PROGRAMMING PEDAGOGY

The Craig'n'Dave approach to teaching programming is revolutionary. Not because it is a new pedagogy, but because it is the first set of resources to truly unify the proven pedagogies for teaching programming effectively. Wrapped into one scheme of learning for teaching and learning the fundamentals of programming in Python and C#, our resources are deeply rooted in these methodologies:

- **Barak Rosenshine's principles**

  Introducing new material in small steps, asking questions, providing models of code and guiding student practice with progressive, logical learning. Checking understanding with an emphasis on teacher-student review, oral feedback and obtaining a high success rate with points-based challenges. Independent practice is central to these resources where students learn programming for themselves.

- **Sue Sentence's PRIMM model**

  Predicting, running, investigating, making and modifying programs as a practical approach to learning and studying code has been adopted and streamlined to "T.I.M.E." in these resources, because becoming proficient at programming takes time and that acronym just makes more sense!

- **Carsten Schulte's block model of program comprehension**

  Comprehending how code works by identifying key programming terminology, recognising structures and syntax, describing the purpose of algorithms, and considering alternative approaches. We have distilled Schulte's work on the duality of structure and function with atoms, blocks, relations, and macro structures in code to something secondary students can understand. Essentially studying program source code and asking structured questions about it.

- **Dale Parson's code sorting**

  Supporting weaker programmers by providing them with all the code they need to create a working program but jumbling the statements up. Creating a working program becomes a card sorting exercise. Scaffolding learning, reducing the comprehension required and enabling a trial and error approach.

- **Alan O'Donohoe's stepped challenges**

  Providing students with the desired output instead of the input. Algorithms can be created in many ways. Encouraging students to adopt paired programming techniques to discuss and justify their approaches with each other before learning from other pairs. Using differentiation by outcome to uncover deeper theoretical conversations about the merits of the different approaches taken.

- ## Richard Pawson and the functional programming paradigm

Independent routines are as critical as sensible variables names and comments for creating well structured, modular programs. All programmers should start by learning to make subroutines from the very beginning of the course. Understanding the necessity for reusable components and parameter passing, not as an advanced topic, but one that is essential to the structure of all programs.

- ## William Lau's little book of algorithms

Building fluency in programming by giving students many different and differentiated challenges. Recognising that repetition reduces the cognitive load by committing fundamental concepts of programming and keywords to long term memory. Working on small, standard algorithms that repeatedly demand sequence, selection and iteration creates more confident programmers.

- ## Craig and Dave's design by doing

The traditional software engineer understands the system life cycle. Analyse the problem and design a solution before you begin to write code. While essential in many historical contexts of programming, rapid and iterative design practices today negate the need for this approach. Instead students learn best by writing real programs. Benefitting from instant feedback modern compilers and run-time environments provide.

Programming taught from the front of the class by a teacher only benefits at best a third of the class. Some can already progress further and some already need additional support. Allowing students to learn independently at their own pace and choosing their own challenges allows greater flexibility for the teacher to stretch and support individual students.