Computer Science, Robotics & AI

IMPACT TONBRIDGE SCHOOL

Computer Science, Robotics and Artificial Intelligence equips students with the technical knowledge and problem-solving skills required to thrive in an increasingly digitalised world. Students will be encouraged to explore both the analytical and creative capacity of computer science, creating algorithmic art, apps, robotics, and games, tackling cyber security and OSINT focused challenges, and evaluating the functionality of modern Gen AI systems, while also considering their safety, and the ethical concerns they raise.

Course structure

Learning is structured around a six-phase challenge that introduces students to a wide range of Computer Science concepts, including computer programing, algorithm design, embedded systems, robotics, cyber security, digital design and generative artificial intelligence. In each phase, students will have the opportunity to work both individually and collectively on 'mini-projects', approaching each of these through an iterative, agile methodology akin to that used in the software development industry.

Skills gained

Python Programing, Data Analysis, Programing and using embedded systems (Microbit), App Design, Game Development, Robotics, AI, Prompt Engineering, Cyber Security, OSINT, algorithm design, problem abstraction and decomposition.

- Students develop their ability to **problem solve**, **designing effective algorithms**, communicating these to teammates and peers, and implementing them using the required technical skills.
- Teamwork and **collaborative development skills** improve, as students are tasked to complete mini-projects in small groups, with each taking on their own important role.
- Students have the ability to **learn to code** in both python, and block-based javascript, as well as being introduced to key programing paradigms and computational thinking concepts.
- Students learn how to **evaluate** and **analyse** the **digital world** around them; considering the validity of sources and data, and the safety and reliability of platforms and systems.
- **Creative skills** are also developed through the process of creating both digital art, and the user-interfaces of apps and games.

Key learning objectives

- Develop an understanding of how agile design methodologies are used in the real world to develop programmatical and technical solutions to problems.
- Learn how to write code in both Python and Javascript, and how to use MakeCode to create programs for the Microbit embedded system.
- Understand some of the social, ethical and environmental issues impacting the digital world, and how these can be navigated.
- Explore different cyber security concepts, developing a deeper understanding of the importance of good security practice and managing your digital footprint.
- Understand how design choices in digital apps and games can be used to both improve user experience, and accessibility.
- Understand what 'dark patterns' are, and how to recognise these.
- Learn how to communicate technical concepts to peers, and how to collaborate and develop solutions.

Example challenge

Students are introduced to the concept of algorithmic art, including its historical precedence, and how it varies from both digital art and artificially generated (Gen AI) art. They are then taught to program in python, by using the python turtle module to create their own algorithmic art; both as individuals, and within small groups. Art is exhibited in the final phase, and students are asked to evaluate their own work and answer the question: is art created through a computer program 'real' art?

PHASE	ACTIVITY	SKILLS DEVELOPMENT
PHASE 1 **Foundation and Orientation**	Students are introduced to the concept of 'algorithmic art' and invited to evaluate and critique various prominent pieces of work from the past century. Alongside this, they are taught about the key terminology, groups and movements associated with the style, as well as prominent exhibitions, such as Cybernetic Serendipity (1968) and Electric Dreams (2025)	**Technical** • Identifying the hardware and software used to create algorithmic art in the past, and exploring how these technologies correspond to more modern mediums. **Problem Solving** • Investigating how these pieces of art were created - abstracting and decomposing their development process. **Analysis & Evaluation** • Evaluating each piece of art, both from a technical perspective, and from a creative one - do they consider it to be 'real' art? **Communication and Collaboration** • Sharing their analysis and evaluation in pairs and small groups.
PHASE 2 **Exploration and Practice**	Students are introduced / refamiliarised with the Python programming language, completing a series of small tasks designed to develop their understanding of variables, basic arithmetic, subroutines and loops.	**Technical** • Learning how to write simple code in python, including how to test and debug programs. **Problem Solving** • Applying their coding knowledge to complete tasks, each of which builds upon the knowledge developed in its predecessor. **Analysis & Evaluation** • Testing their code and assess, to what extent, it fulfills the requirements outlined in the task. **Communication and Collaboration** • Consulting and collaborating with peers to troubleshoot bugs and logic errors in their code.
PHASE 3 **Application and Collaboration**	Students are introduced to the Python Turtle module and complete several paired programming tasks, with one student coding and the other using a turtle command crib sheet to steer development.	**Technical** • Learning how to use the Python Turtle module to create digital drawings, alongside further consolidating the programming skills covered in the previous phase. **Problem Solving** • Completing drawing challenges - starting with basic shapes and gradually progressing to more complex images and patterns. **Analysis & Evaluation** • Collaboratively testing and analyzing their code, troubleshooting issues and improving the efficacy of their solution. **Communication and Collaboration & Reflective Working** • Experimenting with the paired programming methodology, where one student acts as a 'driver' (coder) and the other as a 'guide' - a second set of eyes, informing development. This method helps students develop their ability to articulate technical concepts, and collaboratively troubleshoot code.

PHASE	ACTIVITY	SKILLS DEVELOPMENT
PHASE 4 **Independence and Mastery**	Students create a standalone piece of algorithmic art using the Python turtle module, inspired by the overall theme of the sprint; innovation. They are free to interpret this how they wish, and the theme is kept intentionally vague to allow for a breadth of personal expression.	**Technical** • Consolidating their Python knowledge from the previous phases by applying it in a self-directed programming task. **Problem Solving** • Applying their technical knowledge, and the initial information presented in the orientation phase, to create a piece of algorithmic art that fits the theme given. Once they have decided what they would like their piece to be, they must then decompose the project and implement it. **Analysis and Evaluation** • Comparing and contrasting their work with the pieces presented during the orientation phase, and consider what movement/sub-category their piece would fall into. **Communication & Collaboration** • Discuss their ideas, and their final piece of code, with peers; gathering constructive feedback and suggestions for further improvements and optimisation.
PHASE 5 **Consolidation and Presentation**	In small groups, students collaborate to create art for a mini-exhibition; combining their solo work from the previous phase with collaborative pieces	**Technical** • Prior learning, testing outcomes, and evaluative work are brought together and consolidated in the creation of multiple python programs, developed collaboratively. **Problem Solving** • As both individuals and within their groups, students consider how their will implement their ideas - decomposing each problem, delegating responsibility for tasks and subtasks, and considering both time and personnel limitation. **Analysis and Evaluation** • Evaluate their collective work, considering how each piece fits with the group's exhibition plan and what, if anything, could be done to improve the implementation and efficiency of each piece of code. **Communication & Collaboration** • Working collaboratively on multiple sub-tasks within the same project; developing their communication skills, understanding the importance of compromise, clear time-constrained goals and frequent, constructive feedback.
PHASE 6 **Evaluation and Reflection**	Each group displays their mini exhibition, and gives a short group presentation on the development process.	**Technical** • Explaining how each piece in their exhibition was created, using suitable technical terminology with confidence. **Problem Solving** • Deciding how to arrange their exhibition, and how to deliver their group presentation in an engaging way. **Analysis & Evaluation** • Reflecting on their own work, and evaluating the work of others to provide thoughtful, constructive feedback. **Communication & Collaboration** • Presenting their own ideas for both the exhibition and presentation, and balancing these with those of their peers - respecting the views of others, negotiating and compromising as appropriate to produce an exhibition and presentation representative of all group members.