kidscode jeunesse

Computational Thinking

A Framework for understanding Computational Thinking: Computational Concepts, Computational Practices and Computational Perspectives
(Brennan & Resnick, AERA 2012).

Concepts:

Concepts that designers use as they program; these concepts are common to all programming and transfer to non-programming contexts.

- Algorithm: a series of step-by-step instructions.
- Decomposition: breaking down a complex problem into simpler parts.
- Abstraction: keeping essential characteristics/information (from the latin abs -"away from" and "trahere" -draw).
- Logic: Analyzing and predicting.
- Pattern Recognition: learning to identify and use similarities in order to: simply, shorten, apply similar solutions.

Practices:

Practices designers develop as they program: it is the process of construction and design. Practices focus on the process of thinking and learning, moving beyond the "what" you are learning to the "how" you are learning.

- Experimenting: Trial and error, testing new things, having an idea and trying to see how to make it work (incremental & iterative process).
- Debugging: Answering the question "Does it work the way I want it to?" If the answer is "No", figure out why then fix it (incremental & iterative process).
- Collaboration: For most things in life, you collaborate with others: creating for others and/or creating with others.
- Planning: It is useful to plan things out when tasks are complex or unfamiliar.

This project was supported by a grant from CIRA's Community Investment Program.

Perspectives:

Designers evolve their understanding of the technological world around them, their relationship with others, and of themselves.

- Expressing: Computer programming can be a means of expressing oneself because it is a tool, it can be used in many different ways (web pages, art, music, games, etc…) just like a pencil can be used to write a report or draw a picture.
- Understanding: Being better able to understand today's socio-techno reality - being able to recognize and use some of the tools that create this reality.
- Connecting: Creativity and learning are deeply social practice, facilitate being able to reach out to others in a new ways (creating with others and creating for others).

Computer Programming

Computer

- A programmable machine that can store, retrieve and process data
- Two characteristics of computers:
    - Responds to a specific set of instructions in a well-defined manner.
    - Can execute a pre-recorded list of instructions (a program).
- Is made up of:
    - Hardware: what we can physically touch (the actual machinery.
    - Software: data and instructions allowing a computer to perform a specific task.

Computer Code

- Computer code is how instructions are given to a computer to perform a specific task.

- Computer programming languages:

This project was supported by a grant from CIRA's Community Investment Program.

- All of them serve the same purpose: instructions are written in a human-friendly language which are then be translated (by a compiler) into machine-code that the computer understands.
- Why so many different ones? A programming language is a tool:
  - For humans to express ideas to computers: different people think differently.
  - To accomplish different tasks: languages often make trade-offs to be more effective at the tasks it is mostly going to perform. (Example: A tractor trailer, a bicycle and a Tesla are all vehicles with wheels and steering and will get you from point A to point B: they are used for different things).

Learning and Teaching Computational Practices

Programming in the classroom is much more than a new skill.

By instructing a computer how to create a game or an animation, children are learning about learning. In teaching the computer how to do something, they are teaching themselves how to reflect on behaviours, mathematical systems, scientific methods and creative processes in ways that can deeply enrich future learning.

Planning

Planning can be especially daunting for novice programmers. Allow for warm up, tinkering and pre-planning activities to stimulate ideas. Encourage students to quickly sketch paper and pencil prototypes as a way to think through a game or animation beforehand. And prompt students to break projects down into small, achievable outcomes that clearly relate to the projected goal.

Experimenting

Programming requires *iteration,* which means trying something multiple times. It also means willing to fail before improvement is possible. Students need time for tinkering and reflection. Evaluations focussed on process over product can encourage this type of learning. Student

This project was supported by a grant from CIRA's Community Investment Program.

sheets or design journals can be used to record stumbling blocks and document steps taken to achieve an outcome. Some teachers even give partial marks for "failure," defined as number of iterations attempted, or achieving a more interesting outcome because of an unexpected result. Encourage students to change their plan when appropriate.

Collaboration

Programmers rarely work in isolation. Making software is usually a team project, and even when working on individual projects programmers depend on feedback and advice from peers and mentors. Encourage collaboration with student sheets that document both help received and help given to peers. Or make time for a *think-pair-share* activity, where students take time to think about their project, pair with another student and share any problems or insights. Provide prompts that encourage peer mentorship, like post it notes on a computer that indicate "feedback needed!"or "I've achieved my outcome, and happy to help others."

Debugging

One of the best ways to teach debugging is to model mistakes yourself! Making programming mistakes in front of students, asking for their help in debugging a program, showing interesting mistakes you made and what you learned, does not demonstrate a lack of knowledge or competency. It demonstrates that ALL programmers regardless of skill level must practice debugging. Professional programmers spend much of their working life debugging programs. The more complex the program, the more bugs are likely to pop up!

Internet & Web

Internet

- It is a tangible, physical system made to move information: the infrastructure.
- "The Internet" is a massive combination of millions of computers, connected via cables and wireless signals.
- It is made up of an incredibly large number of independently operated networks: there's no central control.

This project was supported by a grant from CIRA's Community Investment Program.

- End-to-end connectivity: any device connected to the Internet can communicate with any other device.

Web

- Web is short for World Wide Web (www): a worldwide collection of text pages and multimedia files one can access over the Internet.
- Web is the name use for the billions of digital pages written in HTML (Hyper Text Markup Language) code that can be viewed using a web browser (software). Web pages are connected with the programming language HTTP (Hyper Text Transfer Protocol).

Anatomy of a Web Address

- URL: Universal Resource Locator is the official computer name for a web address and (in theory) points to a unique resource on the Web.
- HTTP: Hyper Text Transfer Protocol is the language used to communicate between web browsers and servers - how one computer asks another computer for a document.
- WWW: World Wide Web.
- HTML: Hyper Text Markup Language is the language you use to tell a web browser how to make a page look.
- Finding a page on the web:
    - Start a web browser: the app you use to access web pages.
    - Type the web address (URL) of the website you want to visit.
    - Your computer starts communicating with another computer, called a server, using HTTP (the server can be located anywhere).
    - The information is displayed in your browser's window according to HTML code.

This project was supported by a grant from CIRA's Community Investment Program.