How we teach computing

12 pedagogy principles

**Lead with concepts**
Support pupils in the acquisition of knowledge, through the use of key concepts, terms, and vocabulary, providing opportunities to build a shared and consistent understanding. Glossaries, concept maps, and displays, along with regular recall and revision, can support this approach.

**Work together**
Encourage collaboration, specifically using pair programming and peer instruction, and also structured group tasks. Working together stimulates classroom dialogue, articulation of concepts, and development of shared understanding.

**Get hands-on**
Use physical computing and making activities that offer tactile and sensory experiences to enhance learning. Combining electronics and programming with arts and crafts (especially through exploratory projects) provides pupils with a creative, engaging context to explore and apply computing concepts.

**Unplug, unpackage, repack**
Teach new concepts by first unpacking complex terms and ideas, exploring these ideas in unplugged and familiar contexts, then repacking this new understanding into the original concept. This approach (semantic waves) can help pupils develop a secure understanding of complex concepts.

**Create projects**
Use project-based learning activities to provide pupils with the opportunity to apply and consolidate their knowledge and understanding. Design is an important, often overlooked aspect of computing. Pupils can consider how to develop an artefact for a particular user or function, and evaluate it against a set of criteria.

**Challenge misconceptions**
Use formative questioning to uncover misconceptions and adapt teaching to address them as they occur. Awareness of common misconceptions alongside discussion, concept mapping, peer instruction, or simple quizzes can help identify areas of confusion.

**Model everything**
Model processes or practices — everything from debugging code to binary number conversions — using techniques such as worked examples and live coding. Modelling is particularly beneficial to novices, providing scaffolding that can be gradually taken away.

**Add variety**
Provide activities with different levels of direction, scaffolding, and support that promote active learning, ranging from highly structured to more exploratory tasks. Adapting your instruction to suit different objectives will help keep all pupils engaged and encourage greater independence.

**Structure lessons**
Use supportive frameworks when planning lessons, such as PRIMM (Predict, Run, Investigate, Modify, Make) and Use-Modify-Create. These frameworks are based on research and ensure that differentiation can be built in at various stages of the lesson.

**Read and explore code first**
When teaching programming, focus first on code ‘reading’ activities, before code writing. With both block-based and text-based programming, encourage pupils to review and interpret blocks of code. Research has shown that being able to read, trace, and explain code augments pupils’ ability to write code.

**Foster program comprehension**
Use a variety of activities to consolidate knowledge and understanding of the function and structure of programs, including debugging, tracing, and Parson’s Problems. Regular comprehension activities will help secure understanding and build connections with new knowledge.

Find out more about our principles and add some or all to your personal pedagogy toolkit.