ABSTRACTION IN COMPUTING EDUCATION

Odd Petter Sand

Associate Teaching Professor

IFI, UiO

WHAT IS ABSTRACTION?

- Considered fundamental for computer science (e.g. Dijkstra, 1972)
- Two points of view (Statter & Armoni, 2020)
  - Changing the Resolution
  - Black-box Interface (“what”) ←→ Implementation (“how”)
- Experts move flexibly between levels of abstraction
- “Simultaneously seeing things ‘in the large’ and ‘in the small’” (Knuth, in Hartmanis, 2007)
- Part of generalising activity? (Ellis et al., 2022)

ABSTRACTION IS IMPORTANT

- Navigating complexity - “seeing the forest for the trees”
- Important for understanding (Wiggins & McTighe, 2005):
  - Explaining
  - Interpreting
  - Applying
  - Perspective
- In elementary mathematics, abstraction errors > math errors (Rich, Yadav & Zhu, 2019)

IT IS ALSO CHALLENGING

- Novices gravitate towards lower levels of abstraction (Hazzan & Zazkis, 2005)
  - Get hung up on details, syntax…
  - This particular case in itself (not as a representation of something more general)
  - Unfamiliarity and discomfort increase these tendencies!
- Errors common when students shift between levels (Rich, Yadav, Zhu 2019)
- Meanwhile, experts may operate on several levels of abstraction at once (Hazzan, 2003)
  - Often unconscious behaviour!
  - Almost never explicit

TEACHING ABSTRACTION TO NOVICES

- Armoni's (2013) framework
  - Explicit attention to levels of abstraction (language cues signify which level)
  - Explicit attention to moves between levels
  - Opportunities for students to reflect on their own abstraction processes
- Improved 7th grade CS students' abstraction abilities (Statter & Armoni, 2020)
  - Also improved their general CS performance

COMPUTING → MATHEMATICS?

- Abstraction is important both for computing and mathematics
  - A simple print() in Python represents over 100 lines of code (written in C)!
  - "and then it easily follows that…" → actually means 4 pages of calculations
- Example: Functions (mathematics, programming)
  - Implementation (definition; *how* does the function do what it does)
  - Interface (black-box application; *what* does the function do in this context)
  - Challenging to master for novices in either discipline

PYTHON MAKES ABSTRACTION EXPLICIT

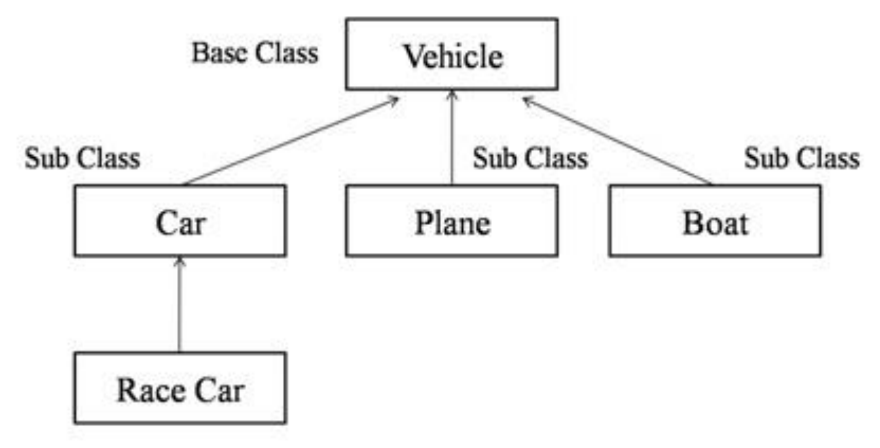
```
def dynsavetxt(fnames, data):

    if( fnames.shape == data.shape ):
        # for every data item
        for index in np.ndindex(data.shape):

            if( type(data[index]) is np.ndarray ):
                print("Saving file:", fnames[index])
                np.savetxt(fnames[index], data[index]) # save to file
            # else use nan for missing data - nan is a float, not an array
    else:
        print("Fatal error in dynsavetxt(...): Array shapes do not match -
```

PYTHON MAKES ABSTRACTION EXPLICIT

- Loops / if-else blocks
  - Indentation = abstraction level?
- Functions
  - Interface: How to use the function / what it does
  - Implementation: How the function works
- Classes (blueprint for objects)
  - May inherit from more general base classes
  - May create subclasses for specialised purposes



TEACHING DESIGN

- Abstraction as a learning goal (in math/programming)
  - Is it fundamental? Important? Or just a bonus?
- Proof of understanding abstraction
  - How to probe abstraction *specifically*?
  - What does understanding abstraction even mean?
- Learning activities
  - Being explicit about abstraction (role model)
  - Activities that not only require abstraction, but are about abstraction?

POSSIBLE RESEARCH QUESTIONS

- How do students think about/work with Python's built-in levels of abstraction in a traditional teaching setting?
- What difference does explicit attention to levels of abstraction make for students' learning in a scientific programming (computing) context?
- How do students who show proof of understanding abstraction approach complex problems compared to those who do not?
- How do students generalise a good understanding of abstraction in one context (such as programming) to other contexts (such as mathematics)?

WHAT ABOUT OTHER CONTEXTS?

- What does abstraction look like in physics?
  - Applying a law/formula in a black-box fashion (interface)
  - Deriving said law/formula and understanding why it works (implementation)
- Chemistry?
- Biology?

REFERENCES

Armoni, M. (2013). On Teaching Abstraction in CS to Novices. *Journal of Computers in Mathematics and Science Teaching*, *32*(3), 265–284.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, *15*(10), 859–866. https://doi.org/10.1145/355604.361591

Ellis, A. B., Lockwood, E., Tillema, E., & Moore, K. (2022). Generalization Across Multiple Mathematical Domains: Relating, Forming, and Extending. *Cognition and Instruction*, *40*(3), 351–384. https://doi.org/10.1080/07370008.2021.2000989

Hartmanis, J. (2007). Turing award lecture: On computational complexity and the nature of computer science. In *ACM Turing award lectures* (p. 1993). Association for Computing Machinery. https://doi.org/10.1145/1283920.1283949

Hazzan, O. (2003). How Students Attempt to Reduce Abstraction in the Learning of Mathematics and in the Learning of Computer Science. *Computer Science Education*, *13*(2), 95–122. https://doi.org/10.1076/csed.13.2.95.14202

Hazzan, O., & Zazkis, R. (2005). Reducing Abstraction: The Case of School Mathematics. *Educational Studies in Mathematics*, *58*(1), 101–119. https://doi.org/10.1007/s10649-005-3335-x

Rich, K. M., Yadav, A., & Zhu, M. (2019). Levels of Abstraction in Students' Mathematics Strategies: What Can Applying Computer Science Ideas about Abstraction Bring to Elementary Mathematics? *Journal of Computers in Mathematics and Science Teaching*, *38*(3), 267–298.

Statter, D., & Armoni, M. (2020). Teaching Abstraction in Computer Science to 7th Grade Students. *ACM Transactions on Computing Education*, *20*(1), 8:1-8:37. https://doi.org/10.1145/3372143

Wiggins, G., & McTighe, J. (2005). *Understanding By Design* (2nd Expanded edition). Assn. for Supervision & Curriculum Development.

DISCUSSION

Also, questions?