for the domain</td>
<td>Trace: Desk-check a solution</td>
<td>Present: Explain a solution to others</td>
</tr>
<tr>
<td></td>
<td>Know</td>
<td>Understand</td>
<td>Analyse</td>
</tr>
</tbody>
</table>

---

Fuller et al. (2007)
Use - Modify - Create

Lee et al. (2011)
Use - Modify - Create

- **Predict** – given a working program, what do you think it will do? (at a high level of abstraction)
- **Run** – run it and test your prediction
- **Explain/Articulate** – What does each line of code mean? (low level of abstraction). I’m not sure that explain is quite the right term.
- **Modify** – edit the program to make it do different things (high and low levels of abstraction)
- **Create/Design** – design/create is a key computational thinking skill.

Sentence. (2017)
## Pulling this together

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONSTRUCT</strong></td>
<td>Novice coders constructing an understanding by using and running code provided</td>
<td>Construct knowledge: new vocabulary and knowledge about construct, reading and tracing, explaining, analysing and evaluating code</td>
<td>The code is 'not theirs'</td>
</tr>
<tr>
<td><strong>CHANGE</strong></td>
<td>Novice coders begin modifying and adapting code provided</td>
<td>Applying knowledge: Implementing code from a given design, adapting code for a different purpose, finding and correcting errors in code (debugging).</td>
<td>Making the code 'partly theirs'</td>
</tr>
<tr>
<td><strong>CREATE</strong></td>
<td>Novice coders begin designing and writing their own code</td>
<td>Reusing solutions in bigger solutions, designing and coding solutions, fixing smelly code (commenting and optimising code).</td>
<td>The code is 'fully theirs'</td>
</tr>
</tbody>
</table>
Historical: Choice of pedagogy

**Constructivism:** The learner is not a passive recipient of knowledge but that knowledge is ‘constructed’ by the learner.

**Constructionism:** The idea that learners’ learn best through building things that are tangible and sharable with the public.

**Social Constructivism:** Groups construct knowledge for one another, collaboratively creating a smaller culture of shared artifacts with shared meaning.
Example pedagogies

- **Code walkthroughs:** Learners step through code predicting outputs
- **Collaborating on solutions:** Writing algorithms and code in groups
- **Scaffolding:** Insert comments into existing code
- **Code debugging:** Finding errors in given code e.g. spot the difference
- **Flipped learning:** Class time to collaborate and compare solutions

Van Gorp and Grissom (2001)
Supporting constructivist

• Using examples that are relevant to students’ own experiences e.g. relating to real-world experiences

• Active learning experiences e.g. unplugged, kinesthetic activities

• Learning by exploration e.g. exploring programming environments and open-ended tasks

• Learning by solving problems e.g. self-directed projects and problem-solving

• Open-ended discussion and working in groups e.g. paired and group problem-solving.
Supporting social constructivist

- Software developers generally spend:
  - 30% of their time working alone
  - 50% of their time working with one other person
  - 20% of their time working with two or more people

DeMarco and Lister (1987)

Three forms of peer-based interaction in the classroom:
1. Tutoring, where the less capable are guided by the more capable;
2. Co-operation, where learners work on different parts of the task;
3. Collaboration, where learners work jointly on almost all parts of the task.

Jehng (1997)
Thanks for listening!

@APCsizmadia, @MarkDorling
References

• AQA GCSE Computer Science: http://www.aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520


References


Lister, R. (2011). *Concrete and other neo-piagetian forms of reasoning in the novice programmer*.


References


References


• WJEC GCSE Computer Science: http://wjec.co.uk/qualifications/computer-science/computer-science-gcse/