Enhancing Python Programming Education Through Creative Computing Environments

Exploring and Developing New Python Tools for Visual Learning Contexts

by

Tristan Bunn

under the supervision of

Dr Aaron Bere

Dr Michelle Douglas

A document submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at

February 2026

Declaration of Originality

This thesis is my own work, and to the best of my knowledge, it contains no materials previously published or written by another person, and no materials which have been accepted for the award of any other degree or diploma at Torrens University Australia or any other institution, except where I have made due acknowledgement. I acknowledge that copyright of published work contained within this thesis resides with the copyright holder(s) of those works. Any contribution made to the research by others is explicitly acknowledged.

Use of Generative AI

I have used AI-assisted tools to support the preparation of this thesis. Specifically, I employed Grammarly [80] extensively for proofreading and editing, and ChatGPT (OpenAI) [154] and Claude (Anthropic) [13] to scaffold parts based on my drafts and to assist with summary sections. In combination with Grammarly, I also used these large language models to refine, condense, and improve the clarity of some language. I confirm that I restricted the use of these tools to my draft text, avoiding any submission of sensitive research data or participant information. All intellectual contributions, interpretations, analyses, and arguments remain my own, except where I have made due acknowledgement. Moreover, I reviewed any AI output to verify, critically examine, and refine it before any integration into the thesis content.

Tristan Alan Bunn | 2026-02-07

Authority of Access Statement

I authorise Torrens University Australia, and any library with which it has a formal agreement, to archive my thesis and to make hardbound and digital copies available through the Library. These copies may be loaned, copied, or communicated in any form of media, whether currently known or developed in the future, subject to the provisions of the *Copyright Act 1968*. I retain the right to use all or part of this thesis in future works.

Tristan Alan Bunn | 2026-02-07

Ethical Conduct Statement

The research associated with this thesis was conducted in accordance with the *National Statement on Ethical Conduct in Human Research* (2023).

Tristan Alan Bunn | 2026-02-07

Abstract

Traditional approaches to teaching programming fundamentals often pose barriers for learners whose primary orientation is artistic- or design-focused. This thesis investigates Python-based creative computing environments as a solution, evaluating existing tools and developing new software and techniques for visual learning contexts, with a focus on students in creative and interdisciplinary fields.

The thesis adopts a hybrid folio/thesis-by-publication format, encompassing software development, scholarly publications, presentations, learning resources, and creative works. Collectively, these outputs demonstrate scholarly contribution, active community engagement, and the artistic applications of novel programming practices.

A central contribution is the creation of Thonny-py5mode, a plugin that integrates the py5 library into the Thonny IDE, serving as a contemporary successor to Processing's discontinued Python Mode (Processing.py). It enables learners to generate graphical and interactive output by writing Python code, bridging technical concepts with creative exploration. An empirical evaluation applies the Task–Technology Fit framework to assess Thonny-py5mode, surveying 143 students in an introductory Python course and analysing their responses with Structural Equation Modelling (SEM) to capture factors influencing user adoption in educational contexts.

The thesis also considers the emerging role of generative AI in programming education. While acknowledging its benefits, it highlights risks of over-reliance and proposes graphical task design with Thonny-py5mode as a strategy to mitigate GenAI misuse. As a relatively late addition, this strand of inquiry foregrounds future research directions and underscores the continuing evolution of the field.

This research demonstrates how Python tools and purpose-built environments, supported by targeted curricula, can lower barriers to programming education by employing graphical and multimedia output to enhance student engagement, conceptual understanding, and motivation.

Acknowledgements

I want to express my sincere gratitude to my supervisors, Dr Michelle Douglas and Dr Aaron Bere, for their invaluable guidance and support, crucial to the successful transition to and completion of my PhD at Torrens University Australia. Also, thank you to TUA for supporting my studies.

I extend my heartfelt thanks to my initial (Victoria University of Wellington) supervisors, Dr Craig Anslow and Dr Karsten Lundqvist, for their excellent guidance in preparing my successful research proposal and seeing me through to full PhD candidature.

I am profoundly grateful to the creators, maintainers, and community surrounding Processing, particularly its Python-related projects, py5 and Processing.py. Your efforts have significantly enriched educational and creative coding communities, inspiring my students and me.

Also, a special thank you to all those whose support and collaboration assisted in completing and disseminating the publications, presentations, and creative works contained herein—this includes funding bodies and sponsors, editors and peer reviewers, study participants, conference and exhibition organisers, and the publishers and academic platforms that provided opportunities to present and share this work with a broader audience.

Technical Details and Credits

I have typeset this document using a modification of the *Mimosis* LaTeX template, available at https://github.com/Pseudomanifold/latex-mimosis. I have prepared all vector graphics (multiple diagram figures) using Inkscape.

Contents

List of Figures

List of Tables

I Introduction

Students studying creative computing, creative technologies, and related fields benefit from learning computer programming in various ways. Firstly, programming expands creative possibilities by enabling more interactive and dynamic work, while also providing the ability to customise or even build entirely new digital tools, removing the constraints of 'off-the-shelf' software [205]. It also supports interdisciplinary practice, equipping students to contribute effectively to projects that merge art, design, technology, and science [83]. Beyond enhancing creativity, programming education fosters industry-relevant capabilities; graduates who possess coding skills can pursue diverse roles in areas such as visual effects, video games, virtual reality, and interactive media [39]. Additionally, literacy in different coding languages can facilitate better collaboration by bridging the gap between artists, developers, and technical specialists, leading to more cohesive and effective project outcomes [53].

Creative computing refers to the use of computer science and programming as media for creativity, employing algorithms, code, and computational processes to generate digital works such as interactive installations, generative art, or digital music. *Creative technology*, by contrast, highlights the application of emerging platforms and tools that extend traditional artistic practices and enable new forms of hybrid or physical expression, such as virtual and augmented reality, 3D printing, sensors, or wearable devices. Both fields sit at the intersection of art, technology, and human innovation, and both prepare students for rapidly evolving creative industries. While one often encounters the terms used interchangeably, the distinction lies in emphasis: creative computing foregrounds computational techniques as a medium of expression, whereas creative technology foregrounds tools and platforms that expand the possibilities of creative practice [213].

This study focuses on the pedagogical aspects of creative computing and creative technologies practice—specifically, the approach to programming education that emphasises creating multimedia outputs. Such methods typically employ creative coding and art- or design-oriented curricula and

teaching methodologies, often aimed at students in creative fields but also applicable to those in other domains [254].

More specifically, this thesis explores Python-based creative computing environments (tools and other software) for education. Python, a high-level programming language, has gained significant prominence due to its simplicity, readability, and versatility [228]. Initially developed by Guido van Rossum and released in 1991, Python's design philosophy emphasises code readability and clear syntax, allowing programmers to express concepts in fewer lines of code than C++ or Java. This efficiency and a comprehensive standard library have made Python a favoured choice among developers across various fields [170]. As the forthcoming chapters demonstrate, the Python language's versatility extends beyond conventional software development, finding substantial applications in creative computing.

In summary, this PhD investigates innovative approaches to enhancing Python programming education by integrating, expanding, and developing new creative computing tools (software) and techniques (creative coding practice). It includes the development of tailored curricula, comprehensive documentation, and a sole-authored book.

Grounded in peer reviewed journal articles (Quartile 2 or higher), the thesis explores and evaluates the effectiveness of Python-based educational environments and creative coding techniques through empirical research methods. Additionally, artistic works showcased at prominent venues illustrate these tools and techniques' practical applications and creative potential. Findings and insights are further disseminated through presentations at esteemed academic and professional conferences, promoting broader awareness and adoption of these advancements in programming education.

1.1 Thesis Presentation

This PhD thesis employs a hybrid structure combining elements of (primarily) a **Folio of Advanced Professional Practice** and (secondarily) a **Thesis by Publication**, as defined in the *Higher Degree by Research Thesis Guidelines*[1] set forth by Torrens University Australia.

The elected format allows for a comprehensive exploration of the PhD topic and new development (and testing) of innovative methods in Python creative computing through bespoke software development, scholarly publications, presentation outputs, and creative works—all completed during the PhD enrolment, which commenced in October 2019 (Appendix A.2), and forming a coherent and

[1] TUA Higher Degree by Research Thesis Guidelines at https://torrens.blackboard.com/bbcswebdav/xid-44216710_1

thematic research narrative. This collection of outputs forms the core of the folio, with each marking a significant milestone in the research journey. Collectively, they contribute to the academic discourse surrounding Python programming education that employs creative computing environments, addressing existing gaps in the literature while proposing novel solutions to contemporary pedagogical challenges.

For the reader's convenience, this thesis organises the outputs by type (Publications, Presentation Outputs, Creative Works) and, within each grouping, presents the contents in chronological order to illustrate the research's evolution over time. Where practical and necessary, each output is presented in full, prefaced by a concise introduction that situates it within the broader research narrative. These works span various stages of inquiry, from initial exploration and hypothesis development to detailed studies and practical applications. Central themes are consistently revisited through diverse methodologies and analytical frameworks, reinforcing the study's contributions.

1.1.1 Chapter Structure

This outline provides a high-level summary of the chapters comprising this PhD thesis, using concise chapter descriptions and a diagram (Figure 1.1) to offer a conceptual overview of the entire document's contents. The elected structure facilitates a thorough understanding of the core concepts and research journey, and includes dedicated chapters that group folio outputs. Moreover, it accommodates the study's practical implications, significance, limitations, and discussion on potential future directions.

1. Introduction (current chapter) Introduces the thesis by outlining the content structure and describing its context, scope, and significance. It lists the research objectives and questions, laying the groundwork for the key themes explored in subsequent chapters, and provides an introduction to creative coding, programming education, and Python.

2. Literature Review Identifies gaps in current research and tools related to creative computing and Python programming education, positioning the work within the broader academic discourse. Given the folio/publication-based framing of this thesis, and to avoid repetition, the literature is distributed across the Publications (as part of the journal articles), the Presentation Outputs, and, to a lesser extent, the Creative Works chapter.

Figure 1.1: Overview of thesis contents and structure

3. Software Covers the Thonny-py5mode plugin, outlining its key features, supporting documentation, and the development process, including associated funding and recognition. The *Evaluating Task–Technology Fit in Thonny-py5mode* article complements this chapter by detailing the plugin's technical implementation and design rationale. Publications, Presentation Outputs, and Creative Works assess the software's effectiveness and its contribution to the research objectives, while also demonstrating its practical applications and broader capabilities.

4. Publications Presents the collected manuscripts—comprising a sole-authored book and peer reviewed articles published in Q1 and Q2 journals—that together form the scholarly core of the thesis. These works explore both new and existing tools, techniques, and curricula for Python-based creative computing, including results of the user studies conducted as part of this research.

5. Presentation Outputs Documents each presentation accompanied by an introduction that outlines its contribution to the thesis and integration into the overall research narrative. Delivered at prominent gatherings, the topics encompass innovative tools, creative coding practices, and teaching methodologies in Python programming that emphasise a focus on creative computing.

6. Creative Works Presents a series of creative works developed using Python-based creative coding tools and techniques explored throughout this PhD research. These works have been exhibited at various online and in-person events and featured in published outputs.

7. Conclusion Consolidates the research findings, tools, publications, presentations, and creative works, demonstrating how they collectively advance Python programming education through creative computing. Specifically, it highlights the interconnections among these outputs and their broader implications for educational practice, summarises the key findings, articulates the thesis' Contribution to Knowledge, and reflects on the strengths and limitations of the developed tools and techniques. The chapter concludes by outlining future research directions and opportunities for further innovation in Python-based creative computing environments.

Appendices Collects supplementary material supporting the thesis, including ethics clearance, folio evidence, survey forms, anonymised data, code samples, and other relevant resources. Where applicable, chapter content will cite resources contained within the appendices.

1.1.2 A Note on Methodology

Notably, this thesis does not include a dedicated Methods or Methodology chapter. Following thesis-by-publication conventions, each output in the Publications and Presentation Outputs chapters—and, to a lesser extent, the Creative Works chapter—details its respective method(s) [56]. The PhD folio is interdisciplinary in nature, spanning digital technology, education, and creative practice across multiple output types. Such a scope necessitates varied methodological framings [192]. Accordingly, the thesis situates its methods within the development process rather than fixing them in advance. A linking commentary and concluding chapter synthesise the diverse strands, articulating the evolution of the PhD process as a whole. Moreover, the structure avoids repetition and supports the coherence of the folio/publication thesis format.

1.2 Research Context and Significance

This PhD argues that an increasing convergence of art, design, and technology has amplified the need for robust programming education tailored to creative disciplines. As technology continues to permeate facets of creative work, the ability to program is a valuable skill that can expand creative boundaries, facilitate interdisciplinary collaboration, and open doors for graduates entering emerging industries. However, teaching programming to students within creative fields can present unique challenges [39, 53, 205]. Creative computing offers a promising approach that invites learners to merge computational thinking with artistic and design-based expression.

Software applications like the Processing[2] IDE (Integrated Development Environment) provide creative computing environments for writing code in Java and other languages. While older versions of the Processing IDE can accommodate Python, work on this aspect is now discontinued and "Python Mode" (Processing.py) will not operate correctly in current versions.

Python is a widely used programming language known for its simplicity and versatility [104]; this thesis argues that there remains significant potential for developing new Python-based tools and pedagogical approaches—akin to Processing's—that make coding more accessible, engaging, and aligned with the goals of learners pursuing foundational programming education for eventual entry into creative industries.

[2] https://processing.org

Figure 1.2 illustrates how this PhD is positioned at the convergence of Creative Coding, Programming Education, and Python.

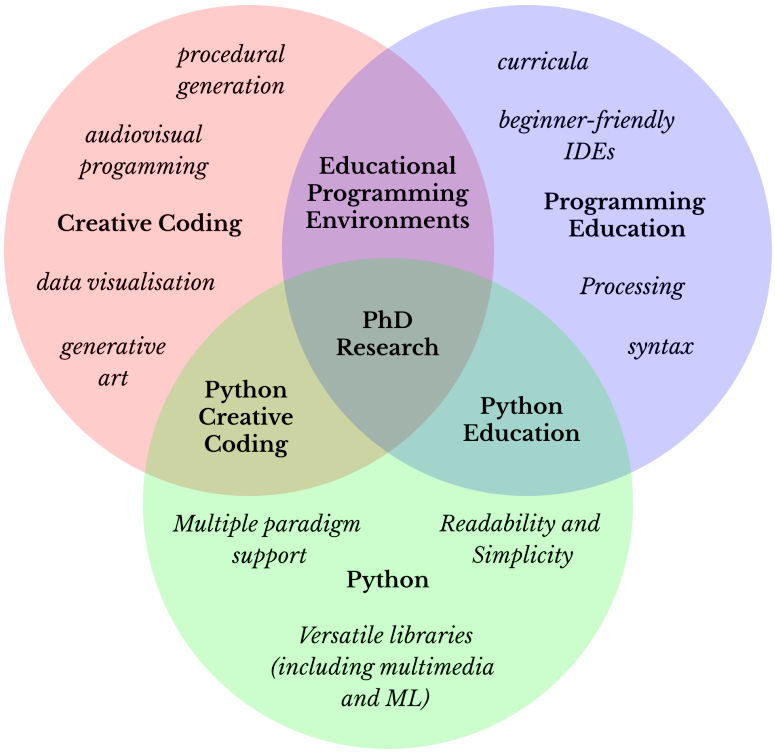


Figure 1.2: Venn diagram positioning this PhD research. Figure by the author.

The PhD focus is developing new tools, curricula, and methods that offer students in creative (and even other) disciplines an approachable way to learn *textual programming*[3] through coding graphical and interactive outputs using Python. As illustrated in Figure 1.3, the research examines the topic from three principal angles:

1. **Tool/Software Development**: Focussing on creating, integrating, and refining Python-based tools, notably the Thonny-py5mode plugin, developed as part of this research to facilitate creative coding practices. Thonny-py5mode is designed to align with the needs of creative disciplines, enabling seamless integration of Python code into artistic processes. Its development emphasises usability and accessibility, striving to accommodate students from varying technical backgrounds to engage with code through programming multimedia outputs.

[3] Textual programming involves typing lines of code instructing computers to perform tasks

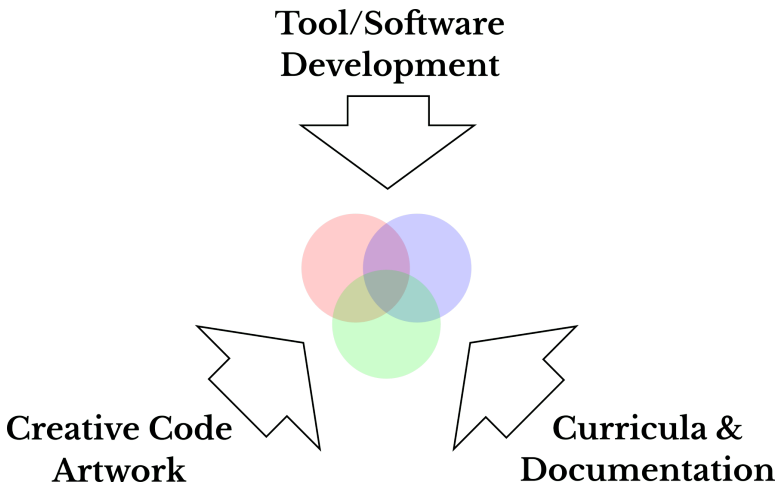


Figure 1.3: Contextualising this PhD research approach. Figure by the author.

2. **Curricula & Documentation**: Involving designing new educational materials and curricula tailored to programming education for creative disciplines; this includes comprehensive texts—a sole-authored book and web-based software tutorials—that promote a hands-on, exploratory learning experience aimed at producing graphical and interactive output using Python code. The goal is to bridge the gap between technical programming concepts and creative practice, accommodating a learning environment where students can apply programming in meaningful, artistic contexts.

3. **Creative Code Artwork**: Exploring the application of Python-based tools and techniques through the creation of digital art and interactive media, including original artworks exhibited at events and in published catalogues. These demonstrate the potential of the environments and tools explored and developed as part of this research, while also showcasing how programming can inspire creative project outcomes.

The significance of this study lies in its potential to influence programming education for creative fields. It contributes to the growing body of knowledge in creative computing pedagogy, providing educators with new strategies and tools to enhance student engagement and learning outcomes. Moreover, the practical implications of this research extend beyond the classroom, as the tools and techniques developed here can offer new ways to produce code-generated output for a broad range of creative applications, including digital art, design, interactive media, and entertainment.

The subsections that follow provide an essential introduction to Creative Coding, Programming Education, and Python, as contextualised within this PhD. These topics are explored in greater detail in the Literature Review chapter, as well as through the outputs presented in the Publications, Presentation Outputs, and Creative Works chapters.

1.2.1 Creative Coding

Creative coding spans applications like generative art, motion graphics, live coding performances, interactive installations, wearables, and robotics. In education, it engages students in programming through hands-on, interdisciplinary projects that blend art, design, and technology. This approach encourages curiosity, lowers the fear of failure, and makes coding more accessible by connecting it to visual, auditory, and tactile experiences [27, 38].

It is common in creative coding to refer to programs as "sketches," especially within Processing environments. This terminology originates from an analogy with visual arts, where a sketch refers to a rough, initial drawing meant to explore ideas. Processing (created by Casey Reas and Ben Fry in 2001) officially adopted this term (Figure 1.4) to reflect that coding should be experimental and iterative, much like drawing a sketch [186].

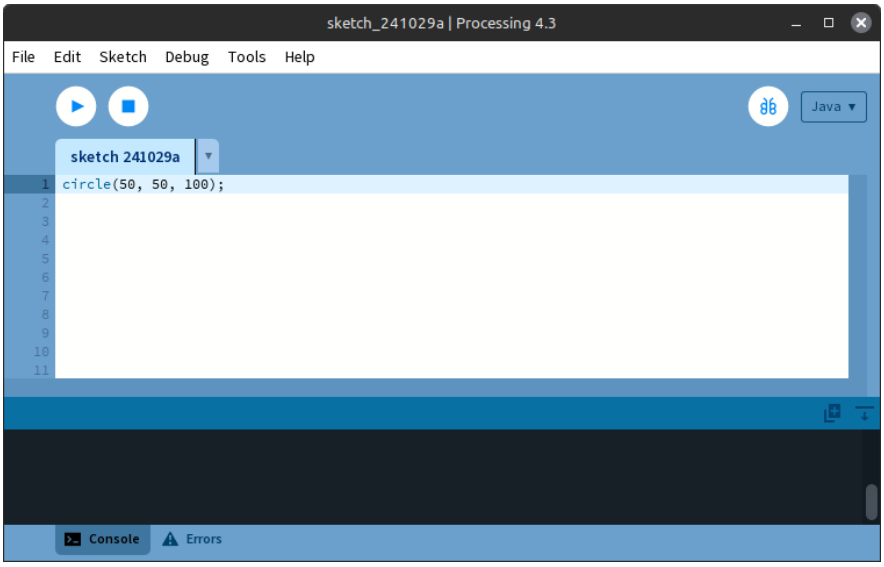


Figure 1.4: Screenshot of the Processing 4.3 IDE. Note the use of "Sketch" within the interface. Screenshot by the author.

This aligns with the philosophy of Processing, which encourages users—especially artists and designers—to approach programming with a creative, low-stakes mindset. The term "sketch" is used

in other creative coding tools and frameworks, including p5.js (a JavaScript version of Processing) and Arduino (microcontroller programming).

Creative coding emphasises creative outcomes over abstract syntax, attracting learners not typically drawn to computer science while engaging students across art, music, and design (and also computing & information sciences) [135, 247]. Its adaptability and emphasis on exploration provide an inclusive framework for programming education, supporting varied learning pathways and modes of skill development [230]. In addition to introducing core technical skills, creative coding can offer several other benefits:

Engagement and Motivation Research affirms that creative coding environments can increase student engagement and motivation by situating programming within contexts that feel meaningful and enjoyable. Walsh *et al.* demonstrate how combining craft with block-based coding in Scratch encourages sustained interest and playful exploration [243]. Dufva highlights how frameworks such as Processing and Arduino stimulate intrinsic motivation by reframing code as a form of creative expression [59]. These findings support broader claims that creative coding projects make the process of learning programming more enjoyable and encouraging [130].

Inclusivity and Appeal to Underrepresented Groups Research increasingly shows that creative coding contexts can broaden participation by engaging groups historically underrepresented in computing. Walsh *et al.* found that arts-integrated coding activities promoted inclusivity among underserved urban youth [243]; Dufva emphasises feminist and critical approaches that frame coding as a cultural and political practice rather than a purely technical skill [59]. Prominent creative coding communities explicitly promote diversity and inclusion [122], such as The Processing Foundation, whose mission statement directly highlights these values [171].

Lowering Barriers to Entry Simplified, beginner-friendly environments lower the threshold for participation by reducing technical overhead [243]. Dufva situates Processing and Arduino as accessible platforms for experimenting with art and code [59], complementing findings that such environments avoid complex setup, while simultaneously providing a pathway between novice and professional developer tools with features like autocomplete and debugging functionality, thereby facilitating a smoother transition to industry-standard practices and coding environments [130, 170].

Code Exploration Through Art Creative coding highlights the intersection of programming, art, and design. As Dufva demonstrates, this bridges a gap between programming and art education, facilitating playful experimentation through live coding, code poetry, and maker practices [59]. Similarly, Walsh *et al.* highlight how students integrated hand-crafted characters into Scratch games, blurring boundaries between craft and computation [243]. Tools like number sliders and colour pickers encourage inventive tinkering, helping students to intuitively discover new ideas while still writing textual code [235]. Some environments, such as Hydra and Visor (Figure 1.5), include REPL characteristics, providing rapid feedback in response to code edits and additions. Code is entered and executed piecewise in a REPL (read–eval–print loop), meaning the output updates without rerunning the entire program [100, 174].

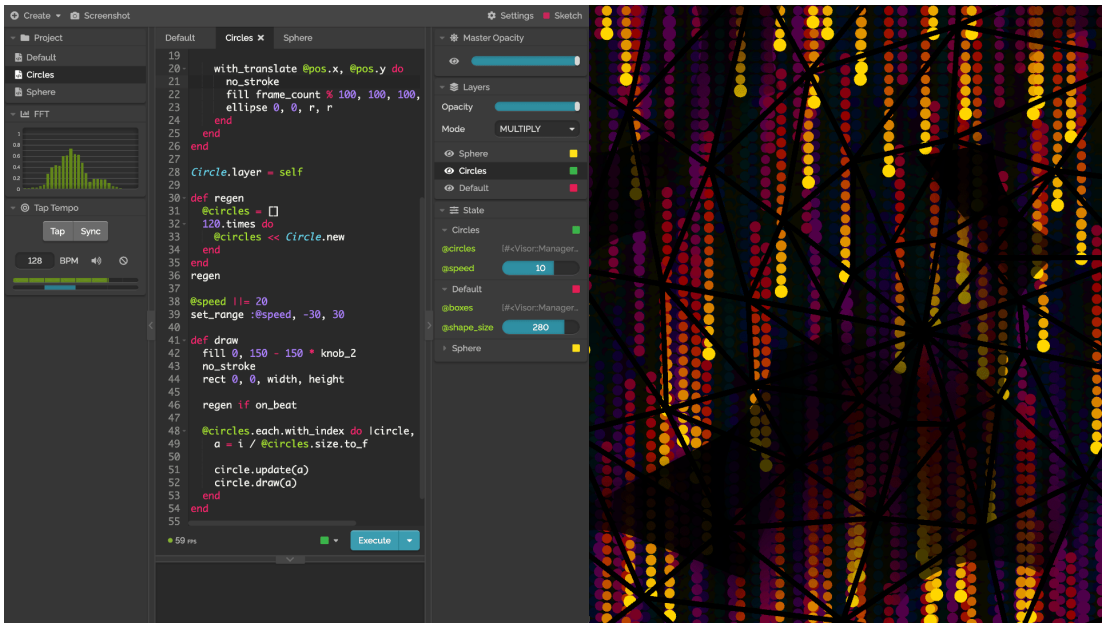


Figure 1.5: Visor is a live coding environment for real-time visual performance, bridging the gap between creative coding and VJing. Screenshot from Jack Purvis (2024). Source: [173].

Building Confidence and Reducing Frustration Evidence suggests that creative coding builds learner confidence by providing observable, incremental successes and reducing common frustrations, helping maintain learner momentum and reinforcing a sense of accomplishment [41]. Walsh *et al.* found that students reluctant to engage with coding gained confidence through hybrid craft–code projects, while those already interested developed patience and creativity [243]. Dufva similarly reports that hands-on, low-stakes assignments (e.g., "human fax machine" sound-to-drawing

tasks) helped students reframe coding as accessible and culturally relevant [59]. Static analysis tools (e.g., linting) and auto-formatting in creative coding environments help students avoid common errors and maintain organised code. Simplified syntax and libraries lower entry barriers, while supportive communities offer resources and guidance to help overcome challenges [170].

In summary, creative coding offers a dynamic and inclusive approach to programming education that resonates with diverse learners and transcends traditional coding paradigms. Beginner-friendly tools like Processing and p5.js, coupled with supportive communities, foster confidence and inclusivity. This PhD aims to combine these benefits with bespoke Python-based tools and pedagogy.

1.2.2 Programming Education

Programming education has long incorporated visual and interactive elements to enhance student engagement and understanding. Turtle graphics, commonly used in computer education during the 1980s and closely associated with the Logo programming language, provided an engaging way for children to interact with programming concepts through visual feedback [2].

Developed by Seymour Papert, Cynthia Solomon, and Wallace Feurzeig, Logo's Turtle graphics allowed children to visualise abstract programming ideas by controlling an on-screen "turtle" to draw shapes and patterns. This embodied Papert's *constructionist* learning theory: learning by making, where students build knowledge through actively constructing projects. This simple yet powerful approach sought to make coding more accessible and enjoyable, promoting problem-solving skills in a creative way. Over time, Turtle graphics evolved through various versions and adaptations of Logo [215].

Python's standard library includes a Turtle graphics module inspired by Logo's. The module allows users to control turtles in a 2D space (Figure 1.6) and remains a popular tool for beginners to learn programming fundamentals using Python's syntax and data structures [144, 239].

Processing was among the first programming environments explicitly developed for creatives—artists, designers, and others in related fields—emphasising accessibility to programming as a medium for visual expression. Unlike many earlier programming environments focused on visual output, Processing was purpose-built to meet the creative community's needs, streamlining the process of transforming artistic concepts into code and, ultimately, code into multimedia outputs [187].

Several Processing variants have appeared since the release of the (Java-based) original, each designed to extend the Processing concept into different programming environments and languages, including

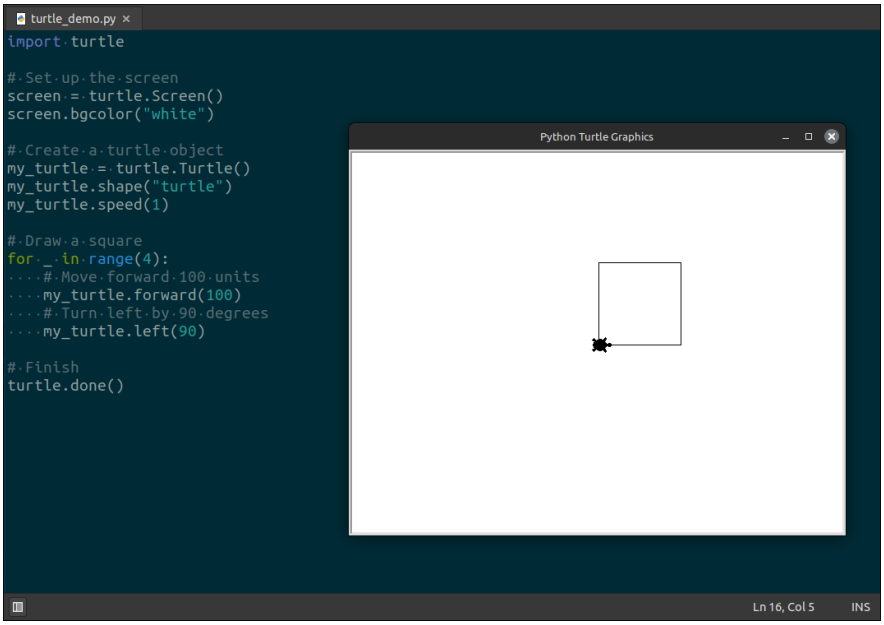


Figure 1.6: A simple square drawn using Python's Turtle graphics library, demonstrating basic commands for movement and rotation. Screenshot by the author.

p5.js (JavaScript), JRubyArt (Ruby), and Processing.py (the now outdated Python mode), among others [245].

Processing has significantly impacted programming education, inspiring learners to see coding as a technical skill *and* a medium for creative expression. There is a wealth of books, reference materials, and resources available for Processing and its variants, making it one of the most well-documented creative coding initiatives. As of writing, the Processing website lists 36 books covering topics from programming generative art to data visualisation, many with an introductory programming focus [168].

Several educational websites use Processing environments to teach programming fundamentals, often within a creative coding context, including The Coding Train, Fun Programming, Happy Coding, The Imaginary Institute, and substantial parts of Khan Academy's "Computer Programming" section. Strive, an EdTech company, leverages a Processing-based environment to teach mathematics, science, and other subjects through code [219].

At this point, it is important to reiterate that *visual programming* languages, as opposed to *textual* programming languages, fall outside the scope of this PhD topic. For instance, Scratch is a notable visual programming language developed in the early 2000s, with the primary goal of making programming accessible, particularly for young learners. It enables them to create stories, games, and animations through an intuitive block-based coding interface (Figure 1.7) [138]. The Scratch com-

munity has encouraged a culture of sharing and remixing projects, significantly contributing to its widespread adoption. Its popularity stems from the simplicity of its drag-and-drop coding interface, which promotes experimentation and iterative learning [190]. Its success has established Scratch as one of the most widely used tools for introducing young learners to programming and computational thinking, influencing the creation of other block-based educational programming environments such as Snap! and Blockly [127].

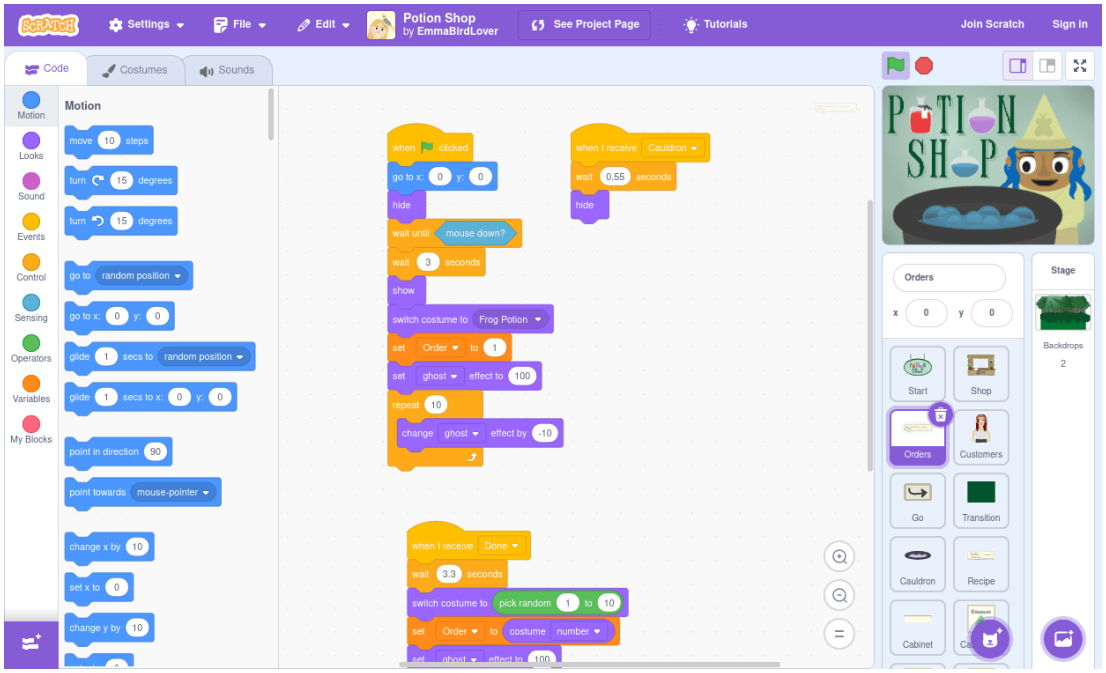


Figure 1.7: The Scratch editor's block-based interface, where users create programs by stacking visual code blocks to form logical sequences. Screenshot of the Scratch web app by the author. Source: [138].

However, while Scratch may be highly effective for introducing programming, particularly to younger learners, textual programming environments such as Processing are arguably better suited for students ready to engage with more complex concepts and develop skills more closely aligned with real-world programming practices [195]. The PhD aim is not to explore the merits of either approach (block-based vs textual), as the elected focus is Python.

1.2.3 Python

Python's readable syntax and expressiveness have contributed to its extensive use in industry and education. Unlike languages such as C++ or Java, which often require verbose implementations, Python allows for more concise expression of programming concepts [124]; it is a versatile, general-

purpose language [128]. Unlike Processing, it does not include a purpose-built IDE for creative coding or built-in functions intended explicitly for multimedia output. However, Python does offer a vast ecosystem of libraries available on PyPI[4] (Python Package Index) and elsewhere, which can extend Python's capabilities for multimedia projects.

PyPI serves as the official repository for Python packages [177]. It is the primary platform for sharing Python packages/libraries and the default source for downloading and installing them. In Python, the terms *libraries* and *packages* are often used interchangeably, though there is a distinction. Regardless, to avoid unnecessary details here, it is enough to know that PyPI primarily hosts packages that can encapsulate libraries. Tools like PIP, Python's standard package manager, connect to PyPI to fetch and install the packages users require.

Table 1.1 highlights some of the most prominent PyPI multimedia packages, all of which one could employ for creative coding projects. The tabulated data is sourced from PyPI Stats: a platform that removes the need for users to directly query raw download records via Google BigQuery [66], that provides aggregate download figures for Python packages hosted on PyPI. Note that Tkinter, a widely used GUI library known for its simplicity and ease of use [140], is part of Python's standard library and therefore not distributed via PyPI. As such, the table does not report download data for it.

This investigation to identify relevant packages is not exhaustive. Instead, Table 1.1 aims to provide a high-level overview of popular Python libraries dealing in domains of 3D graphics, animation, audio processing, GUI & interactive media, image processing, and video handling. Consider this table a cursory approximation rather than an in-depth analysis, supplemented through community discussion data—in forums like Stack Overflow, Reddit (e.g., `r/creativecoding`), and specialised multimedia programming groups on Medium—retrieved using the search term *"Python creative coding"* and limiting the results to the last five years. Moreover, there is no coverage of Python's integration with hardware platforms, such as Raspberry Pi and Arduino, which could enable the creation of interactive systems and devices capable of responding to environmental stimuli, offering physical and immersive user experiences.

One could integrate several of the Table 1.1 libraries to create, for instance, a "dynamic data harmoniser"—a creative tool designed to visualise and sonify real-time data streams, similar to apps like

[4] https://pypi.org

Category	Name	Description	Downloads (October 2024)
3D Graphics	PyOpenGL	Cross-platform, OpenGL Python bindings for 3D graphics programming	1,093,547
	VTK	Visualization Toolkit for 2D/3D scientific data visualisation	963,746
	Panda3D	Real-time 3D engine for games, simulations, and visualisations	180,107
Animation	matplotlib.animation	Packaged as part of Matplotlib for handling animation	66,218,006 (Matplotlib stats)
	Manim	Engine for creating mathematical animations and explanatory videos	48,090
	VPython	3D animation library focused on physics simulations and education	34,387
Audio Processing	Pydub	High-level interface for audio file manipulation and format handling	6,564,261
	librosa	Music and audio analysis tools for feature extraction and signal processing	2,849,617
	sounddevice	Provides bindings for the PortAudio library, with functions to play and record	1,765,578
GUI & Interactive Media	PyQt5	Comprehensive GUI framework based on Qt, with extensive widget collection	4,742,240
	Pygame	Game development framework with graphics, sound, and input handling	3,660,419
	Kivy	App development framework with multi-touch and cross-platform support	716,535
	Tkinter	Standard Python GUI library, included in Python install	–
Image Processing	Pillow (PIL Fork)	Core image processing library for opening, manipulating, and saving image files	115,169,468
	OpenCV	Computer vision library with real-time image/video processing capabilities	14,571,086
	scikit-image	Collection of algorithms for scientific image processing with scipy integration	11,734,284
Video Handling	MoviePy	Video editing, processing, and compositing, with effects and transitions	1,470,720
	ffmpeg (for Python)	Python interface for FFmpeg multimedia framework for video processing	266,091
	VLC-python	Python bindings for VLC media player, enabling video playback and streaming	110,688

Table 1.1: Overview of Python multimedia libraries based on 01 October 2024 PyPI Stats data

TwoTone[5] or DataSonifyer[6]. Such a tool might utilise matplotlib.animation to produce smooth, animated visualisations, librosa to generate audio tones or rhythms reflecting data trends, and PyQt5 to develop a GUI that enables users to adjust visualisation parameters and control audio properties. Another option is to attempt this using just Pygame, which would likely require more effort to implement complex features and potentially lead to performance and scalability limitations.

There are many real-world examples of Python libraries applied in creative projects. Take, for instance, Frederic Brodbeck's *Cinemetrics*: a tool designed to measure and visualise data from films, interpreting their unique characteristics as visual "fingerprints" (see Figure 1.8).

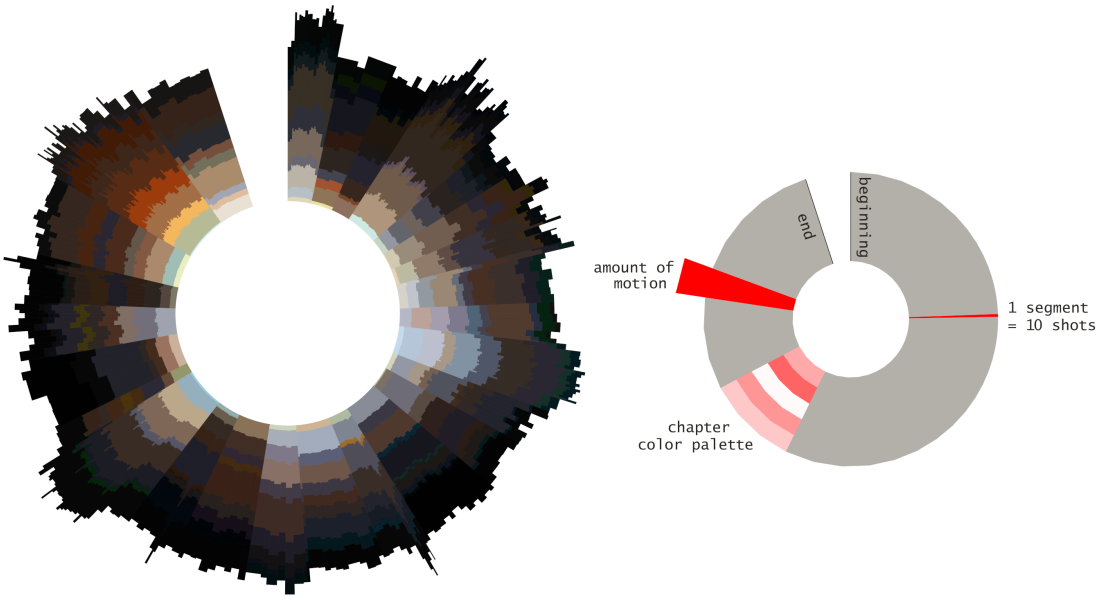


Figure 1.8: Cinemetrics fingerprint of *Quantum of Solace* (2008). Image by Frederic Brodbeck. Screenshot from cinemetrics.site. Source: [30].

Using Python libraries including OpenCV, Cinemetrics extracts and analyses a film's elements, including editing patterns, colour palettes, and motion. It transforms this information into an easy-to-interpret graphical representation.

This PhD research investigates how Python can support creative computing in both educational and practical contexts. It focuses on the language's rich ecosystem of artistic and multimedia-capable libraries while developing new tools and techniques to expand its impact in creative coding contexts.

[5] https://twotone.io

[6] https://datasonifyer.de

1.3 Research Goals

This research aims to design, develop, and evaluate Python-based creative computing tools and techniques that enhance programming education through visual learning contexts. It seeks to empower students and creatives to engage with Python programming concepts through creative coding practices, bridging the gap between technical skills and artistic expression. In doing so, it will contribute new Python tools, pedagogical resources, and creative coding approaches to the broader field of creative computing education.

1.3.1 Research Objectives

To achieve the research goals outlined above, this study defines the following key objectives, designed to break down the overarching research aims into specific, actionable, and measurable outcomes:

- **Tool Development**: Develop and refine Python-based creative coding environments/tools, primarily the Thonny-py5mode plugin, to simplify the process of writing, running, and debugging interactive multimedia projects. These solutions will cater to users with diverse technical backgrounds while offering robust capabilities for creating graphical output using code.
- **Educational Materials**: Create comprehensive and accessible curricula and documentation—including tutorials, a sole-authored book, and other learning materials—to help educators and students effectively integrate creative coding practices into Python-based programming education.
- **Creative Outputs**: Demonstrate the potential of new Python tools and techniques through the creation and exhibition of artworks and interactive media, showcasing the applications in real-world creative projects.
- **Empirical Evaluation**: Conduct user studies with students to assess the effectiveness, usability, and impact of Thonny-py5mode on their Python learning experiences. Collect feedback through surveys and iterative testing to guide design improvements.
- **Broader Dissemination**: Share findings and contributions through scholarly publications, high-impact conference presentations, and creative showcases, promoting wider adoption and inspiring further advancements in Python-based creative computing practice and education.

These objectives provide clear targets for achievement, supporting this PhD study in meaningful contributions to enhancing Python programming education through creative computing environments.

1.4 Research Questions

To achieve the research goals and objectives, this study is guided by a central research question (RQ), supported by a series of sub-questions (SQ #) that further define and focus the thesis.

RQ: How can we enhance creative computing environments through new Python tools and practices to improve programming education in visual learning contexts?

While the primary research question encapsulates the thesis' overarching aim, the sub-questions break it down into specific components, addressing distinct facets of the inquiry and guiding the methodological approaches:

SQ 1: How can Python-based software, similar to Processing.py, be designed and developed to best support creative coding practices?
This focuses on the technical and design aspects of developing a new creative coding solution. It addresses questions of usability, accessibility, and functionality, exploring how environments/tools can facilitate learning through interfaces and features tailored to producing graphical output.

SQ 2: How effective are the proposed tools and techniques in improving learning outcomes and student engagement?
This question evaluates the impact of the software and techniques on students' ability to grasp Python programming concepts and apply them. Empirical studies will assess how these innovations influence learning experiences, confidence, and performance.

SQ 3: How can creative coding outputs demonstrate the applicability of the developed tools in real-world contexts?

This question explores the practical applications of the developed solution(s) in areas such as digital art creation, the implementation of novel techniques, and the development of educational resources. It aims to highlight topics and examples that can encourage students, educators, and practitioners to embrace Python programming as a medium for both artistic expression and pedagogical innovation.

1.5 Timeline

The Figure 1.9 timeline outlines the significant milestones and achievements of this PhD journey, structured to span roughly six years to accommodate the extended duration required for part-time study. While a full-time PhD typically requires a minimum of three years, this part-time pathway allows an approach that balances research, professional obligations (full-time employment for the duration of study), and other personal commitments.

- `2019, Oct.` Commenced PhD studies at Victoria University of Wellington
- `2021, Apr.` Published *Learn Python Visually* (No Starch Press)
- `2021, Aug.` Released software: Thonny-py5mode alpha
- `2022, Feb.` Successfully completed confirmation of candidature
- `2022 →` Continued to work on publication/presentation/creative outputs
- `2023, Nov.` Transferred PhD to Torrens University Australia
- `2024 →` Continued to work on thesis
- `2024, Nov.` Human research ethics approval (Application 0394)
- `2025, Feb.` Conducted software user study
- `2025, Jul.` Final write-up of thesis
- `2025, Oct.` Anticipated submission

Figure 1.9: Timeline of PhD milestones from start to finish

Commencing in October 2019, the timeline reflects steady progress, including major outputs such as the publication of *Learn Python Visually: Creative Coding with Processing.py* in April 2021 and the release of the Thonny-py5mode plugin in August 2021. It also highlights key moments, such as the PhD transfer to Torrens University Australia in November 2023 and the final stages of research and thesis writing, culminating in the anticipated thesis submission indicated as the final milestone.

2 Literature Review

This chapter critically examines literature concerning creative coding software to position this thesis within the broader academic discourse. The insights drawn from this review will inform opportunities to enhance programming education through the development of new Python-based tools and environments tailored to visual learning contexts.

Figure 2.1 illustrates how this literature review integrates within the broader thesis structure. The left column lists the five research objectives outlined in the "Introduction" chapter. **Tool Development** maps directly to section **2.1** of the *Literature Review* (middle column), which in turn informs the *Outputs chapters* on the right. The remaining *Research Objectives* directly link with the *Outputs chapters*, with **Broader Dissemination** encompassing all three.

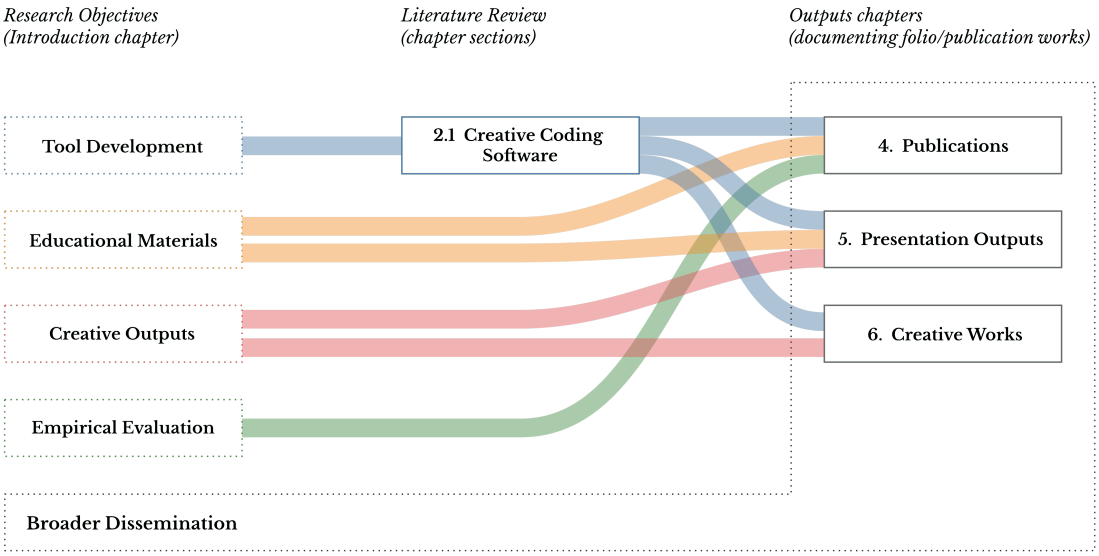


Figure 2.1: Literature review integration into the broader thesis. Diagram by the author.

One should not view the blue paths between the columns of Figure 2.1 as flowing from left to right; rather, the connections between **2.1 Creative Coding Software** and its adjacent nodes are bidirectional,

each informing and shaping the other. Educational Materials, Creative Outputs, Empirical Evaluation, and Broader Dissemination do not feature dedicated sections here.

Due to the folio-based structure of the thesis, this chapter is deliberately concise, with much of the literature review material integrated into the various outputs presented in later chapters—particularly within the Publications chapter (as part of the journal articles), the Presentation Outputs, and, to a lesser extent, the Creative Works chapter.

In summary, this literature review surveys the current landscape of creative computing tools relevant to this study, highlighting opportunities for advancing Python programming education through creative coding environments.

A Note on Python Pedagogy

Creative coding literature offers a mix of theoretical and practice-led perspectives, but proportionately little of it focuses on Python. Unsurprisingly, there is a predominance of studies and curricula that draw on environments such as Processing, p5.js, or Scratch, owing to their design intent and popularity in arts and creative contexts. It is therefore important to situate this project within the broader context of Python pedagogy research, which demonstrates that Python is both widely adopted and well suited as an introductory programming language [58, 163, 191].

Numerous empirical studies and pedagogical reviews attest to Python's effective role in introductory programming education, albeit often within CS contexts. For instance, a study in Taiwan compared students learning to code using Python with their peers learning Java; the former group exhibited significantly improved learning motivation, self-efficacy, and programming performance, and also showed fewer maladaptive cognitions (e.g., avoidance or frustration) [120].

Researchers have employed Python-based teaching to explore new learning tools, confirming its applications in pedagogical innovation. Examples include PyKinetic, which utilises short Python exercise sessions delivered via smartphones to improve coding skills [64]. Using Online Python Tutor, a web-based coding and visualisation tool, educators and students can write scripts directly in a web browser, trace program execution both forwards and backwards, examine the runtime state of data structures, and share their visualisations on the web [86].

Comparative research on block-based versus text-based modalities, as well as recent studies on block-to-text transitions, further informs Python pathways for learners with visual programming experience [196].

As described in Section 1.2.3 of the Introduction, broader reviews of programming pedagogy highlight Python's clear syntax, readability, and extensive standard library as key enablers of learning. In addition to supporting introductory contexts, Eteng *et al.* advocate the benefits of Python in resource-constrained settings [61]. In established institutions and programmes, widely used textbooks and open resources further underscore Python's effectiveness and versatility in instructional and curricular design. Severance's *Python for Everybody*, Downey's *Think Python*, and Zelle's *Python Programming* each emphasise different pedagogical priorities: interactive notebooks, data-driven problems, and problem-solving foundations, respectively [42, 43, 258]. Among other resources, these texts demonstrate Python's broad applicability to various programming domains and teaching styles.

These aspects justify the need for Python-specific pedagogies, motivating the software choices adopted in this PhD. This chapter elects to avoid a dedicated section on Python pedagogy, as this topic is addressed in the *Towards a Python 3 Processing IDE for Teaching Creative Programming* article, which also includes an analysis of exclusively Python-based creative coding environments. The reader will find the article reproduced in full in the "Publications" chapter. Nevertheless, Python references appear throughout this chapter when discussing relevant examples, research, and pedagogical perspectives.

2.1 Creative Coding Software

This section explores various environments and tools specifically designed for creative coding, assessing their features, educational applications, and limitations. It primarily aims to identify key characteristics of Python- and non-Python-based environments that can contribute to and inform the development of a new Python-focused creative coding solution, Thonny-py5mode.

2.1.1 Defining Environments and Tools

Although the terms *environment* and *tool* are often used interchangeably in creative coding discourse, this section adopts a more precise distinction for analytical clarity. While a universally standardised definition appears lacking, several scholarly works across human–computer interaction, media art, and computing education implicitly differentiate between the two. For instance, Reas and Fry contrast the Processing IDE with its underlying Java library, illustrating the distinction between an integrated workspace (environment) and reusable code components (tools) [186]. As Kelleher and Pausch outline in their taxonomy, an environment typically provides an integrated interface that combines code editing, execution, and visual feedback within a single workspace, often designed to lower barriers

for novices. In contrast, tools are narrower in scope, typically facilitating specific tasks within or alongside such environments [108].

However, the distinction between an environment and a tool is often nuanced. For example, Python Mode for the Processing IDE: while technically a plugin (see tool), it transforms the host IDE into something resembling a distinct (albeit related) environment, blurring the conceptual line between the two categories.

This section focuses on environments—or tools that behave like environments, such as Python Mode for the Processing IDE—designed to provide a unified, creative-coding-oriented workspace centered around a specific language, framework, or set of libraries.

Importantly, this PhD research does not entirely exclude other "tools." Notably, the Presentation Outputs chapter considers several, particularly libraries that integrate effectively with Python-based creative coding workflows but do not function through augmenting any existing editor or IDE.

2.1.2 Introduction to Creative Coding Environments

A close link has existed between code and creative work for decades, with computer programming playing a crucial role in shaping computational aesthetics. One can trace this relationship back to the 1960s when pioneers such as Vera Molnar, Frieder Nake, and Georg Nees (Figure 2.2) began using algorithmic methods to create art, demonstrating how textual code—primarily using languages like Fortran, ALGOL, and custom plotting systems—might serve as a tool of creative expression for generating dynamic patterns, compositions, and abstract visuals [143, 164].

In 1975, MIT's *Visible Language Workshop*, led by Muriel Cooper, would investigate how computational methods could revolutionise typography, layout, and publication design. In the late 1990s, MIT's *Aesthetics + Computation Group*, headed by John Maeda, would start on work that eventually led to the Processing IDE, building upon foundations laid by Cooper. When Maeda joined MIT's *Media Lab* in 1996, he continued to explore how textual programming could drive artistic and design-oriented practices. This lineage of computational creativity attracted designers and artists eager to push the boundaries of what text-based code could generate visually. Among them were Ben Fry and Casey Reas, two research assistants in Maeda's group. Inspired by Maeda's pioneering creative coding environment, Design By Numbers (DBN), Fry and Reas examined the accessibility of programming for students in creative fields, questioning how computational languages might be adapted to better

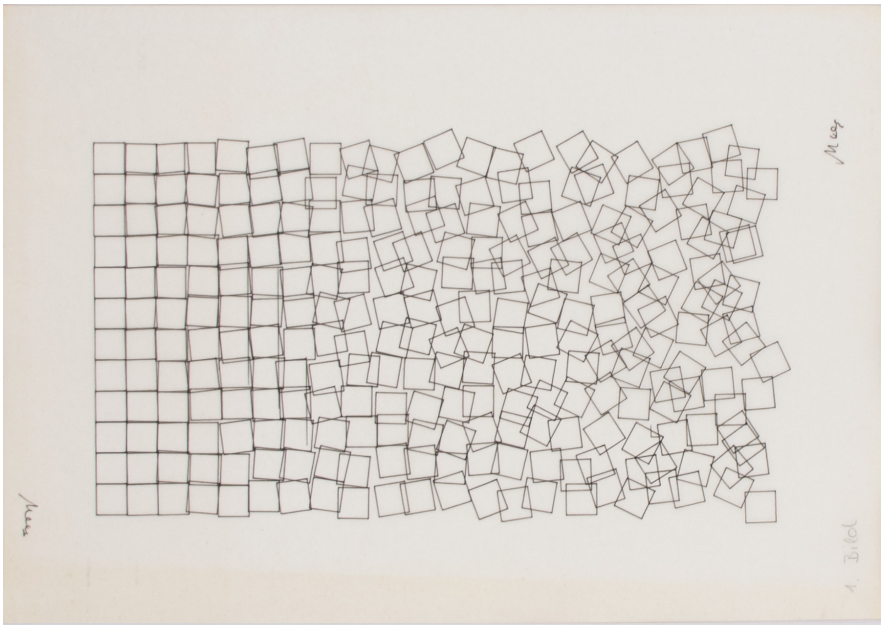


Figure 2.2: **Schotter (Gravel Stones),** 1968, by Georg Nees. Plotter drawing in ink on vellum, 11.25 × 8 in., Inventory ID: Nees-1968-01. Rotated 90° counterclockwise. Image source: [146].

serve artists and designers. For instance, could writing code become a direct and intuitive part of the creative process? [186, 218]

Fry and Reas' efforts would result in Processing (Figure 2.3), a creative coding environment released in the early 2000s, designed to bridge the gap between programming and visual art.

Processing provided an immediate and interactive way to write simplified Java code that generated visuals, much like the early algorithmic art experiments of the 1960s but with modern accessibility. Unlike traditional programming environments that required extensive setup, Processing's minimal execution model—where users could type a few lines of code and instantly see a visual output—resonated with a sketching mindset common in the arts [170].

Processing has inspired and influenced other textual creative coding environments, playing a pivotal role in programming education that equips artists, designers, and educators with powerful tools to explore interactive visualisations, generative design, and multimedia computation [118]. This includes OpenFrameworks (2005), Cinder (2010), and p5.js (2013), which all offer a structured yet flexible approach to creative computing and learning programming concepts.

Unlike node-based or block-based programming environments such as Max/MSP or Scratch, text-based creative coding environments require learners to explicitly structure logic and engage with

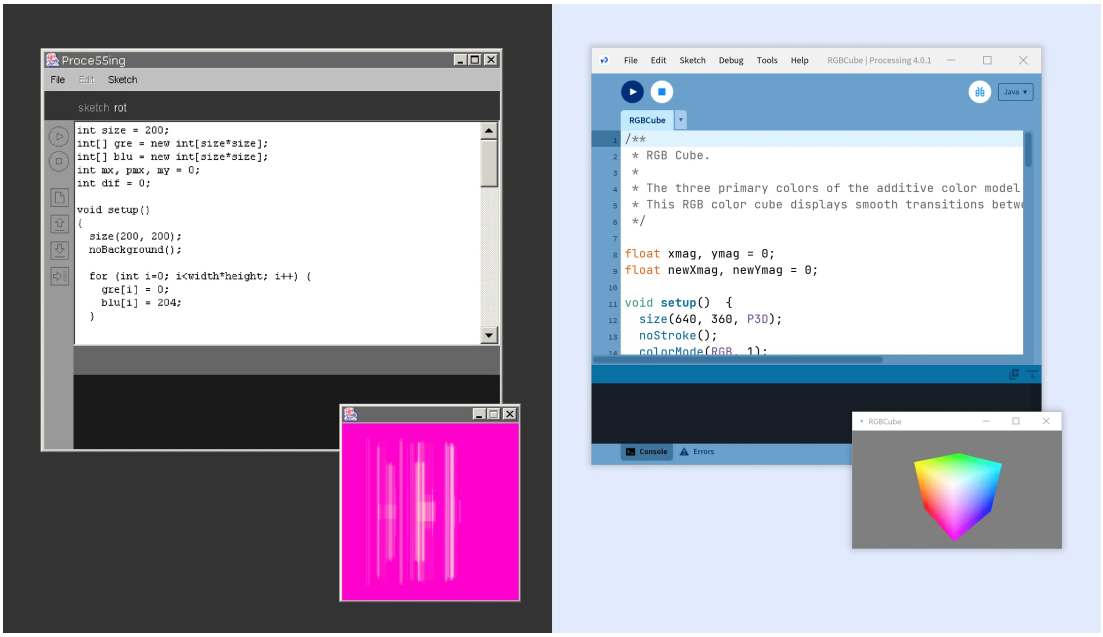


Figure 2.3: Left: alpha version of Processing (then spelled "Proce55ing"); right: Processing 4.0.1, released in 2022. Image from the Processing Foundation. Source: [167].

computational thinking through written code. While node- and block-based environments make programming more accessible to non-technical users, creating compelling projects using those systems often still requires an understanding of core programming principles such as variables, control structures, object-orientation, debugging, and workflow management [94]. Many beginners successfully transition from environments like Scratch to text-based coding with proper scaffolding; Processing can assist in this transition by bridging textual algorithmic logic with visual output while fostering syntax comprehension and creative exploration [184]. Moreover, text-based creative coding environments designed to lower the barrier to entry can help students progress toward more complex concepts, developing skills that align with many real-world programming practices [195].

2.1.3 Prominent Text-Based Creative Coding Environments

This subsection adopts a mixed-method approach that combines a community-curated resource with academic literature. The goal is to triangulate **prominent text-based creative coding environments oriented to generating graphical output** using both grassroots and academic peer reviewed indicators.

Scope

This investigation narrows its focus to contemporary text-based environments that support graphical output, a subset particularly relevant to creative coding practices and this PhD research. However, these are challenging to catalogue. Many are niche, experimental, or in the early stages of development, often labelled alpha or beta; some exist only as components within broader platforms. These factors can complicate identifying, evaluating, or treating them as standalone entities. A more fundamental complication is perhaps definitional: for instance, at what point does a plugin or extension warrant recognition as a distinct environment (rather than a supporting tool)? Similarly, how should one assess sufficient "maturity"? Through sustained development, user adoption, documentation, responsiveness to issues, or some combination thereof?

Several factors, including community involvement, accessibility, and underlying technology frameworks, influence the ongoing development of text-based creative coding environments [10]. The dynamic and evolving nature of the field reflects a landscape lacking clear boundaries, where environments and tools overlap, adapt, and serve different artistic goals and skill levels. Some environments (exemplified by the REPL type) prioritise real-time interaction and immediate visual feedback; others foreground generative processes, algorithmic composition, or evolutionary-type experimentation [113].

This investigation excludes tools that provide only algorithmic or computational functionality without significantly influencing the coding experience—like mathematics or physics libraries. Instead, it highlights environments better described as "IDEs," "lightweight IDEs," or in some cases, "code editors". That is to say: cohesive workspaces designed for creative coding, typically anchored to a specific language, framework, or library ecosystem.

Process

Rather than attempting an exhaustive survey, this section aims to identify the most "prominent" environments for text-based creative coding focused on graphical output. One might establish those using metrics based on historical impact, widespread use, technical innovation, and/or influence on modern creative coding practices. However, this returns to the challenge of discovering and cataloguing them all, compounded by the difficulty of sourcing reliable data to support 'prominence' metrics.

Academic database searches (ACM, IEEE Xplore, and Google Scholar) yield few comprehensive lists, robust datasets, or systematic/comparative studies to support such analysis. Moreover, academic literature concerning creative coding environments remains relatively limited in both scope

and depth. This scarcity appears primarily attributable to the field's origins in art, design, and media practice. Unlike computer-science research, dissemination frequently occurs through channels such as workshops, exhibitions, and community platforms rather than peer reviewed publications [123]. The work of Chibalashvili *et al.* provides one of the few relevant studies [135, 148, 225], offering a table of "the most widespread platforms, environments, and languages of creative programming," categorised into: **text-based** (DBN, Processing, p5.js, OpenFrameworks, SuperCollider); **visual** (Cables, Max/MSP/Jitter, Pure Data, vvvv, Nannou); and **hybrid** (OPENRNDR, TouchDesigner) [46]. However, the **text-based** category is evidently limited, listing only four entries. It includes DBN, a precursor to Processing, as well as SuperCollider, which primarily focuses on audio synthesis (and is, therefore, not directly relevant to this study).

Given the aforementioned challenges, the elected method integrates a community-curated list of creative coding resources—the *Awesome Creative Coding* GitHub repository (Figure 2.4)—with official documentation and academic sources, such as the work of Chibalashvili *et al.*, to evaluate the relevance and significance of each list entry.

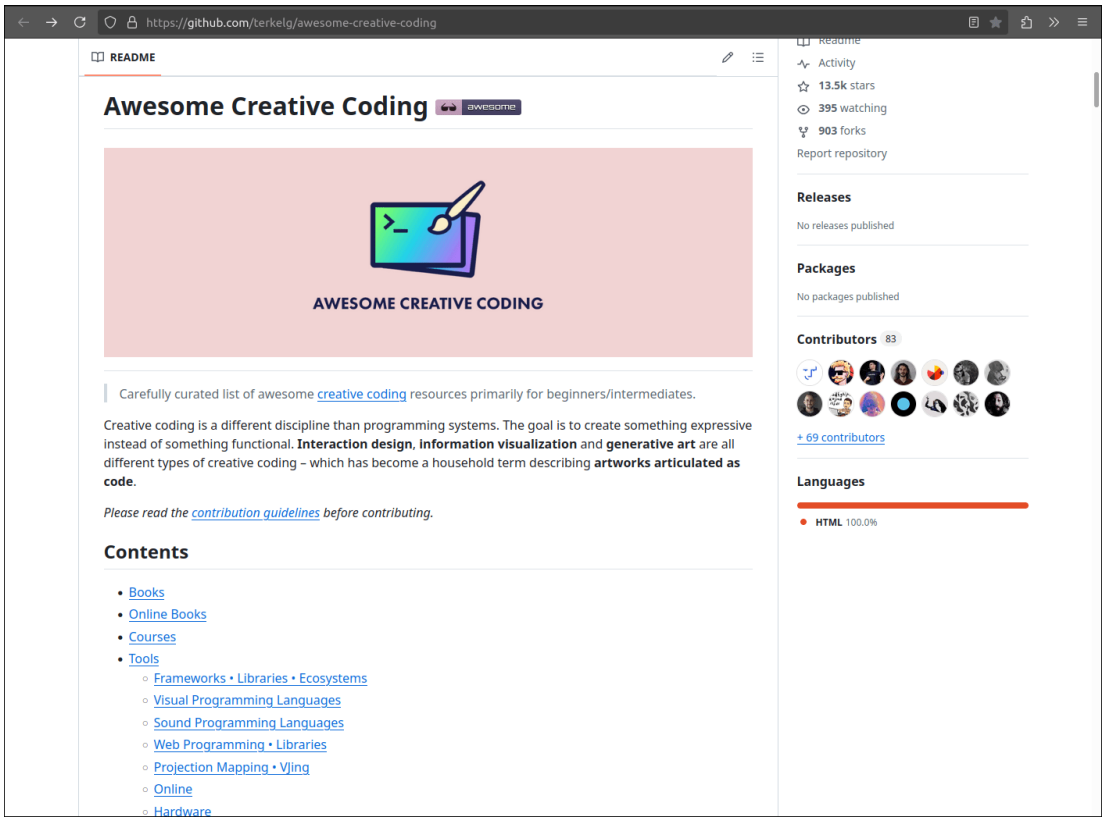


Figure 2.4: *Awesome Creative Coding* GitHub repository by Terkel Gjervig. Screenshot by the author. Source: [73].

This triangulated approach assesses both scholarly relevance and practical adoption, aiming to provide a sufficiently reliable basis to inform the design and feature set of a new Python-based creative coding environment: Thonny-py5mode.

Drawing on a community-maintained and categorised resource leverages the concept of a *folksonomy*: a user-generated classification system that emerges from grassroots participation, as opposed to one imposed solely by researchers or formal institutions [71]. In this context, the folksonomy represents a collaboratively maintained list of creative coding tools curated by an online community of enthusiasts. Folksonomies typically (but not always) rely on tag-based approaches, like the Flickr system for tagging photos. However, an *Awesome Creative Coding* contributor does not explicitly employ tags; instead, the repository functions analogously by organising entries into a thematic taxonomy of lists and sub-lists shaped through collective input and community consensus. This format follows the "awesome list" convention widely used on GitHub, where contributors maintain high-quality lists on specific topics [129, 217].

It is acknowledged that community-curated sources often lack the funding and formal review processes underpinning traditional academic sources—typically deemed authoritative due to peer review, structured validation, and institutional credibility [153]. However, *Awesome Creative Coding* is particularly valuable here, as it reflects bottom-up curation by practitioner communities where scholarly classification seemingly lags behind practice. To reconcile these differing strengths, this investigation employs a critical evaluation framework, the CRAAP Test (Currency, Relevance, Authority, Accuracy, and Purpose) [111], to validate the reliability of *Awesome Creative Coding*.

Sources

With a framework in place, the following points detail the four principal sources employed to derive a list of most prominent text-based creative coding tools:

- **Awesome Creative Coding**: A curated repository hosted on GitHub, available at https://github.com/terkelg/awesome-creative-coding, which presents a "carefully curated list of awesome creative coding resources primarily for beginners/intermediates." As of February 2025, the repository has garnered 13.4k stars—a measure of approval within the open-source community—and contributions from 82 collaborators. First published in November 2016, it remains actively maintained, with approximately 15 commits in 2024 alone. No comparably comprehensive or up-to-date resource was identified in academic databases, positioning this

list as a valuable, community-driven reference point. Repository updates are subject to peer feedback and approval via GitHub pull requests, further reinforcing the credibility of the curation process.

A CRAAP evaluation confirms the repository's reliability. It demonstrates strong *currency*, with over 400 commits reflecting ongoing maintenance and responsiveness to developments in the field. It offers highly *relevant* content focused on tools and libraries for generative art, data visualisation, and interaction design—key domains within creative coding. Regarding *authority*: Terkel Gjervig, a Brooklyn-based creative technologist with a portfolio of digital projects, provides the ultimate oversight of the repository. Its popularity, evidenced by its high star count and widespread community engagement, enhances its credibility. *Accuracy* is supported through direct links to original sources, enabling independent verification. The repository's *purpose* is educational and non-commercial, clearly aimed at supporting practitioners and learners within the creative coding community.

- **ACM Digital Library**: An authoritative academic database accessible at https://dl.acm.org, maintained by the Association for Computing Machinery (ACM), and serving as a vital resource for researchers, practitioners, and students in its field. It hosts a wide range of peer reviewed publications across computing disciplines. This includes significant research relevant to creative computing, including generative art, real-time graphics, human–computer interaction (HCI), and algorithmic design. Key contributions are often published through ACM's special interest groups such as *SIGGRAPH* (covering graphics and interactive techniques) and *SIGCHI* (HCI and user experience), both of which intersect with creative coding practices.

- **IEEE Xplore**: Another critical academic repository, available at https://xploreqa.ieee.org, maintained by the Institute of Electrical and Electronics Engineers (IEEE), and serving as a vital resource for researchers, professionals, and students worldwide. It encompasses extensive literature across engineering and computer science. Areas relevant to creative coding include programming education, generative systems, HCI, and real-time visual computing. Noteworthy journals include: *TVCG* (IEEE Transactions on Visualization and Computer Graphics), which covers real-time rendering and data-driven design; and *CG&A* (IEEE Computer Graphics and Applications), which "bridges the theory and practice of computer graphics topics, including

modelling, rendering, animation, (data) visualisation, HCI/user interfaces, novel applications, hardware architectures, haptics, and virtual- and augmented-reality systems."

- **Google Scholar**: A freely accessible search engine for scholarly literature, available at https://scholar.google.com, which aggregates academic publications across disciplines, including conference proceedings, journal articles, theses, and technical reports. It is beneficial for locating grey literature and tracking citations across institutional boundaries [89]. Although it is less curated than traditional academic databases, Google Scholar indexes a broad range of interdisciplinary sources often not included elsewhere. In this context, it provides a supplementary resource to verify the academic relevance of *Awesome Creative Coding* entries not indexed in ACM or IEEE databases.

The process began with extracting relevant creative coding entries from the *Awesome Creative Coding* repository that met the definition of a text-based coding "environment" rather than a "tool," as defined earlier under Defining Environments and Tools. These entries were then cross-referenced with official documentation (through direct links to original sources) and academic literature from the ACM Digital Library and IEEE Xplore to verify their relevance and usage. Google Scholar served as a supplementary source to ensure broader coverage.

Selection

For inclusion in the final table of prominent text-based creative coding environments (Table 2.1), an *Awesome Creative Coding* entry was required to meet all the following criteria:

1. enable writing and executing creative code within a unified workspace that supports continued project work (across several coding sessions);
2. support writing code that generates graphical output (as opposed to audio or other non-visual forms);
3. use a primarily text-based programming interface;
4. show evidence of active development or an engaged user community; and
5. demonstrate relevance in academic publications or grey literature.

Broadly scoped platforms were excluded, such as CodePen when combined with p5.js, or NEORT with HTML5 Canvas. These platforms function primarily as general-purpose coding playgrounds that provide convenient access to libraries and APIs. The mere ability to import creative coding libraries does not constitute a dedicated environment with structured tooling or workflows specifically tailored to creative coding.

Results

Table 2.1 presents a finalised, alphabetically ordered list of prominent text-based creative coding environments that support graphical output, derived according to the selection criteria and method outlined above.

However, some exclusions from this table warrant further explanation—namely, game engines, (limited) web playgrounds, and shader tools.

Exclusions: Game Engines

Game engines can offer powerful platforms for creative computing, particularly in contexts involving real-time interaction and 3D graphics, including immersive media. Engines such as Unity and Godot provide advanced rendering pipelines, physics systems, animation tools, and cross-platform deployment [19, 28]. These features can significantly extend creative possibilities beyond lightweight environments like Processing or p5.js [29, 147]. For instance, the eTextbook *Generative Unity* demonstrates Unity's creative coding capabilities [72]; CETI (Creative and Emergent Technology Institute)[1] has run workshops on *Creative Coding with Game Engines*.

While game engines can be complex, there are more accessible options for beginners. For instance, for Python, there is Pygame. The best-selling book[2] *Python Crash Course* introduces Pygame through a simple alien shooter, and *Creative Coding in Python* recommends it as a lightweight 2D game library for further exploration [128, 232].

A companion table, Table 2.2, presents entries classified as *game engines*, derived using the same source and method as Table 2.1. As *Awesome Creative Coding* includes no explicit "game engine" category, these entries were drawn from its different lists and sub-lists.

---

[1] https://ceti.institute

[2] https://www.amazon.com.au/Python-Crash-Course-Eric-Matthes/dp/1718502702

Environment	Platform	Language	License	Description
DrawBot	macOS	Python	Open-source	Education-oriented, 2D graphics programming environment
Hydra	Web-browser	JavaScript	Open-source	Live-coding environment for real-time visual synthesis
OpenProcessing	Web-browser	JavaScript	Propietary editor	Online platform for sharing and exploring creative coding, especially with p5.js
P5LIVE	Web-browser	JavaScript	Open-source	Live-coding platform for collaborative p5.js visuals
p5.js Editor	Web-browser	JavaScript	Open-source	Platform for writing, running, and sharing p5.js sketches
Processing	Linux, macOS, Windows	Java	Open-source	Creative coding environment for visuals, interaction, and media art
py5 (Jupyter Notebooks)	Linux, macOS, Windows	Python	Open-source	Library that brings Processing creative coding features to the Python 3 ecosystem
ShaderGif	Web-browser	JavaScript (for Canvas & P5.js)	Open-source	Platform for coding and exporting generative GIFs using GLSL1/2, JavaScript Canvas, and P5.js
Shelly	Web-browser	Bespoke Turtle language	Closed-source, free	Turtle graphics environment for learning coding using simple drawing commands
Shoebot	Linux, macOS, Windows	Python	Open-source	Environment for creating generative vector graphics and animations with code
Turtletoy	Web-browser	JavaScript	Closed-source, free	Platform for generative line art using minimalist Turtle graphics
tixy.land	Web-browser	JavaScript	Open-source	Minimalist platform for animating a 16×16 grid using short expressions limited to 32 characters

Table 2.1: Prominent text-based creative coding environments that support graphical output, alphabetically ordered. Derived from *Awesome Creative Coding*.

Environment	Platform	Language	License	Description
Babylon.js	Web-browser	JavaScript	Open-source	Complete framework for building 3D games with HTML5 and WebGL/WebGPU; includes a robust playground feature
Godot	Linux, macOS, Windows	GDScript, C#	Open-source	2D and 3D development; valued for its lightweight design, scene-based architecture, and flexible scripting with GDScript or C#
Phaser Editor	Desktop, Web-browser	JavaScript, TypeScript	Open-source; desktop version at no cost	2D game framework; supports visual scene editing and asset management using Phaser editor; limited built-in code panel suitable for previews and small edits
PlayCanvas	Web-browser	JavaScript	Open-source	Collaborative 3D game engine using JavaScript and WebGL; well-suited to projects involving real-time rendering and browser-based deployment
TIC-80	Linux, macOS, Windows, Web-browser	Lua, JavaScript, Python, and more	Open-source (non-pro version)	'Fantasy computer' for making, playing, and sharing tiny pixel-art games (similar to PICO-8 or Pyxel)
three.js	Web-browser	JavaScript	Open-source	3D graphics using WebGL, suited to immersive web experiences; web platform includes an editor with scripting functionality (like a game engine)
Unreal Engine	Linux, macOS, Windows	C++, Blueprints	Partially open-source; free to use	High-fidelity 3D engine used in games, simulations, and virtual production; supports creative computing through visual and textual coding
Unity	Linux, macOS, Windows	C#	Proprietary; free version available	Robust 2D/3D rendering, physics, and scripting tools; creative coding applications for installations, real-time multimedia work, and XR

Table 2.2: *Awesome Creative Coding* entries most appropriately classified as game engines, excluded from the final table (Table 2.1)

Table 2.2 includes one entry never listed on *Awesome Creative Coding*: Unreal Engine. This appeared to be a notable omission, and further investigation confirmed its relevance and significance within creative computing practice. Unreal Engine supports advanced real-time rendering with applications for generative art, immersive media, and multimedia installations. Its interoperability with creative tools such as TouchDesigner and Houdini facilitates procedural workflows beyond conventional game development [204]. Additionally, Unreal Engine is found adopted in creative computing curricula. For instance, Parsons School of Design—ranked fourth globally for Art & Design in the QS World University Rankings 2023—offers a creative coding course that explicitly incorporates Unreal Engine (CRN: 2748), alongside another creative coding course focusing on Unity (CRN: 11556).

While game engines are clearly relevant in creative coding contexts involving advanced interactivity or rich media integration, they are often ill-suited to lightweight, improvisational, or beginner-programmer workflows. Environments such as Unity and Unreal Engine typically impose rigid architectural patterns, demand powerful workstations, and require extensive setup and large installer downloads. Their sprawling IDEs and workspace complexity stand in contrast to the immediacy and simplicity prioritised in environments like Processing [48, 214]. Even comparatively lightweight engines like Godot (Figure 2.5) present steeper learning curves and higher system demands than Processing.

Such factors can hinder accessibility, especially in educational contexts focusing is on foundational programming concepts and low-barrier entry points. Consequently, this study treats these game engines as beyond the scope of text-based creative coding environments suitable to inform the design and development of Thonny-py5mode.

Exclusions: Limited Web Playgrounds

Table 2.3 offers another companion table to Table 2.1, listing web-based 'playgrounds' with limited capabilities, extracted from *Awesome Creative Coding*, that did not meet the criterion of providing "a unified workspace that supports continued project work (across several coding sessions)," among other criteria. As with the game engine entries, *Awesome Creative Coding* includes no dedicated sub-list or category for web playgrounds; instead, Table 2.3 identifies relevant entries across multiple groupings.

Broadly, the playgrounds listed in Table 2.3 offer editable code samples to demonstrate a specific library's features. For instance, http://paperjs.org hosts the official documentation for Paper.js, branded as the "Swiss Army Knife of Vector Graphics Scripting." One can browse the "Examples"

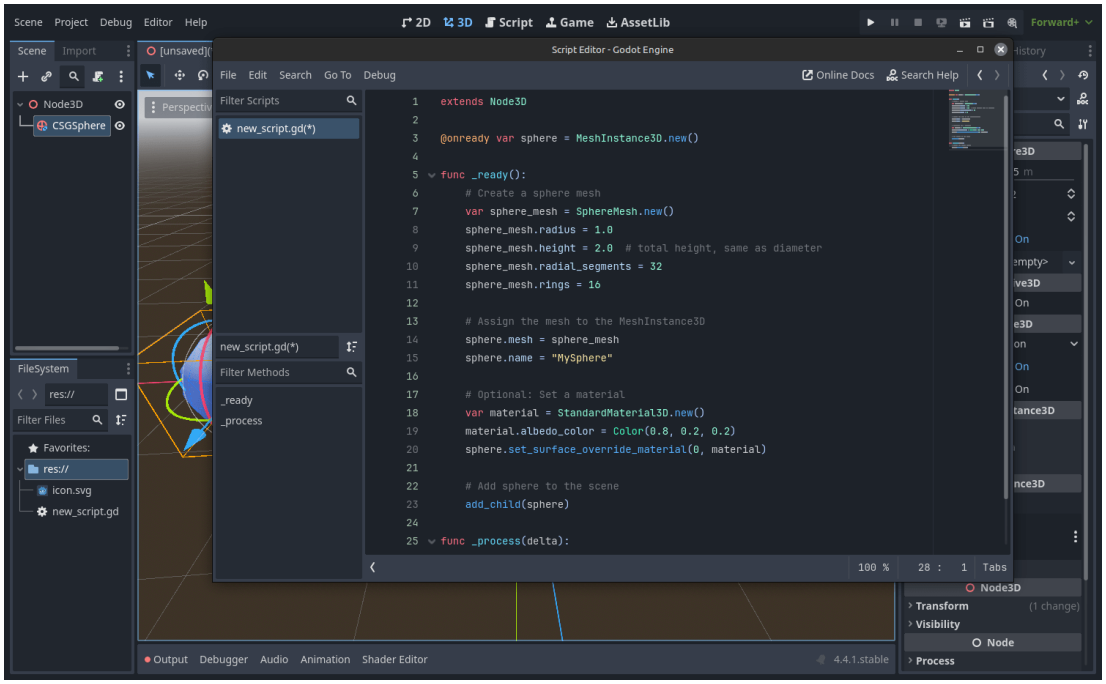


Figure 2.5: The Godot editor (version 4.2) with the Script Editor active. The interface features a broad range of panels and tools, in contrast to the minimalistic Processing IDE. Screenshot by the author.

Environment	Language	License	Description
css-doodle	CSS, JavaScript	Open-source	Web component for generative art using CSS; includes embedded (CodePen-powered) editable demos
Fabric.js	JavaScript	Open-source	Canvas library and SVG-to-canvas parser; includes (broken?) editor for demo scripts
GraphicsJS	JavaScript	Open-source	Lightweight JavaScript library for SVG/VML graphics and animation; playground powered by AnyChart; rendering base for AnyChart libraries
Maker.js	JavaScript	Open-source	Parametric line drawing for SVG, CNC and laser cutting; website includes editor for demo scripts
Paper.js	JavaScript	Open-source	'Swiss army knife' of vector graphics scripting; website includes a "Sketch" area
Pixi.js	JavaScript	Open-source	Fast 2D rendering engine using WebGL with a Canvas fallback; website includes "Playground" area
Pts.js	JavaScript (developed in TypeScript)	Open-source	Library for creative coding and data visualisation; features an online editor for real-time exploration

Table 2.3: Limited web-based 'playground' entries listed on *Awesome Creative Coding* but excluded from the final table (Table 2.1) due to their constrained functionality

section of this website showcasing the library's capabilities using live code, or experiment in the "Sketch" section using a *limited* set of editing features. In contrast, fully-fledged web environments like p5.js offer more comprehensive workspaces, including features for saving and loading scripts, importing assets, and managing and storing projects over time.

Exclusions: Shader Tools

Another category excluded from Table 2.1 comprises *Awesome Creative Coding* shader entries, namely: Cyos, Fragment, GLSL Sandbox, GlslEditor, ISF, KodLife, Shader Park, ShaderTool, Shadertoy, Shdr Editor, Vertexshaderart, and all the shader tutorials (under the *Awesome Creative Coding* > *Learning Resources* > *Interactive* sub-list), some of which feature limited web playgrounds.

Shaders are specialised programs that run on the GPU (Graphics Processing Unit) and are designed for massively parallel processing, typically to compute the colour, position, or appearance of pixels and vertices in real-time graphics. Unlike general-purpose programming languages—such as Python or JavaScript—which run on the CPU and execute instructions sequentially, shaders operate independently for each pixel or vertex. This makes them ideal for tasks such as rendering, lighting, and visual effects. Shader code is typically written in C-like languages such as GLSL or HLSL, which prioritise performance but operate under strict constraints, with no file I/O, minimal global state, and an emphasis on vectorised mathematics [77]. These limitations can pose significant challenges for beginners attempting to grasp foundational programming concepts, such as control flow and variable scope. Moreover, shaders demand considerable mathematical fluency, particularly in areas such as trigonometry and vector algebra, which present additional barriers for novices, especially those from creative or non-technical backgrounds [92, 223].

By contrast, environments such as p5.js and py5 utilise general-purpose programming languages that follow a sequential execution model, providing clearer scaffolding for debugging, feedback, and structured learning; this makes them more accessible for introductory programming education. Notably, both Processing and p5.js include built-in support for shader experimentation, allowing more advanced learners to explore GPU-based techniques within a familiar and supportive creative coding environment [156, 172]

Insights

The environments listed in Table 2.1 reflect a cross-section of tools that exemplify current text-based creative coding practices, while emphasising graphical output. Despite differences in platform, language, and licensing models, these environments share key characteristics: they offer integrated workspaces for coding and rendering visuals, support iterative or even live experimentation, and are shaped by active user communities.

A notable trend in Table 2.1 is the predominance of web-based environments, such as the p5.js Editor, P5LIVE, Hydra, and Turtletoy. These platforms are accessible via a browser, require no installation, and often support live coding or near real-time visual feedback—all features that align well with educational and improvisational use cases. Specifically, their ease of access and immediacy are well-suited to workshops, classrooms, or casual experimentation [209, 210]. Additionally, they tend to adopt open-source licenses, reinforcing their role within the broader collaborative and publicly-engaged creative coding ethos.

Table 2.1 also includes Python-based environments such as py5 (in Jupyter Notebooks), DrawBot, and Shoebot. These reflect a dominant current in creative coding: integrating expressive visual output with general-purpose programming languages widely used in education and research [33]. py5, for instance, brings Processing functionality to Python and benefits from Python's extensive ecosystem of scientific, markup (see SVG) manipulation, web scraping, and data analysis libraries. Similarly, DrawBot offers an education-focused, macOS-native environment for vector-based 2D graphics, while Shoebot revives the spirit of NodeBox in a cross-platform context. These environments can all support pedagogical clarity (through graphical output) in academic contexts where Python is favoured.

Some entries highlight minimalist or domain-specific approaches to creative coding. Shelly, with its bespoke Turtle-like language, and tixy.land, with its grid-based expression syntax aimed at "code golf"[3], strip down the coding environment to focus on constrained yet expressive interaction. Learning settings may employ such environments to encourage creativity through constraint-driven practices [20, 233]. For instance, #tweetcarts (Figure 2.6) exemplify this principle. The inclusion of these environments reflects a broader interest in solutions that foster creativity through simplicity and limitation rather than feature-richness [141].

[3] Code golf is a type of (often competitive) recreational programming where the objective is to solve a problem in the fewest number of characters or bytes of source code possible

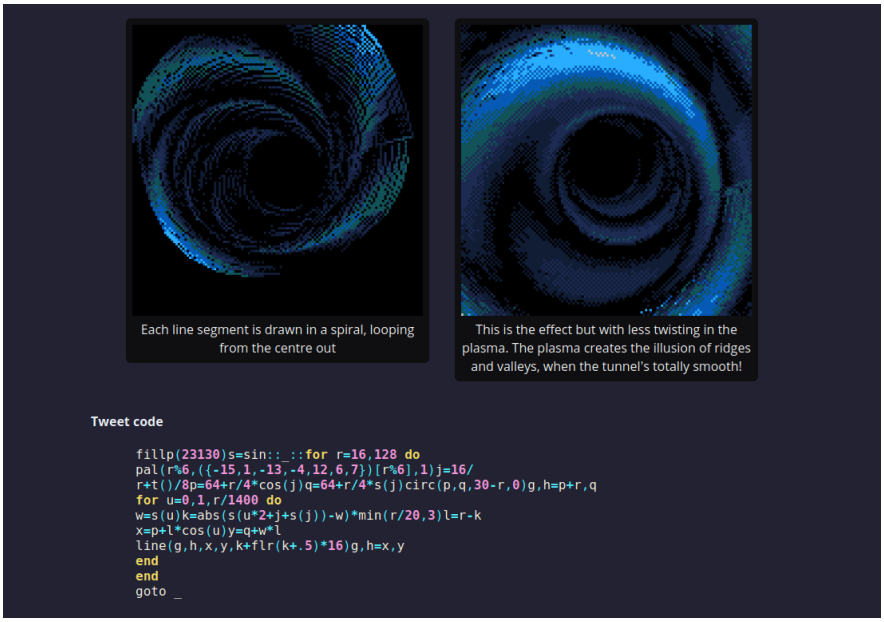


Figure 2.6: A #tweetcart is a tiny code-generated image, animation, or game—typically written in a fantasy console like PICO-8—and written to fit within the character limit of a tweet (originally 140, now 280 characters). Screenshot from Michał Rostocki's website. Source: [193].

In summary, Table 2.1 highlights a diverse range of lightweight, purpose-built environments for creative coding. These are distinct from monolithic software (like game engines) or general-purpose IDEs, and span structured editors like Processing to ephemeral live-coding tools like Hydra, reflecting the field's varied interaction models. JavaScript's dominance, particularly in browser-based tools, underscores its central role in generative and interactive media. However, Python also features prominently, and the Presentation Outputs chapter explores Python's integration into browser environments.

This investigation provided a curated, academically grounded foundation for identifying the types of environments that align with the objectives of this research. In this instance, the development focus is a Python-based, beginner-friendly creative coding platform: Thonny-py5mode. Including long-established environments (like Processing) and newer experimental ones (like tixy.land) ensures that the survey captures the field's historical depth and ongoing innovation. Table 2.1 foregrounds tools that balance simplicity, accessibility, and visual output, helping situate Thonny-py5mode within a lineage of environments designed for technical execution, artistic inquiry, and pedagogical engagement.

2.2 Chapter Summary

This chapter has examined key literature on creative coding software, identifying gaps and opportunities for enhancing Python programming education through creative computing. It established that effective creative coding environments strike a balance between simplicity and expressive capability, supporting both foundational programming concepts and encouraging creative exploration. This underscores opportunities for Thonny-py5mode to address within this domain.

An analysis of prominent text-based creative coding environments revealed an absence of Python-3-based solutions that combine pedagogical clarity with the immediacy and functionalities of tools such as Processing. The *Towards a Python 3 Processing IDE for Teaching Creative Programming* article explores this area in greater depth.

These insights inform the research contributions presented in subsequent chapters. Notably, this chapter alone does not encompass the PhD's entire literature review; instead, the thesis interweaves substantial scholarship components throughout the folio output chapters.

3 Software

This chapter focuses on the Thonny-py5mode plugin, developed as a central component of this research to enhance Python programming education through creative computing environments. It outlines the plugin's key features, development process, and supporting documentation, including funding and formal recognition. The MTAP article, *Towards a Python 3 Processing IDE for Teaching Creative Programming*, complements this chapter by detailing Thonny-py5mode's technical implementation and design rationale, while the *Evaluating Task–Technology Fit in Thonny-py5mode* study examines its impact on students' learning experience. Additionally, several outputs in the Presentation Outputs and Creative Works chapters examine Thonny-py5mode's broader contributions to the research aims, tracing its development journey and demonstrating practical applications across teaching, learning, and creative practice.

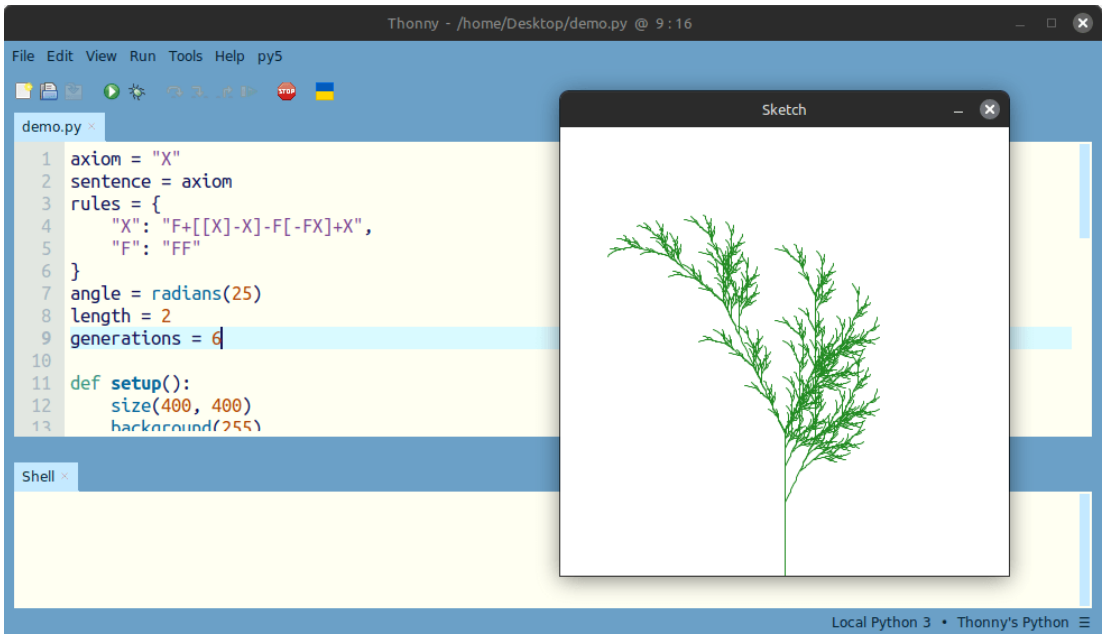


Figure 3.1: Screenshot of Thonny-py5mode in use, rendering an L-system fractal. Screenshot by the author.

Thonny is a beginner-friendly Python IDE developed at the University of Tartu to support programming education. It offers an intuitive interface with a clean editor layout, prioritising simplicity for newcomers. Some key features include a built-in debugger, integrated shell, and streamlined installation process owing to its bundled Python interpreter. Thonny is free, open-source software available for Linux, macOS, and Windows [12].

Thonny-py5mode builds on Thonny's plugin architecture by integrating the py5 library, which brings Processing's programming capabilities into Python [202]. The plugin name follows Thonny's prescribed convention, combining the required "thonny-" prefix with a reference to "py5". Thonny-py5mode serves as a contemporary successor to the Processing IDE's original Python Mode, but offers a Python 3-compatible workflow that supports the complete CPython ecosystem.

The following code demonstrates a basic Thonny-py5mode script (a py5 "sketch"), with Figure 3.2 showing the output (annotated to indicate how the arguments influence the appearance).

```
# setup
size(500, 500)           # canvas size
background('#FFFFFF')    # white background colour
cx = width / 2           # canvas horizontal centre
cy = height / 2          # canvas vertical centre
stroke_weight(5)         # set outline to 5 pixels wide

# draw rectangle
stroke('#FF0000')        # set outline to red
fill('#00FF00')          # set fill to green
rect(100, 50, 120, 340)  # draw rectangle

# draw circle
no_fill()                # set fill to none
stroke('#0000FF')        # set outline to blue
circle(cx, cy, 200)      # draw circle
```

Aligned with the goals of both Processing and Thonny, Thonny-py5mode facilitates easy entry into creative coding through a simplified setup process. It enables students—particularly those from creative disciplines with limited programming experience—to engage with code through the creation

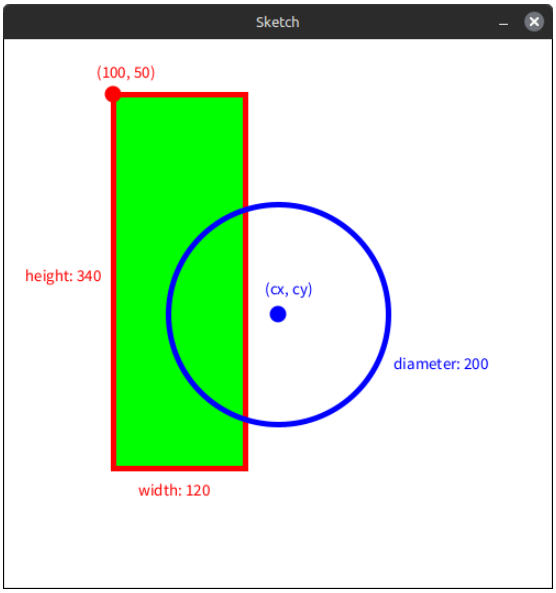


Figure 3.2: Annotated py5 sketch. By the author.

of visual, interactive, and multimedia outputs. The plugin installs a Java Runtime Environment (JRE) and adapts Thonny for py5, delivering a supportive user experience through features including syntax highlighting and autocompletion for py5 code, integrated buttons and shortcuts for running sketches, a colour mixer tool, direct links to reference materials, and a Processing-inspired editor theme.

3.1 Development Process

Development of the Thonny-py5mode plugin began in August 2021 with the creation of a GitHub repository, following several months of preliminary experimentation. This early work, informed by research into Processing-like Python environments, alternative code editors, and the then-recent stable release of py5, suggested that combining py5 with the Thonny editor could provide a suitable successor to Processing's discontinued Python Mode (Processing.py).

Thonny-py5mode's design objectives and user experience intentionally mirror the Processing IDE, which has demonstrated effectiveness in education through its approachable, pedagogically informed workspace [84, 167, 187, 218]. Instead of introducing a new interface, Thonny-py5mode adapts this paradigm to the Thonny environment, combining novice-friendly support and a familiar setting for any Processing users transitioning to Python.

The Thonny-py5mode project relies on GitHub for version control, issue tracking, and community discussion. It distributes releases via PyPI, from which the Thonny package manager retrieves plugins.

Installation instructions are available on both the GitHub and PyPI project pages, and also referenced in the official py5 documentation. Because py5 functions as an upstream dependency, Thonny-py5mode initially specified strict version requirements, but later adopted an 'unpinned' approach, thereby allowing py5 updates to proceed independently of the plugin version.

As of writing, Thonny-py5mode development remains active, with nearly 300 commits on GitHub, four major releases, and contributions from five developers. The initial design, implementation, and all releases up to January 2022 were completed solely by Tristan Bunn (thesis author). After a development hiatus between July 2022 and March 2023, progress resumed with increasing involvement from members of the py5 and Thonny communities.

Bunn first publicly introduced Thonny-py5mode in a presentation at CC Fest in August 2021 (see *Thonny + py5: A Python 3 Environment for Processing*). Since then, Thonny-py5mode has continued to evolve in close coordination with the py5 project, incorporating user feedback and supporting new py5 features. In July 2025, Bunn (GitHub username: tabreturn) formally transferred ownership of the Thonny-py5mode repository to the py5 organisation (from https://github.com/tabreturn/thonny-py5mode to https://github.com/py5coding/thonny-py5mode), along with the associated PyPI project: https://pypi.org/project/thonny-py5mode. Moving forward, Thonny-py5mode is officially maintained under py5 and recommended as "the best [py5] setup for beginners."[1]

3.1.1 Release History

The plugin's development progressed iteratively between August 2021 and June 2024, spanning nine releases during that time, with each aimed at enhancing functionality, stability, and user experience. Initial versions established distribution readiness (e.g., PyPI publication, Windows compatibility), while subsequent updates added integrated JDK support, dynamic sketch paths, and theming options. From 2022 onward, development focused on compatibility with Thonny 4, packaging refinements, and user-experience features, including autocomplete and colour-picking tools. To streamline the setup process, releases during this period introduced features such as progress indicators and enhanced setup logs. Later releases prioritised documentation quality, refreshed visuals, and alignment with evolving py5 requirements, culminating in a stable and polished tool by mid-2024.

[1] https://py5coding.org/content/install.html#install-py5

Community feedback shaped this trajectory, helping to optimise the plugin for both educators and learners and enabling smoother integration of py5 within Thonny across Linux, macOS, and Windows platforms.

For readers interested in granular details, such as incremental fixes or ongoing enhancements, the complete commit history on GitHub provides a record of steady progress, marked by phases of concentrated activity and periods of slower development (notably during 2023). Collectively, the releases demonstrate a responsive and sustained commitment to providing a reliable and accessible environment for creative coding in Python.

3.2 Features

This section documents the principal features of Thonny-py5mode. One may divide these into two categories: (1) bespoke functionality specifically developed for the plugin, described here; and (2) existing Thonny features that complement the execution of py5 sketches within Thonny-py5mode, discussed in the *Towards a Python 3 Processing IDE for Teaching Creative Programming* article and the *Thonny + py5: A Python 3 Environment for Processing* presentation output.

Following installation and activation, the plugin introduces a dedicated *py5* menu (Figure 3.3), placed within the Thonny menu bar.

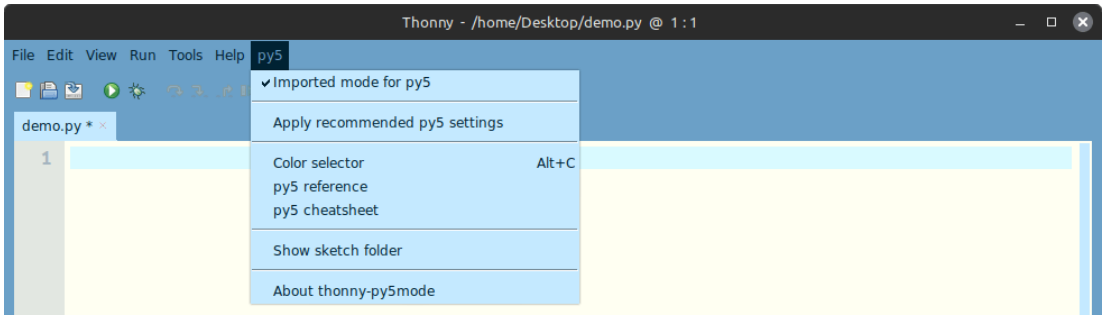


Figure 3.3: Thonny-py5mode activited in Thonny, showing the py5 menu. Screenshot by the author.

From the *py5* menu, users may toggle between *Imported mode*—which removes the need for explicit `import` statements or `py5` prefixes—and the module/class mode[2]. In Imported mode, they can also write Static mode sketches, omitting `setup()` and `draw()` functions when animation or interactivity is unnecessary. Other menu options include *Apply recommended py5 settings*, which

[2] The five py5 modes: https://py5coding.org/content/py5_modes.html

applies a Processing-inspired blue-ish colour scheme to the IDE, among other tweaks. The *Color selector* item opens a mixer window (Figure 3.4) that can be positioned beside the editor for repeated use while writing code. The menu also provides direct links to the *py5 reference* (official documentation) and a *py5 cheatsheet* (a two-page PDF, Figure 3.6, intended for download or printing). Selecting *Show sketch folder* opens the current working directory in the system's default file manager, while *About Thonny-py5mode* displays the plugin metadata (including version number, license, and credits).

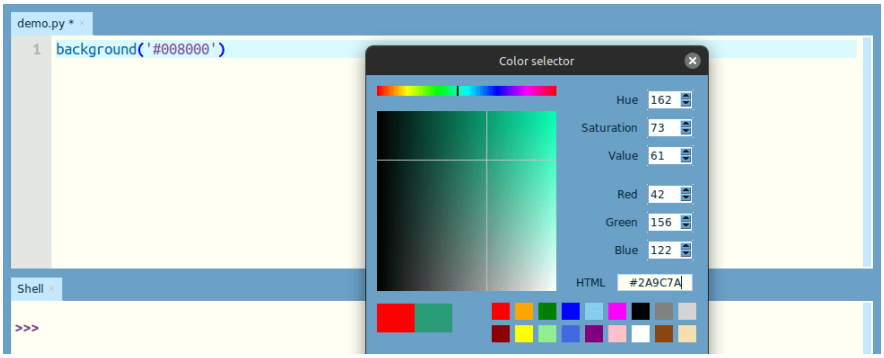


Figure 3.4: Thonny-py5mode colour mixer. Screenshot by the author.

When Thonny-py5mode is enabled, users may also activate autocompletion for py5 commands (Figure 3.5). By default, the plugin applies highlighting to py5 commands and keywords, thereby enhancing code readability and reducing the likelihood of syntactic error.

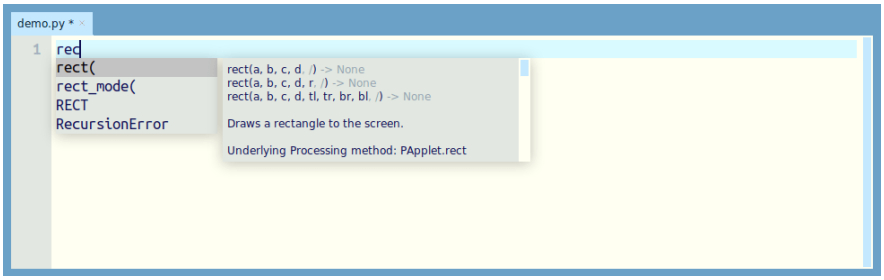


Figure 3.5: Thonny-py5mode autocomplete. Screenshot by the author.

Beyond these core functions, Thonny-py5mode introduces several usability refinements designed to enhance the programming experience. The sketch window, for instance, now remembers its previous position between executions rather than defaulting to the display's top-left corner on every run. Version 0.4.6a0 improved the installer process, adding progress indicators, user-facing notifications, and detailed event logging. Additionally, full cross-platform compatibility ensures consistent behaviour across Linux, macOS, and Windows.

3.3 Documentation

Thonny-py5mode provides setup/installation documentation hosted on both its GitHub and PyPI project pages. As described above, the py5 menu links to two other documentation efforts: (1) a py5 cheatsheet, and (2) the official py5 library documentation.

3.3.1 py5 Cheatsheet

A 'cheatsheet' is a compact reference that distils key syntactic and functional elements of a programming language or library into a concise format, thereby documenting essential concepts, commands, and techniques to support rapid recall [220].

The py5 cheatsheet (Figure 3.6) provides a quick reference for py5, the Python-based implementation of Processing integrated with Thonny through Thonny-py5mode. It presents core syntactic elements and functional constructs for developing sketches, including program structure (static versus animated scaffolds), colour specification with fills and strokes, two-dimensional primitives, methods for creating complex shapes, and Python commenting conventions. Additional sections address typography, mathematical operators, randomisation, system constants, variables, and standard Python control flow. Several topics include a concise, multiline code sample that supports recall rather than extended explanation. The cheatsheet functions as both a pedagogical and practical aid for creative coding. It offers particular value for beginners and those transitioning from other Processing environments (e.g., Java- or JavaScript-based) to Thonny-py5mode.

Reflecting the open-source ethos of both Thonny-py5mode and py5 (and Processing), the author created the cheatsheet with Scribus[3] and Inkscape, using open-source fonts (DejaVu Sans, Enriqueta, and Source Code Pro). A GitHub repository hosts the source files at https://github.com/tabreturn/processing.py-cheat-sheet.

3.3.2 py5 Official Documentation

As noted earlier, the Thonny-py5mode and py5 projects have been closely aligned since their inception, with py5 having recently assumed stewardship of Thonny-py5mode. In 2022, the Processing Foundation accepted a combined Thonny-py5mode–py5 documentation proposal as part of their Google Summer of Code (GSoC) involvement.

[3] Scribus is free, open-source DTP software for designing documents and print-ready PDFs: https://scribus.net

v0.1

py5 cheatsheet

Complete reference at
https://py5.ixora.io/reference

Program Structure

Structuring a static sketch:

```
size(400, 200)
run this code once
```

Structuring an animated sketch:

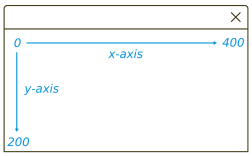
```
def setup():
    size(400, 200)
    run this code once at start
def draw():
    run this code every frame
```

Function anatomy:

```
size(400, 200)
```

function name · width · height · arguments

The default coordinate space:



Comments

```
# this is a single line comment
'''
this is a
multiline comment
'''
```

Fills & Strokes

```
background(color)  # set bg color
fill(color)  # set fill color
no_fill()  # disable the fill
stroke(color)  # set stroke color
stroke_weight()  # stroke width in px
no_stroke()  # disable the stroke
```

A red fill using three different color values:

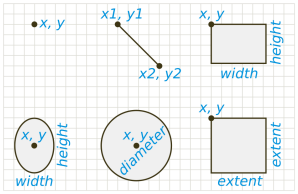
```
fill('#FF0000')  # hexadecimal
fill(255, 0, 0)  # red, green, blue
# HSB (Hue Saturation brightness)
color_mode(HSB, 360, 100, 100)
fill(0, 100, 100)
```

Print

```
print(value)  # prints to console
```

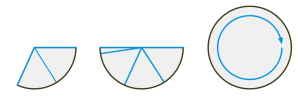
2D Primitives

```
point(x, y)
line(x1, y1, x2, y2)
rect(x, y, width, height)
ellipse(x, y, width, height)
circle(x, y, diameter)
square(x, y, extent)
```




```
arc(x, y, width, height, start, end)
```

For measuring arc angles, use radians:



v0.1

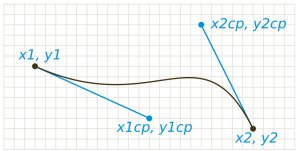
Shapes

Draw complex shapes using vertices nested within `beginShape()` and `endShape()` functions:

```
begin_shape()
vertex(x1, y1)
vertex(x2, y2)
# add more vertices here
end_shape(CLOSE)
```

For curved shapes, use a Bézier-vertex function:

```
begin_shape()
vertex(x1, y1)  # vertex 1
bezier_vertex(
  x1cp, y1cp,   # control point 1
  x2cp, y2cp,   # control point 2
  x2, y2)       # vertex 2
# add more vertices or beziers here
end_shape()
```



Typography

```
text_font(create_font(font, size))
text_size(rendered_size)
text('text', x, y)
```

Math

Use arithmetic operators to add, subtract, etc:

```
+  -  *  /  %
```

Use brackets to override operator precedence:

```
(1 + 2) * 3  # equals 9, not 7
```

Random

For unexpected values, use the random function:

```
random(10)  # 0.0 up to not inc. 10
random(5, 10)  # 5.0 to not inc. 10
random_int(5, 10)  # integer value
random_choice([5, 10])  # choose one
```

To set the seed value for the pseudo-random generator, use:

```
random_seed(integer)
```

Constants & System Variables

For different values of pi, use these constants:

```
PI  HALF_PI  QUARTER_PI  TAU
```

Some useful Processing system variables:

```
width, height  # sketch width, height
mouse_x, mouse_y  # mouse x, y coords
frame_count  # current frame number
```

Control Flow

Python if-then logic:

```
if condition_a:
    if condition_a is true, do this
elif condition_b:
    if condition_b is true, do this
else:
    if condition_a/b false, do this
```

Relational operators:

```
==  !=  >  <  >=  <=
```

Logical operators:

```
and  or  not
```

Iteration using a for-loop:

```
# displays 0, 1, 2 in console
for i in range(3):
    print(i)
```

Looping through a list:

```
numbers = [3, 4, 5] # a list

# displays 3, 4, 5 in console
for i in numbers:
    print(i)
```

Cheat sheet source files available at
github.com/tabreturn/processing.py-cheat-sheet

Figure 3.6: Downloadable/printable py5 PDF cheatsheet. Designed by the author.

This GSoC programme, funded by Google, supports open-source by pairing contributors with real projects and volunteer mentors. Participant stipends vary by location and project size, and offer flexible commitment levels (e.g., 175-hour "medium" or 350-hour "large" projects, depending on the year). Contributors work remotely, typically over the Northern Hemisphere summer. Upon successful final evaluation, they receive their stipend and formal recognition for their work.

Under this initiative, the leaders of Thonny-py5mode and py5—the thesis author, Tristan Bunn, and py5 project lead, Jim Schmitz—secured funding to mentor Zelle Marcovicci in developing a comprehensive set of tutorial materials introducing beginners to Python programming with Thonny-py5mode. Adapted from the book *Learn Python Visually: Creative Coding with Processing.py* and expanded with new sections showcasing CPython capabilities, these materials now form the *Tutorials* section of the official py5 documentation (Figure 3.7).

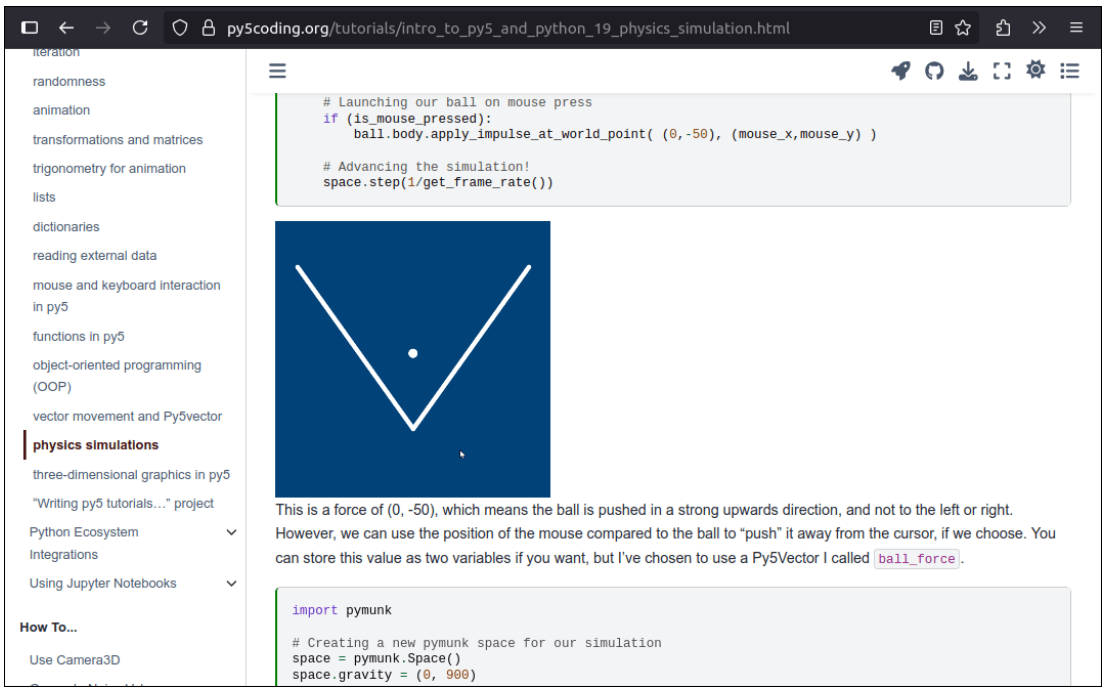


Figure 3.7: py5 official documentation: *Tutorials* section entry on PyMunk integration for 2D physics. Screenshot by the author. Source: [202].

The Processing Foundation published the official project announcement (July 11, 2022) on their Medium account: https://medium.com/processing-foundation/announcing-google-summer-of-code-2022-projects-and-a-few-more-77043ab4d0b4. The GSoC 2022 programme wrap-up post (October 18, 2022) is also available there: https://medium.com/processing-foundation/google-summer-of-code-2022-wrap-up-post-cb64caa840f0.

As the wrap-up post concludes—

> This project was not just to adapt, write, and rewrite some documentation, but to utilize it in teaching and to see what organically developed in the work of [Zelle's] students. In this way py5 has been an enormous success, and the 100-level students using it to learn visual coding have responded incredibly well. The most wonderful part about contributing to open-source projects like this is seeing your contributions make some kind of impact, and the response even from within the small py5 community has been very heartening. Thanks to frequent communication and support from both of her mentors, everything has gone smoothly and Zelle is hoping to continue contributing tutorials and snippets of interesting code to the py5 documentation site even after GSoC 2022 has wrapped up.

3.4 Impact & Recognition

As of writing (August 2025), there are approximately 34,000 recorded downloads and installations of Thonny-py5mode, as recorded in PyPI analytics data[4] Both Massey University (New Zealand) and Torrens University (Australia) have integrated the software into introductory Python courses. In addition, several educators participating in online forums have reported using the environment in their classes. However, no empirical study has yet established the breadth of this adoption or the extent to which the plugin is embedded in formal curricula.

Notably, Thonny-py5mode underpins the online course *Designing with Python: Programming for a Visual Context* (Figure 3.8), offered through Domestika: a platform dedicated to creative professionals sharing their expertise through professionally produced courses.

The Domestika course is designed and delivered by Alexandre Villares, a prominent figure in the Python–Processing community. It introduces programming as a creative medium for artists and designers. Using Thonny-py5mode, Villares teaches foundational Python concepts, including loops, conditionals, functions, and data structures, while demonstrating their application in generating visual output and task automation. Learners are encouraged to experiment with geometric patterns, abstraction, and iterative "baby steps" development. He further situates this practice within the

[4] https://clickpy.clickhouse.com/dashboard/thonny-py5mode

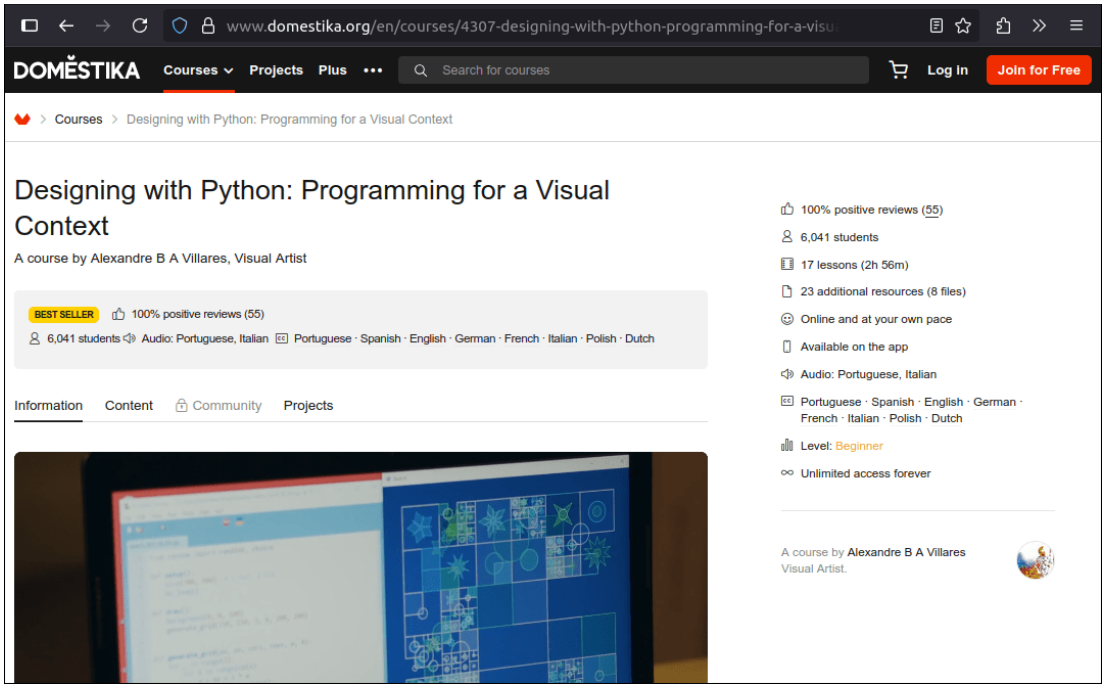


Figure 3.8: *Designing with Python: Programming for a Visual Context*, hosted on Domestika. Screenshot by the author. Source: [240].

broader collaborative creative coding community, framing programming not only as a technical competency but also as an expressive medium for artistic exploration.

At the time of writing, over 6,000 students have enrolled in the course, which has received consistently positive evaluations from 55 reviewers [240]. It is offered with audio in Portuguese, English, French, and Italian, and includes subtitles in additional languages, thereby enhancing its accessibility to an international audience.

Villares maintains an extensive archive of daily sketches employing Thonny-py5mode, which collectively demonstrate its capabilities for iterative, exploratory, and visually oriented coding practices. This archive functions as a living resource, illustrating the potential of Python-based creative coding tools, tracing the shift from Processing.py to Thonny-py5mode/py5, and documenting the evolving affordances of these environments. His practice underscores Python's dual role: both as a technical framework for developing and understanding programming concepts, and as a medium for creative expression. The complete archive, including source code for each sketch, is publicly accessible at https://abav.lugaralgum.com/sketch-a-day.

3.5 Limitations and Future Work

Three main factors currently constrain Thonny-py5mode's development. From these, we identify three concrete directions for further research and development—

First, installation requires a two-step process: users must install Thonny and *then* add Thonny-py5mode via Thonny's plugin manager. A standalone distribution pre-packaging both components, and capable of running fully self-contained, is technically feasible; this might run directly from a USB flash drive without requiring internet access (to complete the plugin manager step). This goal requires further packaging work to ensure cross-platform compatibility and reliable performance across diverse personal and institutional computing environments.

Second, the broader dissemination of py5 coding techniques and sample scripts remains underdeveloped. Processing's web gallery of short, prototypical programs that explore coding basics offers a good model for inspiration. Web-export functionality via pyp5js will enable curated, browsable collections of py5 example sketches. As an early proof of concept, the author's 2021 project, pyde.org (Figure 3.9), demonstrates this potential.

pyde.org (http://pyde.org) is built with Jinja2 for templating, a modified Pygments lexer for syntax highlighting, and pyp5js for transpiling Processing.py to p5.js. Consequently, it is limited to features common to both Processing.py and p5.js—regardless, this overlap is sufficient to convey most foundational and many visually engaging concepts. The New Zealand Open Source Awards[5] selected the project as a finalist in 2021. Implementing a Thonny-py5mode version primarily requires adapting the transpiler for py5 code, a task unlikely to prove overly complex.

Third, although well-suited for introducing Python through visual learning contexts, Thonny-py5mode currently lacks support for dynamic, real-time (see REPL) script execution. Future work should explore live-coding capabilities to enhance classroom engagement and broaden creative applications, including contexts such as VJ performance.

In addition to these primary goals, continued attention to bug fixes, accessibility improvements, and code refinements remains essential. Furthermore, lower-priority but promising directions include automating distribution via GitHub to reduce manual release overhead.

[5] https://nzosa.org.nz/previous-nzosa-winners/nzosa-awards-2021

Figure 3.9: *pyde.org*: short, prototypical programs exploring the basics of programming with Processing.py. Designed and developed by the author.

3.6 Chapter Summary

This chapter outlined the development, features, and impact of Thonny-py5mode, positioning it as both a technical contribution and a platform for Python education through creative computing. It aspires to succeed Processing's discontinued Python Mode, and it has matured into an actively maintained project that extends Thonny and py5, with documentation that further underscores its commitment to accessibility and pedagogical clarity.

The Presentation Outputs chapter explores pathways into Pygame, Blender scripting, and pen-plotter art, highlighting opportunities to extend Thonny-py5mode beyond py5's scope by leveraging Python's ecosystem to integrate additional 2D, 3D, and interactive media workflows within a single beginner-friendly IDE.

4 Publications

This chapter presents the three published works that form the scholarly core of the PhD, arranged chronologically (starting with earliest) to reflect the progression of the research. It builds on earlier chapters, which established the thesis aims, objectives, and the development of the Thonny-py5mode software. Together, these publications mark key milestones in the research journey, spanning the design of Python-based curricula, the exploration and development of new tools, and the empirical evaluation of Thonny-py5mode through a user study. The list below situates each publication within the broader stages of the PhD:

1. **Publication 4.1 – Stage: Processing.py and curriculum foundations.** This phase focused on establishing a curriculum for teaching Python through visual learning contexts, culminating in the sole-authored book *Learn Python Visually: Creative Coding with Processing.py*. This book captures and formalises the educational approaches that initially framed the research.

2. **Publication 4.2 – Stage: Transition to new tools and the development of Thonny-py5mode.** The discontinuation of Processing.py created a gap for a new Python-based creative coding environment. This led to the development of Thonny-py5mode: a plugin intended to replace and extend Processing's Python Mode. The *Towards a Python 3 Processing IDE for Teaching Creative Programming* article positions Thonny-py5mode within the broader context of Python creative coding tools, highlighting its contribution to programming education.

3. **Publication 4.3 – Stage: Empirical testing and evaluation.** This component of the research involved an empirical evaluation of Thonny-py5mode in a 100-level university Python course. The *Evaluating Task–Technology Fit in Thonny-py5mode* article reports on this study, assesses the plugin's effectiveness through the lens of Task–Technology Fit, and discusses its broader implications for related educational contexts.

The Presentation Outputs chapter documents the talks, workshops, and demonstrations that underpinned the publications presented here. Figure 4.1 maps the three publications onto the PhD timeline—using their submission rather than publication dates—thereby illustrating the formative role of presentation outputs in their development.

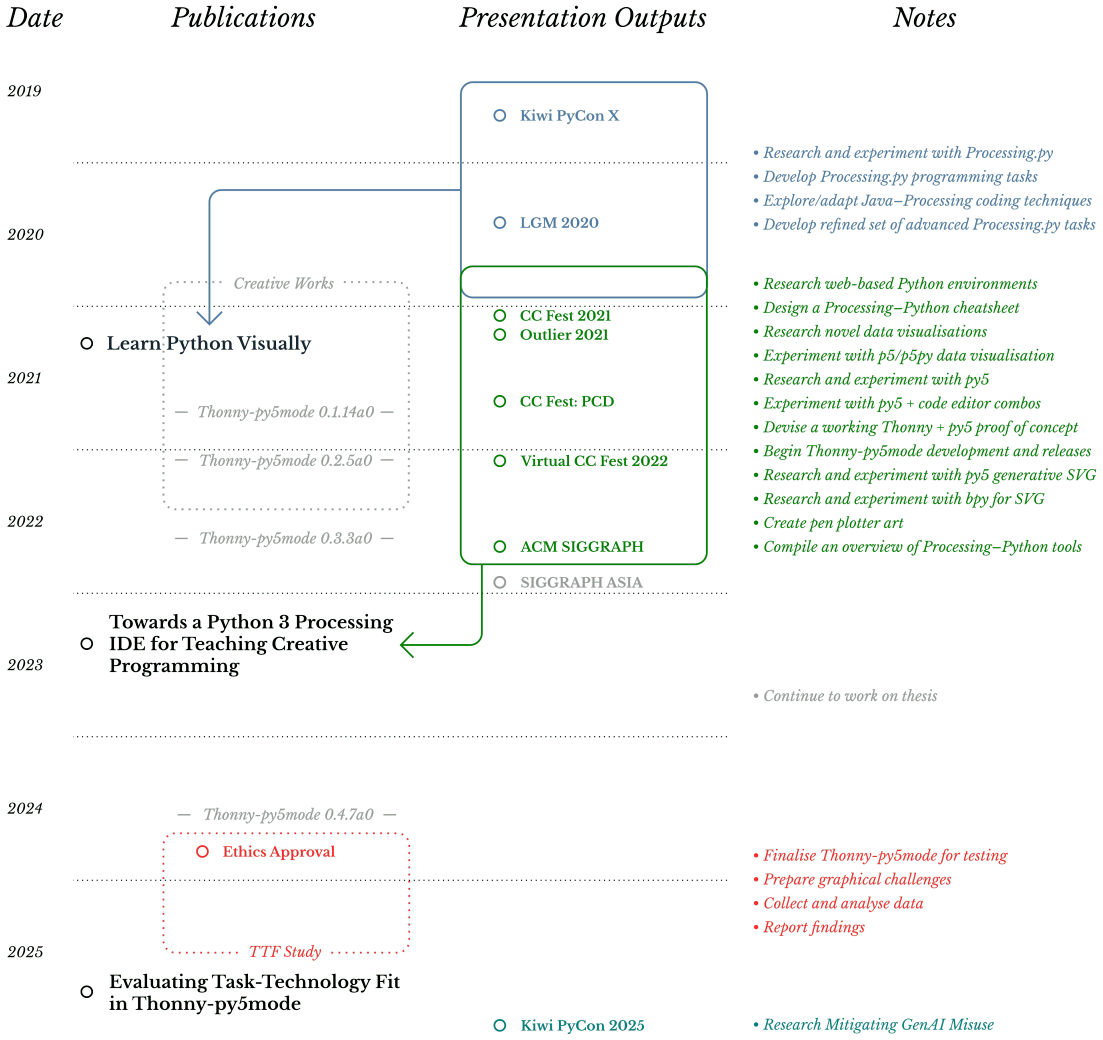


Figure 4.1: Timeline of publications, highlighting the integration of presentation outputs (talks, workshops, and demonstrations) into the research process. Diagram by the author.

The *Publications* column lists the three works discussed in this chapter, shown in **bold black type**. Within this column, several — *Thonny-py5mode...* entries indicate individual Thonny-py5mode releases. Dotted-outline cells mark the periods of producing the *Creative Works* (covered in the

Creative Works chapter) and conducting the empirical evaluation of Thonny-py5mode (*TTF Study*). The *Notes* column summarises the processes and steps that defined each stage.

Viewed within the context of this PhD by folio/publication, the sole-authored book and two peer reviewed Q1/Q2 journal articles balance practical impact with scholarly rigour. *Learn Python Visually: Creative Coding with Processing.py* serves as a resource for educators integrating Python-based creative coding into curricula, while the journal articles contribute to academic discussion surrounding Python tools for visual learning contexts. Collectively, these publications address four research objectives outlined in the "Introduction" chapter: Tool Development, Educational Materials, Empirical Evaluation, and Broader Dissemination.

As outlined in Figure 2.1 in the "Literature Review" chapter, the journal articles engage with much of the relevant literature, as do the Presentation Outputs and, to a lesser extent, the Creative Works. Given the folio/publication thesis format, some of the literature in the journal articles unavoidably overlaps with the Literature Review. Specifically—

- While the "Literature Review" chapter also covers prominent creative coding environments, the article *Towards a Python 3 Processing IDE for Teaching Creative Programming* extends this analysis to explore Python-3-specific creative coding environments in greater depth.
- The other article, *Evaluating Task–Technology Fit in Thonny-py5mode*, reiterates some of the general and introductory literature on creative coding environments in a more condensed form. However, it also provides an additional review of scholarship supporting the Task–Technology Fit framework, which is central to its study.

The next section (4.1) reviews the content and development of the book *Learn Python Visually: Creative Coding with Processing.py*, which cannot be reproduced in full here because of its length and copyright restrictions. Following that, the subsequent sections (4.2, 4.3) present the two journal articles in their preprint (submitted) form, which do not incorporate revisions made after peer review. Publishers permit the distribution and archiving of these versions, which exclude publisher formatting and instead follow the style and conventions of the thesis.

4.1 Learn Python Visually: Creative Coding with Processing.py

Publisher: No Starch Press

Publisher URL: https://nostarch.com

Author(s): Bunn, T.

Date published: 20 July 2021

ISBN-13: 978-1-7185-0096-9

Print length: 296 pages

Catalogue entry: https://nostarch.com/Learn-Python-Visually

Publication evidence: See Appendix C.2

4.1.1 Background & Context

Learn Python Visually (Figure 4.2) represented a significant component of this PhD research, concluding the initial phase of investigation into- and development of novel Python-based creative coding techniques. Aimed at beginner coders and readers transitioning from other programming backgrounds, the book employs Processing's Python Mode (Processing.py) as its pedagogical foundation, combining Python's accessibility with Processing's multimedia capabilities.

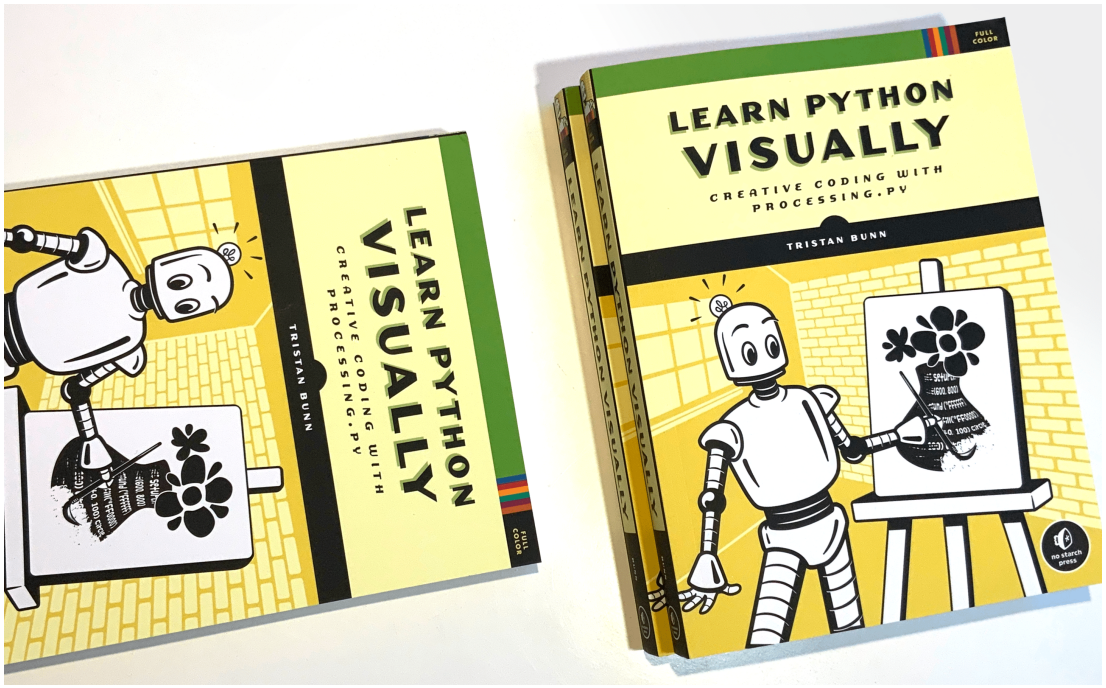


Figure 4.2: *Learn Python Visually: Creative Coding with Processing.py*. Photo by the author.

The book's strength and novelty lie in its visual and practical take on Python instruction. Each chapter builds progressively upon the last, moving from fundamental concepts toward more sophisticated creative coding techniques. This approach directly aligns with the overarching aim of the thesis: enhancing Python programming education through creative computing environments, and specifically the Educational Materials and Broader Dissemination objectives. Two presentations in particular, Processing.py—Creative Coding with Python and Processing Python Mode for Creative Coding and Teaching, provide further insight into the activities and research that informed the book's development.

No Starch Press is a respected independent publisher specialising in technical and educational titles. Established in 1994, it has built a strong reputation for publishing accessible yet rigorous works on programming, cybersecurity, and STEM education, including bestsellers such as *Python Crash Course*, *Python for Kids*, *How Linux Works*, and *Hacking: The Art of Exploitation*. Its titles—widely adopted by learners, educators, and professionals—have received industry recognition, including Independent Publisher Book Awards (from *Independent Publisher* magazine) and inclusion in the prestigious *Communication Arts Design Annual*. This status underscores the book's credibility as both a practical learning resource and a contribution to Python-based programming education [149].

Paddy Gaunt, who served as the book's technical reviewer, studied engineering at Cambridge University (UK). Since 2012, he has maintained pi3d: a Python library designed to deliver high-speed 3D graphics on Raspberry Pi microcomputers [32].

Learn Python Visually is available worldwide in both print and digital formats. It has received favourable reviews on Amazon.com[1] where it features editorial endorsements from Saber Khan (Education Community Director of the Processing Foundation), Dr. Ralf Biedert (Principal Engineer at Tobii AB), and Alfred Abusomwan (Techs Blog). As of June 2025, the book had achieved 2,622 direct sales (Appendix C.3). It was also included in two Humble Bundle collections: *Machine Learning by No Starch Press* (August 2021) and *Python by No Starch Press* (May 2023), that sold 17,669 and 10,881 bundles, respectively [95, 96].

In addition to direct and bundled sales, the book is distributed internationally through No Starch Press partners, including digital platforms such as O'Reilly Online Learning[2] and through academic

[1] https://www.amazon.com/Learn-Python-Visually-Tristan-Bunn/dp/1718500963

[2] https://www.oreilly.com/library/view/learn-python-visually/9781098128937

library collections, further extending its educational reach. Penguin Random House distributes No Starch Press printed titles in the USA and worldwide [150].

Learn Python Visually integrates visual learning throughout its chapters, requiring learners to program output that expresses computational methods through graphical outputs, animations, data visualisations, 2D simulations, and interactive interfaces. In this way, it places emphasis on immediate, observable results to elucidate abstract concepts and support intuitive understanding. The book is printed in full colour, incorporating visual explanations in addition to the results of different code samples. Figures 4.3 and 4.4 present a sample selection of 16 pages each.

The early chapters cover introductory concepts, such as installation procedures, basic syntax, arithmetic operations, and the principles of drawing with code. Once readers are familiar with the fundamentals, later chapters build on these foundations, shifting focus to concepts that require more programmatic reasoning—control flow, iteration, randomness, motion, transformations, data visualisation, and constructing a user interface. The following outline provides a brief overview of each chapter's contents:

Chapter 1: Hello, World!

Covers installation and setup for Processing Python Mode (Processing.py) and introduces the basics of drawing with code. Also: how computers manage colour, how to store and reuse values using variables, and how to perform basic arithmetic operations in Python.

Chapter 2: Drawing More Complicated Shapes

Building on drawing essentials, readers move on to organic (non-geometric) shapes—defining these with points (vertices) and curves, which can describe almost any 2D form in code.

Chapter 3: Introduction to Strings and Working with Text

Introduces Python's string features for manipulating text. To visualise these concepts, readers explore Processing functions for drawing text to the display window in different styles, colours, and fonts.

Chapter 4: Conditional Statements

Moving into control flow, demonstrating how to write programs that make decisions

Figure 4.3: Excerpt from the author's book *Learn Python Visually* (pp. 190–205), published by No Starch Press. This section introduces trigonometric functions and waveforms, and demonstrates the generation of Lissajous figures using Python. Source: [32].

Figure 4.4: Excerpt from the author's book *Learn Python Visually* (pp. 210–225), published by No Starch Press. This section introduces object-oriented principles through the generation of an amoeba simulation using Python. Source: [32].

and execute different actions in response to different situations. Again, readers explore these concepts through graphical output.

Chapter 5: Iteration and Randomness

Covers repeating operations in Python, either a specified number of times or until a condition is met. Readers combine these techniques with randomness to generate tiled patterns.

Chapter 6: Motion and Transformation

Introduces the `draw()` function for animated output. Also covers saving frames as images, transforming groups of visual elements, and employing real-time values for screensavers and clock displays.

Chapter 7: Working with Lists and Reading Data

Unlocks techniques for managing and manipulating collections of values, leading to practical data visualisation examples that read in external CSV data to generate charts dynamically.

Chapter 8: Dictionaries and JSON

Explores richer data structures and visualisation with JSON data, developing readers' skills to work with more complex and novel data visualisations.

Chapter 9: Functions and Periodic Motion

Shows how dividing programs into reusable functions improves modularity and readability, then combines these techniques with trigonometric operations to generate elliptical and wave-type motions.

Chapter 10: Object-Oriented Programming and PVector

Demonstrates object-oriented principles to structure programs that model real-world phenomena, guiding readers through programming a visually engaging amoeba simulation that leverages vector-based motion.

Chapter 11: Mouse and Keyboard Interaction

Concludes the book by adding interactivity to Processing sketches, focusing on mouse

and keyboard input to build a paint app. This requires event functions and techniques for controlling Processing's draw-loop behaviour.

An Afterword discusses where readers might head next, pointing them to the Processing community forum and useful materials, including tutorials on working with Images & Pixels, P3D for 3D work, and the broader Processing ecosystem of libraries (written for Java Mode but typically portable to Python Mode). It encourages translating Java examples, highlights Daniel Shiffman's *The Nature of Code* (with Python ports), and discusses Python's applications in games, the web, data, and AI. Additionally, there is some discussion on adjacent creative-coding environments, including p5.js, JRubyArt, openFrameworks, OPENRNDR, and hardware programming with Arduino.

Notably, readers' prior experience in related areas, including other programming languages, will likely influence their comprehension and the pace at which they progress through the book. However, the text encourages detours wherever inspiration strikes and includes challenges and practical tasks at key moments to reinforce conceptual understanding and promote active learning. For complete beginners, the content aims to reduce the anxiety often associated with textual coding by incorporating visual tasks wherever possible.

4.1.2 Conclusion

Within the framework of this thesis, *Learn Python Visually* consolidates extensive research into Python creative computing, specifically the Processing.py environment, translating experimental work into pedagogical strategies and learning materials. Publication by No Starch Press (est. 1994), with technical review by Paddy Gaunt (maintainer of pi3d), attests to rigour. Uptake indicators include 2,492 direct sales as of Dec 2024, inclusion in two Humble Bundle collections that together sold more than 25,000 digital copies, and worldwide distribution. Collectively, these figures evidence effective pedagogy and and broad accessibility.

4.2 Towards a Python 3 Processing IDE for Teaching Creative Programming

Journal: Multimedia Tools and Applications

Journal Ranking: Q1 (see Scimago Journal & Country Rank)

Author(s): Bunn, T., Anslow, C., and Lundqvist, K.

Statement of Authorship: See Appendix F

Date Received: 06 March 2023

Date Published: 10 October 2024

DOI: https://doi.org/10.1007/s11042-024-20345-1

Publication Evidence: See Appendix C.4

4.2.1 Abstract

Processing is a popular graphical library and IDE developed for electronic art and visual design communities, with a strong focus on teaching art, design, and creative technologies students computer programming fundamentals in a visual context. Processing provides a collection of special commands to draw, animate, and handle user input using Java. Users can enable Python Mode (also called Processing.py) for Processing in the IDE interface. This leverages Jython, a Java implementation of Python, to interface with Processing's Java core, providing a way to write Processing code using Python syntax.

This paper proposes that combining Processing and Python provides an ideal development environment for teaching creative programming fundamentals. Several new Processing-Python environments and tools have emerged, but no attempts to integrate one of the most promising, the *py5* library created by Jim Schmitz, into a Processing-like-IDE experience. py5 offers features not available with Jython, such as compatibility with Python 3 and support for CPython libraries. This paper presents a new coding environment, Thonny-py5mode, developed as a software plugin for the Thonny IDE, which brings a convenient, beginner-friendly setup like that of Processing's Python Mode to users working with py5.

CCS Concepts: Applied computing → Media arts; Interactive learning environments; Education. *Additional Key Words and Phrases: Processing, py5, Python Mode, Processing.py, Python*

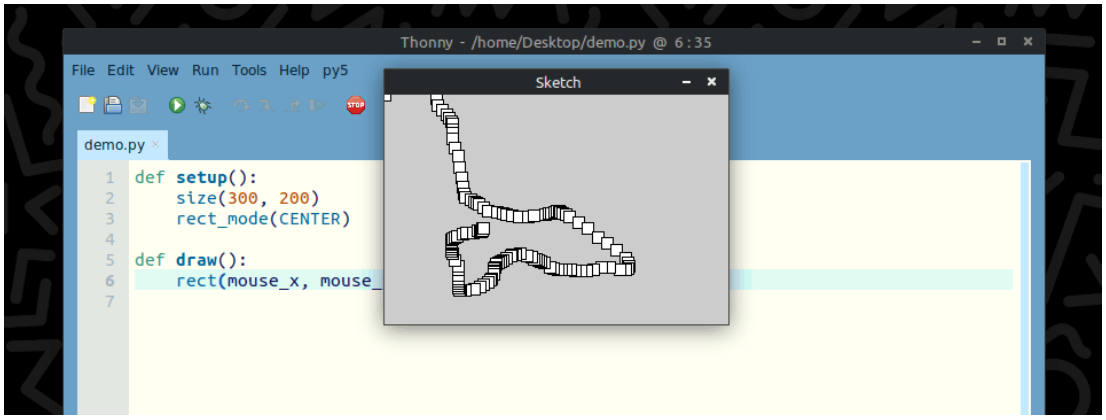


Figure 4.5: Running a py5 sketch using Thonny-py5mode. Screenshot by the author.

4.2.2 Introduction

Processing is an open-source integrated development environment (IDE) popular for teaching non-programmers the fundamentals of computer programming in a visual context. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. Tens of thousands of students, artists, designers, researchers, and hobbyists use Processing and its related projects[3] for learning, prototyping, and a diverse range of creative computing projects [151].

Alternative Processing programming libraries include p5.js (for JavaScript), JRubyArt (for Ruby), and even an effort to create a version for the R language [245]. Python Mode for Processing—which some people refer to as Processing.py—combines the Python programming language and Processing [161], providing users with a Python alternative to Processing's Java syntax. Figure 4.6 contrasts Processing's Java and Python modes, demonstrating how programmers interface with the same API, using the same IDE, with different syntax. In this instance, the resulting output is an animated square that moves from the left to the right of the display window (leaving a trail of visible frames in its wake).

Java and Python for Introductory Programming Lessons

Interestingly, reports from the IEEE (Institute of Electrical and Electronics Engineers) indicate that both Java and Python have been among the top-ten-list of programming languages in recent years [40]. While Processing-Java's success suggests it is a highly effective way to learn to program, the Java syntax

[3] https://processing.org

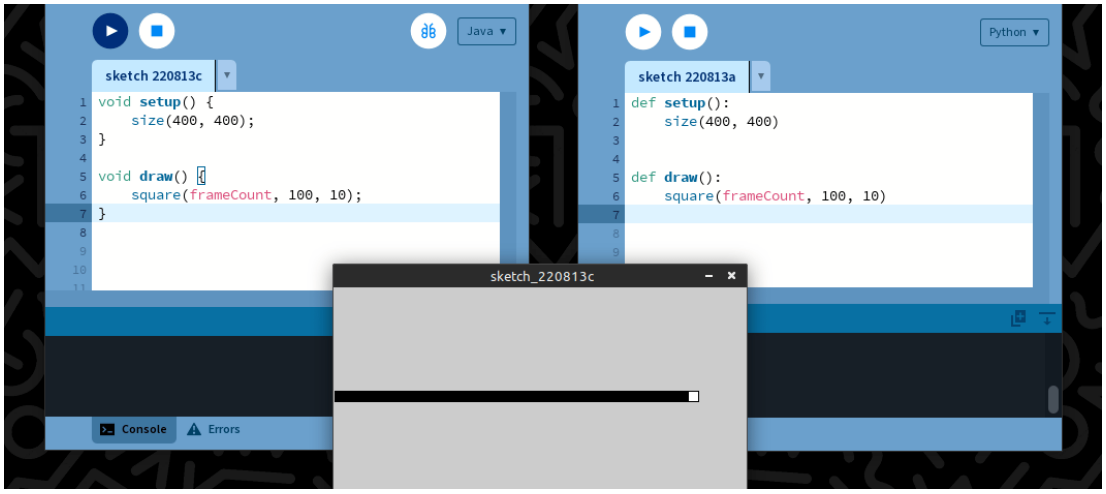


Figure 4.6: Processing in its original Java mode (left) and Python mode (right). Screenshots by the author.

may be less ideal than Python syntax—as this paper argues, particularly for visual design and art students.

There has been an ongoing debate about which language is best for novices and teaching programming [163]. Studies highlight Python's strong and growing presence in introductory Computer Science curricula [86], courses for other domains that employ programming [58, 120, 191], and other educational environments [221]. Python overtook Java as the most popular introductory teaching language at top U.S. Universities in 2014 [86]. To return to Figure 4.6, Python offers a more shallow abstraction gradient. It does not require semi-colons to terminate each line and drops the braces in favour of prescribed indentation. An educator does not need to explain to students what Java's `void` does, which involves understanding how functions return data, something not typically covered in the first lesson of a novice programming course. Learning to code with Processing and Python also provides an entry to other domains of Python programming, including artificial intelligence, data-science and visualisation, game development, web development, desktop applications, and web scraping. Python includes many characteristic features for supporting learning (by abstracting away some more complex concepts), including small and clean syntax, dynamic typing, expressive semantics, immediate feedback, and object orientation [81].

Potential Advantages of Processing-Python over Processing-Java

Educators can leverage the advantages of Python described in the previous subsection by employing Python to write Processing code.

In 2008, Bälter and Dale [18] experimented with an introductory programming course that combined Python, Processing, and Core Java. They opted to introduce coding fundamentals for the first four weeks using Python, shifted to two weeks of Processing-Java following that for its graphics capabilities, and then concluded with a final four weeks of Processing and Java followed by a four-week student-determined final project. The vast majority, 16 of the 26 students, chose Processing for their final projects, while four used text-based Java. In their conclusion, Bälter and Dale mention Nodebox (a Python-based Mac OS X Processing alternative) as an all-Python option for similar courses, which was still under development at the time. A few years later, Processing's Python Mode would combine all those technologies (Python, Processing, and Core Java) into a single IDE, effectively enabling such a course to employ Python throughout.

In 2013, Frederik Berlaen would redesign DrawBot for macOS, a Processing-inspired environment for creating two-dimensional graphics using Python code. DrawBot has proven itself as part of the curriculum within selected courses at the Royal Academy of Art in The Hague.[4] Several other Processing-like Python options have since appeared, reviewed in Section 4.2.4. This proliferation of Python-based Processing-like environments, many led by people involved in education, points to an identified desire for teaching using Python creative coding environments.

The Python language should help novices writing Processing code to produce results (visual output in this case) more rapidly, heightening their sense of accomplishment. As Hsiao-Chi *et al.* conclude in their multi-group study of students learning programming [120], compared with students who learn Java, Python students are more likely to: reduce their maladaptive cognition; improve their self-efficacy and motivation to learn; feel a greater sense of learning achievement. Processing-Python solutions, therefore, provide an arguably more beginner-friendly and approachable (Python-based) experience than Processing-Java and, at the same time, offer many/all the same graphics and multimedia programming features [32].

After the release of Processing version 4 in August 2020, Ben Fry, co-creator of Processing, wrote on the potential of porting the entire project to Python[5], citing issues with Java, particularly Oracle's handling of the product. Fry labelled Python as "terrific," although lacking readily available components for interactive graphics, and described the Python Mode (Jython) implementation as

[4] https://www.drawbot.com/#education

[5] https://github.com/benfry/processing4/wiki/Processing-4

"great". However, he desired the "NumPy [library] and all those other things that make Python wonderful"—a shortcoming addressed in py5. As the official py5 documentation describes [202]:

"Processing.py [see Processing Python Mode] is the spiritual ancestor to and inspiration for py5. [It] is similar to Processing.py in that both use Python syntax but their implementations are very different. Processing.py and py5 do not share any code but py5 benefits from code in the Processing core libraries written to accommodate Processing.py."

Compared with Processing, py5 also includes Python-style naming conventions, various Processing enhancements for named and shorthand hexadecimal colour values, NumPy methods for selecting and manipulating pixels, profiler functions, and OpenSimplex 2 noise.

Processing Python Mode / Processing.py Limitations

Figure 4.7 illustrates at a very high level how Processing Python Mode passes code from the editor to Jython so that Java may interpret it. However, Processing.py's Jython implementation has its limitations: (a) it is source-compatible with Python 2.7 (not 3+), which the Python Software foundation officially sunset[6] on January 1, 2020; (b) it does not support CPython libraries, such as NumPy for handling complex matrix operations or Pymunk for simulating 2D physics.

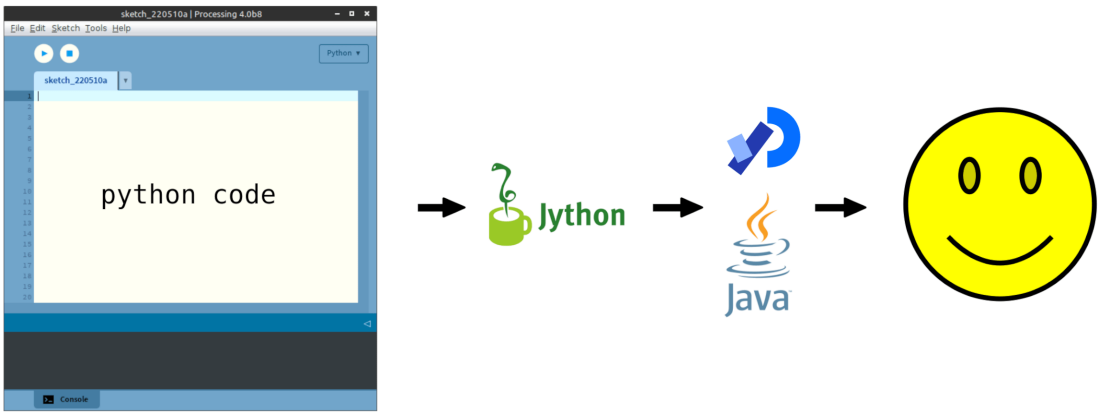


Figure 4.7: Processing Python Mode's Jython implementation. Diagram by the author.

This paper presents a new Processing-inspired development environment, Thonny-py5mode, which supports Python 3 and CPython libraries (via py5) that the author (Bunn) developed as a software plugin for the Thonny IDE.

[6] https://www.python.org/doc/sunset-python-2

4.2.3 The Thonny-py5mode Plugin

Thonny is a beginner-oriented IDE for Python programming, including an embedded CPython (3.x) interpreter, features for explaining variable scope, and a GUI for managing Python packages. The Thonny-py5mode plugin installs JRE and configures Thonny for use with py5. It adds assistive Processing-like features to the Thonny editor, including syntax-highlighting and hinting for py5 functions and keywords, a colour mixer, menu items for converting code between Processing.py and py5, a py5 quick reference, and a Processing-IDE-inspired colour scheme. It provides a much-needed successor to the Processing IDE's Python Mode coding experience, transforming Thonny into a creative computing environment by adding features that employ Processing's core libraries to generate interactive visual output via py5.

py5 provides module, class, imported, and static modes, and the Thonny-py5mode plugin can switch how it runs code to support the mode the user wishes to employ. For example, static mode is the simplest way to start writing code as it does not require any `import` lines or even `setup()` and `draw()` blocks. In static mode, drawing a square requires nothing more than a single line with a `square()` function (and the appropriate three arguments for x-coordinate, y-coordinate, and extent).

The py5 and Thonny-py5mode communities are overlapping groups of people, and ongoing discussion between those developers and users informs new features for both projects. On the other hand, work on the Processing Python Mode / Processing.py project has largely stalled, and the project has sought (unsuccessfully) a new leader for some time [65].

The Thonny-py5mode plugin removes the complexity beginner programmers may face setting up a working py5 environment (installing Python, setting up a Java Runtime Environment, configuring a suitable code editor, and so on). Effectively, it emulates the successful approach adopted in Processing's Python Mode. Namely, a simple-to-activate mode for an existing IDE that conveniently transforms it into a Python environment for creative computing. Thonny-py5mode makes it easy for beginners to start with py5, also serving as an immediately familiar alternative for anybody familiar with the Processing IDE (especially its Python Mode).

Figure 4.8 depicts the Thonny interface with the plugin installed as part of a diagram that approximately contrasts the workings of py5 (which employs JPype over Jython) against those of Processing's Python Mode (Figure 4.7). Note that the code editors may look the same or similar because the Thonny IDE has its Processing-style theme active that Thonny-py5mode applies.

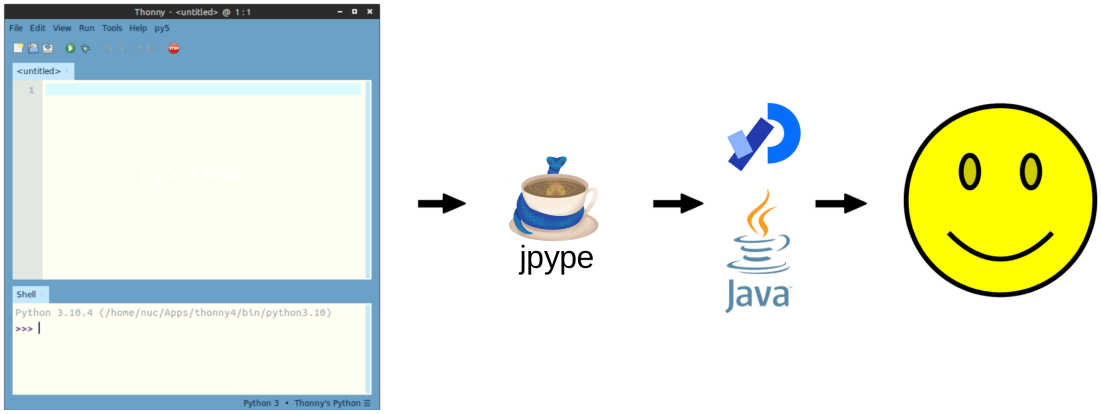


Figure 4.8: The Thonny IDE with Thonny-py5mode activated. py5 uses JPype to interface between Python 3 and Processing's Java libraries. Diagram by the author.

The Thonny IDE includes a graphic interface for managing Python packages (Figure 4.9) to make it easy for users to install additional packages such as Pymunk for incorporating 2D physics in simulations programmed with py5. This interface avoids users having to use a command-line interface for managing Python packages.

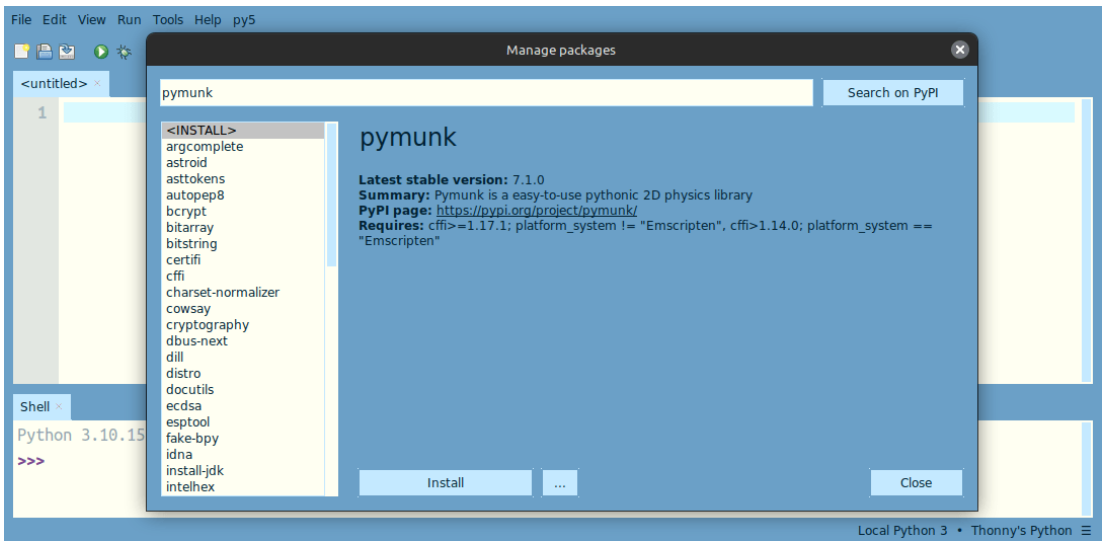


Figure 4.9: Installing Pymunk via the Thonny package manager. Screenshot by the author.

4.2.4 Tools Combining Processing and Python

The introduction section mentions that several new Processing-Python environments and tools have emerged as Python Mode alternatives. This section reviews those, providing technical insight into their differences, and discussing the advantages and disadvantages of each approach. It also reiterates and

highlights the benefits that Thonny-py5mode provides as a new Python-based creative programming environment for teachers, learners or beginners, and creative coders.

In a systematic review of different Python-based programming options for design and architectural education, Villares and de Carvalho Moreira [242] survey 43 environments/tools, "of which 32 at least [are] superficially investigated, [and] at least 20% (7 of 32) have substantial educational aims." The study highlights Python's strong and growing presence in introductory computer science curricula, creative computing courses, and other educational settings. Python is well-suited to novice programmers, and Processing provides a proven environment for teaching textual programming, especially to art, design, and other 'visually-oriented' students [83, 186, 188]. Processing Python Mode may, therefore, seem like an ideal for teaching creative programming, but its Jython component has limitations.

Web-Browser-Based Options

Villares provides a table (Table 4.1) of different Processing + Python software that has emerged in recent years. The bottom four entries in Table 4.1 are web-browser-based. Three of those, pyp5js (depicted in Figure 4.10), Proceso, and BrythonIDE, combine p5.js (JavaScript) and Python using Pyodide, Transcrypt, or Brython; the other web-browser-based approach, employed in SkulptIDE and trinket.io, utilises ProcessingJS (JavaScript), but development on ProcessingJS ceased a while before 2018.[7]

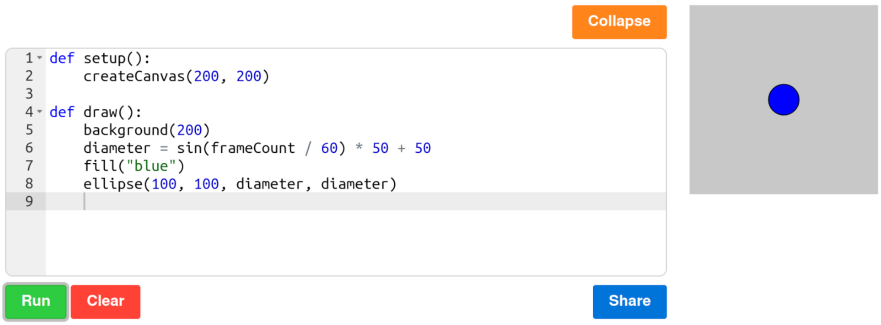


Figure 4.10: Running a p5.js–Python sketch in a web browser using pyp5js. Screenshot by the author.

Browser-based solutions have the advantage of running on any platform with a modern web browser. These solutions offer the quickest means to start writing and executing code and are helpful in many

[7] https://github.com/processing-js/processing-js

Name	Processing features	Based on (and Python ver.)	Python std. library	Libraries ecosystem	Main features	Main limitations
Processing Python Mode (the "reference implementation" for comparison)	Processing 3 Java	Jython (Python 2)	Complete	Java and Processing Java	Available inside Processing IDE; highly Processing-3-compatible	Processing 4 incompatible; no web sharing; no modern Python 3 libs; needs new maintainer
py5	Processing Java graphics via JPype	Python 3	Complete	Python and Processing Java	Truly Python-3-compatible library support; Jupyter notebooks support; core capabilities of Processing 4 Java	Snake_case names; experimental; some Processing Java libraries may not work
p5py	New implementation based on OpenGL (incomplete), and will add Skia back-end	Python 3	Complete	Python only	Truly Python-compatible; no Java/JVM dependency	Snake_case names; experimental; still incomplete; no access to Processing Java libraries
pyp5js (Pyodide or Transcrypt mode)	p5.js	Python 3 via Pyodide or Transcrypt	Complete	Python, JavaScript and p5.js	Web-ready sketches and editor; highly p5.js-compatible; Pyodide is highly Python-compatible	Experimental; still incomplete; p5.js features (as opposed to Processing Java/Python modes)
Proceso	p5.js	Python 3 via Pyodide	Complete	Python, JavaScript and p5.js	Browser-based sketches; highly p5.js compatible; Python-compatible; names similar to py5	p5.js features (as opposed to Processing Java/Python modes)
SkulptIDE and trinket.io	ProcessingJS	Skulpt (Python 2, but migrating to 3)	Partial	Unknown, possibly JavaScript	Appealing and refined web IDE; browser-based sketches	ProcessingJS is defunct; not extensible
BrythonIDE and p5py.com	p5.js	Brython (Python 3)	Fairly complete	JavaScript and p5.js	Browser IDE; browser-based sketches; highly p5.js-compatible	p5.js features (as opposed to Processing Java/Python modes)

Table 4.1: Processing + Python tools table. Adapted from Villares [241].

situations where downloading additional software can be problematic. However, these cannot utilise the rich ecosystem of Processing-Java libraries because none interface with Java in any way. While the p5.js project offers a robust collection of libraries, other Python libraries from the Python Package Index (PyPI) may be unavailable or could prove challenging for beginners to set up.[8]

py5 also supports a browser-based coding environment through Jupyter Notebooks, an open-source web application popular among scientists and researchers [82]. While web-browser-based solutions are preferable in several situations, and Jupyter Notebooks brings a host of valuable features, this paper focuses on a Python 3 alternative for a Processing-Java **desktop** IDE.

Comparing Thonny-py5mode and Processing (and similar) Desktop IDEs

Villares' table 4.1 lists a single solution (Processing Python Mode) using a desktop (opposed to browser-based) IDE. With some relatively simple to involved configuration effort, users can employ a general-purpose desktop IDE (such as Visual Studio Code, Geany, or similar) for Processing, py5, p5py, or pyp5js. However, Thonny-py5mode aims to provide a multi-platform desktop IDE experience for beginners with little to no configuration required. Moreover, it includes many features specific to py5 that a user could not access in a more general-purpose IDE.

Drawbot, Nodebox, and Shoebot are pure Python alternatives to the Processing IDE absent from Villares' table of different Processing + Python environments/tools. None leverage the Processing API. Drawbot and Nodebox provide ready-configured/bespoke desktop IDEs, but those support Apple Mac platforms exclusively. Shoebot is a multi-platform port/rewrite of Nodebox, which also features a bespoke IDE . Shoebot does not provide the extensive features of the Thonny-py5mode plugin, including syntax highlighting and hinting for functions and keywords, a colour mixer, menu items for converting code from Processing.py scripts, and a package manager for installing other Python packages. The Shoebot installation process is also more involved, requiring users to install several dependencies. Shoebot started in 2007 and transitioned from Python version 2 to 3; it is a capable coding environment, particularly for vector graphics, and remains an interesting open-source project to watch.

The p5py project (Figure 4.11) has made good progress on developing a native Python 3 port of the Processing API. However, a few core individuals currently driving the p5py project must effectively recreate the entire Processing API from scratch in Python. py5, instead, leverages the mature Processing-

[8] Refer to Brython, Pyodide, and Transcrypt official documentation for package availability and limitations

Java codebase via JPype, actively developed by a large community of programmers. p5py has no desktop IDE plugin like Thonny-py5mode, or Jupyter Notebooks kernels, and users must install GLFW (a multi-platform OpenGL library) separately. However, p5py is useful for anybody seeking a pure-Python library to programme simulations and interactive art, accumulating 700 stars[9] on GitHub to date.

Figure 4.11: Writing p5py code in a general-purpose code editor (not a bespoke IDE or plugin). Screenshot by the author.

Neither py5 nor p5py provides a Processing kind of IDE experience. That is, a bundled code editor with features specifically to complement working with their libraries. Hence, the opportunity to create a Processing Python Mode / Processing.py successor by way of a Thonny IDE plugin.

4.2.5 Thonny-py5-mode Examples

Accomplished creative coders and beginners alike have employed the Thonny-py5mode environment for programming generative artworks. Figure 4.12 presents an adaptation of Lieven Menschaert's NodeBox script, Aquatics, programmed using Thonny-py5mode. This spawns a creature with a random fill colour, shape (defined by something named the superformula), and no fewer than three eyes. There is a 70 percent chance that hair will grow along the creature's edges, which can be swayed by the force of a randomly directed current.

2-Axis pen plotters provide another fun way for students or artists to experiment with generative output using ink and paper. Again, the Thonny package manager makes it convenient to install the powerful Python library, vpype—a "Swiss-Army-knife" for working with vector graphics for plotters [23]. Using vpype, the programmer can perform some post-processing on py5-generated SVG

[9] https://github.com/p5py/p5

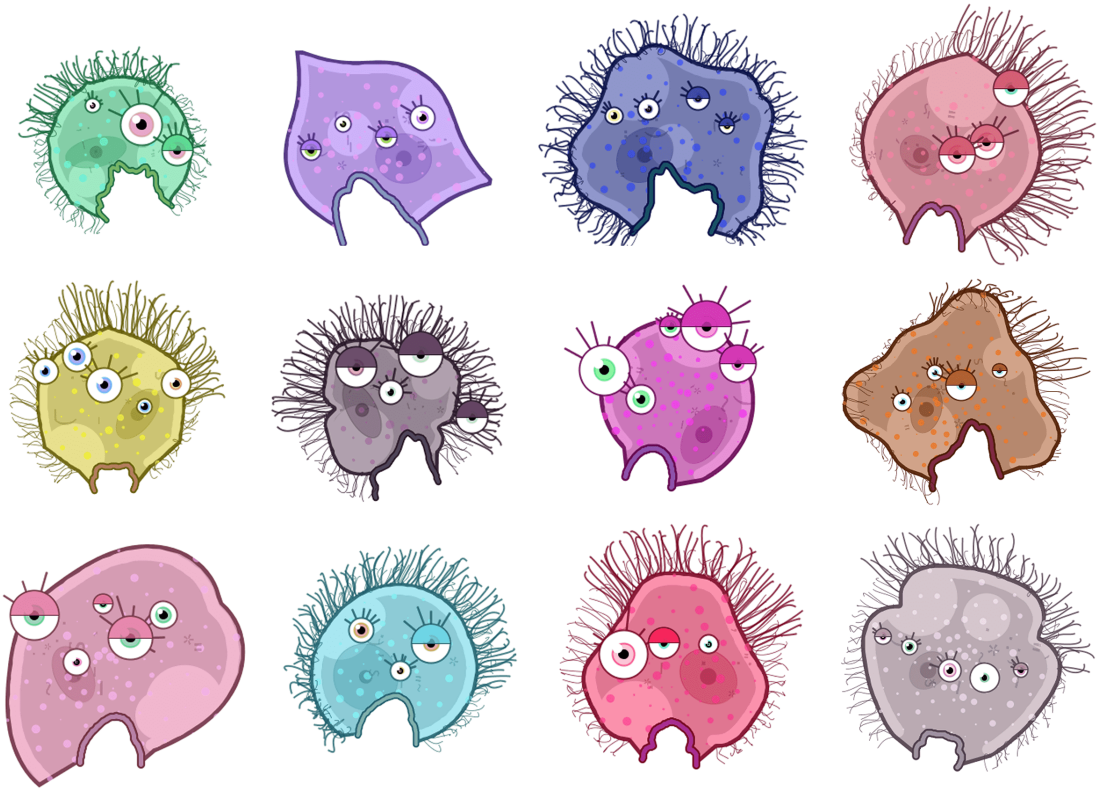


Figure 4.12: Artwork inspired by Lieven Menschaert's NodeBox script *Aquatics*, programmed using Thonny-py5mode. Artwork by the author.

files to optimise paths for pen plotting. Figure 4.13 shows a Thonny-py5mode program that generates SVG files that the creator will plot using different colour pens.

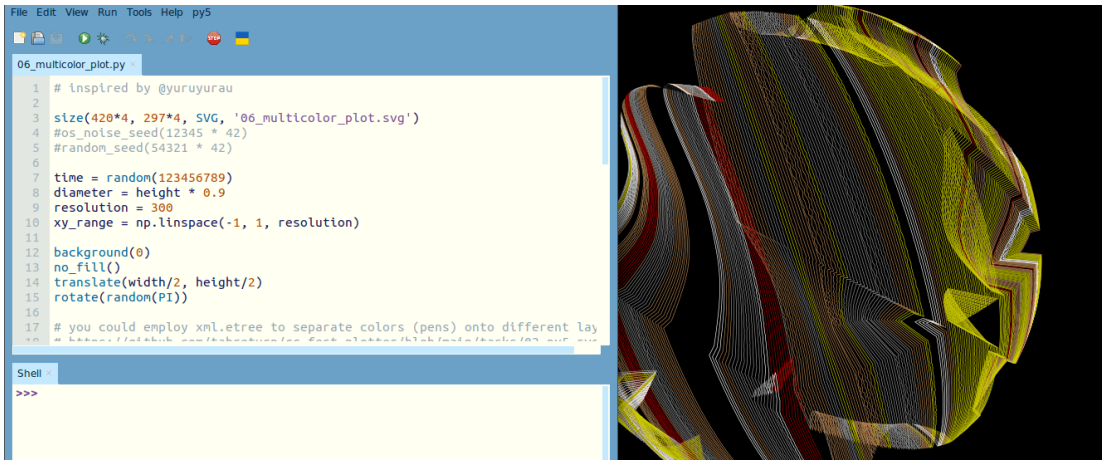


Figure 4.13: Programming generative (multi-pen) plotter art in the Thonny-py5mode environment. Screenshot by the author.

A wide variety of other projects might employ Thonny-py5mode, including for visual art and design work, video games, installation artworks and projections, sound art, prototypes, and other creative applications. Reviewing different results created using Processing, OpenFrameworks, and OPENRNDR offers inspiration for what is possible.

4.2.6 Thonny-py5mode in the Classroom

At Massey University's College of Creative Arts in New Zealand, educators adapted a course that used Processing's Python Mode to one using Thonny-py5mode for 2022. This required revising the course learning materials, which also served as part of the py5 project's efforts to develop tutorial documentation. The Processing Foundation co-funded Zelle Marcovicci for the documentation work, mentored by Jim Schmitz and Tristan Bunn. Materials went 'live' on the official py5coding.org website later in 2022 [169]. To date, there is no comparative study to assess the 2022 cohort's experience and abilities against those of the previous cohorts who used Processing's Python Mode. Figure 4.14 is an example of one student's submission at the mid-point of the course, after roughly 5 × 2-hour training workshops using Thonny-py5mode.

A cursory review of the 2022 assignments indicates that the work is at least as good as the previous year's. With some tweaking, the Python Mode materials adapted appropriately for Thonny-py5mode. This switch also enabled a few improvements to the curriculum. For instance, Pymunk replaced

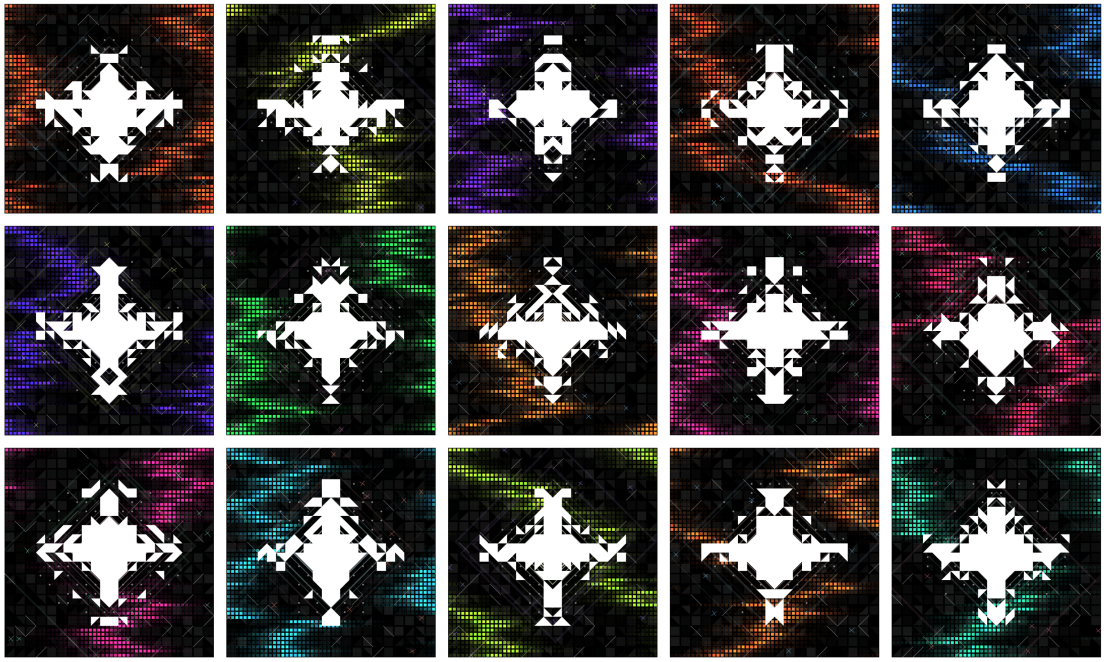


Figure 4.14: Abstract avatars created by student Toby Penno. Shared with permission.

pypybox2d for physics, providing improved performance due to its C bindings and more complete and up-to-date documentation than pypybox2d. Access to NumPy was helpful for anything that involved matrices, and several py5 functions incorporate NumPy for manipulating arrays of pixels. The `py5Vector` class works seamlessly with NumPy arrays.

Educator, researcher and visual artist Alexandre Villares has employed Thonny-py5mode in secondary and tertiary, face-face and online settings (synchronous and asynchronous) in Brazil, and for the online Brazilian-Portuguese language offering *Designing with Python: Programming for a Visual Context* on the online learning platform Domestika. The Domestika course has seen 1320 student enrolments to date with positive student feedback [240].

4.2.7 Future Work and Thonny-py5mode Development

In the near future, we aim to conduct and report on Thonny-py5mode user studies, creative outputs, and technical achievements toward a Python 3 development environment for teaching computer programming in a visual context.

Thonny-py5mode is under active development again in 2023. The plugin is closely aligned with the py5 project, responding to its user feedback, new features, and developments. GitHub hosts the Thonny-py5-mode code repository. Jim Schmitz leads the py5 project, and Bunn leads Thonny-

py5mode development; both related projects rely on GitHub for source control, its built-in *Issues* feature for issue tracking, and *Discussions* for collaborative communication around the project. Both projects host releases on PyPI (The Python Package Index). py5 developments effectively operate upstream of Thonny-py5mode; Thonny-py5mode dependencies are specific, but there are plans to 'unpin' this so that new versions of py5 update independently of the Thonny-py5mode plugin version. Users can update the Thonny-py5mode plugin via the Thonny package manager, which tracks and retrieves packages in PyPI. There is documentation for installing the Thonny-py5mode plugin on PyPI and GitHub, also linked in the py5 documentation.

4.2.8 Project Development Priorities

There is a discussion among the developers of py5 and Thonny-py5mode regarding potential support for live coding, which would enable users to alter py5 code while it is running. This feature may prove helpful in teaching and as software for users looking to VJ, which involves the real-time creation and manipulation of visuals, often synchronised with music, for live performances using software, hardware, and generative techniques [174]. Specific priority goals for the Thonny-py5mode project moving forward are to:

- package a 'portable' version of Thonny-py5mode that includes the plugin pre-bundled, which could additionally run off of a flash drive;
- carry out ongoing bug fixes (reported via GitHub *Issues*) and other minor enhancements;
- to more actively position py5/Thonny-py5mode as a compelling option for educators to teach students about Python and creative coding.

Lower priority goals include:

- GitHub *Actions* to publish packages (and other useful actions);
- a pyp5js-powered exporter (to export Thonny-py5mode scripts for JavaScript/Web);
- providing a feature for downloading bundles of py5 examples (from online repositories such as GitHub) to browse and run on the user's device.

The Thonny IDE also presents some interesting opportunities beyond what the Processing IDE might manage. For example, it is easy to move into using Pygame Zero (for creating basic games) by

simply enabling the relevant plugin via a Thonny menu. Blender scripting offers a way to explore more advanced 3D content than Processing's 3D capabilities can offer. Using Python to interface with Blender's powerful render engine, one can output creations in high-resolution image and video formats that employ particle systems, rigid-body/fluid/cloth dynamics, metaballs, and volumetrics, among other graphically intensive techniques [34]. To accomplish this in Thonny, perhaps one could add a menu item using the plugin architecture, which in turn sends scripts to Blender running in 'headless' mode (so that it renders an image without opening the Blender application). The significance of discussing these two examples is to illustrate how the Thonny IDE can provide a suite of creative computing features that employ different libraries (including Processing core libraries via py5) using a plugin approach. In contrast, this is potentially more difficult to accomplish in a more focused IDE like that of Processing.

4.2.9 Conclusion

Thonny-py5mode combines a popular beginner Python IDE, Thonny, and the most promising successor to the Processing.py project, py5—which does not otherwise include a Processing-like IDE experience. The plugin offers a capable and approachable integrated development environment to complement py5. Like Python Mode in the Processing IDE, installing it is an easy enough process for a beginner. One can also package this setup into a portable application for easy distribution, including everything pre-configured. Several educational environments currently employ Thonny-py5mode for teaching. Researchers, developers, and users seeking to get involved with the project can refer to Appendix for the relevant online resources.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

4.3 Evaluating Task–Technology Fit in Thonny-py5mode: Supporting Graphical Output in Introductory Python Education

Journal: Journal of Information Systems Education

Journal Ranking: Q2 (see Scimago Journal & Country Rank)

Author(s): Bunn, T., Bere, A., and Douglas, M.

Statement of Authorship: See Appendix F

Date Received: 20 August 2025

Date Published: *Under review*

Publication Evidence: see Appendix C.1

4.3.1 Abstract

Coding environments that integrate graphical output into programming instruction can enhance engagement, comprehension, and motivation among beginners. This study evaluates Thonny-py5mode, a plugin for the Thonny IDE that emulates the Java-based Processing creative coding IDE, but instead supports Python via the py5 library. Guided by the Task–Technology Fit (TTF) framework, we extend the model with personalised learning, hedonic motivation, and effort expectancy to capture factors influencing Thonny-py5mode user adoption in educational contexts. The study analyses survey data from 143 students in a 100-level university Python course that incorporated four weeks of Thonny-py5mode use, including an assessment requiring participants to recreate six graphical challenges. We employ Structural Equation Modelling (SEM) to test nine hypotheses mapped to our extended TTF framework. Our results show that Thonny-py5mode's usability and task support, along with its ease of use, contributed to a strong perceived TTF, which in turn influenced students' intention to continue using it. These results highlight Thonny-py5mode design priorities that can encourage adoption and sustained use in related educational coding environments and tools.

Keywords: Task–technology fit (TTF), Python, Introductory programming, Integrated development environment

4.3.2 Introduction

Programming graphical output through textual code can make learning to program more accessible and engaging by providing immediate visual feedback. Environments supporting visual output encourage experimentation, deepen conceptual understanding, and boost learner motivation [16, 18, 47]. One of the earliest examples is Logo's Turtle graphics, developed in the 1970s at MIT, where learners controlled an on-screen turtle to draw shapes [215]. Inspired by this approach, Python has included a Turtle module since version 1.5.2, released in 1999 [97, 178]. Building on this tradition, environments such as Processing and p5.js introduce programming through graphical output but rely on Java or JavaScript [68, 130, 206]. With Processing's Python mode (Processing.py) now unmaintained and incompatible with current IDE versions [65], a gap exists for educators seeking Processing-based learning experiences in Python.

Python remains a leading choice for introductory programming due to its readability and industry relevance [63, 86], yet many courses emphasise text-only output, focusing on syntax and logic through the console rather than graphical or interactive elements [83, 88]. This emphasis can limit engagement for learners from creative or interdisciplinary backgrounds; by contrast, Processing-style environments have been shown to enhance understanding, retention, and motivation among these audiences [118, 188, 200] suggesting potential benefits for Python-based tools with similar capabilities.

This study forms part of the ongoing research and development of Thonny-py5mode (https://github.com/tabreturn/thonny-py5mode), a bespoke plugin that introduces the Processing creative-coding paradigm to Python through the beginner-friendly Thonny IDE via the py5 library [11, 202]. The plugin addresses the gap left by Processing.py, enabling students to produce graphical and interactive output in Python without the complexity of managing manual package installations. Instead, the setup process is handled entirely through Thonny's built-in plugin manager in just a few clicks. Thonny-py5mode's interface (Figure 4.15) is deliberately designed to emulate the Processing IDE, with a similar layout, colour scheme, built-in sketch runner, and syntax highlighting. A dedicated "py5" menu provides quick access to the py5 reference, a colour mixer, an online/printable cheatsheet, and the sketch folder.

Thonny-py5mode provides switchable coding modes, including an *Imported* mode, which allows learners to use familiar Processing functions such as `draw()` and `square()` without explicitly adding Python `import` statements or `py5` prefixes. When running a script (or "sketch"), Thonny-py5mode automatically manages the output window while preserving full Shell compatibility for debugging.

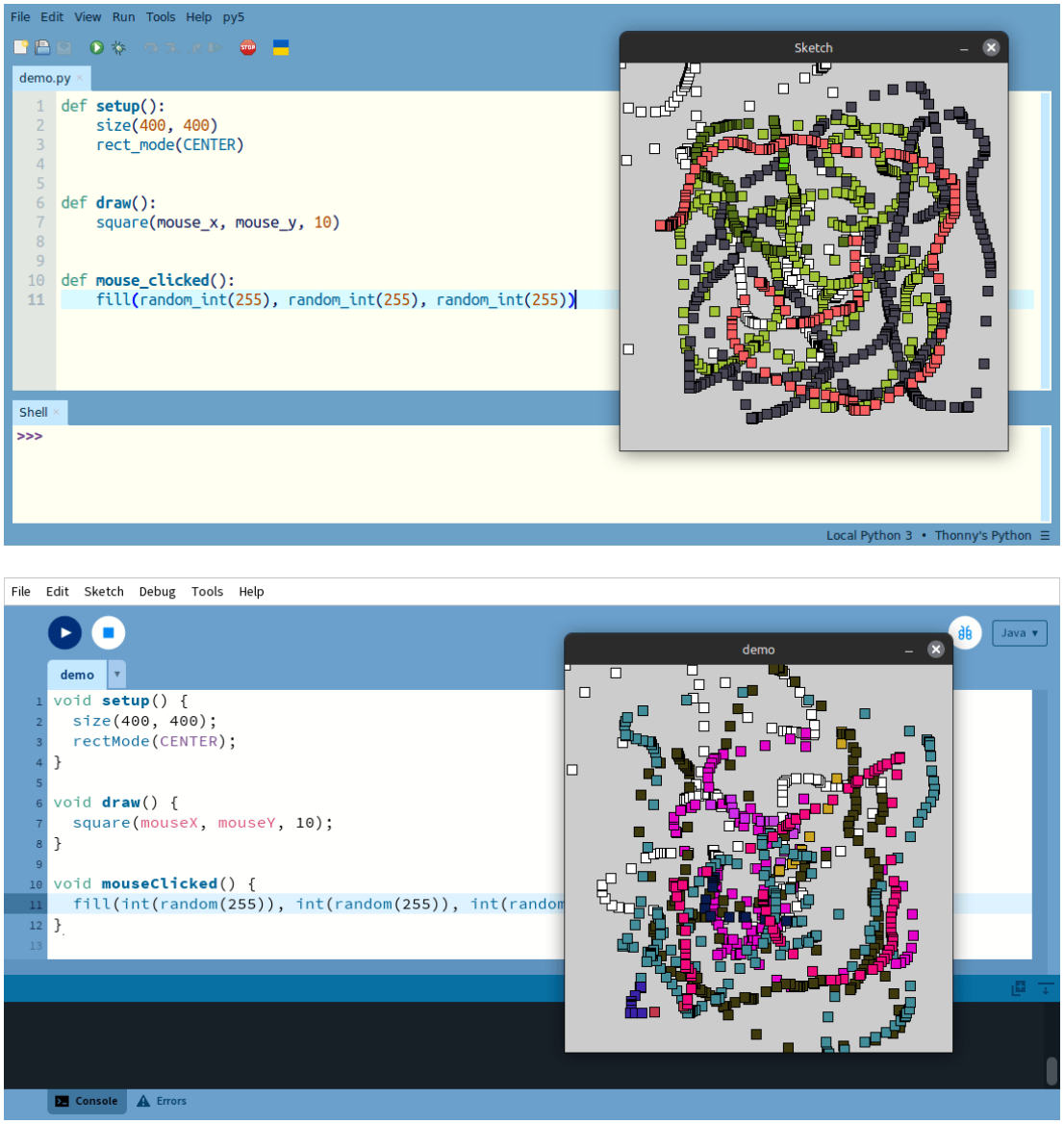


Figure 4.15: Top: Thonny-py5mode plugin activated, running a script. Bottom: Processing 4.x running a script. Screenshots by the author.

Guided by the TTF framework [69], we surveyed students in an introductory Python course to evaluate Thonny-py5mode's usability, task support, and overall effectiveness.

4.3.3 Background

Goodhue and Thompson (1995) introduced the Task–Technology Fit (TTF) model to explain how well a technology supports users in performing their tasks, and how this alignment influences both system use and performance outcomes. They argued that users achieve better results and are more likely to continue using a system when its capabilities align with the demands of the task [76, 126]. In this view, *fit* occurs when a tool's features directly support the requirements of the work, enabling more effective task completion. TTF extended earlier adoption models including the Technology Acceptance Model (TAM) and Theory of Planned Behaviour (TPB) by incorporating the direct relationship between task–technology alignment and task performance, which those models did not explicitly address [5, 52]. By taking a post-adoption perspective, TTF highlights that positive attitudes alone do not guarantee improved performance or sustained use, unless the technology fits the task. Goodhue and Thompson (1995) formalised this idea through the "Technology-to-Performance Chain," which posits that performance gains occur only when a strong TTF and actual system use occur [75].

Education encompasses diverse activities such as instruction, collaboration, and assessment. It employs technologies ranging from LMSs to coding platforms, providing multiple scenarios for TTF analysis. However, simply adopting educational technology is insufficient; tools must align with pedagogical needs to generate meaningful outcomes [251]. Several recent studies have applied the TTF framework to evaluate how well digital tools support teaching and learning. For example, Alyoussef (2021) found that higher TTF in MOOCs improved satisfaction, performance, and sustained use [7]. Yaakop *et al.* (2020) linked strong alignment with collaborative and resource-sharing tasks to continued engagement with online environments [250]. TTF studies of Google Workspace have shown that matching tool features to collaborative learning tasks predicts adoption [31, 207]. Wang and Kartika Sari (2024) demonstrated that aligning game mechanics with learning goals enhanced engagement and reduced cognitive load [244]. Lim and Lee (2021) connected fit to creative and collaborative skills development [119]. Together, these studies confirm TTF as a valuable framework for evaluating educational technologies, where alignment between tool features and learning tasks may drive engagement, performance, and sustained use.

Although previous research has proven that the fit between learning task requirements and technological characteristics is a central antecedent of technology adoption, these studies overlooked students' personal characteristics. This omission weakened the effectiveness of the TTF and reduced its explanatory power, since technology must not only meet task requirements but also align with individual learners' traits [21]. Researchers have observed similar limitations in TPB, which emphasises attitude, subjective norm, and perceived behavioural control, but has faced criticism for neglecting individual-level characteristics beyond its core constructs [5, 229]. Beyond generic adoption models, studies in South African higher and basic education underscore that curriculum-level technology integration and educators' ICT self-efficacy meaningfully shape how they appropriate tools in practice [139, 182].

For these reasons, the current study extends TTF by introducing a personal characteristics construct, operationalised through personalised learning, hedonic motivation, and effort expectancy. Incorporating these traits strengthens the framework and aligns it with the enriched TPB approach, which has shown to outperform the original TPB in predicting behavioural intentions [5, 229].

4.3.4 Hypothesis Development

Figure 4.16 presents our extended TTF model, integrating nine hypotheses. To better frame these, we broadened the *Personal Characteristics* construct to include *Personalised Learning*, *Hedonic Motivation*, and *Effort Expectancy*—factors shown to influence learners' technology engagement and capture individual differences in perceived fit [203]. These highlight pedagogical affordances of Thonny-py5mode that go beyond functional alignment, incorporating insights from acceptance models such as TAM and UTAUT/2, where tailored experiences, perceived enjoyment, and ease of use are recognised as critical drivers of behavioural intention [22, 142].

The subsections below describe each construct and its associated hypotheses, while Section 4.3.6 details the analysis method.

Task Characteristics

Physical and cognitive features of a task such as complexity, structure, frequency, interdependence, and cognitive load that shape performance and determine how well technology can support it. In the TTF model, better alignment between task requirements and technology capabilities leads to improved performance, satisfaction, and adoption [76, 93]. In this study, Python programming

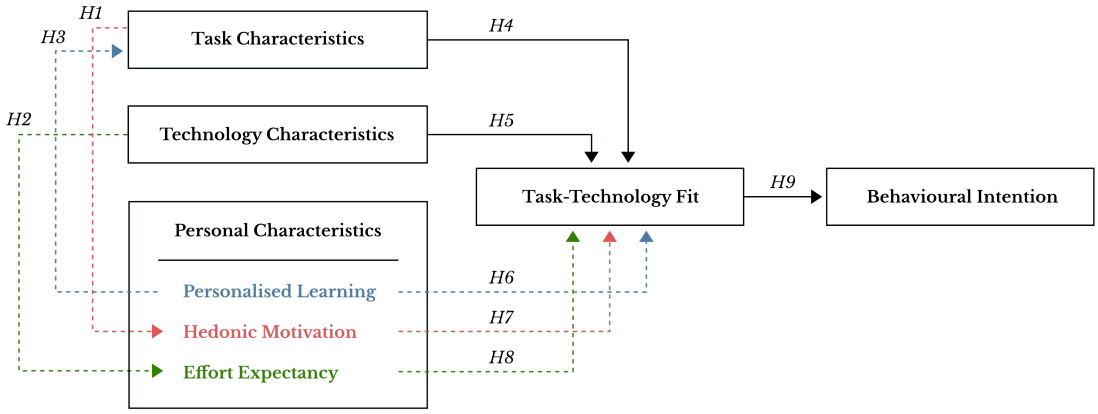


Figure 4.16: Extended TTF model for Thonny-py5mode. Constructed by the authors.

tasks include writing and debugging code, incorporating py5 drawing functions, and solving graphical challenges, which require rapid feedback, low cognitive load, and ease of use—capabilities Thonny and Thonny-py5mode aim to provide. The following hypotheses have been formulated based on the task characteristic construct relationship with hedonic motivation and task–technology fit, respectively.

H1 Task Characteristics positively influence the Hedonic Motivation associated with Thonny-py5mode-supported learning.

H4 Task Characteristics positively influences the Task–Technology Fit associated with Thonny-py5mode-supported learning.

Technology Characteristics

Functional features, properties, and usability aspects of a system that determine how effectively it supports users' tasks, including reliability, accessibility, data quality, interactivity, and alignment with user abilities and needs [55, 76]. In the TTF framework, strong alignment between technology characteristics and task requirements enhances performance, adoption, and satisfaction; while poor alignment can hinder task completion, particularly in education [159]. In this study, Thonny-py5mode's beginner-friendly interface, graphical output, and real-time debugging aim to meet the needs of Python novices, lowering cognitive load and supporting tasks such as py5 coding, visualisation, and problem-solving. The following hypotheses have been formulated based on the technology characteristic construct relationship with effort expectancy and task–technology fit, respectively.

H2 Technology Characteristics positively influences the Effort Expectancy associated with Thonny-py5mode-supported learning.

H5 Technology Characteristics positively influences the Task–Technology Fit associated with Thonny-py5mode supported learning.

Personal Characteristics

Individual user traits, motivations, preferences, and cognitive capabilities that can affect how a user perceives and interacts with technology, as well as how they perform tasks. These attributes shape perceptions of fit, ease of use, and enjoyment, and can ultimately affect technology adoption and outcome [69, 224, 237, 244]. As Figure 4.16 illustrates, we operationalise personal characteristics through three constructs: *Personalised Learning*, *Hedonic Motivation*, and *Effort Expectancy*, examined in detail in the three subsections that follow.

Personalised Learning

Tailoring the environment to an individual's needs, preferences, and pace of learning to enhance engagement, satisfaction, and performance. In TTF terms, personalisation enables system features and tasks to adapt to learners' cognitive abilities, motivations, and learning styles [44, 136]. Thonny-py5mode supports this through optional tool panels, real-time visual feedback, and coding activities that can be approached at varying levels of complexity, allowing learners to progress at their own speed and focus on relevant concepts. By aligning with students' cognitive and emotional needs, these features can increase engagement and enjoyment in programming, particularly for novice learners [6, 210]. In resource-constrained contexts, higher ICT self-efficacy supports more constructive classroom uses of technology, suggesting that personalisation, which builds confidence, can strengthen fit [139]. The following hypotheses have been formulated based on the relationship between the personalised learning construct and task characteristics, as well as task-technology fit.

H3 Personalised Learning positively influences the Task Characteristics associated with Thonny-py5mode-supported learning.

H6 Personalised Learning positively influences the Task–Technology Fit associated with Thonny-py5mode supported learning.

Hedonic Motivation

The extent to which a technology is perceived as enjoyable or fun to use. Higher enjoyment can increase users' intention to adopt the technology and lead to their improved task performance [15, 45]. Within the TTF framework, hedonic motivation can also promote engagement and reduce cognitive resistance in learning contexts [227]. In this study, Thonny-py5mode's visual feedback, interactive coding environment, and immediacy aim to create an enjoyable learning experience that sustains interest—particularly for novice programmers—enhancing satisfaction and encouraging continued use [44, 244]. The following hypothesis has been formulated based on the relationship between the hedonic motivation construct and task–technology fit.

H7 Hedonic Motivation positively influences the Task–Technology Fit associated with Thonny-py5mode-supported learning.

Effort Expectancy

The perceived ease of use of a technology; alongside TTF, it is a key construct for user acceptance and can influence performance in educational contexts. TTF suggests that the easier a technology is to use, the better its fit to the task, the stronger the learning outcomes [7, 15, 210], and the lower the resistance to adoption [1]. In this study, Thonny-py5mode is designed to offer high perceived ease of use through a simplified, learner-friendly interface, clear error messages, and near-instant visual feedback. These features minimise the effort required for novices to complete coding tasks effectively and confidently [11, 12, 44, 210]. The following hypothesis has been formulated based on the effort expectancy construct relationship with task–technology fit.

H8 Effort Expectancy positively influences the Task–Technology Fit associated with Thonny-py5mode supported learning.

Task–technology Fit

Describes the degree of alignment between a technology's capabilities and the requirements of a user's task. High fit is typically associated with positive outcomes such as productivity, satisfaction, and behavioural intention to use the technology [76, 244]. As Figure 4.16 illustrates, it encompasses three dimensions in our model: *Task Characteristics*, *Technology Characteristics*, and *Personal Characteristics*. In this study, Thonny-py5mode-supported learning demonstrates task–technology fit when these

three dimensions align, meeting the demands of graphical Python programming tasks and the needs of novice learners. The following hypothesis has been formulated based on the task–technology fit construct relationship with behavioural intention.

H9 Task–technology fit positively influences students' Behavioural Intention to learn using Thonny-py5mode.

Behavioural Intention

A person's willingness or intent to use a technology in the future, over a sustained period. This is a core psychological variable in technology acceptance models such as UTAUT and UTAUT2, where it not only reflects interest but also predicts actual system use. Several aspects shape behavioural intention, including performance expectancy, effort expectancy, and TTF [1, 44, 244]. Higher behavioural intention is associated with greater engagement, improved performance, and stronger adoption continuity [45, 227]. In Thonny-py5mode-supported learning, beginner-friendly features, graphical feedback, extendable libraries, and a low learning curve (with the capacity to support more advanced projects over time) aim to address both task and personal needs, which should increase students' willingness to adopt and continue using it [244].

4.3.5 Course Setting

This study evaluated the perceptions of Torrens University Australia (TUA) students using Thonny-py5mode in *ITP122 Introduction to Programming*: a 12-week Bachelor level course introducing Python to beginners, delivered in Trimester 1, 2025. Students used Thonny-py5mode during Weeks 5–8, after completing Assessment 1 (in Week 4) and leading into Assessment 2 (due at the end of Week 8). In the first four weeks, they worked in IDEs such as Jupyter Notebook, PyCharm, and Spyder, producing only console output. Assessment 2 required them to complete graphical coding challenges using Thonny-py5mode. Figure 4.17 presents the six assessment tasks, to be recreated as accurately as possible using Python code.

These tasks required students to apply `if-else` statements, and `for` and `while` loops, in combination with a limited set of approximately 15 py5 drawing functions, with additional marks awarded for clear structure and effective commenting. The weekly content leading up to Assessment 2 covered: (Week 1) Introduction to Computer Programming and IDEs; (Week 2) Fundamentals of Decision

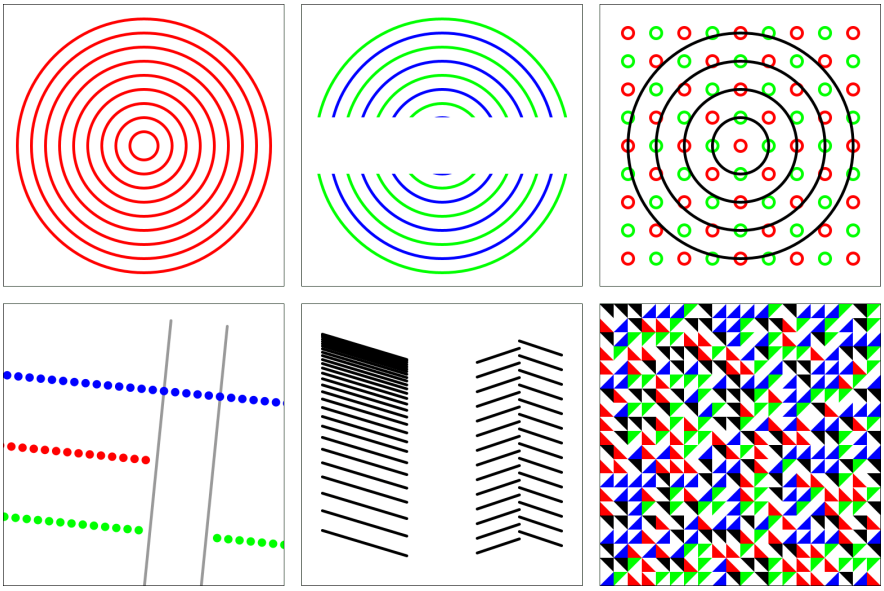


Figure 4.17: Thonny-py5mode Assessment 2 challenges. Developed by the author.

Logic; (Week 3) Intermediate Decision Logic; (Week 4) Lists and Dictionaries; (Week 5) Nested Decision Logic; (Week 6) Simple Loops; (Week 7) Basic Functions; and (Week 8) Intermediate Loops.

4.3.6 Method

Our proposed model (Figure 4.16) examined key factors shaping novice learners' behavioural intention to learn Python programming using Thonny-py5mode. Our study method employed a quantitative, survey-based approach, selected for its confirmatory and deductive qualities, and common application in studies applying TTF and other technology acceptance frameworks [6, 31, 244]. The model involved testing a priori hypotheses and examining theoretically grounded relationships using data from authentic educational settings [69, 142, 159].

We applied a structured validation approach to develop and validate the measurement instrument: a survey questionnaire comprising seven sections (Table 4.2) aligned with our model constructs (see Figure 4.16). We first developed our model through an extensive literature review to identify and define theoretical constructs, refining these through specialist input and analysis. Experts in Python teaching and learning reviewed the draft questionnaire to improve its structure and wording. Based on feedback, we amended potentially ambiguous statements to establish the instrument's content validity [31, 250].

We recruited ITP122 participants aged 18 years or older from TUA's campuses in Adelaide, Brisbane, Melbourne, Sydney, and online. From Week 9, following the submission of Assessment 2, we invited students to complete a voluntary, anonymous questionnaire either online or on paper, with no incentives offered and assurances that participation would not affect their grades. All items used a 5-point Likert scale (1 = strongly disagree to 5 = strongly agree) addressing constructs across the study's extended TTF model. Table 4.2 outlines the model constructs alongside their corresponding item/indicator groupings.

The researchers conducting the study did not teach or grade the ITP122 course, and students were aware that the Thonny-py5mode was under evaluation. The online questionnaire closed one week after the 12-week course concluded, and we distributed final paper questionnaires for completion during Week 12 classes. Completion time averaged 10–15 minutes, observed in both in-class and online administration. All participants provided informed consent, and the study complied with research ethics guidelines approved by the TUA Human Research Ethics Committee (HREC; Application 0394), classified as Low and Negligible Risk (LNR).

We employed Structural Equation Modelling (SEM), which requires an adequate number of participants to ensure consistent and reliable results. Guidelines for SEM recommend that the minimum sample size should exceed the number of distinct parameters in the model, with a common rule of thumb of 5–10 observations per indicator [90, 91]. The conceptual model in this study contained 24 indicators, suggesting an appropriate sample size between 120 and 240 participants. We collected data from 157 respondents. Data screening procedures addressed missing values, outliers, normality, and multicollinearity [90], resulting in the removal of 14 cases. Table 4.3 presents an overview of the screened respondent statistics, comprising 143 valid responses, exceeding the minimum (120) recommended for SEM and thereby supporting the reliability of the analyses.

We further examined the dataset for common method bias using Harman's single-factor test and the marker variable test [91]. Harman's test showed that no single factor accounted for the majority of variance, and the marker variable results indicated that substantive construct variance exceeded any method-related variance. Together, these findings suggest a low likelihood of common method bias.

4.3.7 Data Analysis and Results

Following data preparation and bias checks, we applied SEM to test and validate the research model, examining the critical determinants of students' intention to learn Python programming using the

Construct	Items for Modelling
Task Characteristics (TC)	• Thonny-py5mode supports effective hands-on learning through Python graphics creation. • Programming graphics using Thonny-py5mode encourages critical and analytical skills. • Overall, Thonny-py5mode is an effective environment for learning Python fundamentals.
Technology Characteristics (TCS)	• Thonny-py5mode features an intuitive, user-friendly interface. • I can complete graphical coding tasks efficiently using Thonny-py5mode. • Thonny-py5mode provides reliable, stable performance during coding sessions.
Personalised Learning (PL)	• Using Thonny-py5mode to code graphical output supports learning at my own pace. • I can customise the Thonny-py5mode environment to match my learning preferences. • Thonny-py5mode accommodates different learning styles and strategies. • Overall, Thonny-py5mode promotes a personalised learning experience.
Hedonic Motivation (HM)	• Thonny-py5mode allows me to express my creativity through coding. • Learning to code graphical output with Thonny-py5mode is fun and enjoyable. • Thonny-py5mode motivates me to experiment with different coding techniques and ideas. • Overall, coding graphics with Thonny-py5mode is a positive and enjoyable experience.
Effort Expectancy (EE)	• Thonny-py5mode is easy to install and set up. • Thonny-py5mode is easy to learn for beginners. • Thonny-py5mode makes it easy to debug code. • Overall, Thonny-py5mode is efficient and easy to use.
Task–Technology Fit (TTF)	• Thonny-py5mode makes it easy for me to learn new coding concepts. • Thonny-py5mode provides useful and efficient debugging tools. • Overall, Thonny-py5mode supported the enhancement of my coding skills.
Behavioural Intention (BI)	• I intend to continue using Thonny-py5mode for future projects. • I would recommend Thonny-py5mode to others learning to code. • I expect to use Thonny-py5mode frequently for regular graphics and interactive coding.

Table 4.2: Measurement items for the study

Measure	Item	Percentage	Frequent
Participant Campus	Melbourne	23.78%	34
	Sydney	20.28%	29
	Adelaide	10.49%	15
	Brisbane	09.09%	13
	Online	36.36%	52
ITP122 Gender	Male	63.64%	
	Female	36.36%	
ITP122 Age	18–22	49.65%	
	23–27	32.87%	
	28–32	12.59%	
	33+	04.90%	

Table 4.3: Study demographics

Thonny-py5mode coding environment. To empirically test the validity and reliability of the constructs presented in Figure 4.16, we conducted a Confirmatory Factor Analysis (CFA) in AMOS version 26 using the survey data. The goodness-of-fit (GOF) statistics assessed were the likelihood ratio chi-square (χ^2), the ratio of χ^2 to degrees of freedom (χ^2/df), the root mean square error of approximation (RMSEA), and the Comparative Fit Index (CFI). Table 4.4 reports the measurement model evaluation results.

We analysed convergent validity using the factor loading (FL) value and the composite reliability (CR) value. The general rule is that the FL and CR values should be at least 0.50, and preferably 0.70 or higher, with all FLs statistically significant. Following this rule, we dropped five items in the original conceptual model [90]. Other items with FLs ranged from 0.70 to 0.94, and constructs with CRs above 0.88, as shown in Table 4.4, indicating a high convergent validity.

Construct reliability testing consists of item reliability (IR) and construct reliability [90]. IR is determined by the squared multiple correlation value, with values above 0.50 indicating acceptable reliability. The IR values for all the items are higher than 0.50, indicating that these are sufficient for measuring the construct. We evaluated the construct reliability by calculating Cronbach's alpha (α) coefficient, with a value of 0.70 or higher considered acceptable [90]. Six constructs have high Cronbach's alpha coefficients, all above 0.85, indicating strong construct reliability.

Construct	Item	FL	IR	CR	χ^2/df	p	CFI	RMSEA	α
Task Characteristics (TC)	TC1	0.84***	0.71	0.93	1.67	0.14	0.98	0.06	0.91
	TC2	0.89***	0.79						
	TC3	0.91***	0.83						
Technology Characteristics (TCS)	TCS1	0.83***	0.70	0.90					0.90
	TCS2	0.79***	0.68						
	TCS3	0.85***	0.73						
Personalised Learning (PL)	PL1	0.77***	0.65	0.88					0.86
	PL2	0.83***	0.70						
	PL3	0.86***	0.74						
	PL4	0.86***	0.74						
Hedonic Motivation (HM)	HM1	0.77***	0.75	0.89					0.85
	HM2	0.73***	0.63						
	HM3	0.84***	0.71						
	HM4	0.81***	0.68						
Effort Expectancy (EE)	EE1	0.94***	0.88	0.94					0.92
	EE2	0.92***	0.82						
	EE3	0.88***	0.77						
	EE4	0.91***	0.81						
Task Technology Fit (TTF)	TTF1	0.79***	0.68	0.98					0.86
	TTF2	0.86***	0.74						
	TTF3	0.83***	0.70						
Behavioural Intention (BI)	BI1	0.84***	0.71	0.93					0.91
	BI2	0.94***	0.88						
	BI3	0.92***	0.85						
Recommended value		≥ 0.70	≥ 0.50	≥ 0.50	≤ 0.30	≥ 0.05	≥ 0.90	≤ 0.08	≥ 0.70

Note(s): ***$p \leq 0.001$, **$p \leq 0.01$, *$p \leq 0.05$

Table 4.4: Measurement model validation and reliability statistics

We assessed the GOF statistics for the final measurement model after conducting validity and reliability tests. The normalised chi-square value ($\chi^2/df = 1.67$) fell below the recommended cut-off of 3.00, indicating that the model did not significantly differ from the observed data. The RMSEA value (0.06) was below the 0.08 threshold, and the CFI value (0.98) exceeded the 0.90 threshold. Together, these results indicate a good fit between the model and the data, making it suitable for hypothesis testing.

Discriminant validity is supported when the square root of the AVE for each construct is greater than its correlations with other constructs [90]. Table 4.5 presents the correlation matrix for the seven constructs. All constructs demonstrated high discriminant validity, with AVE values ranging from 0.77 to 0.85. For example, the AVE for Effort Expectancy (0.85) exceeds its correlations with other constructs, which range from 0.22 to 0.77, indicating strong discriminant validity for this construct.

	TC	**TCs**	**PL**	**HM**	**EE**	**TTF**	**BI**
TC	**0.82**						
TCs	0.56	**0.79**					
PL	0.48	0.42	**0.84**				
HM	0.62	0.52	0.47	**0.82**			
EE	0.39	0.49	0.53	0.38	**0.85**		
TTF	0.51	0.44	0.41	0.36	0.28	**0.77**	
BI	0.45	0.38	0.49	0.42	0.41	0.22	**0.81**

Table 4.5: AVE and squared correlation matrix

We assessed the overall fit of the structural model using multiple fit indices to evaluate the proposed hypotheses. Table 4.6 summarises the results. The chi-square (χ^2) value normalised by the degrees of freedom (χ^2/df) was 2.34, below the recommended cut-off of 3. The GFI (0.92) and AGFI (0.93) exceeded the recommended threshold of 0.80, while the TLI (0.98) and CFI (1.12) were above the 0.90 threshold (note that AMOS occasionally reports CFI values slightly above 1.0 due to rounding). The RMSEA (0.07) was below the recommended maximum of 0.08. Overall, these results indicate that the structural model provides a good fit to the data [90].

Table 4.7 presents the hypothesis testing results from the structural model. The path coefficients (β) and significance levels (p-values) provide statistical support for eight hypotheses (H1, H2, H4, H5, H6, H7, H8, and H9), while there is insufficient support for H3.

Model Fit Indices	Recommended Value	Actual Value
χ^2/df	$1 < NC < 3$	2.34 ($p \leq 0.001$)
RMSEA	< 0.08	0.07
GFI	> 0.80	0.92
AGFI	> 0.80	0.93
TLI	> 0.90	0.98
CFI	> 0.90	1.12

Table 4.6: Research model's Goodness-of-Fit (GOF) indices

Hypothesis	Path			Coefficient (β)	SE	p	Result
H1	TC	→	HM	0.43	0.065	0.003	Supported
H2	TCs	→	EE	0.18	0.067	0.000	Supported
H3	PL	→	TC	0.02	0.072	0.170	Not Supported
H4	TC	→	TTF	0.56	0.055	0.000	Supported
H5	TCs	→	TTF	0.66	0.049	0.005	Supported
H6	PL	→	TTF	-0.33	0.061	0.003	Supported
H7	HM	→	TTF	0.26	0.054	0.000	Supported
H8	EE	→	TTF	0.41	0.060	0.000	Supported
H9	TTF	→	BI	0.33	0.057	0.000	Supported

Table 4.7: Structural model path coefficients and hypothesis testing results

Figure 4.18 presents the model from Figure 4.16 with the corresponding path coefficients, standard errors, and significance levels for each hypothesised relationship in the extended TTF model.

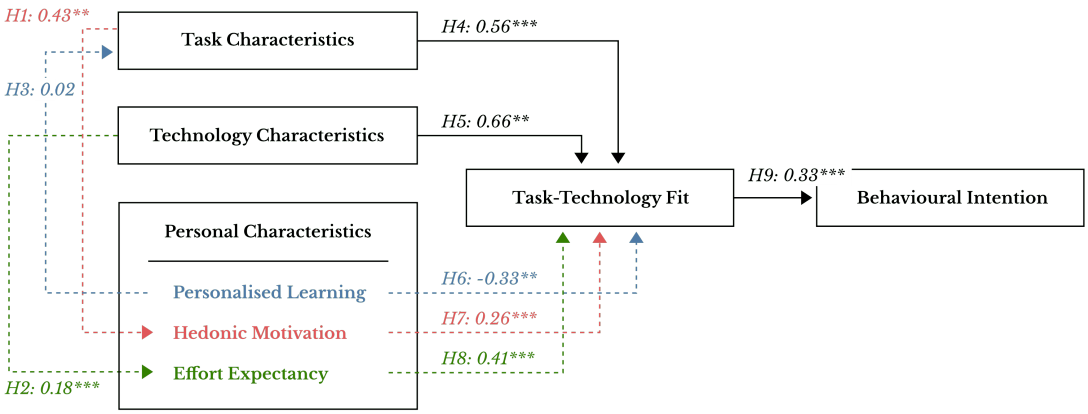


Figure 4.18: Extended TTF model with path estimates and significance. Constructed by the authors.

The hypothesis testing results indicated strong support for most proposed relationships in the extended TTF model. **H1** – Task characteristics positively influenced hedonic motivation in Thonny-py5mode-supported learning (β = 0.43, p = 0.003), indicating that engaging, well-structured tasks enhanced enjoyment and emotional engagement. **H2** – Technology characteristics positively influenced effort expectancy (β = 0.18, p < 0.001), suggesting that user-friendly features reduced the perceived effort required to use Thonny-py5mode. **H3** – Personalised learning did not significantly influence task characteristics (β = 0.02, p = 0.170), indicating that adaptive or individualised elements did not alter perceptions of the learning tasks. **H4** – Task characteristics positively influenced task–technology fit (β = 0.56, p < 0.001), showing that clarity and relevance shaped perceptions of tool–task alignment. **H5** – Technology characteristics positively influenced task–technology fit (β = 0.66, p = 0.005), underscoring the role of usability and functionality. **H6** – Personalised learning negatively influenced task–technology fit (β = -0.33, p = 0.003), suggesting that greater personalisation sometimes reduced perceived alignment. **H7** – Hedonic motivation positively influenced task–technology fit (β = 0.26, p < 0.001), affirming that emotional engagement improved perceptions of alignment. **H8** – Effort expectancy positively influenced task–technology fit (β = 0.41, p < 0.001), indicating that ease of use strengthened perceived fit. **H9** – Task–technology fit positively influenced behavioural intention to use Thonny-py5mode (β = 0.33, p < 0.001), showing that students were more likely to adopt the environment when it aligned with their learning needs.

4.3.8 Discussion

Our structural model (Figure 4.18) shows that task characteristics, technology characteristics, personalised learning, hedonic motivation, and effort expectancy all have direct, significant effects on the TTF of Thonny-py5mode. Moreover, this fit has a direct, significant effect on students' adoption of Thonny-py5mode for learning Python across TUA's four campuses in Australia and in its online cohort. However, personalised learning did not significantly affect technology characteristics.

H1 tested whether task characteristics influence hedonic motivation, demonstrating that the model supports H1 (β = 0.43, p = 0.003). This aligns with previous findings that well-structured, cognitively engaging learning tasks can significantly enhance learners' enjoyment. For instance, Škorić *et al.* (2021) found that aligning programming task demands with technology capabilities improves student satisfaction and continued use intentions [210]; similarly, Melzer (2019) noted that task design (an aspect of task characteristics) can strengthen motivation and engagement in information systems education [136]. In our context, Thonny-py5mode's support for step-by-step debugging, output visualisation, and structured logic-based graphical challenges appears to heighten students' emotional involvement, thereby raising hedonic motivation. These task effects also ripple into effort expectancy perceptions. When learners like using a tool because it structures tasks clearly and responds instantly, they tend to see the tool as less effortful. In this study, Thonny-py5mode matched the structure of students' tasks and supplied immediate feedback, increasing intrinsic motivation and helping create a more pleasant Python learning experience.

H2 examined the link between technology characteristics and effort expectancy, confirming that Thonny-py5mode significantly influenced the latter for students learning Python programming (β = 0.18, p = 0.007). Prior research finds that usability, clear feedback, and responsive system design reduce cognitive load and increase perceived ease of use. Wang and Kartika Sari (2024) reported that user-friendly interface features and visual feedback mechanisms enhance perceived ease of use in educational technology contexts [244]; Škorić *et al.* (2021) observed that web-based programming tools offering simplified navigation, real-time debugging, and platform accessibility support higher levels of perceived task–technology fit and reduce effort required [210]. Thonny-py5mode's intuitive layout, error reporting, and graphical output lower the cognitive and physical effort students expend during learning. This pattern suggests that effort expectancy is shaped by how well the system aligns with novices' cognitive needs. When friction drops, learners can focus on grasping Python concepts rather than wrestling with the environment.

H3 tested whether personalised learning shapes task characteristics. Contrary to some prior work, our model did not support H3 ($\beta = 0.02$, $p = 0.170$). Melzer (2019) reported that personalisation, when grounded in cognitive fit theory, can improve learners' perceptions of task clarity in information systems education [136]. However, our findings suggest that personalisation alone does not automatically yield clearer or better-structured tasks. To influence task characteristics, system designers likely need to embed personalisation into the logic and scaffolding of the tasks themselves, not only into interface-level features.

H4 concerned the impact of task characteristics on task–technology fit, demonstrating that the effect is significant and positive ($\beta = 0.56$, $p < 0.001$). Prior research confirms that when tasks are clearly defined and supported by interactive affordances such as real-time feedback and visualisation, students perceive a stronger TTF [121]. Yaakop *et al.* (2020) found that collaborative and well-structured tasks in cloud-based learning systems improved perceived alignment between tasks and technological tools [250]. Thonny-py5mode's syntax alerts, integrated visual feedback, and simplified debugging strengthen the perceived fit.

H5 tested whether technology characteristics directly influence task–technology fit, showing a strong positive relationship ($\beta = 0.66$, $p < 0.005$). Research from Thabet *et al.* (2024) shows that system quality elements—particularly intuitive interfaces, real-time visual output, and error-handling assistance—improve perceptions of how well a technology matches task demands [227]; similarly, Wang and Kartika Sari (2024) observed that platform responsiveness and adaptability enhance user engagement and strengthen TTF perceptions [244]. In this study, the approachable Thonny-py5mode interface, convenient install process, graphical output, and shell panel support closely match the requirements of creative coding tasks. Through this alignment, students can focus on problem-solving without hindrance from the environment, thereby reinforcing perceptions of strong TTF.

H6 indicates that personalised learning had a significant negative effect on task–technology fit ($\beta = -0.33$, $p = 0.003$), suggesting that greater personalisation was associated with lower perceived alignment between Thonny-py5mode and the programming tasks. This aligns with H3's finding that personalisation did not significantly improve task characteristics, implying limited integration with the task structure. In contrast, Melzer (2019) found that tailoring environments to individual preferences can strengthen perceived fit [136]; Ishaq *et al.* (2024) reported that adaptive difficulty, personalised feedback, and real-time code visualisation can enhance alignment in programming contexts [99]. One possible explanation is that aspects of Thonny-py5mode that could support personalisation, such as

immediate visual feedback and the creative flexibility of py5 commands and graphical output, were not tailored to individual learners or sufficiently integrated into the assessment scaffolding. This may have made them seem loosely connected to the programming tasks and fostered a perception of mismatch rather than synergy.

H7 addresses hedonic motivation's role in task–technology fit, indicating a significant and positive relationship ($\beta = 0.26$, $p < 0.001$). Škorić *et al.* (2021) found that when students perceive interactive programming environments as enjoyable, their assessment of TTF improves [210]; Ishaq *et al.* (2024) similarly reported that adaptive difficulty, personalised feedback, and real-time code visualisation can heighten both enjoyment and perceived alignment in programming contexts [99]. Thonny-py5mode's graphics-oriented output provides a form of immediate visual feedback (alongside shell output), and its simplified interface likely make programming less intimidating for Python novices.

H8 tested the effect of effort expectancy on task–technology fit, showing a significant and positive relationship ($\beta = 0.41$, $p < 0.001$). Prior research supports this link between perceived ease of use and stronger perceptions of task–technology alignment. Lim and Lee (2021) found that intuitive interfaces and reduced complexity enhance users' sense of fit with learning tasks [119]; Thabet *et al.* (2024) reported that clear navigation structures, immediate feedback, and low cognitive load significantly improve the perceived match between system capabilities and intended tasks [227]. Thonny-py5mode's clean layout, integrated debugging, and graphical output likely reduced effort, reinforcing perceptions of fit with the graphical programming tasks.

H9 links task–technology fit to behavioural intention; the effect is significant and positive ($\beta = 0.33$, $p < 0.001$). DeLone and McLean's (2003) IS success model emphasises the role of system–task alignment in driving continued use [55]; Škorić *et al.* (2021) confirmed that when students perceive web-based programming tools as fitting well with learning activities, they show stronger continuance intentions [210].

Taken together, these results highlight an integrated picture of adoption. Task–technology fit is a central driver of behavioural intention, shaped by clear, well-scaffolded tasks (H1, H4), strong technology characteristics (H2, H5), and user-centric elements that make the experience both enjoyable (H7) and easy (H8). In Thonny-py5mode, features such as syntax alerts, intuitive debugging, and integrated graphical output help align the environment closely with the demands of graphical Python tasks. Personalised learning, however, shows a more complex pattern: it does not significantly alter task characteristics (H3) and, in this study, was associated with lower TTF (H6). This suggests that

when adaptive elements, such as immediate visual feedback or the creative possibilities offered by py5 commands, are not tightly embedded in task scaffolding, they may reduce rather than enhance perceived alignment.

For practice, three implications stand out: (1) design tasks that are structured, cognitively engaging, and tightly mapped to learning outcomes, while ensuring the environment provides immediate, meaningful feedback; (2) adopt environments with novice-friendly interfaces, visual feedback, and streamlined debugging features that reliably lower effort, raise enjoyment, and strengthen TTF; and (3) treat personalisation with care: ensure that task scaffolding and learning pathways closely integrate with adaptive pacing and feedback, rather than layering these superficially at the interface level.

In the specific case of Thonny-py5mode, the convergence of usability, visual immediacy, and supportive debugging appears to deliver a strong perceived fit to core Python programming tasks. Students who experienced this fit reported greater enjoyment, lower perceived effort, and stronger intentions to continue using the environment. The overall pattern suggests that Processing-like, graphically-oriented Python environments such as Thonny-py5mode can meaningfully enhance early programming experiences when they balance clear task design, robust interactive affordances, and thoughtfully embedded personalisation.

4.3.9 Research Implications

The study offers both theoretical and practical contributions for the use of Thonny-py5mode. Theoretically, it extends TTF theory by incorporating three personal characteristics: personalised learning, hedonic motivation, and effort expectancy. While traditional TTF frameworks focus on aligning task needs and technology characteristics [69], our model posits that learners' experiences and motivations also mediate this relationship. Testing hypotheses 1 through 9, we confirm that perceived fit depends on both system functionality and users' enjoyment, learning needs, and intention to use the system, echoing prior work demonstrating the joint influence of system quality and user factors on TTF [211, 227], thereby expanding TTF to include emotional and motivational dimensions alongside functional alignment. The insignificant effect between personalised learning and technology characteristics further suggests that adaptive learning systems must consider not only the quality of system features but also the user's context and capability, consistent with insights from Melzer (2019) on the role of cognitive fit in personalised learning design [136]. Design implications resonate with findings that effective curriculum transformation requires embedding authentic uses of technology in programme design

[182]. That sustained self-efficacy support through hands-on, scaffolded training enables educators to adopt more constructive, knowledge-building uses of ICT in classrooms [139].

Practically, the findings suggest selecting or designing coding environments like Thonny-py5mode for their ability to support structured learning tasks while remaining stimulating and user-friendly. In Thonny-py5mode, features such as graphical output via py5 sketches, seamless integration between the sketch window and Thonny panels, and a simplified, non-command-line interface that avoids package management reduce cognitive load and boost satisfaction [119, 227]. These elements align closely with our results on task–technology fit, as the environment's layout and immediacy in generating graphics help beginners stay oriented and engaged. The strong link between hedonic motivation and TTF is illustrated in Thonny-py5mode's ability to make coding sessions enjoyable through real-time, graphical results, encouraging sustained use [210, 250]. For educators, these findings can inform the adaptation of existing IDEs or the design of new ones tailored to specific Python multimedia libraries, without sacrificing ease of use. By pairing beginner-friendly tools with scaffolded coding tasks and challenges, educators can help novices build confidence and skills. Finally, the significant influence of TTF on behavioural intention shows that when students feel Thonny-py5mode fits their needs, they are more inclined to continue using it; this underscores the importance of aligning the tool's features with both pedagogical goals and learner characteristics [55].

4.3.10 Conclusion

This study employed an extended Task–Technology Fit framework to investigate factors influencing university students' intention to use Thonny-py5mode for learning Python. The findings demonstrate that task characteristics, technology characteristics, hedonic motivation, effort expectancy, and personalised learning all contributed to perceived fit, with TTF subsequently driving behavioural intention. While personalisation did not significantly affect task characteristics and was, in this case, associated with lower TTF, other factors proved more influential. The integration of structured tasks, a user-friendly design, and an engaging graphical approach fostered strong alignment between Thonny-py5mode and learners' needs.

For practice, our findings underscore the value of beginner-friendly IDEs that integrate graphical output and beginner-friendly design, complemented with scaffolded tasks to reduce cognitive load and sustain engagement. We demonstrated this through extending TTF theory, integrating the role of motivational constructs alongside functional alignment in predicting technology adoption for

programming education. In Thonny-py5mode, a balance of usability and visually fostered enjoyment, lowered effort, and encouraged continued use. Future research should explore how to embed personalisation in task scaffolding more tightly, and how contextual factors such as prior programming experience and educator support influence adoption outcomes.

4.4 Chapter Summary

In summary, this chapter presented the three publications that form the scholarly core of the thesis. Together, they trace the research journey from curriculum design with Processing.py, through the development of Thonny-py5mode, to its empirical evaluation in a university setting. These works complement the Literature Review and Software chapters, while connecting to the dissemination efforts documented in Presentation Outputs and Creative Works. Collectively, they demonstrate the originality, significance, and scholarly validation of the research, achieved through peer reviewed rigour.

The following chapter, Presentation Outputs, continues the folio narrative, documenting a series of prominent academic and professional conference presentations that laid the groundwork for the three publications.

5 Presentation Outputs

This chapter documents a series of presentation outputs delivered at prominent academic and professional conferences. Together with the outputs in Chapters 4 (Publications) and 6 (Creative Works), they constitute this PhD folio.

These presentations address the central themes of this thesis: Python tools for visual learning contexts, associated educational materials, and creative coding techniques. Each event offered a valuable opportunity to support the Broader Dissemination of the research findings, share innovative approaches, and engage with both academic and creative computing communities.

Figure 5.1 presents a timeline illustrating how the preparation and delivery of each presentation contributed iteratively to the development of the PhD research. Notably, it excludes the final presentation output, documented in Section 5.10, which explores the application of Thonny-py5mode in mitigating the misuse of generative AI in introductory Python courses. Kiwi PyCon 2025 has accepted a proposal for the conference, scheduled to take place after the anticipated submission of this thesis for examination. It would otherwise appear as a distant outlier on the Figure 5.1 timeline, visually distorting the chart. Hence its omission. Nevertheless, it represents a significant, albeit late, strand of enquiry within the PhD.

This chapter organises the outputs chronologically (ending with most recent), with each section structured around the following points to highlight the presentation's relevance, contributions, and integration into the overarching research narrative:

1. **Presentation Abstract**: A presentation summary describing the topic and highlighting aspects such as the session's content focus, target audience, and key objectives. Generally, a short abstract quoted directly or appropriately adapted from the conference programme or proposal.

2. **Conference/Event Overview**: A brief description of the conference, including its focus and audience, revealing its relevance to Python creative computing and/or education.

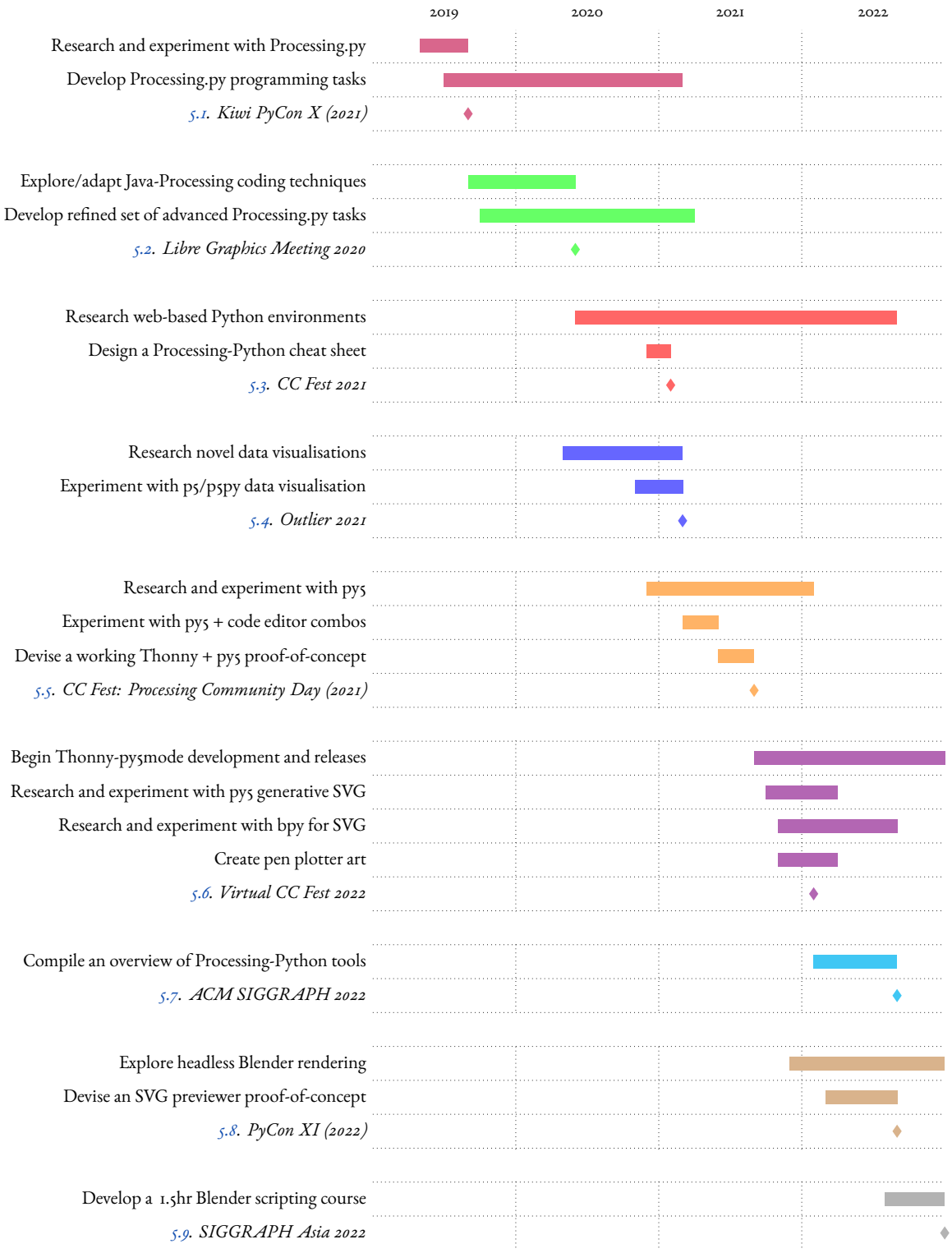


Figure 5.1: Chronological overview of presentation outputs contributing to this PhD study's research, software development, and folio

3. **Content & Method**: A detailed description of the presentation's content, including the educational tools, techniques, and creative computing environments discussed. Where relevant, this describes any methods, experimentation, and research for developing the session content.

4. **Audience Engagement & Feedback**: A brief account of the audience response, including any relevant questions, comments, and constructive feedback received during and after the presentation.

5. **Reflections & Implications**: An analysis of how the presentation research and preparation, and subsequent audience feedback, informed the direction and progression of the PhD study.

As the presentations varied in scope, impact, and audience engagement, the length and depth of documentation for each likewise differ.

It is pertinent to add that these conferences and events invite speakers through a robust selection process, which involves consideration by a panel of expert reviewers, ensuring that the selected presentations are of high quality and contribute meaningfully to the field. Each proposal is subject to evaluation criteria, including originality, relevance, potential impact, and novelty/innovation. This rigour underscores the significance and credibility of the outputs documented within this chapter.

The preparatory work for each presentation entailed research into extant literature. As with the Publications and Creative Works chapters, the content of this chapter further supplements that of the Literature Review. The reader will find citations in the respective *Content & Method* and *Reflections & Implications* of each section.

5.1 Processing.py—Creative Coding with Python

Conference/Event Name: Kiwi PyCon X (2019)

Author(s): Bunn, T.

Venue: Wellington, New Zealand: Rutherford House

Start/Finish Dates: 23 August 2019 – 25 August 2019

URL: https://kiwipycon.nz

Acceptance Letter/Invitation or Event Programme: See Appendix D.7

5.1.1 Conference/Event Overview

Kiwi PyCon is an annual conference dedicated to Python programming and its applications, serving as New Zealand's national Python conference. Organised by the New Zealand Python User Group (NZPUG), it is part of a global network of PyCon[1] conferences aimed at facilitating collaboration and knowledge exchange among Python users worldwide.

The event provides a space for Python enthusiasts, developers, educators, researchers, and industry professionals to share ideas and discuss emerging trends in Python programming. While primarily focused on the New Zealand Python community, Kiwi PyCon also draws international speakers and participants, offering local and global perspectives.

The conference features a variety of topics, including software development, data science, machine learning, web development, programming education, and creative computing. As part of the PyCon movement, Kiwi PyCon reflects a shared commitment to supporting and growing the Python community.

5.1.2 Presentation Abstract

Processing is a programming language and integrated development environment that caters to the electronic arts, new media art, and visual design communities. Initially released in 2001, Processing featured a Java-based syntax, but 2014 saw the release of an additional Python mode: Processing.py.

This session begins with some context, namely the creative coding scene, then moves into a series of programming tasks using Processing.py within the Processing IDE, investigating topics like graphics,

[1] https://en.wikipedia.org/wiki/Python_Conference

randomness, noise, animation, and interactivity. All this should make for a fresh and inspiring session; something that will appeal to novices and creatives but also offers something for experts.

5.1.3 Content & Method

The session focused on Processing's Python mode (Processing.py) as a tool to provide designers, artists, and aspiring coders with an accessible and visual way to learn Python programming, highlighting inspiring projects and examining an early iteration of a programming fundamentals curriculum built around Processing.py. It introduced the creative coding landscape beginning at the *demoscene*, a subculture originating in the 1980s focused on creating real-time multimedia demos to showcase programming and artistic skills [165] (Figure 5.2).

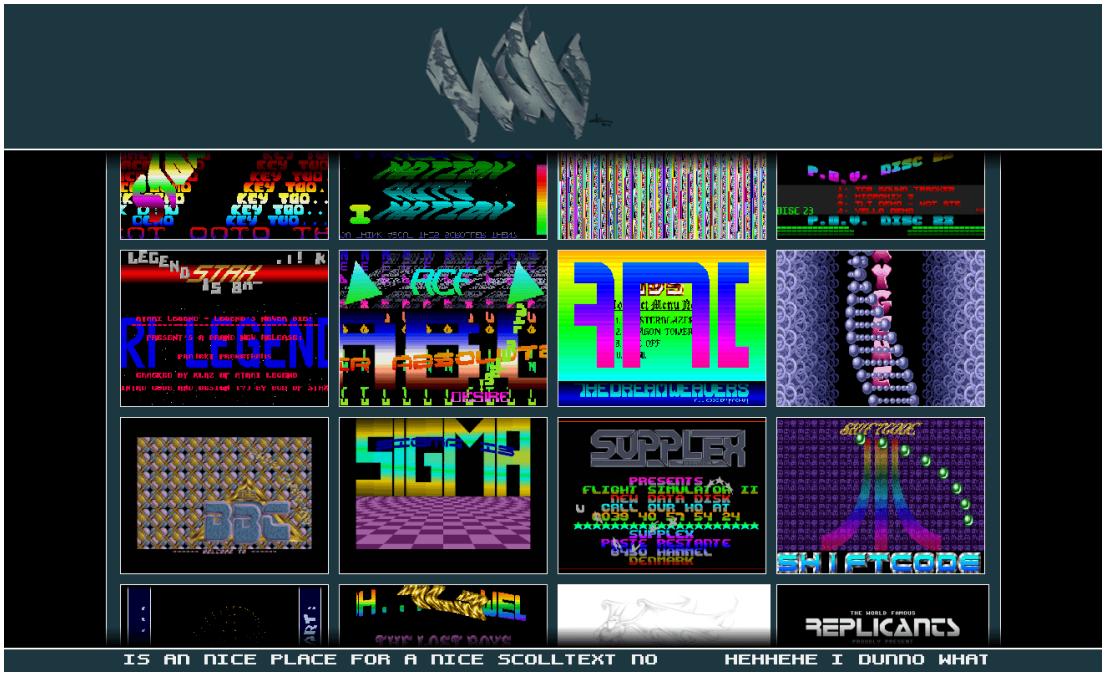


Figure 5.2: *WAB.com (We Are Back)* is a CODEF-powered platform featuring remakes of cracktros and other retro demoscene productions. Screenshot from the website of Antoine Santo. Source: [198].

The session was initially planned for a 45-minute timeslot and then adapted to a 2-hour workshop at the request of the event organisers. It covered four major areas, with point 4 (*Programming Tasks*) consuming around half the allocated slot:

1. **An Introduction to Creative Coding**: A concise historical overview of the demoscene, highlighting its influence on the evolution of creative coding. This included discussions on Flash, which supported a smaller yet comparable creative scene in the late 1990s, focused on developing

interactive and animated web-based "demos" using Macromedia Flash [101]. Additionally, the introduction featured a curated selection of inspiring works created with Processing and Python, showcasing the distinctive opportunities offered by Processing.py.

2. **Processing**: Exploring the history of the software, this section highlighted its impact as a transformative tool for creative coding [46]; it also examined alternative environments to Processing along with concise code examples that illustrated the syntax and capabilities of Processing.py, providing a practical glimpse into its creative potential.

3. **A Proposed Processing.py-Based Programming Fundamentals Curriculum**: Outlining the early iterations of a course designed for students with little to no prior programming experience, tailored to those in creative or related fields of study. This draft curriculum suggested topics and tasks aimed at guiding students in developing foundational programming skills through engaging and practical creative coding exercises.

4. **Programming Tasks**: Building on the previous point, this largest segment delved into a variety of practical coding examples designed to create and manipulate visual elements, utilise randomness and noise for procedurally generated graphics, and incorporate animation and interactivity. These tasks aimed at developing an understanding of the Processing paradigm by applying its Python Mode to both creative and practical contexts, bridging theoretical concepts with hands-on exploration.

The research that supported the development of the session content integrates into the Literature Review chapter and is further reflected in the outputs of the Publications and Creative Works chapters.

5.1.4 Audience Engagement & Feedback

The session, attended by 26 people, attracted a significant number of Visual Effects (VFX) professionals, largely due to the prominence of Wētā FX[2] in Wellington, New Zealand. The workshop format offered ample opportunities for discussion, and more than a traditional talk or lecture might typically allow.

The historical context of the *demoscene* and Flash struck a chord with many attendees, evoking nostalgic memories of the creative pursuits that initially sparked their journeys into the industry.

[2] Wētā FX is a world-renowned visual effects company headquartered in Wellington, NZ (https://www.wetafx.co.nz)

The discussion of randomness, noise, and algorithms for simulations resonated as well, reflecting foundational techniques in VFX workflows, albeit presented in a more simplified and stripped-back form aimed at educational contexts.

While it was evident that Processing.py lacks the scalability, power, and integration capabilities required for complex, industry-standard VFX scripting tools, some attendees commented that it might harbour potential as a resource for rapid prototyping and visualising mathematical concepts. This lightweight approach could offer VFX professionals a quick and creative means to explore ideas and techniques relevant to their work, providing a bridge between artistic experimentation and technical execution.

5.1.5 Reflections & Implications

While I, the PhD candidate, had been active in related domains for several years prior, the inquiry into the specific combination of Python and Processing begins here. This presentation effectively marks the starting point of the PhD research journey, shaping the direction it would ultimately take: the exploration and development of new Python tools for visual learning contexts.

The session's *Programming Tasks* content would lay the early foundations for publishing the book *Learn Python Visually: Creative Coding with Processing.py* (No Starch Press).

5.2 Processing Python Mode for Creative Coding and Teaching

Conference/Event Name: Libre Graphics Meeting 2020

Author(s): Bunn, T.

Venue: Rennes, France (moved online for COVID-19)

Start/Finish Dates: 27 May 2020 – 29 May 2020

URL: https://libregraphicsmeeting.org/2020/en/program.html

Acceptance Letter/Invitation or Event Programme: See Appendix D.8

5.2.1 Conference/Event Overview

The Libre Graphics Meeting (LGM) is an annual international symposium that brings together developers, researchers, artists, and designers to advance and apply open-source software in creative and technical fields. This interdisciplinary event serves as a platform for sharing innovative research, exploring methodologies, and fostering collaboration across areas such as graphic design, animation, 3D modelling, typography, and computational art.

The programme includes academic presentations, technical workshops, and collaborative sessions, emphasising the role of open-source tools in expanding access to creative expression, promoting sustainable design, and democratising technology. By centring on community-driven innovation and open knowledge, LGM critically examines and advances the intersections of technology, creativity, and collaboration.

5.2.2 Presentation Abstract

Processing Python Mode (also known as Processing.py) provides designers, artists, and aspiring coders with an accessible and visual way to learn Python programming. This presentation introduces the software, creative coding scene, inspiring projects, and a programming fundamentals curriculum based on Processing.py.

5.2.3 Content & Method

This presentation represented a significant evolution of the PyCon X session (see Section 5.1), expanding upon its foundational concepts to deliver a more in-depth exploration of creative coding with Processing.py. The refined scope incorporated a broader range of topics, deliberately moving away

from practical, code-along activities to examine advanced techniques at a higher level and the technical dimensions of Processing.py. Unlike its predecessor, this session adopted a 50-minute talk format, rather than a workshop format.

This provided an opportunity to present further curriculum development and ongoing research, including exploring (hitherto undocumented) Processing.py-compatible ways to construct user interfaces and implement 2D physics. These solutions incorporated ControlP5 for GUI elements and pypybox2d for physics simulation, both of which can operate within Processing.py's Jython limitations [116, 201].

The topics encompassed an exploration of more advanced computational art techniques, including animated trigonometry and matrix-based geometric transformations, generative design through algorithmic randomness, bespoke data visualisation techniques, vertex-based curve generation, and kernel-based image processing methods. Figure 5.3 displays a selection of code samples to accompany the presentation, provided for attendees as a resource archived at https://github.com/tabreturn/processing.py-tabreturn.github.io.

The research that supported the development of the session content integrates into the Literature Review chapter and is further reflected in the outputs presented in the Publications and Creative Works chapters.

5.2.4 Audience Engagement & Feedback

Like many similar events during the early stages of the COVID-19 pandemic, the Libre Graphics Meeting 2020 had abruptly transitioned to an online format. The sudden shift to virtual conferencing likely resulted in limited audience engagement and interaction, perhaps attributable to participants' and organisers' unfamiliarity with digital conference platforms and the new challenges of remote collaborative video experiences. The presentation feedback, albeit scant, was encouraging.

5.2.5 Reflections & Implications

The repository of code samples produced in the lead-up to and shared at the event marked significant progress in developing novel creative coding techniques for Processing.py, contributing to new learning approaches, many of which were subsequently refined and integrated into the comprehensive text, *Learn Python Visually: Creative Coding with Processing.py* (No Starch Press).

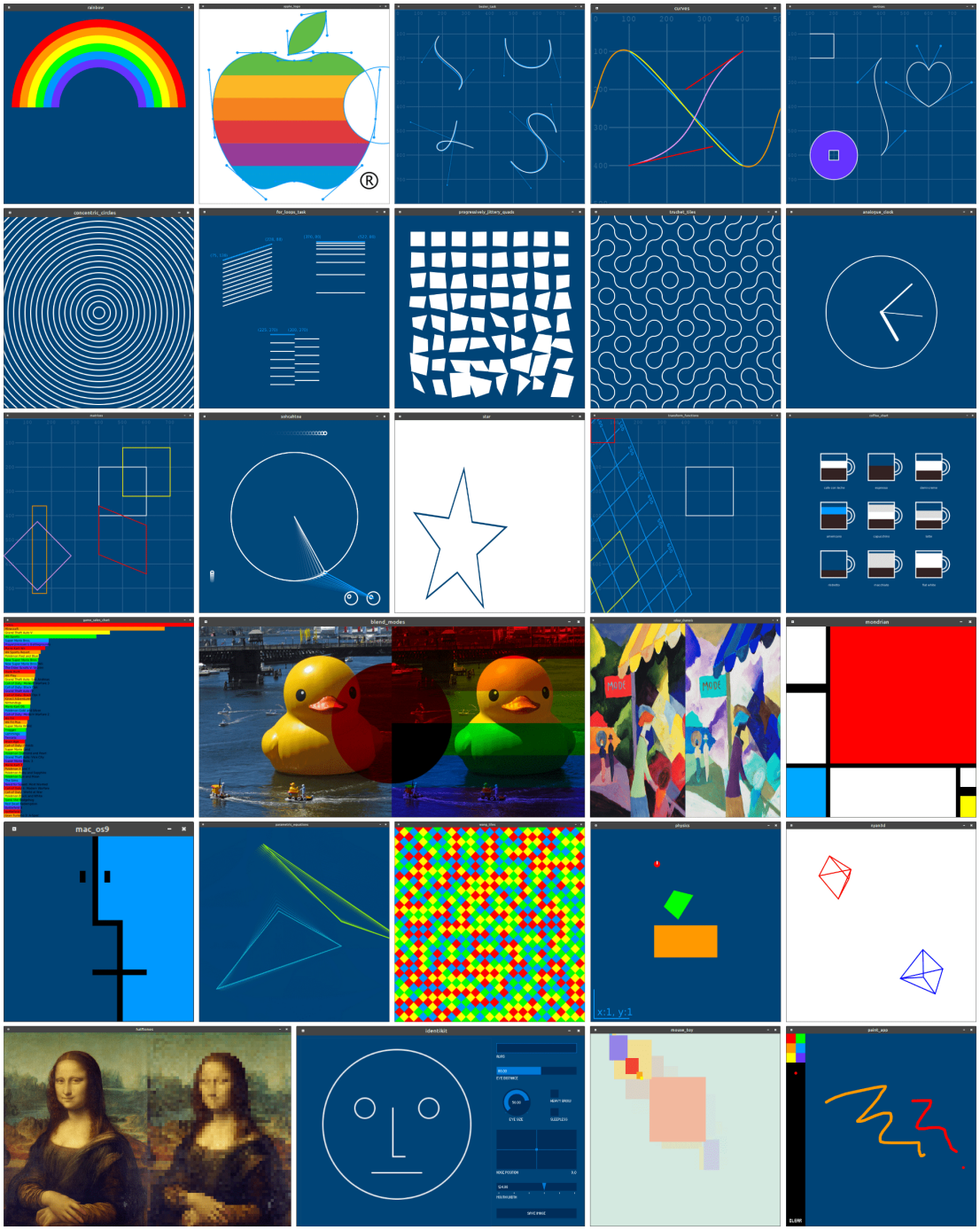


Figure 5.3: A selection of code samples prepared by the author for the Libre Graphics Meeting 2020 presentation. Images generated the author. Source: [222].

5.3 Creative Coding with Python & Processing

Conference/Event Name: CC Fest 2021

Author(s): Bunn, T.

Venue: New York, USA: NYU Tisch (online)

Start/Finish Dates: 24 January 2021

URL: https://ccfest.rocks

Acceptance Letter/Invitation or Event Programme: See Appendix D.3

5.3.1 Conference/Event Overview

CC Fest (Creative Coding Fest) is a free, inclusive event that celebrates creative coding in all its forms. It brings together a diverse community of participants, from students and artists to hobbyists, educators, technologists, and anyone curious about the field. Alongside talks, the event offers opportunities to engage in hands-on activities such as crafting digital art, creating animations and games, exploring AI, working with hardware, and more.

CC Fest has been held in various locations, including New York, Los Angeles, and San Francisco, and has offered both in-person and virtual experiences.

5.3.2 Presentation Abstract

This talk introduces creative coding techniques using a new, browser-based creative coding environment: Computiful. Computiful enables users to program interactive digital art for the Web using the Python programming language. The session includes a review of different Python programming environments for creative coding and their history and development.

5.3.3 Content & Method

Computiful, developed by Nick McIntyre, was an online code editor based on the p5.js web editor[3] adapted to run Python code. It used the Brython[4] interpreter to execute Python directly in a web browser by translating it into JavaScript, allowing seamless integration with p5.js.

[3] https://editor.p5js.org

[4] https://brython.info

A key modification in Computiful was the reorientation of the p5.js/Processing coordinate system to align with the top-right quadrant of the Cartesian plane ("Quadrant I"). This adjustment, aimed at improving intuition for math students, makes y-values increase upwards (from the bottom) rather than downwards (from the top)—unlike traditional Processing environments where y-values increase downward (from the top). These changes, along with other refinements, supported the teaching and learning of mathematical concepts through coding. Computiful has since transformed into the Strive Editor (Figure 5.4).

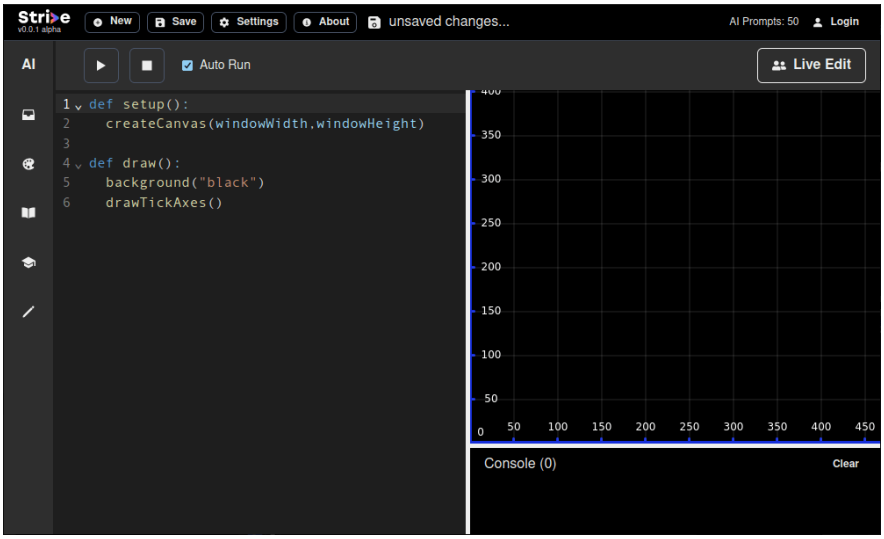


Figure 5.4: The Strive Editor uses a coordinate system where origin (0, 0) is positioned at the bottom left of the display pane (rather than top left); y-values increase *upwards*, and x-values increase to the right. Screenshot by the author. Source: [219].

Strive, an EdTech company, leverages Processing-based environments to teach mathematics, science, and other subjects through writing code [219]. Nick McIntyre, Tamir Shklaz, and Maxim Schoemaker maintain the project source code on GitHub at https://github.com/StriveMath/p5-python-web.

The talk provided an overview of the Processing ecosystem, explaining its variants, leading into web-based implementations and a Computiful introduction. The demonstration section of the session reviewed building an amoeba simulation, covering key concepts like object-oriented programming using Computiful. A GitHub repository link for all participants provided access to the complete code samples and other resources, including a cheatsheet (designed as part of the session's preparation) for those transitioning from Java/JavaScript to Python. These materials can be found archived at https://github.com/tabreturn/cc-fest-computiful.

5.3.4 Audience Engagement & Feedback

The session concluded with an invitation for the audience to explore the topic further and ask questions. However, most feedback occurred at the beginning of the talk when the speaker asked participants several questions to gauge their familiarity with different tools. Of around 60 attendees: approximately six indicated (by raising a virtual hand) that they had used the original Java-based Processing; around ten had experience with p5.js; and a larger portion (about 15) had some background in Python. However, this quick survey aimed to tailor the talk's content to the audience's experience rather than to gather quantitative data.

5.3.5 Reflections & Implications

Python code typically cannot run directly in a web browser. However, Computiful enabled the execution of Python sketches by leveraging a combination of p5.js and Brython. Later, the Strive Editor replaced Brython with Skulpt. Within the context of this PhD, these developments initiated research into potential web-based solutions aimed at Processing-like Python learning environments.

The traditional Python interpreter, or *reference* implementation (commonly known as CPython), runs like traditional computer software—i.e., on systems where users must manually download, install, and configure it [176]. In contrast, both Brython and Skulpt allow Python code to execute directly in web browsers, providing several advantages typical of browser-based solutions [79, 180]. These include eliminating the need for setup or installation, ensuring cross-platform compatibility, enabling more instant/beginner-friendly access, and facilitating seamless integration with online (learning?) platforms and tools [14]. However, Brython and Skulpt come with certain disadvantages when compared to CPython, such as:

- **Performance Overhead**: Both Brython and Skulpt are slower than standard Python because they must convert Python code into JavaScript for execution; this introduces additional performance overhead [179]. However, some experimentation demonstrated that this limitation is not a significant concern for beginner-level- and other projects that are not computationally intensive.
- **Limited Library Support**: Neither Brython nor Skulpt fully supports the Python standard library, and particularly libraries depending on C extensions, such as NumPy, pandas, or TensorFlow. Additionally, these environments cannot execute Python code that relies on native system-level libraries or features [79, 180].

- **Browser Restrictions**: As Brython and Skulpt run within a web browser, their use is restricted to web-based applications; this limits their ability to access system-level resources or hardware features (with which CPython can interact), such as file system access, network operations, or local hardware interfacing [158].
- **Debugging Challenges**: Brython and Skulpt can make debugging more challenging than with standard Python. Transpiling Python to JavaScript adds an extra layer of complexity, making it harder to track and resolve errors. However, one can argue that something like Processing.py presents relatable issues using Jython to compile Python code to Java bytecode [103].

The research that supported the development of the presentation and the subsequent insights into browser-based solutions, such as Computiful, Brython, and Skulpt, contributed considerably to the publication of the MTAP (Q1-ranked) journal article, *Towards a Python 3 Processing IDE for Teaching Creative Programming*.

5.4 Novel Visualisations with Python and p5

Conference/Event Name: Outlier 2021

Author(s): Bunn, T.

Venue: Online

Start/Finish Dates: 04 February 2021 – 07 February 2021

URL: https://www.outlierconf.com

Acceptance Letter/Invitation or Event Programme: See Appendix D.5

5.4.1 Conference/Event Overview

Outlier is a data visualisation conference organised by the Data Visualization Society (DVS) to provide a platform for the 'data viz' community to exchange knowledge and ideas. The conference emphasises advancing technical practices and critically examining the norms of data visualisation to explore new possibilities within the discipline. It brings together a diverse group of professionals, including designers, academics, and business intelligence developers, who contribute to the field through varied approaches and techniques. In addition to organising the conference, the DVS publishes *Nightingale*, a journal dedicated to sharing high-quality articles and ideas that advance knowledge and practice.

Outlier 2021, initially planned as an in-person event but moved online due to COVID-19, attracted nearly 1000 participants worldwide [102].

5.4.2 Presentation Abstract

Python data visualisation libraries—like Matplotlib, Bokeh, Plotly, and others—provide a broad range of chart types. But what if you're looking to visualise data in more unique or creative ways? This is a brief overview of p5: a high-level drawing library to help you quickly create novel visualisations, simulations, and interactive art using Python.

5.4.3 Content & Method

This lightning talk explored innovative approaches to creating unique visualisations using Python and p5 (also known as p5py), a library that brings the Processing programming paradigm to *native Python* environments [157]. Unlike tools such as Computiful and Processing.py, which rely on mechanisms to

convert Python code for execution in JavaScript or Java, p5 operates directly within Python, ensuring seamless integration with the Python ecosystem.

Traditional Python data visualisation libraries like Matplotlib, Seaborn, and Plotly have proven highly effective for producing standard plots and charts [134]. However, crafting more unique, expressive, and creative visualisations can require alternative approaches [26, 67]. Notable examples include Frederick Brodbeck's *Cinemetrics*[5]—a project that visualises films as colour-coded "fingerprints"—and 3Blue1Brown's compelling COVID-19 spread simulations[6] using dynamic graphics to communicate complex concepts in an accessible, engaging way. Such projects highlight the power of visualisation that transcends conventional charting methods to tell rich, nuanced stories.

Combining Processing's intuitive creative coding model with Python's versatility and readability, p5 can empower users to develop generative data visualisations and real-time simulations, with features to explore data interactively. This fusion opens possibilities for artists and scientists alike, enabling the creation of dynamic and bespoke visualisations that can inspire and inform audiences beyond traditional data representation techniques.

The talk showcased p5's capabilities through several bespoke examples, including visualising the frequency of letters in a given passage of text (Figure 5.5), illustrating p5's potential as a versatile alternative to traditional Python plotting libraries.

5.4.4 Audience Engagement & Feedback

Outlier's *Lightning Talk* format does not accommodate engagement during the session time.

5.4.5 Reflections & Implications

The presentation underscored the potential of Processing-inspired environments, such as p5, to enable novel approaches to data visualisation. Several experimental p5 code samples developed for the talk demonstrated that this paradigm applies well to crafting unique, dynamic visualisations. Moreover, it presents opportunities to employ data-driven techniques in creative coding, providing ways for learners and practitioners to explore various data file formats (including CSV, JSON, and XML) [133].

[5] http://cinemetrics.fredericbrodbeck.de/old.html

[6] https://www.youtube.com/watch?v=gxAaO2rsdIs

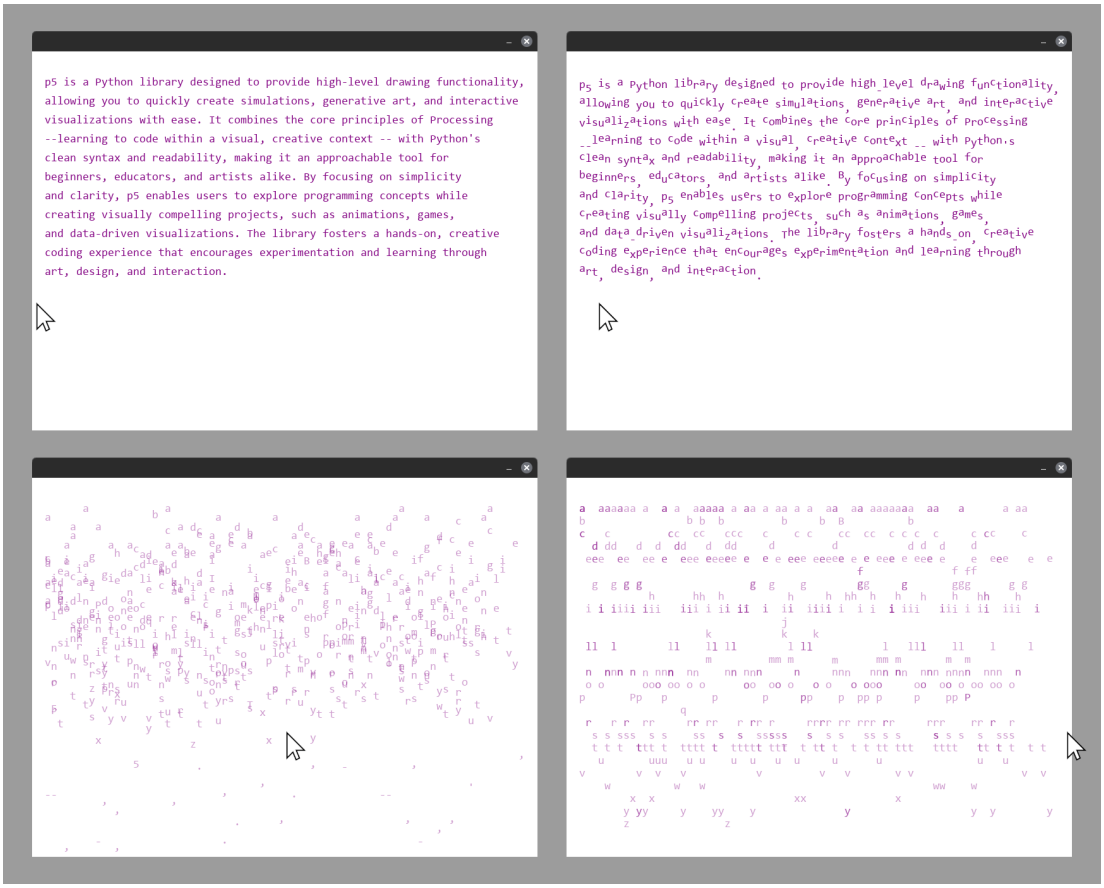


Figure 5.5: An interactive p5.js sketch by the author, presented at Outlier 2021, designed to visualise the frequency of letters in a given passage of text. As the mouse pointer moves from left to right, the characters transition from continuous text to grouped lines. Adapted from an example by Benedikt Groß *et al.* in *Generative Gestaltung*. Source: [85].

Preparing the talk and accompanying code samples provided valuable insights into the strengths and trade-offs of native Python libraries compared to their (Brython, Jython, Skulpt) alternatives. Notably, as a native library, p5 benefits from its:

- **Seamless Integration with CPython**: Offering access to a full range of features expected in native Python, including support for widely used, high-performance libraries—particularly those with C extensions.
- **Direct Execution of Python Code**: Eliminating the need for translation layers required by alternatives like Processing.py or Computiful, thereby reducing complexity and potential performance bottlenecks [199].

- **Potential for Broader Adoption**: Driven by its integration into standard package management tools, and convenient availability to CPython's existing user base, which includes educators, researchers, and practitioners [40].

However, p5 also present challenges, specifically:

- **Lacking Adoption and Stability**: As p5 is still evolving, it has yet to reach a level of development and adoption comparable to Processing and p5.js, which are highly mature environments with extensive feature sets, stable performance, active developer communities, and significant user followings [157].
- **Setup Complexity**: Python-based Processing tools, such as p5, often require users to install additional dependencies (e.g., GLFW) and manage virtual environments for package installation; this stands in contrast to the streamlined, 'turnkey' experience Processing.py and Strive offer, which provide immediate usability with minimal configuration, enabling users to focus on creative coding with minimal setup process.

The research that supported the development of this presentation and the resulting insights into native Python Processing-inspired solutions, such as p5, DrawBot, and Shoebot, contributed considerably to the publication of the MTAP (Q1-ranked) journal article, *Towards a Python 3 Processing IDE for Teaching Creative Programming*.

5.5 Thonny + py5: A Python 3 Environment for Processing

Conference/Event Name: CC Fest: Processing Community Day (2021)

Author(s): Bunn, T.

Venue: Online

Start/Finish Dates: 22 August 2021

URL: https://processingfoundation.org/advocacy/pcd-2021

Acceptance Letter/Invitation or Event Programme: See Appendix D.9

5.5.1 Conference/Event Overview

A special edition of CC Fest held to coincide with *Processing Community Day (PCD) 2021*, marking the 20th anniversary of Processing. This celebration highlighted the intersection of technology and creativity, bringing artists, educators, technologists, and enthusiasts together to share projects and explore new tools, techniques, and ideas. Through online presentations, virtual meetups, collaborative artworks, and hands-on coding challenges, the event showcased the pivotal role of Processing and creative coding in fostering accessible, inclusive, and innovative practices over the past two decades.

5.5.2 Presentation Abstract

This session delves into the integration of py5—a Python adaptation of Processing built on its core libraries—with Thonny, a user-friendly Python IDE designed for beginners. The outcome is a portable, self-contained IDE that replicates the simplicity and functionality of the Processing Development Environment (PDE), creating an accessible and easily distributable platform for creative coding enthusiasts and learners. The presentation highlights py5's capabilities, including support for Static, Module, and Imported programming modes, while demonstrating how Thonny can be customised to emulate a PDE-like experience.

5.5.3 Content & Method

The presentation provided an overview of creative coding using Python and Processing, with a focus on transitioning from Processing.py to py5 using Thonny-py5mode, exploring workflows, technical differences, and practical demonstrations of the tools.

Processing traditionally uses Java as its foundation for running sketches. Processing's Python Mode (Processing.py) integrates Python by leveraging Jython, a tool that translates Python 2.7 code into Java-compatible bytecode [103]. However, this approach has limitations, including a lack of Python 3 support and the inability to use popular C-extension libraries like NumPy.

py5, introduced as a "spiritual successor" to Processing.py, addresses these limitations by replacing Jython with JPype. This change enables support for Python 3 and seamless integration with C-extension libraries [137]. The aim of py5 is not just to replicate and extend Processing.py, but to integrate Processing's capabilities into the broader Python ecosystem while maintaining its creative programming focus [202].

The session demonstrated setting up Thonny, a beginner-friendly Python IDE, configured with the Thonny-py5mode plugin; this setup (Figure 5.6) allows users to run py5 sketches in an environment tailored for education and creative coding. There was also a brief demonstration using py5 with Jupyter Notebooks, providing a flexible, browser-based alternative for writing and running sketches.

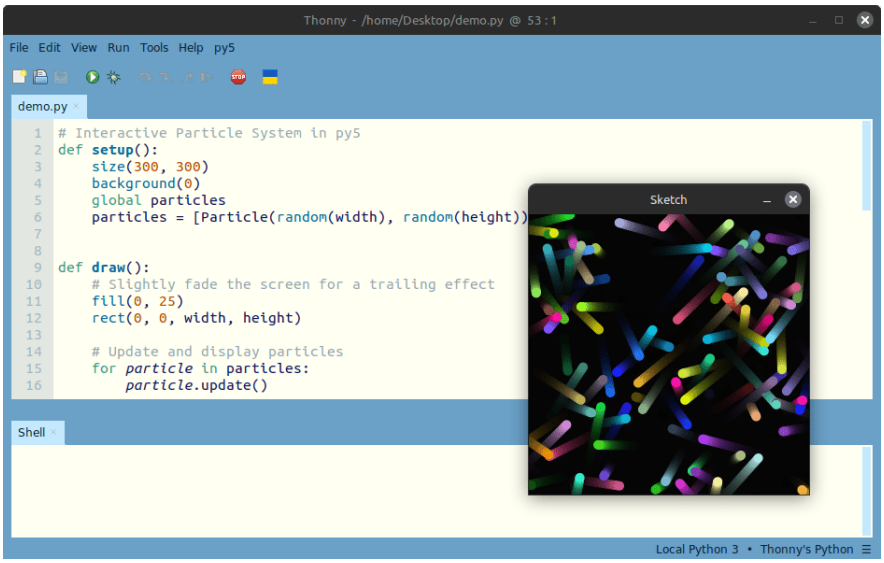


Figure 5.6: The Thonny IDE with the Thonny-py5mode plugin activated, running an animated particle sketch. Plugin and screenshot by the author.

The session also showcased the debugging capabilities of Thonny-py5mode, using Python's PDB[7] debugger to step through code execution interactively, and Thonny's "Plotter" (Figure 5.7) feature to visualise real-time data output, such as waveforms.

[7] https://docs.python.org/3/library/pdb.html

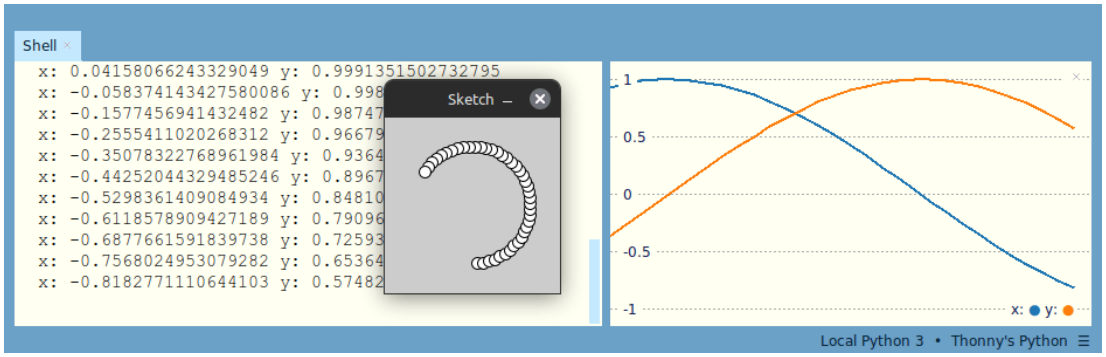


Figure 5.7: Running a py5 sketch to draw a circle, while plotting the x- and y-coordinates as sine and cosine waves using Thonny's *Plotter* feature. Screenshot by the author.

The presentation concluded with exploring additional features, such as how Thonny-py5mode customises Thonny's themes and syntax highlighting for a more Processing-like experience, and guidance on managing Python packages within Thonny. Participants were directed to a GitHub repository containing examples, setup instructions, and links to learning materials, still archived at https://github.com/tabreturn/cc-fest-py5.

5.5.4 Audience Engagement & Feedback

Audience engagement primarily focused on technical questions related to the code demonstrations.

5.5.5 Reflections & Implications

This presentation marked the first public introduction of the Thonny-py5mode plugin to a gathering of experts and users, following months of development informed by research into Processing-like Python solutions.

Experimentation with various code editors and the (then) recent stable release of py5, revealed that combining py5 with the Thonny editor could offer a promising successor to Processing's Python Mode (Processing.py). This led to the creation of the Thonny-py5mode plugin, providing a new Python 3 environment designed to enhance programming education through creative computing. As a central component of this PhD study, its development is detailed in Chapter 3 (Software).

In addition to Python 3 support, py5 offers many compelling features, including its five modes[8], three of which the presentation highlighted:

[8] http://py5coding.org/content/py5_modes.html

- **Static Mode**: Like Processing's *Static Mode*, this allows users to create graphic output with as little as a single line of code. For example, the following code draws a circle in the centre of the display window:

```
circle(width/2, height/2, 10)
```

- **Imported Mode**: This mode enables users to create animated and interactive output by adding a `draw()` and (optionally) `setup()` function. For instance, to draw circles at the position of the mouse cursor:

```
def draw():
    circle(mouse_x, mouse_y, 10)
```

- **Module Mode**: Offering a more traditional Python experience, requiring an `import` statement and accessing py5 features via a `py5` prefix. Below is a *Module Mode* adaptation of the *Imported Mode* example:

```
import py5

def draw():
    py5.circle(py5.mouse_x, py5.mouse_y, 10)

py5.run_sketch()
```

The presentation also emphasised py5's ability to extend Processing's functionality through integration with C-extension Python libraries, including tasks involving:

- **NumPy Integration**: By demonstration of direct manipulation of pixel data in sketches, specifically using the py5 `np_pixels[]` feature[9], a NumPy array containing the values for all the pixels in the display window.

[9] https://pyscoding.org/reference/sketch_np_pixels.html

- **Pymunk Integration**: Using an example combining py5 drawing functions to render Pymunk physics simulations, such as 2D falling and bouncing objects.

The exploration of various Python–Processing environments and libraries, including py5, provided valuable insights that shaped the direction, design, and development of Thonny-py5mode. These findings, along with research on learner experiences using Thonny-py5mode, are documented in Q1/Q2 journal articles in the Publications chapter of this thesis.

5.6 Generate SVG for Pen Plotters using Python

Conference/Event Name: Virtual CC Fest 2022

Author(s): Bunn, T.

Venue: Online

Start/Finish Dates: 30 January 2022

URL: https://ccfest.rocks

Acceptance Letter/Invitation or Event Programme: See Appendix D.4

5.6.1 Conference/Event Overview

As described previously (see Section 5.5. Thonny + py5: A Python 3 Environment for Processing), CC Fest is a free, inclusive event that celebrates creative coding in all its forms. It brings together a diverse community of participants, from students and artists to hobbyists, educators, technologists, and anyone curious about the field.

5.6.2 Presentation Abstract

This session explores the intersection of creative coding, vector graphics, and modern plotting technologies. It begins with an overview of plotters and their resurgence in digital art, then moves on to the fundamentals of programmatically generating vector art using Python, introducing the technical and artistic potential of py5, Blender scripting (bpy) for 3D line-graphics, and Inkscape to manipulate SVG-based generative art for pen plotters. Attendees will learn techniques for optimising plotting workflows with Python libraries like vpype, and handling SVGs through programmatic layer management. This session provides code-along examples and resources for beginners and seasoned creators alike.

5.6.3 Content & Method

A pen plotter is a mechanical device akin to a robotic arm with a pen, used for drawing vector-based art and design work. The session's introductory segment discussed the revival of vintage plotters, DIY plotter projects, and modern pen plotters with a specific focus on the Axidraw range manufactured by Evil Mad Scientist [62]. These are simple, modern, and relatively inexpensive devices that can hold most types of pens, markers, and even brushes (Figure 5.8).

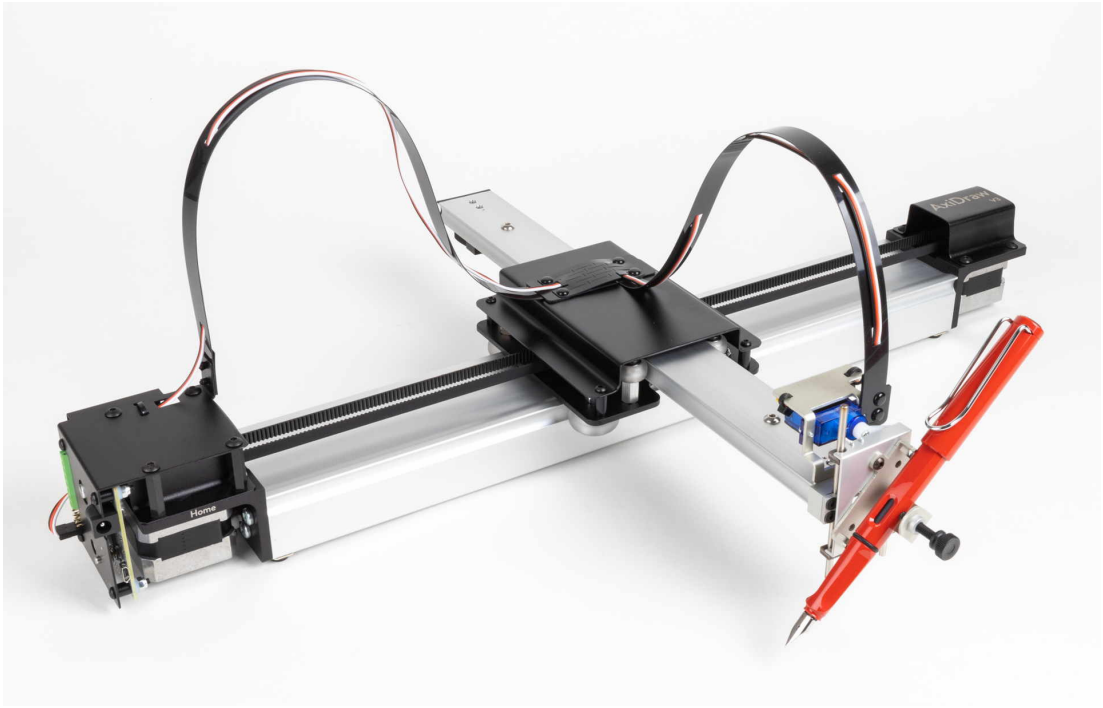


Figure 5.8: The most popular AxiDraw model, the V3, which can accommodate an A4-size plotting area. Image from Evil Mad Scientist. Source: [62].

Vector graphics are defined mathematically, ensuring they are resolution-independent and maintain smooth, sharp edges regardless of size. In contrast, raster graphics are composed of a fixed grid of pixels and become visibly pixelated or blurry when enlarged [35]. SVG is a vector graphic file format containing human-readable code based on the XML standard, with tags and attributes to describe different shapes and properties [60]. Most artists and designers rely on tools with graphic interfaces (like Adobe Illustrator, Affinity Designer, or Inkscape) to create SVG graphics. A more technical alternative involves generating SVGs programmatically using languages like Python.

While hand-coding logos and illustrations is almost always less practical than using software with a graphic interface to draw them, generating SVG markup with Python (or similar programming languages) opens up unique possibilities. It allows for the automation of repetitive tasks, ensures precise control over coordinates, and facilitates the creation of generative or data-driven visuals [115, 185]. Creatives can delve into algorithmic and procedural vector graphic art, leveraging mathematical principles, randomness, and rules-based logic to produce intricate patterns and dynamic compositions [164].

Preparation for the session involved extensive research and hands-on experimentation with py5, Thonny-py5mode, and an *AxiDraw V3/A3* pen plotter. Based on insights, the session showcased novel

Python-based generative art techniques, including employing py5's `beginRecord()` and `endRecord()` methods to create layered SVG files.

Layered SVG files are particularly useful for plots requiring different colours and line styles, as one can assign a different layer to each 'pen,' thereby simplifying the process of switching drawing implements. Axidraw plotters integrate a 'print driver' within Inkscape, and the session demonstrated how to enable, disable, or combine SVG layers in Inkscape and other techniques for streamlined plotting workflows.

Figure 5.9 compares a procedurally generated SVG and its plotter output for a multi-pen, layered SVG plot. This is one of several experiments using a combination of Thonny-pymode and Inkscape.

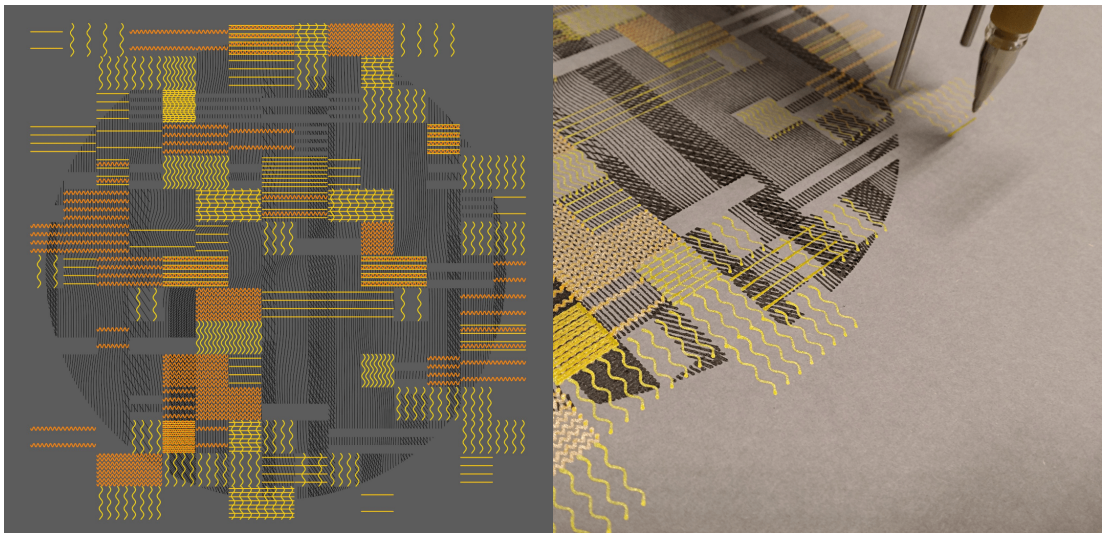


Figure 5.9: Comparison of a Python-generated SVG graphic (left) and its plotted result (right). Screenshot and artwork by the author.

The session also demonstrated techniques employing *vpype*, a *"Swiss-Army-knife command-line tool for plotter vector graphics"* [23]. However, rather than operating this via the command line, the code samples integrated vpype into the same py5 script to generate the graphic, helping to better organise and streamline the process of outputting ready-optimised SVG files. Several vpype examples helped highlight some key features, including:

- **Path Optimisation**: Simplifying paths to improve plot efficiency, including line-merging and related commands that can 'prune' overlapping paths and closely positioned points by merging them where possible.

- **Plotting Optimisation**: To sort the order of paths for efficient plotting, including a *Travelling Salesman Problem (TSP)* solver to minimise pen-up travel distance; and visualisation features to provide a preview of the vector graphic with an overlay of the plotter's path for verification.
- **Occluded Line Removal**: The *occult*[10] plugin from Loic Goulefert to remove lines 'occulted' by polygons; this refers to lines or segments of a polygon that should be hidden or obscured by other elements within the same drawing. In other words, occult can 'cover' parts of shapes that otherwise 'show through' others (Figure 5.10). The order of paths is important, as occult will consider the last geometry in an SVG file to be 'on top' of all other geometries.

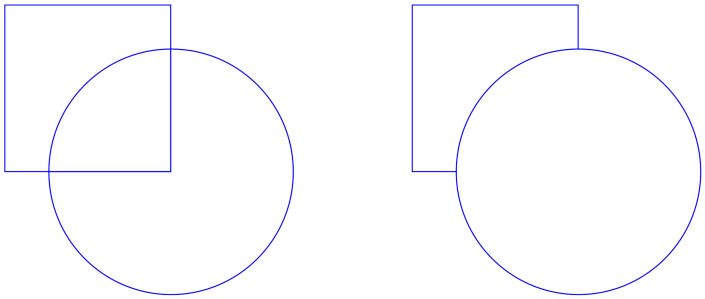


Figure 5.10: An overlapping square and circle (left), and the same drawing after applying occult (right). SVG generated by the author.

Blender is a free and open-source 3D creation software suite that supports the entire 3D production pipeline. It also features a comprehensive Python scripting API (bpy) to customise workflows and extend Blender's functionality. This allows one to programmatically manipulate objects, materials, and animations, enabling automation of repetitive tasks, the creation of custom tools or add-ons, and procedural content generation [3].

Furthermore, Blender can render SVG files using its edge- and line-based non-photorealistic (NPR) rendering engine, Freestyle [25]. When combining the bpy library (to generate 3D forms) and Freestyle, one can utilise Python to create aesthetically compelling and visually intriguing results. Figure 5.11 showcases one of many bpy SVG experiments, exploring the creation of complex structures blending cones and metaballs.

The session covered setting up Blender scenes for SVG output. In such applications, vpype proves particularly useful for optimising the Freestyle SVG renderings, as these can include many overlapping

[10] https://github.com/LoicGoulefert/occult

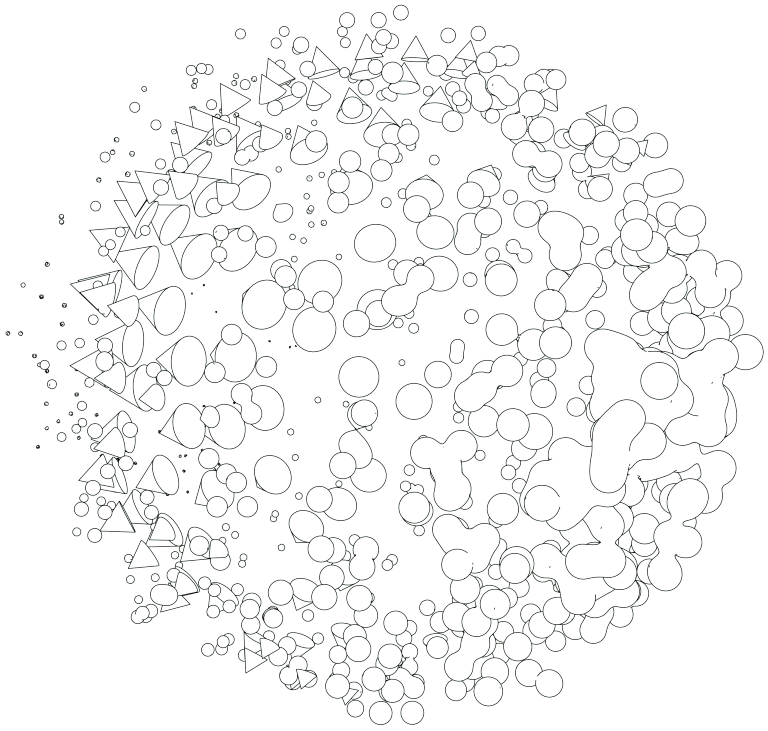


Figure 5.11: An SVG sphere comprising cones and metaballs, procedurally generated using the bpy API. Generated by the author.

and disconnected lines, as well as excessive points that would otherwise dramatically slow the plotting process.

A GitHub repository was made available to attendees, offering helpful resources, including task files designed to guide users through various techniques for generating plottable SVG files using Thonny-py5mode. It also featured links to a curated collection of useful plotter utilities and inspiring artworks from the plotter art community. The repository is archived at https://github.com/tabreturn/cc-fest-plotter.

5.6.4 Audience Engagement & Feedback

This CC Fest proved exceptionally popular, with organisers capping registrations at 750 attendees days before the event. However, it is not clear exactly how many people attended this talk. Platform analytics were opaque, timing was tight, and participant feedback was minimal, consisting primarily of a few technical queries.

The GitHub repository received 26 stars (effectively a "favourite" or "like" for open-source projects) following the event, suggesting that significantly more than 26 people viewed the talk. In previous CC Fest sessions, similar repositories typically received about one star per five attendees.

5.6.5 Reflections & Implications

This presentation showcased the ongoing development and refinement of Thonny-py5mode, representing continued research into innovative creative coding techniques and the tool's potential for enhancing Python programming education within creative computing contexts.

Pen plotters can offer unique, hands-on learning opportunities by introducing students to modern automation and digital fabrication concepts. The tangible outputs of their work can provide an added layer of satisfaction, fostering deeper engagement and enhancing the educational experience [24, 105].

By utilising Thonny-py5mode for plotting applications, students can explore creative coding concepts by generating SVG files. This process introduces XML markup and exposes them to real-world challenges in translating digital designs into physical art. Unlike virtual environments, pen plotters bring inherent variability due to factors such as paper imperfections, pen inconsistencies, and adjustments to plotter speed, all of which can affect line quality and accuracy in the final output.

Several of the artworks in the Creative Works chapter incorporate pen plotters, leveraging the techniques shared in this session to produce compelling results.

The Blender Python (bpy) scripting aspects explored in this session inspired further research into Blender as a creative coding tool, culminating in a more comprehensive presentation on the topic at SIGGRAPH Asia 2022: *Blender Scripting for Creative Coding Projects*.

5.7 Demystifying the Python–Processing Landscape: An Overview of Tools Combining Python and Processing

Conference/Event Name: ACM SIGGRAPH 2022

Author(s): Bunn, T., and Carrasco, T.

Statement of Authorship: See Appendix F

Venue: Vancouver, Canada & Virtual

Start/Finish Dates: 08 August 2022 – 11 August 2022

URL: https://dl.acm.org/doi/10.1145/3532836.3536231

Acceptance Letter/Invitation or Event Programme: See Appendix D.1

5.7.1 Conference/Event Overview

SIGGRAPH is a premier annual conference and exhibition focused on computer graphics and interactive techniques, drawing a global audience of researchers, artists, developers, and industry leaders. It is hosted in North America and serves as a central platform for showcasing advancements in 3D animation, visual effects, virtual reality, and other creative technologies. The event includes technical papers, workshops, keynote presentations, and exhibits from leading studios and institutions, offering a comprehensive view of the industry's latest developments and future directions. In 2022, ACM SIGGRAPH Vancouver drew over 11,700 attendees [246].

5.7.2 Presentation Abstract

Processing, launched in the early 2000s, combines Java with an integrated development environment (the PDE) for creative coding. 2010 saw the introduction of Processing's Python Mode, offering a way to write Python syntax in the PDE. However, this relies on Jython, which is restricted to Python 2.7 and lacks compatibility with many CPython libraries. Other options have since emerged to address these limitations and provide Python 3 support; among those, p5 provides a native Processing port, while py5 leverages JPype to interface with Processing's core libraries.

This talk explores the often confusing landscape of Python–Processing tools, covering their history, development, and differences. It provides detailed comparisons and weighs the advantages and disadvantages of various solutions, offering insights into the possible future of Python–Processing environments. In

addition, the presenter will provide practical guidance on selecting the most suitable tools for beginners, educators, and experienced programmers eager to employ Python as a medium for creative coding.

5.7.3 Content & Method

This presentation largely represented a summary of all the research, development, and experimentation that informed the previous presentations detailed in this chapter.

The talk explored Python-based tools inspired by the Processing paradigm, targeting artists, educators, and programmers interested in creative coding. It highlighted tools like Processing.py (Python Mode), pyp5js, p5py, and py5, emphasising their features, distinctions, and applications. For instance, pyp5js[11] supports Processing-like Python programming for the web, while py5 integrates deeply with the CPython ecosystem.

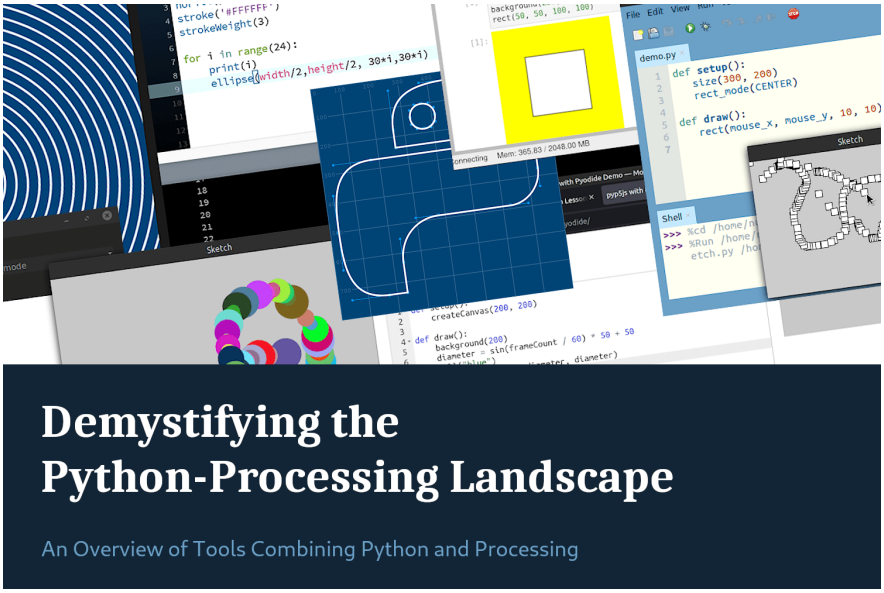


Figure 5.12: Digital image created to represent the talk, displayed in the ACM SIGGRAPH 2022 event programme. Image by the author.

By way of practical examples, the talk demonstrated static and interactive sketches in Java, contrasting these with implementations in Processing.py, py5, and other environments, also covering features like animation, interactivity, and the integration of libraries with C extensions (NumPy and Pymunk). The use of py5 in Thonny (using the Thonny-py5mode plugin) and its Jupyter Notebook support

[11] https://berinhard.github.io/pyp5js

highlighted the opportunities for learning and teaching Python fundamentals within a creative coding context.

The session concluded with resources for further exploration, including links to GitHub repositories, cheat sheets, and useful forums. The supporting materials are archived in the ACM digital library at https://dl.acm.org/doi/10.1145/3532836.3536231.

5.7.4 Audience Engagement & Feedback

The presentation was delivered in a pre-recorded format. During the subsequent live Q&A session, the presenter, Bunn, addressed several technical questions and engaged in discussions with the audience, which (perhaps unsurprisingly) included several creative coding enthusiasts.

One attendee suggested that others share their Python projects at *SPARKS*[12] events. These are monthly gatherings featuring a series of 5- to 10-minute lightning talks followed by discussions centred on the month's topic, hosted by ACM SIGGRAPH DAC. The mission of the DAC (Digital Arts Committee) is to foster year-round engagement and dialogue within the digital, electronic, computational, and media arts. The committee strives to facilitate dynamic scholarship and creative programming within the ACM SIGGRAPH organisation, aiming to promote collaboration between artists and the broader computer graphics and interactive techniques communities [4].

5.7.5 Reflections & Implications

The event represented the most prominent platform to date for showcasing the research, software development, and experimental work conducted for this PhD to a global audience, serving as the culmination of those efforts. Subsequently, the researcher attended several *SPARKS* gatherings, which offered valuable inspiration and insights that shaped some of the outputs in the Creative Works chapter.

[12] https://dac.siggraph.org/sparks-overview

5.8 Generative Art with Python (using py5 and bpy)

Conference/Event Name: PyCon XI (2022)

Author(s): Bunn, T., and Carrasco, T.

Statement of Authorship: See Appendix F

Venue: Christchurch, New Zealand: The Arts Centre Te Matatiki Toi Ora

Start/Finish Dates: 19 August 2022 – 21 August 2022

URL: https://kiwipycon.nz

Acceptance Letter/Invitation or Event Programme: See Appendix D.10

5.8.1 Conference/Event Overview

As described previously (see Section 5.1. Processing.py—Creative Coding with Python), Kiwi PyCon is an annual conference dedicated to Python programming and its applications, serving as New Zealand's national Python conference.

5.8.2 Presentation Abstract

This talk will introduce Python as a tool for generative art, which entails creating artistic multimedia output using computer code. Think: writing Python code that makes visual art, usually with some randomness thrown in for cool and unpredictable results. The presentation focuses primarily on two Python libraries: py5 for 2D visuals and Blender's bpy module for 3D. It includes an overview of generating artworks for NFTs, pen plotters, and other creative output using a new creative coding environment, Thonny-py5mode.

5.8.3 Content & Method

The presentation content represented the ongoing progress of this PhD study's investigation into innovative Python tools for visual learning contexts, Python-based creative coding techniques, and the development of the Thonny-py5mode plugin.

To provide relevant context, the presentation highlighted various techniques employed by creators on *fx(hash)*[13], illustrating how similar methods could be emulated or adapted using Thonny-py5mode.

[13] https://www.fxhash.xyz

fx(hash) serves as a platform and ecosystem for creating, collecting, and trading generative NFT art, leveraging the Tezos blockchain for its operations [226]. During this period, the art world's engagement with NFTs peaked notably. In March 2021, digital artist Beeple's work, *Everydays: The First 5000 Days*, sold for $69 million at Christie's, marking a significant moment in digital art history [106]. This event solidified NFTs as a legitimate and potentially profitable art market segment, while prominent NFT generative art collections like *Bored Ape Yacht Club* and *CryptoPunks* gained substantial popularity [36].

In preparation for the event, this research focused on exploring new methods for previewing output that do not rely on the traditional py5/Processing sketch window. This entailed presenting a solution that utilises a web browser to preview SVG files generated using Python scripts, enabling users to take advantage of built-in developer tools for navigating XML-based structures (Figure 5.13). This setup automatically reloads the preview whenever the image is regenerated, triggered by rerunning the script within Thonny-py5mode.

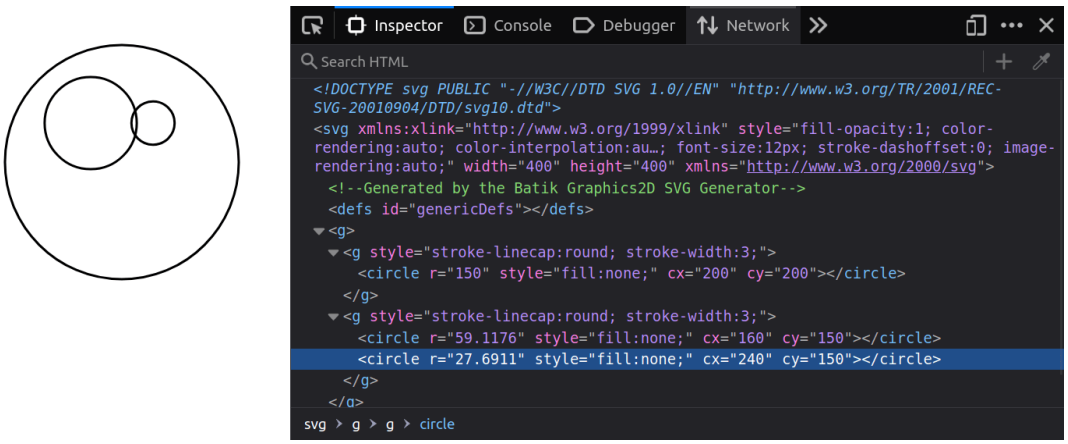


Figure 5.13: Using the Firefox web developer panel to explore SVG markup generated with Thonny-py5mode. Screenshot by the author.

For Blender Python (bpy) scripts, the proposed approach leveraged Blender's *headless* mode to generate 3D outputs directly. This eliminates the need to launch the full Blender interface, instead using Blender's Python interpreter to run scripts from within Thonny. Practical examples showed how the resulting images could automatically reload in a preview window (see Figure 5.14). This workflow incorporated the `fake-bpy-module`: a set of mock Blender Python API modules that enable code completion and syntax highlighting in editors like Thonny.

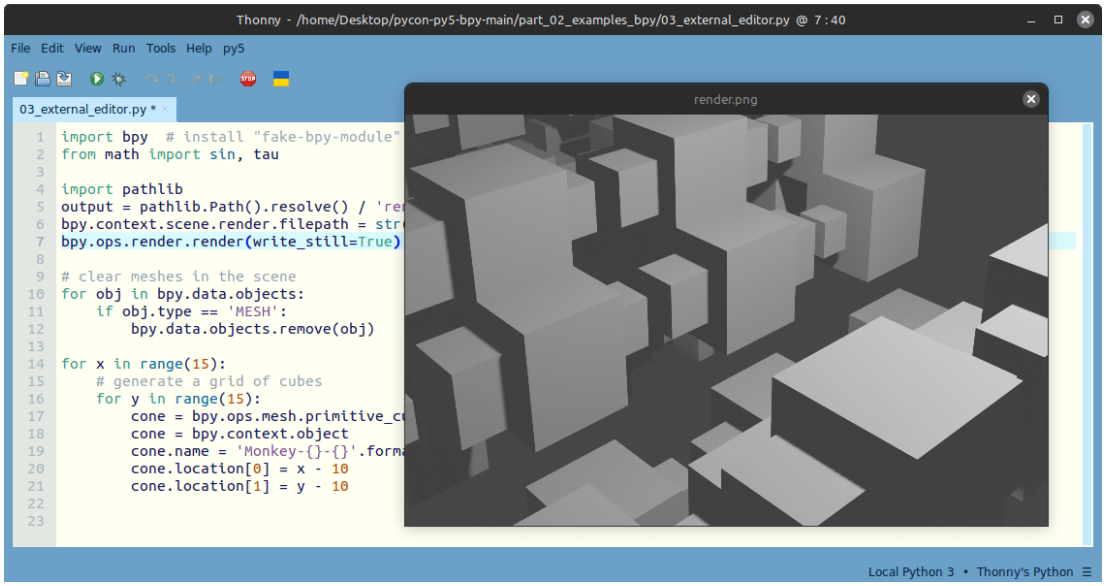


Figure 5.14: Generating a Blender render directly from Thonny, without having to launch the Blender application. Screenshot by the author.

5.8.4 Audience Engagement & Feedback

As the event schedule ran over time, there was insufficient opportunity for questions and discussion. However, several participants shared their creative work after the talk and sought discussion on creative coding techniques.

The talk followed Paris Buttfield-Addison's presentation on *Machine Learning with Python and a Game Engine*, based on his book *Practical Simulations for Machine Learning: Using Synthetic Data for AI* [37]. This inspired further consideration on how educators might utilise Thonny-py5mode to teach foundational machine learning concepts. One possibility is to employ py5 to visualise ideas similar to those in Tariq Rashid's *Make Your Own Neural Network*, described as a "step-by-step, gentle journey through the mathematics of neural networks, and creating your own using the Python programming language" [184]. Another option is to adapt material from David Glassner's *Deep Learning: A Visual Approach* [74]. A further direction could involve reworking Daniel Shiffman's *Neural Networks* chapter from *The Nature of Code*, which demonstrates building "the simplest possible example of [a neural network]" using only p5.js, Processing's JavaScript environment [206].

To delve into machine learning more directly, educators can bypass the programming of a basic neural network from scratch and move immediately into model training. For this approach, we can draw on Python's established ecosystem of pre-built libraries and frameworks, such as *PyTorch*, *TensorFlow*, and *Scikit-learn* [70, 112, 183].

Although this PhD study does not focus on machine learning or generative AI, its potential relevance to future work, as well as its impact on programming education, is explored in Section 5.10 and readressed in the "Conclusion" chapter.

5.8.5 Reflections & Implications

This was an opportunity to collaborate with co-presenter Taylor Carrasco[14], a practitioner, researcher, and educator in VFX, animation, post-production, and visual narratives, whose expertise in 3D environments and VFX scripting directly informed the bpy component.

The presentation research highlighted opportunities to extend Thonny-py5mode beyond py5 by integrating additional creative coding capabilities. The bpy experiments, for example, serve as a proof of concept for seamless Blender scripting integration, enabling students to explore advanced 3D content. Using Python to interface with Blender's render engine enables high-resolution outputs that incorporate particle systems, rigid-body dynamics, fluid dynamics, cloth dynamics, metaballs, and volumetrics [34].

Another possible enhancement involves adding a preview tool for inspecting SVG markup alongside py5 graphics, with features such as browsing nested tags and attributes—akin to browser developer tools. These examples point to future directions for creative coding support in Thonny-py5mode.

Thonny-py5mode takes the second part of its name from py5, but renaming the tool may become necessary if it expands to include additional features (such as bpy support). The *thonny-* prefix follows Thonny's required naming convention for plugins.

While the focus of Thonny-py5mode and this PhD research is not NFT art, the scene offers relevance, thought-provoking class discussion topics, and even a potential income-generating hobby for students learning Python programming using creative coding environments. Although many observers note that the hype around NFT art has subsided, others argue that NFTs remain in a state of ongoing evaluation, particularly in the context of the art business. Several prominent NFT platforms continue to sell generative art, demonstrating some enduring appeal and market presence [49].

The bpy exploration also inspired further research into Blender as a creative coding tool, leading to an expanded collaboration with Carrasco, culminating in a more in-depth presentation at SIGGRAPH Asia 2022 (see *Blender Scripting for Creative Coding Projects*).

---

[14] https://www.imdb.com/name/nm3576725

5.9 Blender Scripting for Creative Coding Projects

Conference/Event Name: SIGGRAPH Asia 2022

Author(s): Bunn, T., and Carrasco, T.

Statement of Authorship: See Appendix F

Venue: Daegu, South Korea

Start/Finish Dates: 06 December 2022 – 09 December 2022

URL: https://sa2022.siggraph.org/en/?post_type=page&p=2122&id=crs_111&sess=sess135

Acceptance Letter/Invitation or Event Programme: See Appendix D.2

5.9.1 Conference/Event Overview

SIGGRAPH Asia is a leading annual conference and exhibition covering advancements in computer graphics, interactive techniques, and emerging technologies across the Asia-Pacific region. Its diverse programme includes technical papers, workshops, keynotes, panel discussions, and showcases like the Art Gallery, Computer Animation Festival, and Emerging Technologies demos. By bridging disciplines, SIGGRAPH Asia delivers fresh perspectives on 3D graphics, VFX, virtual reality, artificial intelligence, computer vision, and game design, aiming to drive innovation and shape the future of digital creativity. The SIGGRAPH Asia 2022 event drew over 3,000 attendees [246], from 52 countries and regions, including researchers, artists, developers, and industry leaders.

5.9.2 Presentation Abstract

This session introduces Blender as a creative coding tool. Blender is open-source 3D modelling and animation software with a Python API for coding 3D objects, animations, and effects. We'll cover Blender's scripting tools, the bpy module, and techniques for creating and manipulating 3D objects with Python. Participants will generate animated 3D patterns entirely through code, learn to import and manage different Python modules, and employ external code editors to interface with Blender. Attendees can code along or observe and ask questions.

5.9.3 Content & Method

This session content built on previous the presentations, *Generate SVG for Pen Plotters using Python* and *Generative Art with Python (using py5 and bpy)*, to further explore Blender as a tool for cre-

ative coding, focusing on deeper insights into Blender's bpy API and more involved techniques for programming generative 3D art.

While Blender is widely known for 3D modelling and animation using its GUI and node-based workflows [234], one can also employ Python scripting for a broad range of Blender tasks. These range from programmatically generating basic objects to advanced operations, such as particle systems, physics simulations, procedural modelling, and scene automation. This approach offers high-quality 3D renderings beyond what more 2D-oriented creative coding tools like Processing can output [156].

This session spanned approximately 1.5 hours, guiding participants through calibrating a Blender scene, and using Python to programmatically generate 3D objects and algorithmically control transformations and animations. The content was structured as follows:

1. **Setup**: Installing Blender across Windows, macOS, and Linux, launching it via the command line, and running basic Python scripts using the Blender *Scripting* tab (Figure 5.15).

2. **Blender Scripting Features**: Exploring core tools within the Blender interface, including the *Info Editor* for logging Python commands from GUI actions, the *Python Console* for executing Python commands interactively, and configuration settings for developer extras like Python tooltips.

3. **bpy Fundamentals**: Introducing the bpy module, focusing on key components like `bpy.data` and `bpy.context`; this included addressing and manipulating objects via bpy.

4. **Programming Animations**: Demonstrating the creation of animations using programmatically generated keyframes and sine wave functions to produce intricate wave-like motion.

5. **Advanced Techniques**: Covering how to manage external files, integrate third-party libraries, and execute Blender scripts directly from the command line for 'headless' execution.

6. **Using External Editors**: Describing how to edit scripts in external code editors (such as Thonny or Atom) and methods to auto-reload rendering previews during headless execution.

The session materials and resources are archived at https://github.com/tabreturn/blender-creative-coding. The repository includes additional resources for further exploration, inspiring examples, and scripts incorporating procedural generation and related techniques.

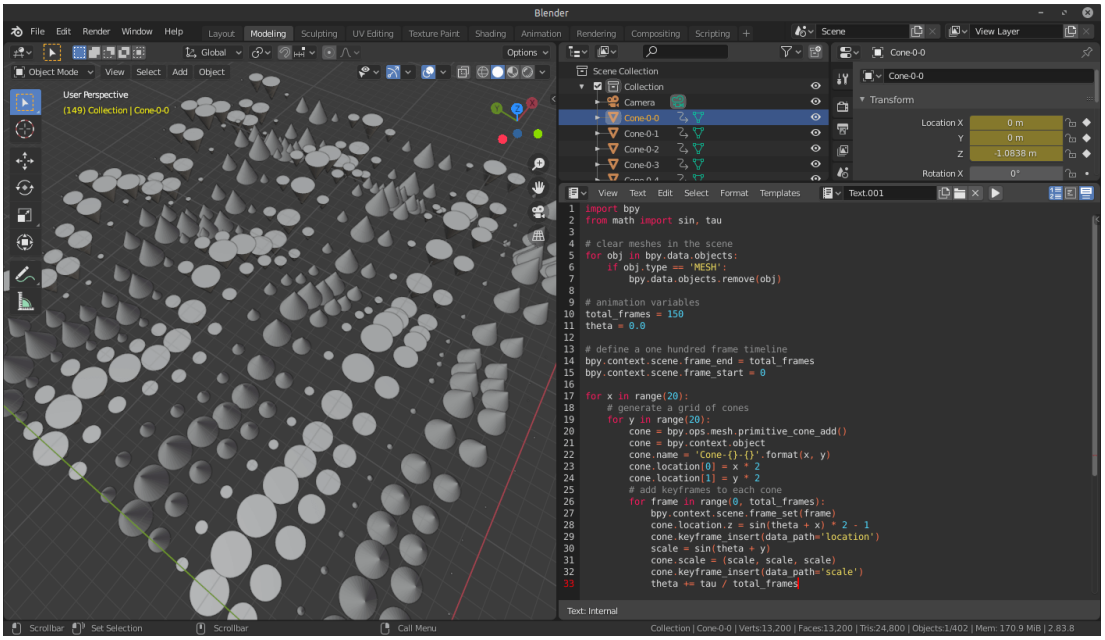


Figure 5.15: Demonstrating the creation of programmatically generated patterns using Blender's scripting tools, before moving into an external editor. Screenshot by the author.

5.9.4 Audience Engagement & Feedback

Bunn, the PhD candidate who prepared the proposal and materials for the accepted talk, transitioned to a role with a new employer prior to the SIGGRAPH Asia event, forfeiting his funding and availability to present. As a result, collaborator Carrasco (the second-listed author) delivered the session solo.

5.9.5 Reflections & Implications

This session highlighted Blender's potential for Python-based creative coding—a capability that may be overlooked by practitioners using Processing, OpenFrameworks, and similar frameworks.

Consider GitHub *topics*: these are tags developers assign to repositories to categorise them by technology, framework, programming language, or subject area, aiding discoverability and filtering. For example, the topic "p5js" includes 4,405 public repositories[15], while "openframeworks" lists 765[16]. By contrast, "blender-scripts," identified as the most representative tag for Blender scripting after

[15] https://github.com/topics/p5js

[16] https://github.com/topics/openframeworks

a review of various repositories, contains only 325[17]. The topic "Processing" (4,442 repositories) is excluded here because its broad usage across unrelated domains makes it an unreliable indicator.

Processing's Python Mode, py5, p5py, DrawBot, and Shoebot provide valuable gateways into creative coding, the Python language, and programming fundamentals in general. While these environments offer robust support sufficient for most 2D and some light 3D creative coding applications, they lack the powerful 3D capabilities Blender can provide. Through Blender scripting, users gain access to a high-performance rendering engine via Python, enabling the creation of high-resolution images and videos. Moreover, bpy enables the exploration of advanced 3D concepts and graphically intensive techniques.

Building on prior presentations, the research and experimentation for this session revealed further opportunities to enhance Thonny-py5mode by integrating additional creative coding features beyond py5, potentially adding Blender scripting support for generating non-interactive 3D output.

[17] https://github.com/topics/blender-scripts

5.10 Mitigating AI Misuse in Introductory Python Courses with Graphical Programming Tasks

Conference/Event Name: Kiwi PyCon 2025

Author(s): Bunn, T.

Venue: Wellington, New Zealand

Start/Finish Dates: 21 November 2025 – 23 November 2025

URL: https://kiwipycon.nz

Acceptance Letter/Invitation or Event Programme: See Appendix D.6

5.10.1 Presentation Abstract

As LLMs reshape how students engage with programming, educators face increasing challenges in maintaining academic integrity. Drawing on the presenter's doctoral research, this talk explores how graphical programming tasks can meaningfully resist unauthorised AI assistance, offering a robust alternative to conventional text-based exercises that are highly susceptible to automation. We present a series of tasks to complete in a Python-based creative coding environment (Thonny-pysmode), employed in undergraduate assessments, and evaluate the performance of leading LLMs in replicating them. This session presents a scalable, discipline-agnostic strategy for designing resilient assessments in the GenAI era that is adaptable to other Python graphics libraries.

5.10.2 Content & Method

General artificial intelligence (GenAI) technologies, particularly large language models (LLMs), are transforming programming education. Since the introduction of tools such as ChatGPT and GitHub Copilot in 2022, it has become increasingly easy for students to generate functional Python code with little to no conceptual understanding of its workings [248]. On one hand, these tools can benefit students by assisting with code interpretation, debugging, and modification tasks. On the other, LLM-assisted programming risks compromising the educational experience, reducing learning to surface-level engagement, where it is difficult for educators to determine genuine understanding from AI-assisted mimicry [107]. Moreover, this presents serious challenges for institutions seeking to uphold academic integrity, ensure fair assessment practices, and effectively grade students' code comprehension, computational reasoning, and problem-solving abilities [152].

Kiesler & Schiffner (2023) demonstrated that GPT-4 performs strongly on beginner programming tasks, solving 94.4% correctly. The study utilised 72 appropriately scoped Python tasks from CodingBat [160], typical of CS1 courses, covering foundational topics such as strings, lists, and logic [109]. Prather *et al.* (2024) investigated the differential impact of GenAI tools on novice programmers and found that such technologies tend to amplify existing disparities rather than mitigate them. In their study, students with higher prior performance and self-efficacy were able to leverage GenAI to accelerate their progress, often using it strategically to test ideas or refine code. In contrast, students who already struggled relied on it in ways that masked their lack of understanding, leading to superficial task completion without deeper learning [166].

However, LLMs are now capable of performing tasks far more complex than simple scripts or CS1-level assessment work, including the generation and deployment of complete applications spanning multiple coding languages, libraries, and frameworks. For instance, Replit Agent can create fully functional web apps from a single natural language prompt [189].

Institutions have responded to the challenges presented by GenAI in varying ways, sometimes inconsistently across departments or even within the same diploma or degree program [114, 175]. Some universities have adopted open and exploratory stances, encouraging students to engage with GenAI for its creative or professional value; others have implemented restrictive policies borne of concerns around plagiarism, over-reliance, and the erosion of foundational programming competencies [132]. However, even within environments that restrict GenAI use, it remains difficult for educators to enforce such policies—automated detection systems remain unreliable, policies can be challenging to interpret and apply consistently across different assessments, and many staff face limited support for conducting time-consuming investigations into suspicious work [9].

In creative computing disciplines, many educators consider expression and experimentation as central tenets of their pedagogy. Recognising the opportunities that GenAI presents, some have begun to integrate it more deliberately into coursework. For instance, *Creative Coding and Generative AI* at Carleton College is a project-based course that engages students using tools such as Inform 7, Tracery, p5.js, and ml5.js—blending narrative, visuals, code, and AI [197]. Similarly, Victoria University of Wellington offers *Creative Coding and AI*, which integrates GenAI to support both design workflows and code review processes [238].

It is also relevant to note that programming graphical output with Python provides a tangible way for students to explore the mechanics underlying AI through graphics, animation, and visualisation.

For example, Daniel Shiffman's *The Nature of Code* includes chapters on evolutionary algorithms and neural networks, demonstrating these concepts using p5.js. Educators can readily adapt these approaches to Python using py5 or other Processing-inspired frameworks [206].

Even in settings where educators wish to avoid GenAI-powered tools, they may find that the software they teach now includes embedded LLM features for contextual guidance, conversational prompting, and real-time assistance [8, 252, 256]. For instance, the latest version of Anaconda integrates the *Anaconda Assistant* within Jupyter Notebook (Figure 5.16). These notebooks serve as environments for Python-based data science, machine learning, and other cross-disciplinary computing. Jupyter is popular in educational settings, particularly in disciplines outside computer science or software engineering, as it combines executable code, narrative text, and inline visuals. This aligns naturally with literate programming paradigms and supports documented experimentation [236]. As GenAI tools continue to permeate standard learning workflows, educators are increasingly rethinking teaching and assessment, considering ways to emphasise interpretation, personalisation, and reflection, rather than simply producing working code [57, 166].

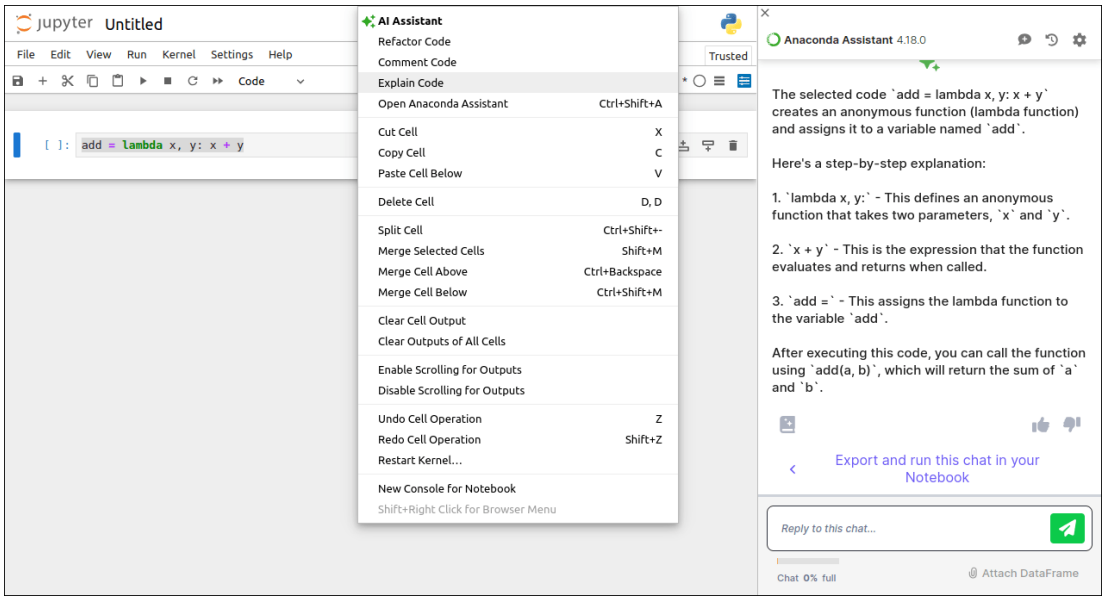


Figure 5.16: Anaconda's Jupyter Notebook environment with integrated AI Assistant (v4.18.0) explaining a `lambda` expression. Screenshot by the author.

Given the prevalence and convenient accessibility of LLM technologies, educators are exploring ways to mitigate their negative impacts on learning and assessment. To enhance the reliability and trustworthiness of assessment practices, research suggests a combination of complementary approaches

that mitigate the misuse of GenAI [125, 249]. Current literature outlines a layered framework of strategies encompassing:

1. **Code Tracking and Authorship Methods**: Employing version control and IDE-integrated trackers (e.g., zyBooks' Coding Trails), as well as stylometric analysis to verify authorship [87, 98, 231]. In high-stakes contexts, secure exams administered online or in physically proctored settings remain among the most effective deterrents to cheating. However, it is also important to acknowledge that such systems can raise ethical concerns, including surveillance overreach, invasive data collection, algorithmic bias, and restrictions on students' control over their physical space and movement [117].

2. **Integrity Culture and Student Engagement**: Strategies that do not focus on code itself, but instead shape student attitudes and behaviours around academic integrity and the responsible use of GenAI. These approaches aim to establish clear and specific policies that encourage the appropriate use of such tools, promoted through explicit guidelines, reflective activities, and ethical discussion [17, 110]. Additionally, educators can remind students to seek help with assessments through office hours, peer collaboration, or tutoring services, thereby mitigating pressures that may drive dishonest behaviour [17].

3. **Modified Assessment Design and Expository Practice**: Utilising authentic, individualised, or expository processes to resist automation [125, 257]. Assignments tied to current events or newly released APIs can exploit knowledge gaps in GenAI training data; bespoke datasets introduce contextual dependencies that are unlikely to be accommodated in public models [181]. Instructors may also redesign tasks to emphasise higher-order thinking and minimise rote GenAI use by focusing on problem-solving rather than implementation details (where LLMs tends to excel) [257]. Expository strategies such as reflective writing or recorded walkthroughs can reveal discrepancies between claimed and actual code authorship [125].

Extending on Item 3 (Modified Assessment Design), McDanel and Novak (2025) evaluated the performance of GPT-3.5, GPT-4o, and Claude Sonnet on a set of representative Python programming tasks from SIGCSE's *Nifty Assignments* [208]. The study sought to identify additional areas in which GenAI performs ineffectively on assessment tasks. Figure 5.17 visualises this sample according to visual complexity and task length.

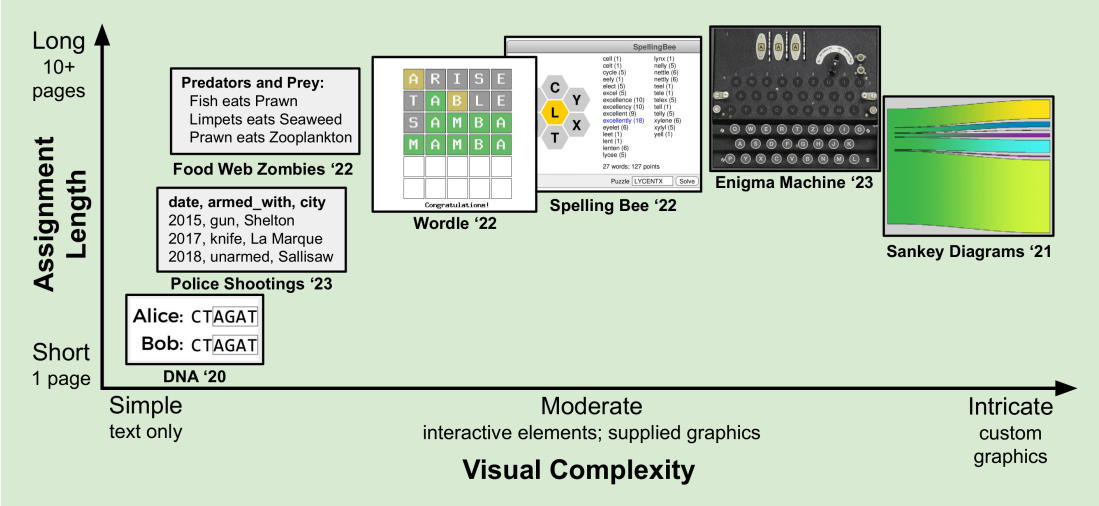


Figure 5.17: McDanel and Novak's spectrum of Nifty Assignments across visual complexity and task length. Source: [131].

McDanel and Novak analysed both qualitative and quantitative aspects of model performance and proposed practical strategies to help educators design assignments that resist automation by GenAI. The authors explicitly note that "certain assignments, particularly those involving visual elements, proved challenging for all models." Furthermore, they observed that "when the input is an image or the solution is defined as a correct-looking visual, current-era LLMs literally do not have a way to interpret such information and struggle." However, "due to the fuzzy nature of correctness often found in graphical programs, they are usually less testable as well." [131]

Thonny-py5mode Graphical Tasks

Inspired by McDanel and Novak's findings, this presentation posits that assessment designers can apply similar techniques to Thonny-py5mode-based tasks. More accurately, this ancillary insight emerged during the study reported in *Evaluating Task–Technology Fit in Thonny-py5mode*, which examined students' use of Thonny-py5mode for graphical coding challenges. Devising the py5 assessment tasks revealed that the combination of visual reasoning and algorithmic logic required to complete them effectively limited the capabilities of LLMs. It was this observation that prompted further exploration of existing literature, leading to the work of McDanel and Novak, among others. Appendix B.1 provides the complete assessment brief, comprising all six challenges, each including scaffolded starter code and contextual hints as provided to students.

Thonny-py5mode makes it straightforward for students to access programming for visual output. Using just a few functions, they can render a variety of shapes with different fill and stroke attributes, enabling them to engage with graphical commands after a brief introduction. Note that a two-page introduction (pp. 2–3) in the brief provided sufficient instruction for students to begin the assessment tasks. Consequently, the original learning outcomes—focused on control flow and loops in particular—remained unchanged, while the refreshed brief substantially enhanced the assessment's resistance to GenAI assistance.

Testing this hypothesis entailed feeding the same six task graphics into three popular LLMs: Claude Sonnet 4, Gemini 2.5 Flash, and OpenAI GPT-4o, along with the associated starter code and prompts requesting py5-based solutions. Figure 5.18 presents the six visual tasks for students to reproduce. These replaced a series of six textual-output tasks that comprised the original assessment brief, trivial to complete using modern LLMs.

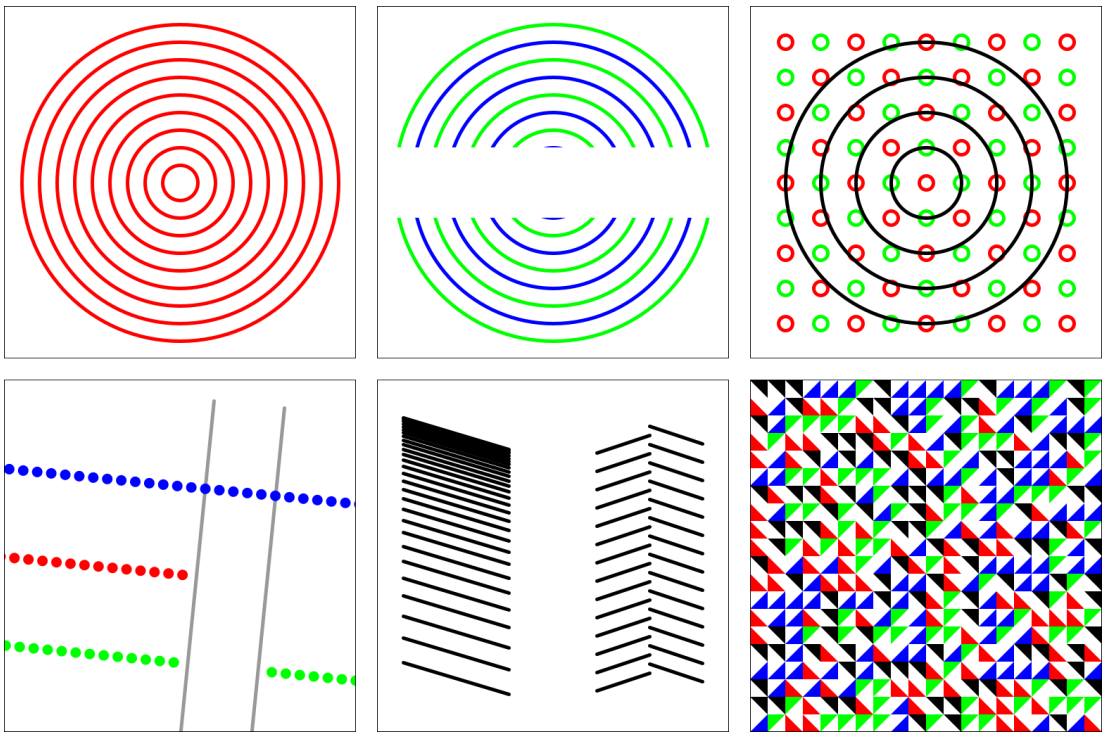


Figure 5.18: Thonny-py5mode graphical challenges used in Assessment 2. Developed by the author using py5.

Figures 5.19, 5.20, and 5.21 present the most accurate results from three attempts per model. Someone with a reasonable understanding of Processing or py5 can identify a high-level approach to each task before writing a single line of code. While some outputs were partially correct, none successfully replicated the target images.

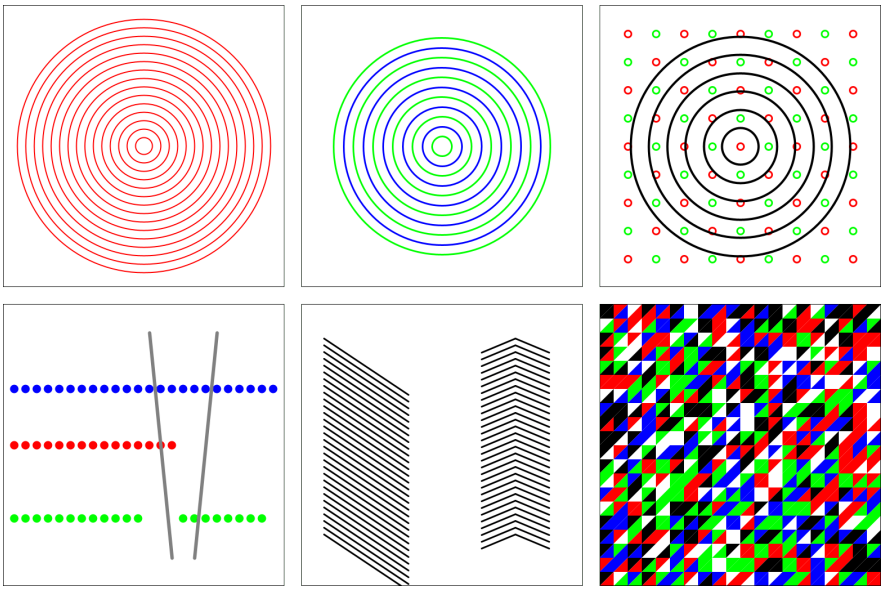


Figure 5.19: **Claude Sonnet 4** attempts at recreating graphical challenges.

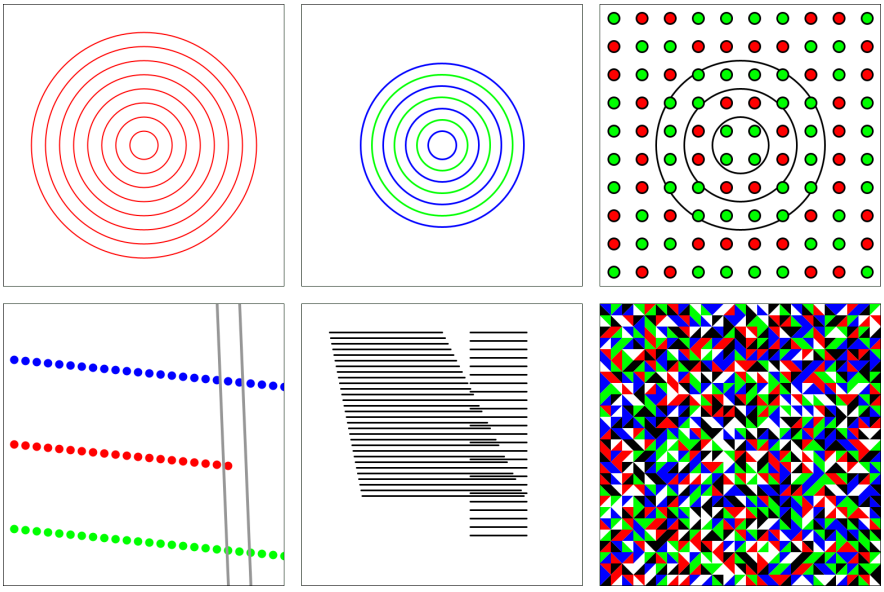


Figure 5.20: **Gemini 2.5 Flash** attempts at recreating graphical challenges.

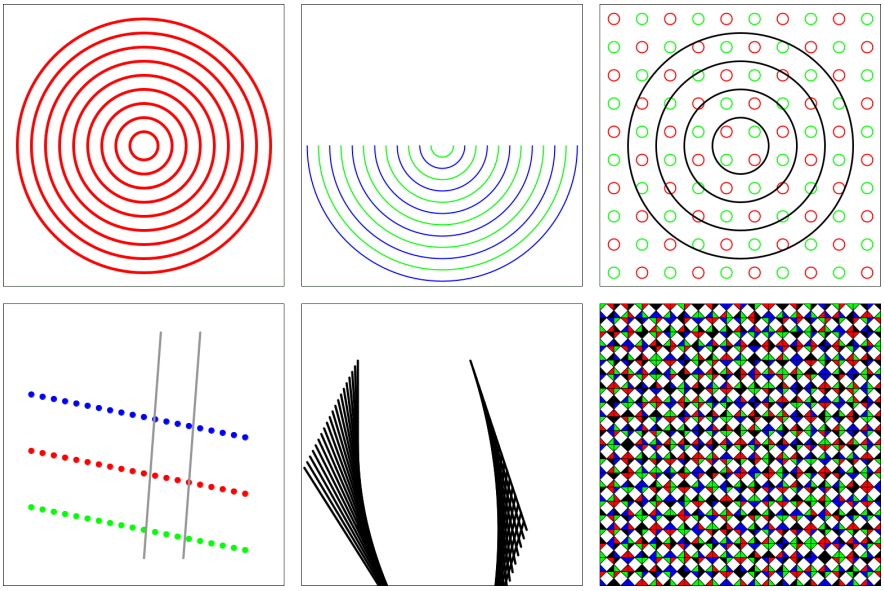


Figure 5.21: **OpenAI GPT-4o** attempts at recreating graphical challenges.

As the reader can observe, inaccuracies range from relatively minor—such as variations in the number of circles and stroke weight for the first (top-left) task—to more conspicuous failures. For instance, none of the models identified that a white rectangle obscures the centre of the green-blue concentric circles (top row, middle graphic). The remaining outputs similarly fall short, often in ways that appear suspicious, as if a student had approximated the layout and managed to code the complex Python logic yet overlooked obvious visual characteristics. While multiple approaches and code implementations could produce identical visual results, the intended solution path should be fairly evident.

These findings suggest that visually oriented coding exercises reveal specific weaknesses in popular GenAI models. Consequently, educators might employ similar tasks to mitigate unauthorised assistance. Even with starter code and clear visual targets, the models consistently failed to produce accurate results, highlighting persistent difficulties in translating graphical goals into functional code, particularly in terms of geometric precision.

5.10.3 Audience Engagement & Feedback

Kiwi PyCon 2025 is scheduled to take place following the submission of this thesis for examination.

5.10.4 Reflections & Implications

Research suggests that Python tasks involving spatial reasoning and visual composition remain challenging for GenAI tools. Thonny-py5mode graphics exploit this limitation, requiring geometric reasoning. The tests presented here demonstrate promising resistance to GenAI automation and merit continued exploration. However, it will be crucial to establish clear rubrics for evaluating visual fidelity in future studies. Further studies should also test whether iterative prompting, added context, or alternative libraries (e.g., p5.js, Pillow) improve GenAI accuracy.

Animated challenges would further amplify this effectiveness. Moreover, parameter variation, where each student receives a unique dataset or task specification, would help reduce the risk of shared or copied code [54]. To individualise tasks, educators can generate variations programmatically.

If GenAI improves at these tasks, future research might shift toward intelligent tutoring systems (ITSs). The boundary between coding assistants and ITSs is increasingly blurred, with GenAI tools now assuming pedagogical roles once reserved for human tutors. While ITSs offer adaptive instruction and feedback [50], GenAI assistants can now provide similar support when prompted strategically [57]. Recent work on Python-focused AI tutoring reflects this convergence [78], including that of Yang *et al.* (2024), who found that PyTutor improved engagement while also cautioning against over-reliance [253].

For creative coding, ShiffBot[18] is a GenAI tutoring experiment co-developed with Daniel Shiffman, integrating Retrieval-Augmented Generation (RAG) into the p5.js web editor to deliver context-aware, pedagogically aligned feedback [194]. No Python-based equivalent appears to exist, though tools like GitHub Copilot and the Anaconda Assistant can generate py5 code with moderate accuracy for beginner tasks. Unlike ShiffBot, though, their educational value depends on constructive use rather than quick solutions.

In conclusion, the research and experimentation that informed this presentation offer new approaches to mitigate AI misuse in introductory Python courses through Thonny-py5mode tasks, highlighting a promising direction for future research on AI-resistant assessment and human–AI collaboration in programming education.

[18] https://shiffbot.withgoogle.com

5.11 Chapter Summary

This chapter presented the outputs forming the presentations component of the PhD folio, tracing the evolution of the research journey and its progression. These activities contributed to the development of new tools, resources, and techniques for enhancing Python programming education through creative computing environments. They also offered valuable opportunities to share findings, spark collaborations, and receive critical feedback. Collectively, these outputs explored the following significant themes:

- **Creative Coding for Beginners**: Demonstrated the accessibility of Python creative coding tools to support learning programming fundamentals through graphical and creative tasks, establishing foundations for pedagogical development. These insights and materials later informed the writing and publication of the sole-authored book, *Learn Python Visually: Creative Coding with Processing.py* (No Starch Press, 2021).

- **Web-Based Creative Coding**: Explored browser-based environments, such as Computiful and Strive, that make Python creative coding more accessible by removing software installation barriers. This work examined the trade-offs between web-based and desktop-based tools, as well as frameworks and libraries that support cross-platform deployment.

- **Interactive Data Visualisation**: Highlighted the p5/p5py library and Thonny-py5mode's capacity to generate unique and interactive data visualisations, expanding Python's capabilities beyond traditional charting libraries. These explorations enabled presentations across events targeting different disciplines, including data visualisation, simulation, numerical methods, and analytics.

- **Innovative Educational Tools**: Documented the development of Thonny-py5mode, a Python 3-based environment designed to succeed the PDE's Python Mode (Processing.py). The Software chapter details the plugin's design, rationale, educational uses, accompanying materials, and development challenges.

- **Plotting Art from Code**: Showcased Python's potential to produce vector-based (SVG) designs for pen plotters using Thonny-py5mode, and optimised these workflows with tools such as vpype and Inkscape to translate digital creations into tangible artworks. The works presented in Creative Works further demonstrate these techniques, experiments, and insights.

- **Comparative Tools Analysis**: Analysed the strengths and limitations of Python–Processing tools—such as Processing.py, py5, p5py, and others—guiding the selection of suitable tools for diverse creative coding and educational applications. These research insights integrate into the MTAP (Q1-ranked) journal article, *Towards a Python 3 Processing IDE for Teaching Creative Programming*.

- **Generative 3D Art**: Demonstrated the capabilities of Blender's Python API (bpy) for creating high-fidelity generative 3D art, bridging creative coding with advanced rendering engines. This work also highlighted opportunities to extend Thonny-py5mode with a Blender 'back-end.'

- **Expanding Creative Coding Frontiers**: Illustrated how combining 2D and 3D programming techniques with various Python libraries, including py5 and bpy, enables the creation of innovative generative art that can appeal to learners and artists alike. Several workshops provided opportunities to share these practical methods.

- **Mitigating GenAI Misuse with Graphical Tasks**: Investigated strategies to reduce students' potential over-reliance on generative AI for completing programming assessments. By integrating graphical tasks, these approaches encouraged a deeper understanding of Python programming fundamentals and computational thinking.

Each output drew upon prior research, experimentation, and audience feedback, illustrating how creative coding environments can democratise Python programming by making it more accessible and engaging for both novices and experienced coders. Together, these contributions point toward future directions, including enhanced 3D and SVG tools and integration with emerging technologies such as artificial intelligence and machine learning.

6 Creative Works

This chapter presents creative works generated using Python-based creative coding tools and techniques developed through this PhD research. Each output has been exhibited at an event and/or featured in a publication and, in all instances, selected through a review process that validates significance and quality. Alongside the outputs in Chapters 4 (Publications) and 5 (Presentation Outputs), these works constitute the PhD folio and support three of the five research objectives: Tool Development, Creative Outputs, and Broader Dissemination.

Traditional artistic practice often engages with personal, cultural, or philosophical themes. In contrast, the works in this chapter focus on pushing the boundaries of tools and exploring new forms of computationally-driven visual expression. As Brown and Sorensen suggest, creative artefacts in practice-led research serve as both a method of inquiry and an outcome, enabling a feedback loop between experimentation and reflection [212]. More accurately, this research's iterative, exploratory process treated creative artefacts as byproducts of experimentation rather than standalone artistic pieces. This reflects a tradition of artists creating custom tools to expand creative boundaries, challenging the limitations of existing instruments to explore new possibilities in digital media [145]. As Paul writes—

> Code has also been referred to as the medium, the 'paint and canvas,' of the digital artist, but it transcends this metaphor in that it even allows artists to write their own tools—to stay with the metaphor, the medium in this case also enables the artist to create the paintbrush and palette. [162]

Accordingly, the chapter contents evidence the creative potential of Thonny-py5mode and other Python tools realised through creative work, rather than focusing on traditional artist statements typical of gallery or student projects.

Inspired by open-source principles and hacker cultures, the conceptualisation phases, tool development, and coding practices adopted agile processes, where software and techniques evolved through experimentation, testing, and community feedback [255]. Similar to Processing or p5.js, environments

like Thonny-py5mode embody the dual role of enabling innovation while serving as artefacts of creative exploration.

The chapter organises content into distinct sections, with each dedicated to a specific output (constituting one or many creative works), presented in chronological order and concluding with the most recent. The individual creative works presented here remain largely untitled; instead, names such as "North" serve as project labels, where experimentation encapsulated threads exploring particular themes and techniques. For clarity and consistency, each section provides a short account of the following aspects:

- **Description & Context**: Outlines the work's format, thematic focus, and dissemination platform, identifying any technical, creative, or conceptual challenges that it addressed.
- **Creative Process**: Briefly describes tools, techniques, and methods used in making the artwork(s), with emphasis on the Python-based creative coding approaches central to this research.
- **Impact & Reflection**: Summarises the work's influence on subsequent outputs, reflecting on artistic, technical, and conceptual learnings, and where these insights shaped the research.

Additionally, each write-up includes **visual documentation**, which may comprise final outputs, process work, and photographs illustrating key stages of development.

While the creative works in this chapter pursue varied technical and aesthetic directions, they all serve the same research purpose: to iteratively explore, test, extend, inform, and refine the tools, methods, and pedagogical strategies developed through this PhD. The primary function of this chapter is to document the material evidence and process work of this endeavour. **Accordingly, it provides a deliberately concise treatment of the descriptions, details of creative process, and reflective commentary for each output**, leaving the Publications and Presentation Outputs chapters to contextualise and elaborate on the broader insights and implications.

6.1 North

Exhibition/Event Name: ADA Network Symposium 2021

Author(s): Bunn, T.

Venue: Wellington, New Zealand: Massey University College of Creative Arts

Start/Finish Dates: 04 December 2021

URL: https://ada.net.nz/tag/symposium2021

Acceptance Letter/Invitation or Event Programme: See Appendix E.1

6.1.1 Description & Context

These works consisted of generative pen plotter drawings created with Python code and rendered in coloured inks on a range of paper stocks. They featured as part of ADA's 10th symposium, *Indeterminate Infrastructures*, which examined how creative practices engage with the uncertain architectures of contemporary media systems. ADA (Aotearoa Digital Arts Network) connects New Zealand artists, curators, educators, and the public to advance digital and electronic art through collaboration, exhibitions, and education.

6.1.2 Creative Process

The workflow relied almost entirely on Python, utilising Thonny-py5mode for coding and executing sketches, with Inkscape solely for sending plotting instructions. The code generated wave- and ripple-based mathematical patterns, producing spherical and radial line formations that shifted between structured order and chaotic variation. Transferring precise vector coordinates into ink through a pen tip and onto a textured paper surface introduced subtle irregularities and 'failures,' adding further layers of visual complexity and interest to the plotted output.

6.1.3 Impact & Reflection

This series contributed to the PhD research by demonstrating how Thonny-py5mode and SVG Python libraries can support pen-plotting workflows for generative art. Beyond generating line art, the project tested new approaches to layering SVG markup (for colour switching) and vector path optimisation. The resulting insights informed several of the outputs discussed in the Presentation Outputs chapter—most directly the talks detailed in Sections 5.6 and 5.8.

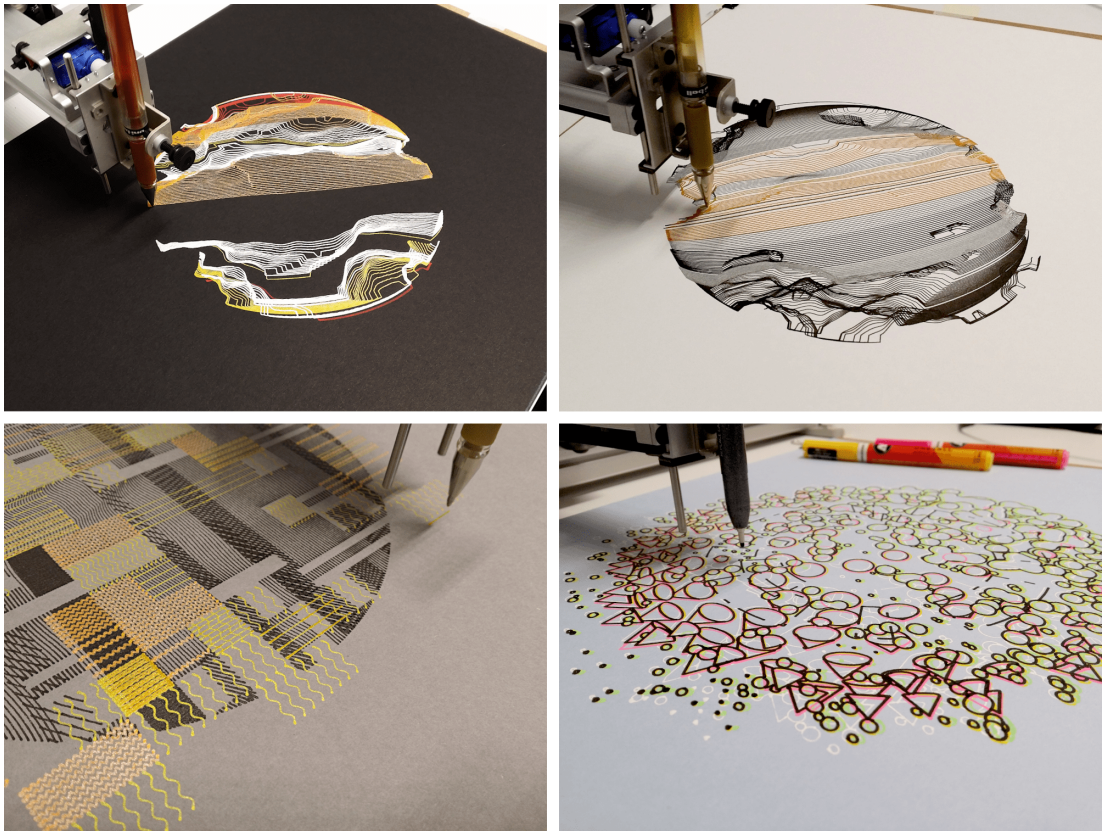


Figure 6.1: Plotting spherical line formations generated using Thonny-py5mode. Photos by the author.

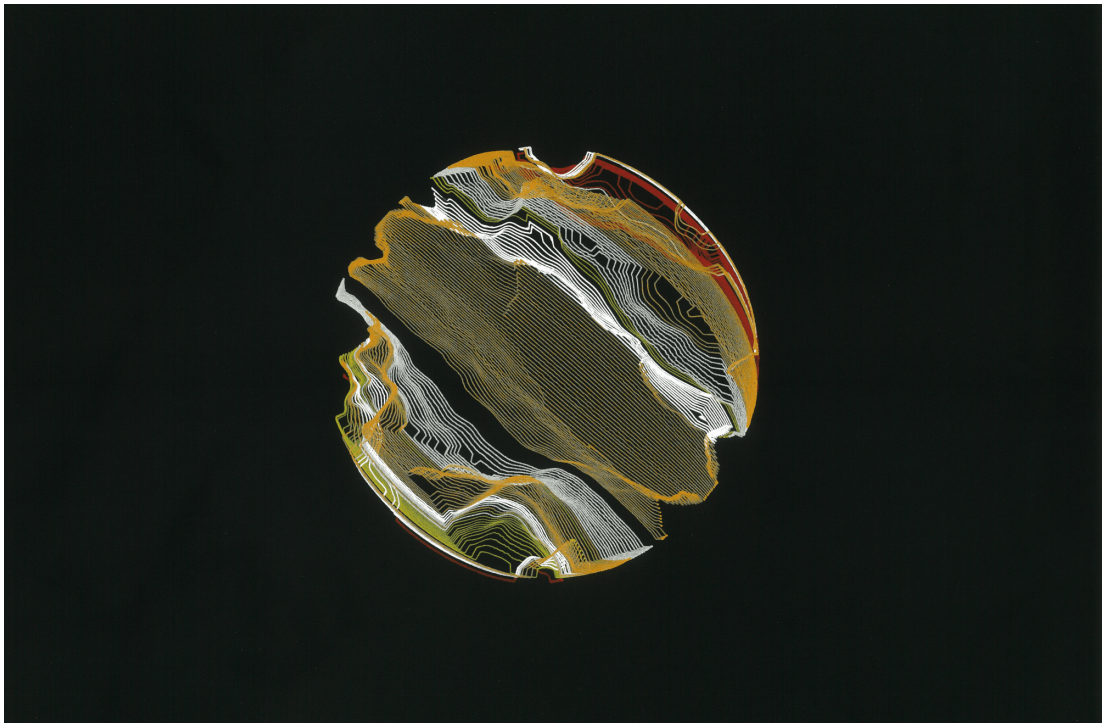


Figure 6.2: A finalised plot from the *North* series. Metallic inks on black card. Artwork by the author.

6.2 Destruction of Kepler-186f

Exhibition/Event Name: {Between} 2021

Creator(s): Bunn, T.

Venue: Coimbra, Portugal: Nest Collective

Start/Finish Dates: 08 December 2021 – 10 December 2021

URL: https://pcdcoimbra.dei.uc.pt/2021/exhibition/between

Acceptance Letter/Invitation or Event Programme: See Appendix E.2

6.2.1 Description & Context

A generative, plotted artwork exhibited at *{Between} An Inventory of Anachronic Practices*, an international exhibition themed around anachronism. Panellists received 60 submissions in response to an open call and, through a double-blind review process, selected 24 artworks to exhibit. This piece speculated on humanity's precarious place in the universe, presenting intelligent life as likely elsewhere, and positioning our technological and societal state as a potential *anachronism* in cosmic time. Specifically, this final poster depicted the annihilation of Kepler-186f, one of multiple generative outputs that visualised the destruction of some planet.

6.2.2 Creative Process

Again, this work employed a generative coding approach, utilising Python code within the Thonny-py5mode environment. Its algorithm relied on hash-based seeding functions—in this case, a deterministic integer derived from the MD5 hash of the string `Kepler-186f`. Each unique string/planet name generated a distinct arrangement of ten panels depicting the annihilation of a that planet, producing outcomes that appeared chaotic yet remained reproducible.

6.2.3 Impact & Reflection

The project further experimented with techniques for generating SVG markup, informing several of the outputs discussed in the Presentation Outputs chapter—most directly the talks detailed in Sections 5.6 and 5.8.

Creatively, it investigated programmatic approaches to comic panel layout and engaged with procedurally generated pictorial narratives (or "silent" comics).

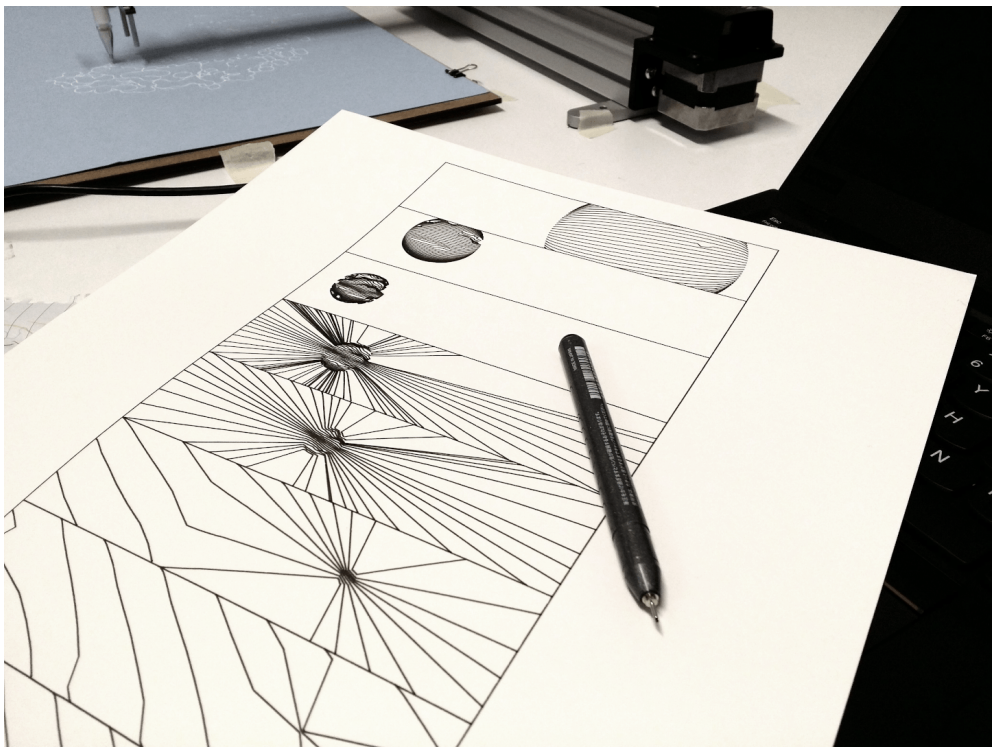


Figure 6.3: Generative plot depicting the destruction of Kepler-186f. Black fine-tip pen on white paper. Photo by the author.

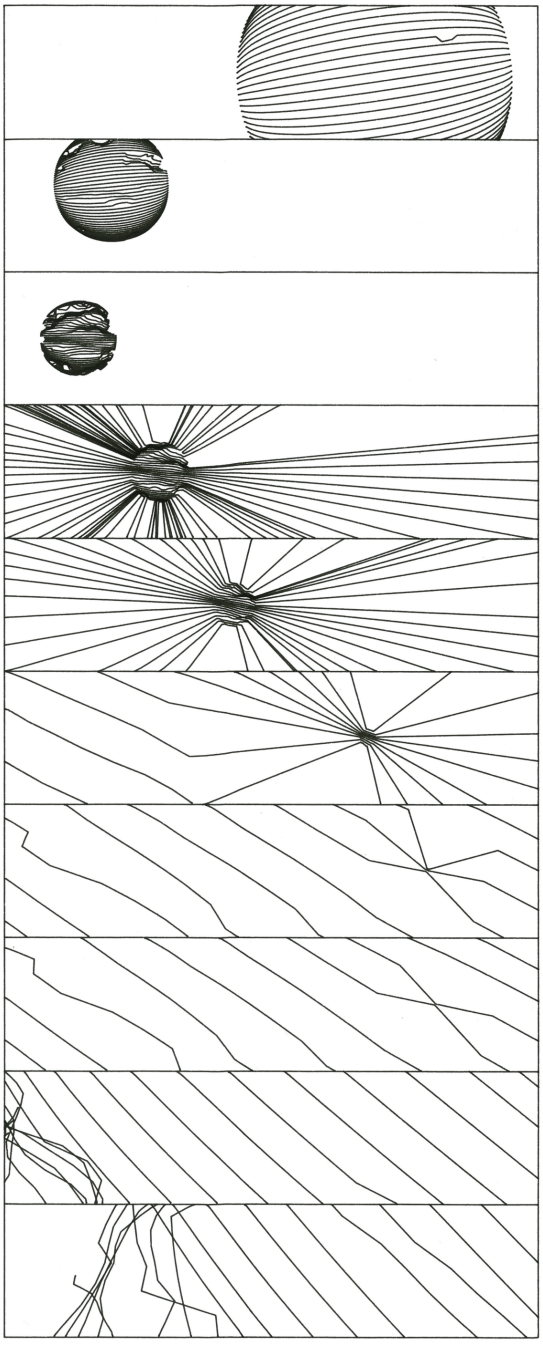


Figure 6.4: *Destruction of Kepler-186f*. Exhibited at *{Between}*, 2021. Artwork by the author.

6.3 South

Exhibition/Event Name: LINK 2021

Author(s): Bunn, T.

Venue: Auckland, New Zealand: Auckland University of Technology

Start/Finish Dates: 01 December 2021 – 03 December 2021

URL: http://dx.doi.org/10.24135/link2021.v2i1.167

Acceptance Letter/Invitation or Event Programme: See Appendix E.3

6.3.1 Description & Context

A series of generative artworks exploring wave motion through pen-plotter drawing, presented at the LINK Conference. Established in 2019, LINK is a platform for dialogue and knowledge exchange, particularly across the Global South, aiming to create links between diverse practices, contexts, and epistemologies in art and design. South extends techniques developed in the North series.

6.3.2 Creative Process

Whereas the North works sought to establish a Thonny-py5mode workflow for plotter art, this series foregrounded experimentation. Figure 6.5 tested the limits of the 2-axis plotter machine, burning out servo motors through highly detailed and prolonged plotting. The series also incorporated experiments with moiré patterns (Figure 6.6, top right) and with layering plotter drawings over prints of generative raster imagery (Figure 6.6, bottom right).

6.3.3 Impact & Reflection

The series reflected continued experimentation with Python-based creative coding techniques, feeding back into Thonny-py5mode development and informing several of the outputs discussed in the Presentation Outputs chapter—most notably the talks detailed in Sections 5.6 and 5.8, as well as aspects discussed in Section 5.5.

As with earlier plotter works, it demonstrated the translation of algorithmic techniques into material outcomes, foregrounding the variability inherent in such processes. This includes the deliberate use of pens that intermittently dried out before resuming, visible in the red lines at the top right of Figure 6.6.

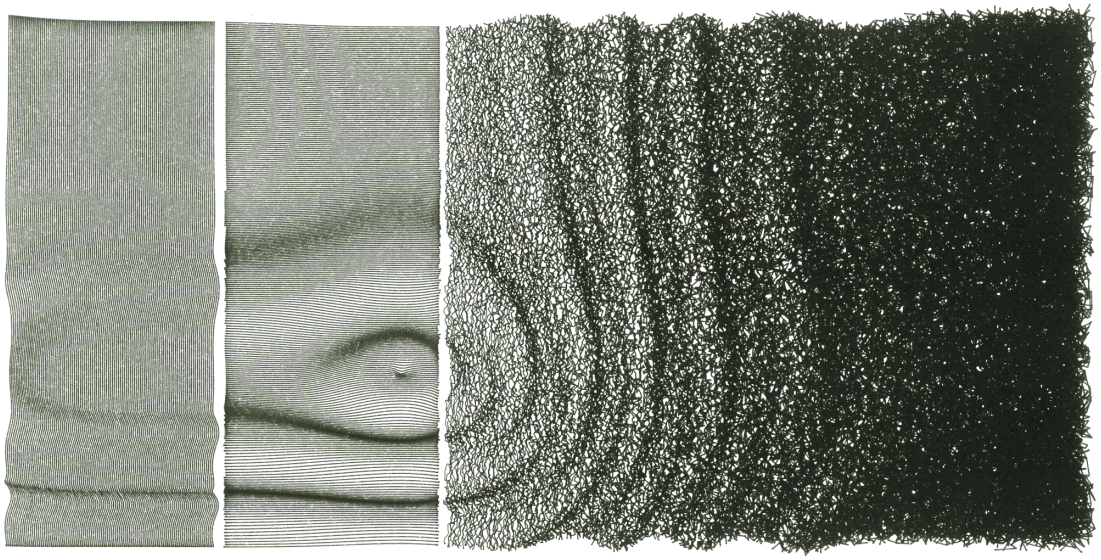


Figure 6.5: Black fine-tip pen on white paper. Approximate plotting time: 12 hours. Artwork by the author.

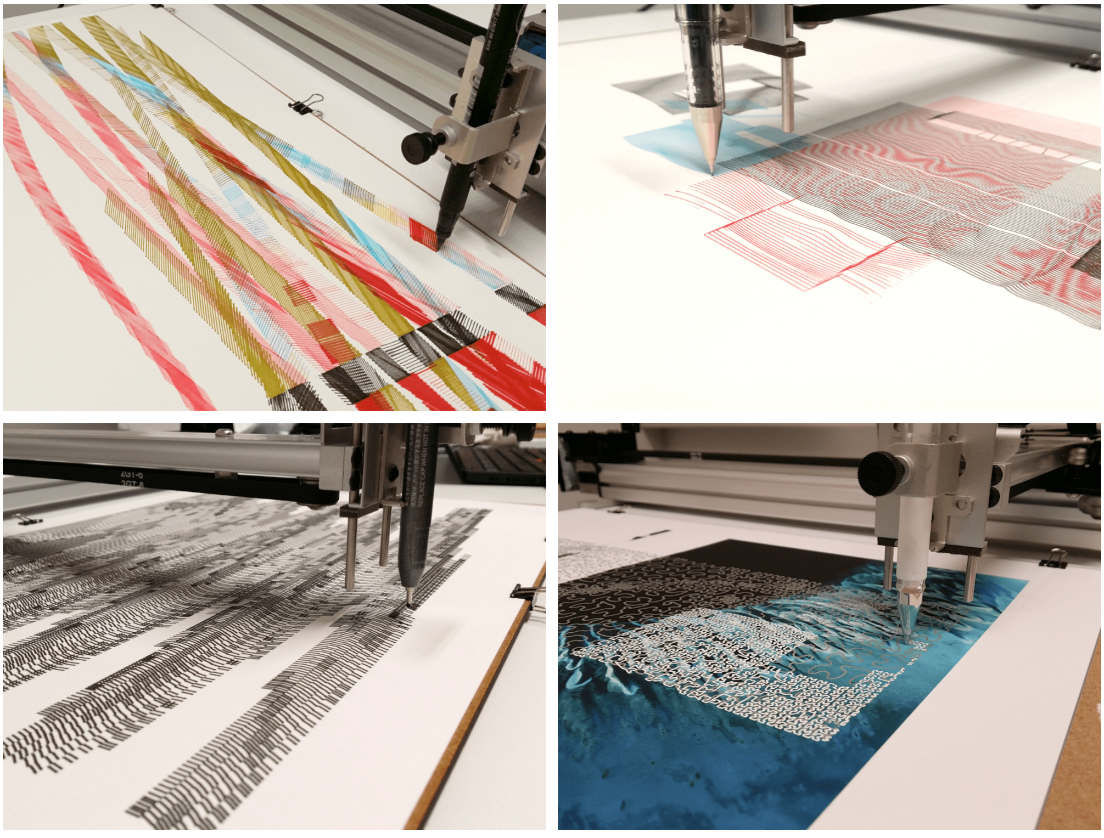


Figure 6.6: Plotting moiré patterns (top right), and over raster prints (bottom right). Photos by the author.

6.4 Relics U+130C8

Exhibition/Event Name: Motyf 2022

Author(s): Bunn, T.

Venue: Warsaw, Poland: PJAIT

Start/Finish Dates: 25 March 2022 – 15 April 2022

URL: https://motyf2021.webflow.io/#motyf_exhibition

Acceptance Letter/Invitation or Event Programme: See Appendix E.4

6.4.1 Description & Context

A generative plotter-drawing series exhibited at Motyf: an international symposium and media art exhibition held biennially. Motyf 2021 (delayed to 2022 due to the pandemic) centred around the theme "Communicating Complexity." This work marked a shift away from mechanical to more organic forms, seeking to emulate hand-drawn qualities and processes of gradual unravelling—suggesting that complexity is best understood through its vulnerabilities as much as its structures.

6.4.2 Creative Process

Again, this series employed Python code written and run through the Thonny-py5mode environment, intentionally employing randomness and decay to introduce irregularities, resulting in robotically plotted drawings that straddle the line between algorithmic precision and human-like mark-making. This included experimenting with the broadest markers the plotter could handle (Figure 6.8, left) and the simulation of roughly masked areas (Figure 6.7).

6.4.3 Impact & Reflection

The series reflected ongoing experimentation with diverse Python-based creative coding techniques, feeding back into *Thonny-py5mode* development and informing several of the talks discussed in the Presentation Outputs chapter—most notably those in Sections 5.6 and 5.8, as well as aspects of Section 5.5.

Beyond technical exploration, *Relics U+130C8* examined how generative systems might mimic, diverge from, or potentially hybridise with human drawing practices.

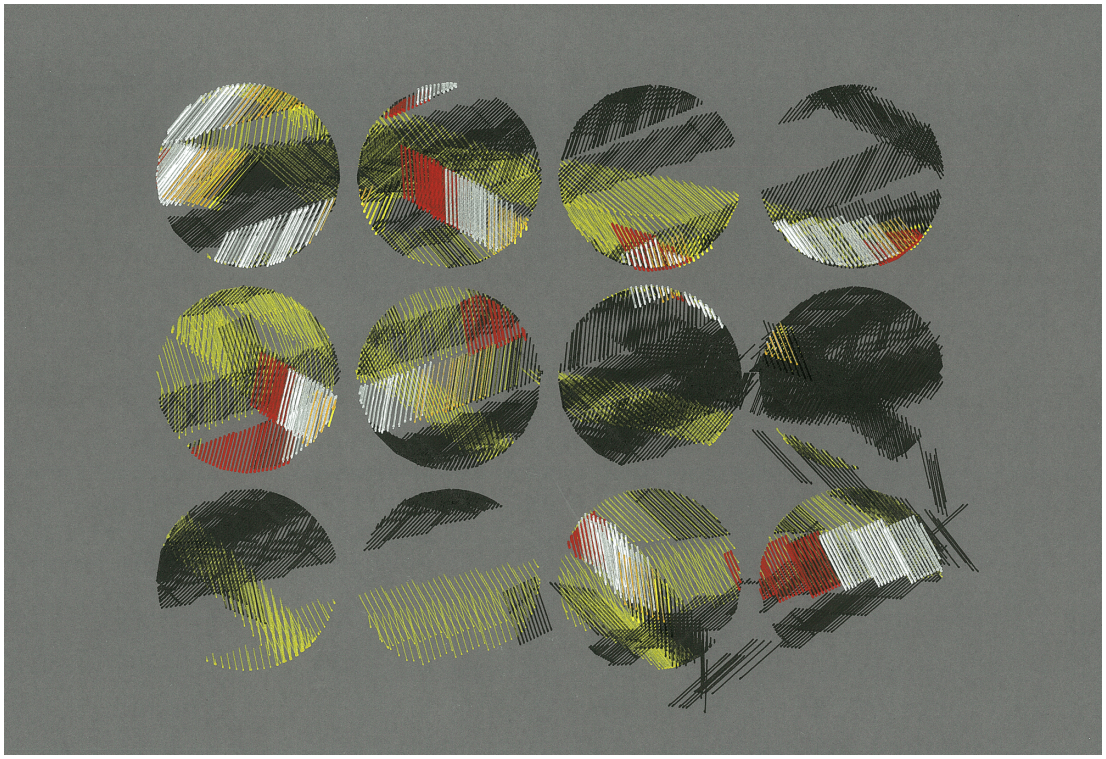


Figure 6.7: Coloured pens on grey card. Accuracy decreases progressively as the rows of circles advance rightward and downward. Artwork by the author.

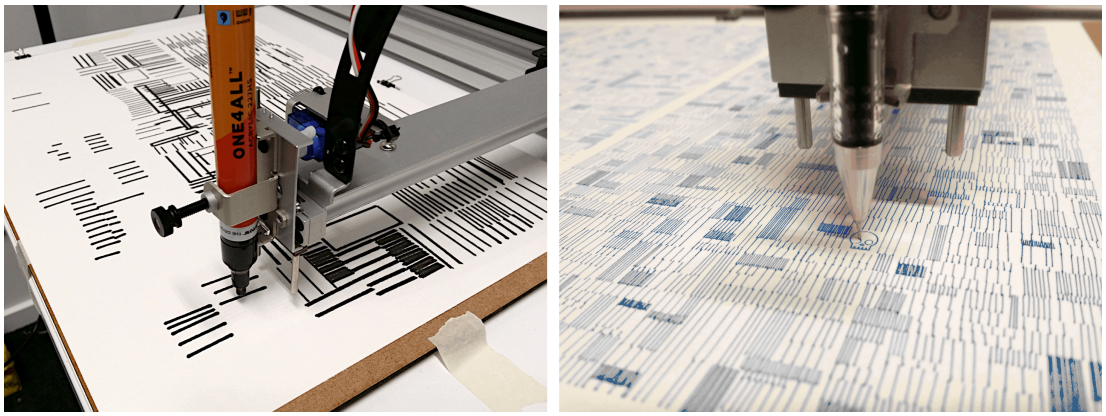


Figure 6.8: Experimenting with plotting more 'human-like' markings. In effect, machines emulating humans emulating a machines. Photo by the author.

6.5 Digital Aquatics

Publication: Processing Community Catalog 2001–2021

Author(s): Processing Foundation; Editors: Reas, C., and McCarthy, L.

Details: Hardcover, ISBN-13: 978-0999881323

Publication Date: January 2022

URL: https://archive.org/details/processing-community-catalog-2021

Acceptance Letter/Invitation or Event Programme: See Appendix E.5

6.5.1 Description & Context

A contribution to the *Processing Community Catalogue*, published to commemorate two decades of Processing and its related initiatives, including its JavaScript, Python, Ruby, and Raspberry Pi variants. This work is a Thonny-py5mode adaptation of a sketch that transforms mathematical formulas into playful, amoeba-like forms. Additionally, it featured in promotional materials for the catalogue's international open call: https://twitter.com/ProcessingOrg/status/1439283507679797250.

6.5.2 Creative Process

The Python code adapted Johan Gielis' Superformula, building on Lieven Menschaert's NodeBox project[1] *Aquatics!*. Each run of the sketch generated a unique organism with varying shapes, colours, and other visual details. No two creatures are alike. However, they always possess at least three eyes, may (or may not) sprout hair along their outlines, and can appear to sway in currents simulated by directional noise.

6.5.3 Impact & Reflection

Again, this work employed Python code written and run through Thonny-py5mode, demonstrating how this approach can produce outputs that evoke a humorous and whimsical feel.

It also represents a bridge between Processing.py techniques (Sections 4.1, 5.1, 5.2) and py5, demonstrating that Thonny-py5mode can reliably replace Processing.py. In doing so, it establishes Thonny-py5mode's credibility as a Python 3 successor to Processing's discontinued Python Mode.

[1] https://nodebox.net/code/index.php/Aquatics

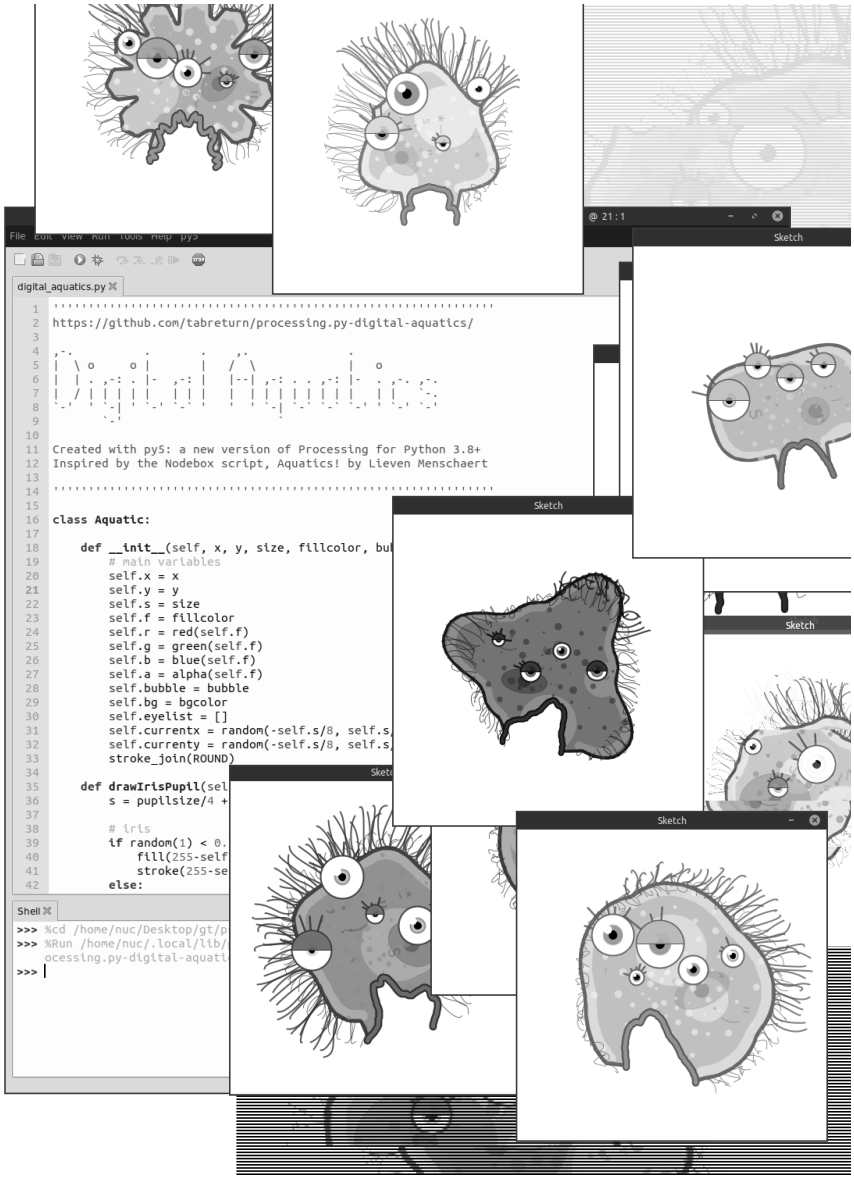


Figure 6.9: Greyscale artwork published in the *Processing Community Catalog 2001–2021* (printed on lilac paper), inspired by Lieven Menschaert's NodeBox script *Aquatics!*. Artwork by the author.

6.6 Chapter Summary

This chapter has presented a portfolio of creative works developed over the course of the PhD, collectively demonstrating how Thonny-py5mode facilitated novel approaches to generative drawing, while the processes and outputs, in turn, informed its ongoing development. Together, these works contribute practical examples to academic discourse on advancing Python programming education through creative computing environments, and can inspire further exploration and experimentation within the creative coding community.

These works substantively advance three of the thesis' Research Objectives: showcasing the development and application of custom Python-based tools; evidencing the production of credible creative outputs; and extending dissemination and engagement through international exhibitions and publications.

This chapter deliberately maintained concise descriptions, deferring to the Publications and Presentation Outputs chapters to contextualise and elaborate on the broader insights and implications of the creative work. These subsequent discussions demonstrate the innovative use of Python-based creative coding environments, tools, and techniques, and highlight the artistic and technical achievements these solutions enable.

7 Conclusion

This PhD study encompassed the development of new software and its empirical evaluation, the design of learning materials, and the production of creative works. Adopting a folio/publication format, the thesis charted the research journey through a curated folio of publications, presentations, and creative outputs unified by a cohesive narrative. These efforts primarily focused on novel Python tools and techniques to advance pedagogical practice and scholarly understanding of Python programming education concerning creative computing environments.

The Introduction situated creative coding within the broader context of graphical programming education, starting with early environments such as Turtle Graphics and progressing to Processing and p5.js. It outlined Python's established multimedia ecosystem and framed three interrelated strands of investigation: **tools/software development**, **curricula** & **documentation**, and **creative code artwork**. These co-evolved to produce new tools, materials, and techniques that lower barriers to entry, broaden participation, and enhance Python learning. The outcomes foster programming literacy, which enhances graduates' industry-relevant capabilities, particularly in disciplines at the intersection of creativity and technology, such as VFX, video games, VR, and interactive media.

The research journey originated with Processing's Python Mode (Processing.py), which inspired the publication of the sole-authored book, *Learn Python Visually: Creative Coding with Processing.py*. Published by the reputable No Starch Press, it formed a significant component of the folio. However, Jonathan Feinberg's subsequent discontinuation of the Processing.py project motivated inquiry into alternative approaches, leading to the development of a replacement solution: Thonny-py5mode, another key element of the folio.

Peer-reviewed articles published in Q1/Q2 journals examined both established and newly developed tools, techniques, and curricula for Python-based creative computing, incorporating findings from Thonny-py5mode user studies conducted as part of this research. A series of presentations at leading

international conferences further disseminated and contextualised these contributions, underscoring the research's significance and credibility.

This "Conclusion" chapter summarises the thesis' key findings, contributions, and implications. It revisits the research objectives and questions, situating the results within broader scholarly and pedagogical discussions surrounding creative computing environments for Python education. The chapter concludes with a reflection on the PhD's strengths and limitations, followed by a discussion on directions for future research.

7.1 Revisiting the Research Objectives

A set of overarching Research Objectives (ROs) guided this study, defined to break down the research aims into specific, actionable, and measurable outcomes. The following subheadings address each objective in turn, outlining the original contributions this thesis makes to each domain.

RO1 Tool Development

The Software, Publications, and Presentation Outputs chapters chronicle the PhD study's exploration and development of new Python-based creative computing tools/environments, with a principal focus on the conception and refinement of Thonny-py5mode: a bespoke plugin that integrates the py5 library into the Thonny IDE. Inspired by Processing's Python Mode, it lowers barriers to programming by providing an environment for exploring graphical output with Python 3 code. Specifically, it streamlines access by simplifying setup and providing a purpose-built workspace with assistive features.

Through an extensive survey of existing tools, the MTAP article, *Towards a Python 3 Processing IDE for Teaching Creative Programming*, highlighted the need for a new Python 3 successor to Processing's Python Mode. It detailed Thonny-py5mode's technical implementation and design rationale, demonstrating how it effectively integrates Thonny and py5 to provide a beginner-friendly environment. The JISE article, *Evaluating Task–Technology Fit in Thonny-py5mode*, applied the Task–Technology Fit framework to evaluate Thonny-py5mode and concluded that its design and functionality enhanced usability, engagement, and learning outcomes.

Thonny-py5mode demonstrates sustained development and widespread adoption, affirming its effectiveness in educational contexts. Solo development began in 2021 and progressed through nine releases by mid-2024, after which multiple developers joined the project. In 2025, the py5 organisa-

tion assumed stewardship, cementing its status as an official Python creative coding environment. The project also received support from the Processing Foundation through its Google Summer of Code (GSoC) initiative. Distributed via GitHub and PyPI, Thonny-py5mode has surpassed 34,000 installations. Educational providers have incorporated it into curricula at institutions such as Massey University and Torrens University, as well as in Domestika's online course, *Designing with Python*. For further detail, the Software chapter provides comprehensive coverage.

Together, these elements demonstrate the pedagogical impact of the tools developed through this PhD, positioning them as both technical contributions and platforms for advancing Python education within creative computing. Thonny-py5mode now stands as a prominent and accessible successor to Processing's discontinued Python Mode, having matured into an actively maintained project that extends the py5 library, supported by comprehensive official documentation hosted at py5coding.org.

RO2 Educational Materials

Early folio entries centre on Processing.py, notably *Learn Python Visually: Creative Coding with Processing.py*, which synthesised extensive research, adaptation, and the development of new pedagogical techniques for Processing's Python Mode. As part of the book's development, the author presented components at major Python venues (§§ 5.1, 5.2), affirming the quality and novelty of these contributions while enabling expert feedback and reflection. The book translated these insights into structured learning materials, informed by strategies documented across numerous works on Processing and the author's own classroom experience. Its publication by No Starch Press (est. 1994), together with a technical review by Paddy Gaunt (maintainer of pi3d), underscores the work's rigour. Indicators of reach include 2,492 direct sales as of Dec 2024, inclusion in two Humble Bundle collections that together sold more than 25,000 digital copies, and global distribution. Collectively, these outcomes demonstrate both pedagogical effectiveness and broad adoption and accessibility.

The shift from Processing.py to the development of Thonny-py5mode necessitated new learning materials focused on the latter. Thonny-py5mode provides setup documentation on GitHub and PyPI, and once installed, a drop-down menu links to a py5 cheatsheet—a printable reference summarising key syntax, structures, and functions to support rapid learning and recall. The (PhD) author has licensed the cheatsheet design files as open-source, allowing educators to adapt and extend them. The Thonny-py5mode workspace also includes a link to the official py5 documentation, containing a set of tutorials developed as part of a joint Thonny-py5mode–py5 project accepted under the Processing

Foundation's Google Summer of Code 2022 initiative; these materials essentially represent a Thonny-py5mode adaptation of *Learn Python Visually: Creative Coding with Processing.py*, with additional sections that accommodate py5's ability to leverage CPython-3 libraries (§§ 3.3, 5.5).

These educational materials collectively provide educators and learners with accessible, practical entry points that bridge abstract programming concepts with graphical and multimedia outcomes. As demonstrated in the Literature Review, such pedagogical methods make programming both approachable and meaningful for students in creative fields of study. Moreover, this material extends the pedagogical reach of Thonny-py5mode.

RO3 Creative Outputs

The Creative Works chapter documented a portfolio of artistic outputs that demonstrated the creative potential of the tools and techniques developed through this PhD research, predominantly employing Thonny-py5mode. Each output underwent peer or curatorial review, evidencing both creative and technical significance.

Notably, the chapter positioned creative artefacts as outcomes of experimental tool development rather than traditional art practice, aligning with practice-led research methodologies in which making and reflection operate in continuous feedback (as described by Brown & Sorensen and Paul). Rooted in open-source and hacker-culture principles, the creative process and subsequent tools development adopted iterative, community-informed, and agile methods.

The works *North*, *Destruction of Kepler-186f*, *South*, and *Relics U+130C8* all employed a two-axis pen plotter. Along with *Digital Aquatics*, they collectively: (1) demonstrated the innovative application of new Python-based creative coding environments and techniques; (2) highlighted the artistic and technical achievements made possible through these tools; (3) inspired further exploration and experimentation within the creative coding community; and (4) contributed practical examples to academic discourse on advancing Python programming education through creative computing environments. These contributions are formally articulated in the Presentation Outputs (§§ 5.5–5.8) which detail how Thonny-py5mode enables generative, algorithmic drawing and SVG experimentation expressed through tangible plots.

Together, these works advanced several of the research objectives by informing the development of Thonny-py5mode, which facilitates visual learning contexts, supports artistic exploration, and strengthens the link between computational thinking and creative expression.

RO4 Empirical Evaluation

Extensive experimentation, survey work, and analysis of related creative-coding environments and Python tools informed the design and development of Thonny-py5mode.

To evaluate its effectiveness, the JISE article *Evaluating Task–Technology Fit in Thonny-py5mode* reported a quantitative study that employed an extended Task–Technology Fit (TTF) framework. This model incorporated *personalised learning*, *hedonic motivation*, and *effort expectancy* to predict behavioural intention. The study analysed 143 valid responses from a 100-level university Python cohort following four weeks of Thonny-py5mode use and an assessment with six graphical tasks. Researchers collected data via a structured 5-point Likert survey refined through expert review and administered with ethics approval (TUA HREC application #0394).

Using a CFA/SEM pipeline (AMOS v26), the measurement model demonstrated strong reliability and convergent validity, with high factor loadings and composite reliabilities, and indices indicating good model fit. The structural model then tested nine hypotheses, finding support for eight. *Task characteristics*, *technology characteristics*, *hedonic motivation*, and *effort expectancy* each positively influenced perceived TTF, which in turn predicted adoption. However, personalisation did not improve task characteristics or TTF (showing a negative association), suggesting a need to embed adaptive elements more tightly into task scaffolding rather than at the interface layer.

Collectively, and consistent with the broader literature (§§ 1.2.1–1.2.3, §§ 4.3.3–4.3.8), the results show that Thonny-py5mode enhances usability and reduces cognitive load, thereby increasing enjoyment, strengthening perceived fit, and supporting both learning outcomes and continued use. It achieves this through an integrated sketch runner, immediate graphical feedback, simplified debugging, and a low-friction setup.

Methodologically, the work advances TTF by foregrounding motivational constructs alongside functional alignment within a creative-computing context. Practically, it provides evidence-based design priorities for Python IDEs in education, including: scaffolded, graphics-driven tasks; intuitive interfaces that minimise effort; and personalisation strategies that align with assessment structures. Overall, the study establishes both the scholarly contribution and applied value of Thonny-py5mode—positioning it as a robust successor to Processing's Python Mode and an effective environment for improving engagement and comprehension in introductory Python offerings.

RO5 Broader Dissemination

The Publications and Presentation Outputs chapters documented the dissemination of this research through Q1 and Q2 journal articles (§§ 4.2–4.3), international conference presentations, and community workshops (§§ 5.1–5.10).

The journals *MTAP* and *JISE*, as well as established venues such as PyCon, Libre Graphics Meeting (LGM), and SIGGRAPH, employ rigorous peer review and curation processes that attest to the quality, originality, and technical contributions of these outputs. These channels not only validated the research's scholarly merit but also facilitated dialogue with global communities of educators, developers, and artists working at the intersection of Python and creative coding.

Complementing these scholarly and professional channels, the Creative Works chapter demonstrated the practical and artistic applications of the tools and techniques developed through this PhD study (§§ 6.1–6.5). These exhibitions, invited demonstrations, and published catalogues further extended the visibility of Thonny-py5mode, providing tangible evidence of its creative potential.

Together, these dissemination activities—spanning academic publishing, open-source software releases, professional conferences, and artistic exhibitions—ensured that the research reached both scholarly and practitioner audiences. Collectively, the outputs have fostered the ongoing adoption of Thonny-py5mode and new techniques across university curricula, workshops, and online learning platforms, evidencing a real-world impact and inspiring further advancements in Python-based creative computing education and practice.

7.2 Readdressing the Research Questions

The preceding discussion connected specific contributions to each research objective; in this section, these elements integrate to address the central questions. The responses situate the PhD within the landscape of Python-based learning approaches that incorporate visual output, linking the technical outcomes of Thonny-py5mode and its surrounding ecosystem to broader implications for curriculum design, pedagogy, and creative practice.

RQ: How can we enhance creative computing environments through new Python tools and practices to improve programming education in visual learning contexts?

The thesis addressed the research question through a combination of software innovation, empirical investigation, and creative experimentation. The development (RO1) and evaluation

(RO4) of Thonny-py5mode demonstrated how new Python tools and practices can advance creative computing environments. The plugin's accessible, graphically oriented features bridged the gap between technical programming instruction and creative exploration, while supporting comprehension through visual representations of Python programming concepts (RO2). The environment's low-friction setup and immediate visual feedback exemplify this advancement, building on motivational principles established in studies of Processing, p5.js, and similar software (§§ 4.3, 2.1.3). Integration into university curricula and professional courses, recognised through academic and professional dissemination (RO3, RO5), provides evidence of both educational utility and community adoption.

Collectively, these outcomes demonstrate that creative computing contexts can meaningfully enhance engagement and learning in Python programming fundamentals settings, particularly for students from creative disciplines.

Three sub-questions further defined the thesis focus and research goals—

SQ1: How can Python-based software, similar to Processing.py, be designed and developed to best support creative coding practices?

The design and evolution of Thonny-py5mode demonstrated how to extend Processing.py's pedagogical strengths to a Python 3 environment while maintaining accessibility and familiarity (RO1). By embedding py5 within Thonny, the research created a sustainable successor to Processing's Python Mode within a modern, beginner-friendly IDE (§ 4.3). Developed through an iterative, open-source process, Thonny-py5mode aligns with best practices in participatory software design and educational tool research, with its development informed through practical experimentation and the creation of artwork (RO3). The project's adoption by the py5 organisation in 2025 (§ 3.4) further validates its success and long-term viability within the creative coding ecosystem.

SQ2: How effective are the proposed tools and techniques in improving learning outcomes and student engagement?

Empirical evaluation (RO4), conducted through a TTF study published in JISE (§ 4.3), confirmed that Thonny-py5mode significantly enhances perceived task–technology fit, enjoyment, and intention to continue use. These findings empirically substantiate long-held assumptions

in creative computing pedagogy—that immediate graphical feedback and reduced cognitive friction can improve both engagement and learning outcomes. The inclusion of *hedonic motivation* and *effort expectancy* within the extended TTF model provided new insight into the motivational dynamics of creative computing. While personalisation features did not directly strengthen fit in this study, this result highlights directions for future design (§ 4.3.8).

SQ3: How can creative coding outputs and creations showcase the potential of the developed tools in real-world applications?

The creative works (RO3) serve as both artefacts of artistic research and practical demonstrations of Thonny-py5mode's real-world capabilities, positioning creative outputs as integral to tool development and embodying practice-led research principles where making and reflection operate reciprocally. Through exhibitions, catalogues, and curatorial review (RO5), these works demonstrated how Thonny-py5mode supports generative design, SVG-based workflows, and physical plotting, while integrating myriad Python libraries and opportunities for 3D exploration (§§ 5.8–5.9). The resulting artefacts not only validated the expressive and technical potential of the tools, but also offered exemplars that bridge computational literacy and creative practice, inspiring learners, educators, and practitioners alike.

7.3 Contribution to Knowledge

This thesis makes **five original contributions to knowledge** in the field of Python programming education, particularly within creative computing contexts:

1. **Software Contribution**: Developed Thonny-py5mode, the first fully integrated Python 3 desktop environment for Processing-based creative coding. Through integration with py5, this innovation establishes a viable successor to Processing.py, lowering barriers to entry while supporting graphical and interactive programming.

2. **Pedagogical Contribution**: Authored original, research-informed learning materials—including a sole-authored book and Thonny-py5mode tutorials—that translate creative coding principles into accessible, Python-based curricula. These resources provide educators and learners with practical frameworks for exploring programming through visual and creative contexts.

3. **Empirical Contribution**: Applied and extended the Task–Technology Fit (TTF) framework to creative coding environments, generating new empirical insights into how IDE design influences motivation, usability, and learning outcomes in programming education.

4. **Practice-Based Contribution**: Produced and exhibited original artworks that operationalise the tools developed through this research. These works serve as both artistic outputs and practical demonstrations of Thonny-py5mode's expressive and educational potential, embodying practice-led inquiry.

5. **Dissemination Contribution**: Delivered a multi-modal dissemination of creative computing research spanning peer-reviewed journals, open-source software releases, international conferences, and artistic exhibitions—thereby linking scholarly impact with educational and community adoption.

7.4 Discussion and Future Directions

This thesis has demonstrated the potential of creative computing approaches in Python education, with a particular emphasis on graphical and creative coding techniques. Its contribution lies in integrating software development, pedagogy, and empirical study within a coherent narrative supported by a folio of peer-reviewed outputs.

Nonetheless, it is important to acknowledge several limitations. The empirical evaluation of Thonny-py5mode primarily involved Information Technology students, selected mainly for their availability rather than for disciplinary diversity. Further validation with learners from creative and design-focused disciplines would help extend and strengthen the findings.

The rapid emergence of LLM-powered tools such as ChatGPT and GitHub Copilot has already begun to reshape programming education, warranting further examination of their pedagogical impacts. Had these technologies been available at the outset, this PhD may have followed a markedly different trajectory. Nevertheless, its contributions remain novel, spanning new pedagogical strategies, software tools, and learning materials that can complement or integrate with emerging GenAI-supported teaching practices. Consequently, the topic merits rementioning before identifying directions for future research.

7.4.1 Generative AI

The presentation output, *Mitigating AI Misuse with Graphical Programming Tasks*, illustrated that while GenAI technologies can enhance novice programmers' productivity, they also tend to encourage superficial, prompt-dependent learning behaviours. In the current environment, educators remain concerned with ensuring authentic learning and assessment, especially as LLMs perform strongly on text-based CS1 tasks.

Institutional responses to GenAI have varied widely. Some universities have adopted open and exploratory approaches that promote use for creative and professional development. In contrast, others have implemented restrictive measures, citing concerns over plagiarism, dependency, and the erosion of core coding competencies. Yet even in contexts where GenAI use is restricted, enforcement remains problematic—automated detection tools are unreliable, policy interpretation varies across assessments, and teaching staff often lack the resources to investigate suspicious work.

That said, current GenAI systems exhibit persistent weaknesses in accurately completing visually defined problems, particularly tasks requiring geometric precision, diagrams, or animation. Trials conducted in this research (§ 5.10) suggest that Python-based graphical assessments using Thonny-py5mode can effectively resist GenAI automation. Nevertheless, it remains to be seen how durable this resistance will prove over time, and which specific design features assessment developers might further leverage to sustain robustness in this domain.

7.4.2 Future Research

Building on this thesis' insights and findings, and recognising the growing impact of GenAI technologies, future research might prioritise the following:

1. **GenAI-Resilient Thonny-py5mode Assessment Design**: Conduct a systematic benchmarking of iterative LLM prompting versus one-shot generation for visual programming tasks, and design instructor-facing tools to produce dynamically varied, student-specific assessment tasks.

2. **A py5-Aligned Tutoring Companion**: Investigate approaches inspired by tools such as ShiffBot to design a retrieval-augmented generation (RAG) system that draws on py5 documentation and example repositories, enabling adaptive, context-aware tutoring and feedback mechanisms. Leveraging py5's integration with Jupyter Notebooks, coupled with the Anaconda Agent and

strategic prompting, will help emulate this system (Figure 7.1), providing a rapid environment for experimentation and feasibility evaluation.

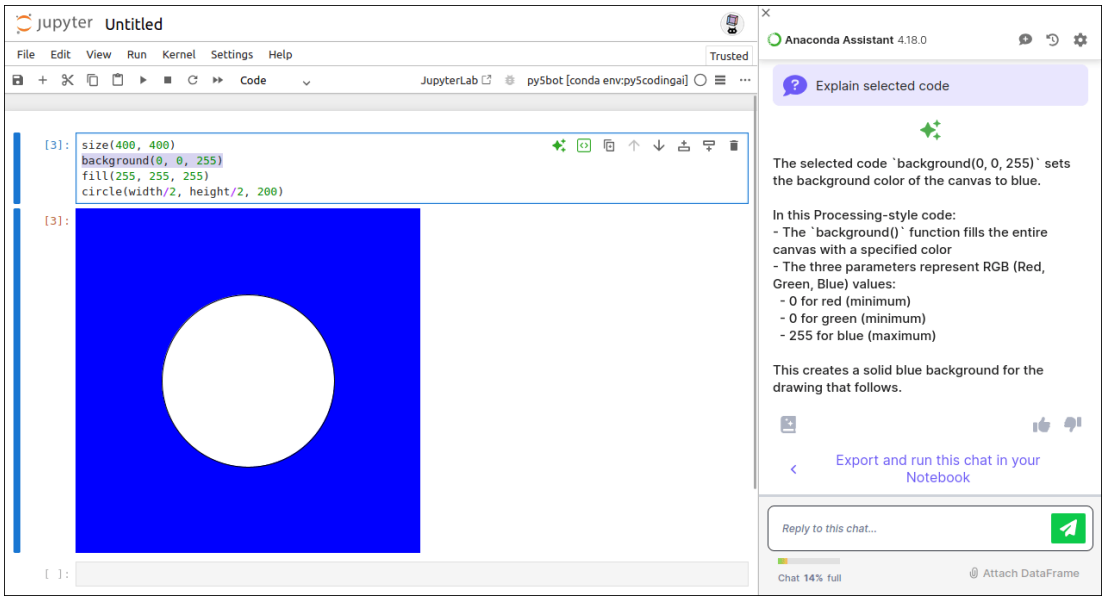


Figure 7.1: py5 running in a Jupyter Notebook, with Anaconda Navigator's AI Assistant used to explain the workings of the py5 `background()` function. Screenshot by the author.

3. **Longitudinal Evaluation**: Conduct multi-semester or multi-institutional studies to assess learning retention, creative self-efficacy, and sustained engagement when using Thonny-py5mode. Such work should also involve students who are more exclusively creative-disciplines-focussed, rather than predominantly IT cohorts.

In addition to these research directions, it is important to address questions of open-source sustainability regarding Thonny-py5mode.

7.4.3 Sustainability

As with many open-source ecosystems, the durability of these interventions depends on sustainability. Both Thonny-py5mode and py5 offer substantial potential for extension but require a clear and realistic funding strategy. This might combine competitive grants, institutional partnerships through curricular adoption, and community sponsorship. Ensuring long-term maintenance, documentation, educator support, and release cadence is vital to sustaining these contributions and ensuring their continued impact [51, 155, 216].

7.5 Closing Statement

In closing, this PhD makes an original and sustained contribution to knowledge at the intersection of Python programming education, creative computing, and software design. It demonstrates how research-led tool development, empirical evaluation, and creative practice can together advance both pedagogical theory and artistic production. Through establishing Thonny-py5mode as a viable successor to Processing's Python Mode, the thesis delivers new creative coding solutions of clear scholarly and practical value.

Presented in a predominantly folio format, the work captures the reflexive interplay between explorative making and reflection-informed software development, setting a precedent for future practice-led investigations in creative computing education. It extends frameworks such as Task–Technology Fit to creative coding contexts and provides open, accessible resources that continue to shape teaching, learning, and creative practice internationally.

Overall, the research deepens understanding of how creative computing enhances engagement and learning in Python education, while leaving an enduring legacy of software, curricula, and artworks that embody this knowledge in practice.

End

Bibliography

[1] Abbad, M. M. M. 2021. Using the UTAUT Model to Understand Students' Usage of E-Learning Systems in Developing Countries. en. *Education and Information Technologies*, 26, 6, (Nov. 2021), 7205–7224. DOI: 10.1007/s10639-021-10573-5.

[2] Abelson, H. and diSessa, A. 1981. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. en. The MIT Press, (June 1981). ISBN: 978-0-262-36274-0. DOI: 10.7551/mitpress/6933.001.0001.

[3] Acampora, P. 2023. *Python Scripting in Blender: Extend the Power of Blender Using Python to Create Objects, Animations, and Effective Add-Ons*. eng. Packt Publishing. ISBN: 978-1-80323-422-9.

[4] ACM SIGGRAPH DAC. 2022. Community. (2022). Retrieved Oct. 9, 2022 from https://dac.siggraph.org/community.

[5] Ajzen, I. 2020. The Theory of Planned Behavior: Frequently Asked Questions. en. *Human Behavior and Emerging Technologies*, 2, 4, (Oct. 2020), 314–324. DOI: 10.1002/hbe2.195.

[6] Alyoussef, I. Y. 2021. E-Learning Acceptance: The Role of Task–Technology Fit as Sustainability in Higher Education. en. *Sustainability*, 13, 11, (June 2021), 6450. DOI: 10.3390/su13116450.

[7] Alyoussef, I. Y. 2021. Massive Open Online Course (MOOCs) Acceptance: The Role of Task-Technology Fit (TTF) for Higher Education Sustainability. en. *Sustainability*, 13, 13, (July 2021), 7374. DOI: 10.3390/su13137374.

[8] Amiri, S. and Islam, M. 2025. Enhancing Python Programming Education with an AI-Powered Code Helper: Design, Implementation, and Impact. en. *Software Engineering*, 11, 1, (Apr. 2025), 1–17. DOI: 10.11648/j.se.20251101.11.

[9] An, Y., Yu, J. H., and James, S. 2025. Investigating the Higher Education Institutions' Guidelines and Policies Regarding the Use of Generative AI in Teaching, Learning, Research, and Administration. en. *International Journal of Educational Technology in Higher Education*, 22, 1, (Feb. 2025). Publisher: Springer Science and Business Media LLC. DOI: 10.1186/s41239-025-00507-3.

[10] Angert, T., Suzara, M. I., Han, J., Pondoc, C. L., and Subramonyam, H. 2023. Spellburst: A Node-Based Interface for Exploratory Creative Coding with Natural Language Prompts. en. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. (Oct. 2023), 1–22. DOI: 10.1145/3586183.3606719.

[11] Annamaa, A. 2015. Introducing Thonny, a Python IDE for Learning Programming. en. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. ACM, Koli Finland, (Nov. 2015), 117–121. ISBN: 978-1-4503-4020-5. DOI: 10.1145/2828959.2828969.

[12] Annamaa, A. 2024. Thonny, Python IDE for Beginners. (Dec. 2024). Retrieved Dec. 1, 2024 from https://thonny.org.

[13] Anthropic PBC. 2025. Claude Opus 4.1. (2025). https://claude.ai.

[14] Ater, T. 2017. *Building Progressive Web Apps: Bringing the Power of Native to the Browser*. eng. (1st ed.). O'Reilly, Beijing. ISBN: 978-1-4919-6165-0.

[15] Ayasrah, F. T. M. 2020. Exploring E-Learning Readiness as Mediating Between Trust, Hedonic Motivation, Students' Expectation, and Intention to Use Technology in Taibah University. *Journal of Education & Social Policy*, 7, 1. DOI: 10.30845/jesp.v7n1p12.

[16] Bakar, M. A., Mukhtar, M., and Khalid, F. 2019. The Development of a Visual Output Approach for Programming via the Application of Cognitive Load Theory and Constructivism. en. *International Journal of Advanced Computer Science and Applications*, 10, 11. DOI: 10.14569/IJACSA.2019.0101142.

[17] Balalle, H. and Pannilage, S. 2025. Reassessing Academic Integrity in the Age of AI: A Systematic Literature Review on AI and Academic Integrity. en. *Social Sciences & Humanities Open*, 11, 101299. Publisher: Elsevier BV. DOI: 10.1016/j.ssaho.2025.101299.

[18] Bälter, O. and Bailey, D. A. 2010. Enjoying Python, Processing, and Java in CS1. *ACM Inroads*, 1, 4, (Dec. 2010), 28–32. DOI: 10.1145/1869746.1869758.

[19] Barczak, A. and Woźniak, H. 2020. Comparative Study on Game Engines. en. *Studia Informatica*, 23, (Dec. 2020), 5–24. DOI: 10.34739/si.2019.23.01.

[20] Bartoli, A., De Lorenzo, A., Medvet, E., and Tarlao, F. 2014. Playing Regex Golf with Genetic Programming. en. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, Vancouver BC Canada, (July 2014), 1063–1070. ISBN: 978-1-4503-2662-9. DOI: 10.1145/2576768.2598333.

[21] Bere, A. 2018. Applying an Extended Task-Technology Fit for Establishing Determinants of Mobile Learning: An Instant Messaging Initiative. en. *Journal of Information Systems Education*, 29, 4, 239. Retrieved Aug. 18, 2025 from https://jise.org/Volume29/n4/JISEv29n4p239.html.

[22] Bere, A. and Rambe, P. 2016. An Empirical Analysis of the Determinants of Mobile Instant Messaging Appropriation in University Learning. en. *Journal of Computing in Higher Education*, 28, 2, (Aug. 2016), 172–198. DOI: 10.1007/s12528-016-9112-2.

[23] Beyeler, A. 2022. Vpype. (2022). Retrieved Oct. 9, 2023 from https://hydra.ojack.xyz/docs.

[24] Bircher, M. 2022. *Numerically Controlled Pen Plotters in Art: Building an Experimental Pen Plotter Allowing for Additional Creative Possibilities*. MA thesis. University of Lapland, Rovaniemi, Finland. https://lauda.ulapland.fi/handle/10024/65132.

[25] Blender Documentation Team. 2024. Freestyle — Blender Manual. (Jan. 2024). Retrieved Oct. 9, 2023 from https://docs.blender.org/manual/en/4.0/render/freestyle.

[26] Bohnacker, H., Groß, B., Laub, J., Bohnacker, H., and Bohnacker, H. 2012. *Generative Design: Visualize, Program, and Create with Processing*. eng. C. Lazzeroni, editor. Princeton Architectural Press, New York. ISBN: 978-1-61689-077-3.

[27] Bown, O., Fraietta, A., Loke, L., and Ferguson, S. 2020. Creative Coding and Interaction Design for Media Multiplicities: Challenges, Paradigms and Frameworks. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction* (TEI '20). Association for Computing Machinery, New York, NY, USA, 877–880. ISBN: 978-1-4503-6107-1. DOI: 10.1145/3374920.3374966.

[28] Bradfield, C. 2023. *Godot 4 Game Development Projects: Build Five Cross-Platform 2D and 3D Games Using One of the Most Powerful Open Source Game Engines*. eng. (2nd ed.). Packt Publishing. ISBN: 978-1-80461-562-1.

[29] Braun, A. 2023. *XR Development with Unity: A Beginner's Guide to Creating Virtual, Augmented, and Mixed Reality Experiences Using Unity*. eng. (1st ed.). Packt Publishing Limited, Birmingham. ISBN: 978-1-80512-004-9.

[30] Brodbeck, F. 2024. Cinemetrics. (2024). https://cinemetrics.site/?selection=Quantum%20of%20Solace.

[31] Budiartha, C. I. W. E., Redi, A. A. N. P., Halim, G. R., Rachmah, A., and Grahani, B. P. 2024. Using Task-Technology Fit to Analyze the Adoption of Google Workspace for Education in Teaching and Learning Process. In (May 2024), 873–882. DOI: 10.22492/issn.2189-101X.2024.69.

[32] Bunn, T. 2021. *Learn Python Visually: Creative Coding with Processing.py*. No Starch Press, San Francisco, California, USA. ISBN: 978-1-7185-0097-6. https://nostarch.com/learn-python-visually.

[33] Bunn, T., Anslow, C., and Lundqvist, K. 2024. Towards a Python 3 Processing IDE for Teaching Creative Programming. en. *Multimedia Tools and Applications*, 83, 38, (Oct. 2024), 86247–86260. DOI: 10.1007/s11042-024-20345-1.

[34] Bunn, T. and Carrasco, T. 2023. Blender Scripting for Creative Coding Projects. In *SIGGRAPH Asia 2022 Courses* (SA '22). Association for Computing Machinery, New York, NY, USA. ISBN: 978-1-4503-9474-1. DOI: 10.1145/3550495.3558222.

[35] Burrough, X. and Mandiberg, M. 2009. *Digital Foundations: Intro to Media Design with the Adobe Creative Suite*. New Riders in association with AIGA Design Press, Berkeley, Calif. ISBN: 978-0-321-55598-4.

[36] Busolli, L. and Rangone, A. 2023. NFT in Art Business: Real Opportunity or Short-Term Novelty? A Theoretical Approach. en. *Economia Aziendale Online*, Vol 14, (Dec. 2023), 1235–1252 Pages. DOI: 10.13132/2038-5498/14.4.1235-1252.

[37] Buttfield-Addison, P., Buttfield-Addison, M., Nugent, T., and Manning, J. 2022. *Practical Simulations for Machine Learning: Using Synthetic Data for AI*. (First ed.). O'Reilly Media, Inc, Sebastopol, CA. ISBN: 978-1-4920-8992-6.

[38] Caravati, M. and Tatar, K. 2024. Interfacing ErgoJr with Creative Coding Platforms. en. In *Proceedings of the 9th International Conference on Movement and Computing*. ACM, Utrecht Netherlands, (May 2024), 1–4. DOI: 10.1145/3658852.3659082.

[39] Carey, H., Florisson, R., and Giles, L. 2019. Skills, Talent and Diversity in the Creative Industries. en. Tech. rep. The Work Foundation, United Kingdom, (Jan. 2019).

[40] Cass, S. 2022. Top Programming Languages 2022. English. (Sept. 2022). Retrieved Aug. 15, 2022 from https://spectrum.ieee.org/top-programming-languages-2022.

[41] CC2020 Task Force. 2021. Chapter 7: Curricular Design. In *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, New York, NY, USA. ISBN: 978-1-4503-9059-0.

[42] Center for Open Education. 2025. Python for Everybody: Exploring Data Using Python 3. (2025). Retrieved July 7, 2025 from https://open.umn.edu/opentextbooks/textbooks/336.

[43] Center for Open Education. 2025. Think Python: How to Think Like a Computer Scientist - 2e. (2025). Retrieved July 7, 2025 from https://open.umn.edu/opentextbooks/textbooks/43.

[44] Chao, C.-M. 2019. Factors Determining the Behavioral Intention to Use Mobile Learning: An Application and Extension of the UTAUT Model. *Frontiers in Psychology*, 10, (July 2019), 1652. DOI: 10.3389/fpsyg.2019.01652.

[45] Cheng, Y., Sharma, S., Sharma, P., and Kulathunga, K. 2020. Role of Personalization in Continuous Use Intention of Mobile News Apps in India: Extending the UTAUT2 Model. en. *Information*, 11, 1, (Jan. 2020), 33. DOI: 10.3390/info11010033.

[46] Chibalashvili, A., Savchuk, I., Olianina, S., Shalinskyi, I., and Korenyuk, Y. 2023. Creative Coding as a Modern Art Tool. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, 14, 2, (June 2023), 115–127. DOI: 10.18662/brain/14.2/447.

[47] Chiodini, L., Sorva, J., Hellas, A., Seppälä, O., and Hauswirth, M. 2025. Two Approaches for Programming Education in the Domain of Graphics: An Experiment. en. *The Art, Science, and Engineering of Programming*, 10, 1, (Feb. 2025), 14. DOI: 10.22152/programming-journal.org/2025/10/14.

[48] Chover, M., Marín, C., Rebollo, C., and Remolar, I. 2020. A Game Engine Designed to Simplify 2D Video Game Development. en. *Multimedia Tools and Applications*, 79, 17-18, (May 2020), 12307–12328. DOI: 10.1007/s11042-019-08433-z.

[49] Chun, D. 2023. When the NFT Hype Settles, What Is Left Beyond Profile Pictures? A Critical Review on the Impact of Blockchain Technologies in the Art Market. en. *Arts*, 12, 5, (Aug. 2023), 181. DOI: 10.3390/arts12050181.

[50] Crow, T., Luxton-Reilly, A., and Wuensche, B. 2018. Intelligent Tutoring Systems for Programming Education: A Systematic Review. In *Proceedings of the 20th Australasian Computing Education Conference*. ACM, Brisbane Queensland Australia, (Jan. 2018), 53–62. DOI: 10.1145/3160489.3160492.

[51] Curto-Millet, D. and Corsín Jiménez, A. 2023. The Sustainability of Open Source Commons. en. *European Journal of Information Systems*, 32, 5, (Sept. 2023), 763–781. DOI: 10.1080/0960085X.2022.2046516.

[52] Davis, F. D., Bagozzi, R. P., and Warshaw, P. R. 1989. User Acceptance of Computer Technology: A Comparison of Two Theoretical Models. en. *Management Science*, 35, 8, (Aug. 1989), 982–1003. DOI: 10.1287/mnsc.35.8.982.

[53] Davis, T. A. and Kundert-Gibbs, J. 2006. The Role of Computer Science in Digital Production Arts. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (ITICSE '06). Association for Computing Machinery, New York, NY, USA, 73–77. ISBN: 1-59593-055-8. DOI: 10.1145/1140124.1140146.

[54] Deitrick, E. 2022. Individualizing Student Assessments with Parameterized Assessments. en. (Nov. 2022). Retrieved June 26, 2025 from https://www.codio.com/blog/individualized-student-questions-parameterized-assessments.

[55] Delone, W. and McLean, E. 2003. The DeLone and McLean Model of Information Systems Success: A Ten-Year Update. en. *Journal of Management Information Systems*, 19, 4, (Apr. 2003), 9–30. DOI: 10.1080/07421222.2003.11045748.

[56] Denholm, C. J. and Evans, T. D. 2007. *Supervising Doctorates Downunder: Keys to effective supervision in Australia and New Zealand*. eng. OCLC: 166330456. ACER Press, Camberwell, Vic. ISBN: 978-1-4294-8334-6.

[57] Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., and Reeves, B. N. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. en. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. ISBN: 979-8-4007-0423-9. ACM, Portland OR USA, (Mar. 2024), 296–302. DOI: 10.1145/3626252.3630909.

[58] Duda, M., Sovacool, K. L., Farzaneh, N., Nguyen, V. K., Haynes, S. E., Falk, H., Furman, K. L., Walker, L. A., Diao, R., Oneka, M., Drotos, A. C., Woloshin, A., Dotson, G. A., Kriebel, A., Meng, L., Thiede, S. N., Lapp, Z., and Wolford, B. N. 2021. Teaching Python for Data Science: Collaborative Development of a Modular & Interactive Curriculum. en. *Journal of Open Source Education*, 4, 46, 138. DOI: 10.21105/jose.00138.

[59] Dufva, T. S. 2021. Creative Coding as Compost(ing). en. In *Post-Digital, Post-Internet Art and Education*. Springer International Publishing, Cham, 269–283. ISBN: 978-3-030-73770-2. DOI: 10.1007/978-3-030-73770-2_16.

[60] Eisenberg, J. D. and Bellamy-Royds, A. 2014. *SVG Essentials*. eng. (2nd ed.). O'Reilly, Beijing Köln. ISBN: 978-1-4493-7435-8.

[61] Eteng, I., Akpotuzor, S., Akinola, S. O., and Agbonlahor, I. 2022. A Review on Effective Approach to Teaching Computer Programming to Undergraduates in Developing Countries. en. *Scientific African*, 16, (July 2022), e01240. DOI: 10.1016/j.sciaf.2022.e01240.

[62] Evil Mad Scientist Laboratories. 2023. AxiDraw Writing and Drawing Machines. (2023). https://www.axidraw.com.

[63] Ezenwoye, O. 2018. What Language? The Choice of an Introductory Programming Language. en. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, San Jose, CA, USA, (Oct. 2018), 1–8. ISBN: 978-1-5386-1174-6. DOI: 10.1109/FIE.2018.8658592.

[64] Fabic, G. V. F., Mitrovic, A., and Neshatian, K. 2019. Evaluation of Parsons Problems with Menu-Based Self-Explanation Prompts in a Mobile Python Tutor. en. *International Journal of Artificial Intelligence in Education*, 29, 4, (Dec. 2019), 507–535. DOI: 10.1007/s40593-019-00184-0.

[65] Feinburg, J. 2023. Python Mode for Processing. (Oct. 2023). Retrieved Apr. 4, 2024 from https://github.com/jdf/processing.py.

[66] Flynn, C. 2024. About PyPI Stats. (2024). Retrieved Oct. 2, 2024 from https://pypistats.org/about.

[67] Fry, B. 2007. *Visualizing Data*. (1st ed.). O'Reilly Media, Inc., Sebastopol, California, USA.

[68] Funk, M. and Zhang, Y. 2024. *Coding Art: A Guide to Unlocking Your Creativity with the Processing Language and p5.js in Four Simple Steps*. eng. (2024, 2nd ed.). *Design Thinking*. Apress, Berkeley, CA. ISBN: 978-1-4842-9780-3. DOI: 10.1007/978-1-4842-9780-3.

[69] Furneaux, B. 2012. Task-Technology Fit Theory: A Survey and Synopsis of the Literature. en. In *Information Systems Theory*. Vol. 28. Y. K. Dwivedi, M. R. Wade, and S. L. Schneberger, editors. Series Title: Integrated Series in Information Systems. Springer New York, New York, NY, 87–106. ISBN: 978-1-4419-6108-2. DOI: 10.1007/978-1-4419-6108-2_5.

[70] Gallatin, K. and Albon, C. 2023. *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*. eng. (Second ed.). O'Reilly Media, Inc, Beijing Boston Farnham Sebastopol Tokyo. ISBN: 978-1-0981-3572-0.

[71] Gendarmi, D. and Lanubile, F. 2006. Community-Driven Ontology Evolution Based on Folksonomies. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Vol. 4277. D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, R. Meersman, Z. Tari, and P. Herrero, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 181–188. ISBN: 978-3-540-48272-7. DOI: 10.1007/11915034_41.

[72] Gieselmann, H. 2019. Generative Unity. en. eTextbook. (2019). Retrieved Feb. 24, 2022 from https://generativeunity.de.

[73] Gjervig, T. 2025. Awesome Creative Coding. (2025). https://github.com/terkelg/awesome-creative-coding.

[74] Glassner, A. S. 2021. *Deep Learning: A Visual Approach*. No Starch Press, San Francisco. ISBN: 978-1-7185-0072-3.

[75] Goodhue, D. L. 1995. Understanding User Evaluations of Information Systems. *Management Science*, 41, 12, 1827–1844. Publisher: INFORMS. Retrieved July 1, 2025 from http://www.jstor.org/stable/2633074.

[76] Goodhue, D. L. and Thompson, R. L. 1995. Task-Technology Fit and Individual Performance. *MIS Quarterly*, 19, 2, (June 1995), 213. DOI: 10.2307/249689.

[77] Gordon, V. S. and Clevenger, J. 2021. *Computer Graphics Programming in OpenGL with C++*. (Second ed.). Mercury Learning and Information, Dulles, Virginia. ISBN: 978-1-68392-672-6.

[78] 2023. GPTutor: A ChatGPT-Powered Programming Tool for Code Explanation. en. In *Communications in Computer and Information Science*. ISSN: 1865-0929. Springer Nature Switzerland, Cham, 321–327. ISBN: 978-3-031-36336-8. Retrieved July 12, 2025 from https://link.springer.com/10.1007/978-3-031-36336-8_50.

[79] Graham, S. and Miller, B. 2019. Programming Skulpt. English. (June 2019). Retrieved Dec. 10, 2024 from https://brython.info/static_doc.

[80] Grammarly Inc. 2025. Grammarly. (2025). https://www.grammarly.com.

[81] Grandell, L., Peltomäki, M., Back, R.-J., and Salakoski, T. 2006. Why Complicate Things? Introducing Programming in High School Using Python. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (ACE '06). Australian Computer Society, Inc., AUS, 71–80. ISBN: 1-920682-34-1.

[82] Granger, B. and Pérez, F. 2021. Jupyter: Thinking and Storytelling with Code and Data, (July 2021). DOI: 10.22541/au.161298309.98344404/v3.

[83] Greenberg, I. 2016. *Processing: Creative Coding and Computational Art*. Apress. ISBN: 978-1-4842-2025-2. https://link.springer.com/book/10.1007/978-1-4302-0310-0.

[84] Greenberg, I., Kumar, D., and Xu, D. 2012. Creative Coding and Visual Portfolios for CS1. en. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, Raleigh North Carolina USA, (Feb. 2012), 247–252. ISBN: 978-1-4503-1098-7. DOI: 10.1145/2157136.2157214.

[85] Groß, B., Bohnacker, H., Laub, J., and Lazzeroni, C. 2018. *Generative Design P_3_1_3_01*. Hermann Schmidt. http://www.generative-gestaltung.de/2/sketches/?01_P/P_3_1_3_01.

[86] Guo, P. 2014. Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. English. (July 2014). Retrieved Aug. 15, 2022 from https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext.

[87] Gurioli, A., Gabbrielli, M., and Zacchiroli, S. 2024. Is This You, LLM? Recognizing AI-Written Programs with Multilingual Code Stylometry. (Dec. 2024). DOI: 10.48550/arXiv.2412.14611.

[88] Guzdial, M. 2005. *Introduction to Computing and Programming in Python: A Multimedia Approach*. eng. *An Alan R. Apt book*. Pearson/Prentice Hall, Upper Saddle River, NJ. ISBN: 978-0-13-117655-3.

[89] Haddaway, N. R., Collins, A. M., Coughlin, D., and Kirk, S. 2015. The Role of Google Scholar in Evidence Reviews and Its Applicability to Grey Literature Searching. en. *PLOS ONE*, 10, 9, (Sept. 2015), e0138237. K. B. Wray, editor. DOI: 10.1371/journal.pone.0138237.

[90] Hair, J. F., Black, W. C., Babin, B. J., and Anderson, R. E. 2019. *Multivariate Data Analysis*. eng. (Eighth ed.). Cengage, Andover, Hampshire. ISBN: 978-1-4737-5654-0.

[91] Hair, J. F., Hult, G. T. M., Ringle, C. M., Sarstedt, M., Danks, N. P., and Ray, S. 2021. An Introduction to Structural Equation Modeling. In *Partial Least Squares Structural Equation Modeling (PLS-SEM) Using R: A Workbook*. J. F. Hair Jr., G. T. M. Hult, C. M. Ringle, M. Sarstedt, N. P. Danks, and S. Ray, editors. Springer International Publishing, Cham, 1–29. ISBN: 978-3-030-80519-7. DOI: 10.1007/978-3-030-80519-7_1.

[92] Halladay, K. 2019. *Practical Shader Development: Vertex and Fragment Shaders for Game Developers*. eng. Apress L. P, Berkeley, CA. ISBN: 978-1-4842-4457-9.

[93] Hidayat, D., Pangaribuan, C. H., Putra, O. P. B., and Irawan, I. 2021. Contemporary Studies of Task-Technology Fit: A Review of the Literature. In *2021 International Conference on Information Management and Technology (ICIMTech)*. IEEE, Jakarta, Indonesia, (Aug. 2021), 309–313. ISBN: 978-1-6654-4937-3. DOI: 10.1109/ICIMTech53080.2021.9535028.

[94] Hirzel, M. 2023. Low-Code Programming Models. en. *Communications of the ACM*, 66, 10, (Oct. 2023), 76–85. DOI: 10.1145/3587691.

[95] Humble Bundle Inc. 2023. Humble Tech Book Bundle: Python by No Starch. (2023). Retrieved Aug. 30, 2025 from https://www.humblebundle.com/books/python-no-starch-books.

[96] Humble Bundle Inc. 2021. Machine Learning Bookshelf by No Starch Press. (2021). Retrieved Aug. 30, 2025 from https://www.humblebundle.com/books/machine-learning-bookshelf-no-starch-press-book.

[97] Hunt, J. 2019. Python Turtle Graphics. en. In *Advanced Guide to Python 3 Programming*. Undergraduate Topics in Computer Science. Springer International Publishing, Cham, 13–21. ISBN: 978-3-030-25943-3. Retrieved July 4, 2025 from http://link.springer.com/10.1007/978-3-030-25943-3_3.

[98] Idialu, O. J., Mathews, N. S., Maipradit, R., Atlee, J. M., and Nagappan, M. 2024. Whodunit: Classifying Code as Human Authored or GPT-4 Generated - a Case Study on CodeChef Problems. en. In *Proceedings of the 21st International Conference on Mining Software Repositories*. ACM, Lisbon Portugal, (Apr. 2024), 394–406. DOI: 10.1145/3643991.3644926.

[99] Ishaq, K., Alvi, A., Haq, M. I. U., Rosdi, F., Choudhry, A. N., Anjum, A., and Khan, F. A. 2024. Level Up Your Coding: A Systematic Review of Personalized, Cognitive, and Gamified Learning in Programming Education. en. *PeerJ Computer Science*, 10, (Nov. 2024), e2310. DOI: 10.7717/peerj-cs.2310.

[100] Jack, O. 2022. Documentation Portal. (Sept. 2022). Retrieved Oct. 2, 2024 from https://hydra.ojack.xyz/docs.

[101] Jankowski, T., Purgason, T., Van De Grift, I., Turner, L., Odendaal, A., Jordan, E., Clement, M., Dawes, B., Ke, T., Chan, I., Nakamura, Y., Singe, J., Besson, O., Paterson, J., Suriani, V., Baumann, J., Suga, Y., and Davis, J. 2000. *New Masters of Flash*. en. Apress, Berkeley, CA. ISBN: 978-1-59059-209-0. DOI: 10.1007/978-1-4302-5145-3.

[102] Judeikytė, E. and Wong, Y. N. 2022. Outlier Is Back 2022. (Jan. 2022). Retrieved Aug. 8, 2023 from https://nightingaledvs.com/outlier-is-back-in-2022-apply-to-speak.

[103] Juneau, J., Baker, J., Ng, V., Soto, L., and Wierzbicki, F. 2009. *The Definitive Guide to Jython: Python for the Java Platform*. eng. *The Expert's Voice in Software Development*. Apress Distributed by Springer-Verlag, [New York] New York. ISBN: 978-1-4302-2528-7.

[104] Kaila, E., Luukkainen, M., Laaksonen, A., and Lemström, K. 2023. On Changing the Curriculum Programming Language from Java to Python (Discussion Paper). en. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. ACM, Koli Finland, (Nov. 2023), 1–7. DOI: 10.1145/3631802.3631814.

[105] Karthik, S., Thirumal Reddy, P., and Prakash Marimuthu, K. 2017. Development of Low-Cost Plotter for Educational Purposes Using Arduino. *IOP Conference Series: Materials Science and Engineering*, 225, (Aug. 2017), 012019. DOI: 10.1088/1757-899X/225/1/012019.

[106] Kastrenakes, J. 2021. Beeple Sold an NFT for $69 Million. English. *The Verge*, (Mar. 2021). Edition: Featured Storeis, March 2021 Place: Lower Manhatten, New York City, USA. Retrieved Dec. 18, 2024 from https://www.theverge.com/2021/3/11/22325054/beeple-christies-nft-sale-cost-everydays-69-million.

[107] Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., and Grossman, T. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. (Apr. 2023), 1–23. DOI: 10.1145/3544548.3580919.

[108] Kelleher, C. and Pausch, R. 2005. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. en. *ACM Computing Surveys*, 37, 2, (June 2005), 83–137. DOI: 10.1145/1089733.1089734.

[109] Kiesler, N. and Schiffner, D. 2023. Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. (Aug. 2023). Retrieved May 5, 2025 from https://your-ai-staff.com/large-language-models-in-introductory-programming-education-chatgpts-performance-and-implications-for-assessments.

[110] A. E. King, ed. 2025. *Artificial Intelligence, Pedagogy and Academic Integrity*. en. *Advances in Artificial Intelligence in Education*. ISSN: 3004-9903, 3004-9911. Springer Nature Switzerland, Cham. ISBN: 978-3-031-92534-4. Retrieved July 18, 2025 from https://link.springer.com/10.1007/978-3-031-92534-4.

[111] Klopfer, L. 2004. Evaluating Information – Applying the CRAAP Test. en. *LOEX Quarterly*, 31, 3, Article 5. Retrieved Mar. 6, 2024 from https://commons.emich.edu/loexquarterly/vol31/iss3/5.

[112] Kneusel, R. T. 2021. *Practical Deep Learning: A Python-Based Introduction*. (First ed.). No Starch Press, Inc, San Francisco, CA. ISBN: 978-1-7185-0075-4.

[113] Knochel, A. D. and Patton, R. M. 2015. *If* Art Education *Then* Critical Digital Making: Computational Thinking and Creative Code. en. *Studies in Art Education*, 57, 1, (Oct. 2015), 21–38. DOI: 10.1080/00393541.2015.11666280.

[114] Kohen-Vacs, D., Usher, M., and Jansen, M. 2025. Integrating Generative AI into Programming Education: Student Perceptions and the Challenge of Correcting AI Errors. en. *International Journal of Artificial Intelligence in Education*, (July 2025). DOI: 10.1007/s40593-025-00496-4.

[115] Latini, M. and Köbben, B. 2005. A Web Application for Landslide Inventory Using Data-Driven SVG. en. In *Geo-information for Disaster Management*. P. Van Oosterom, S. Zlatanova, and E. M. Fendel, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 1041–1054. ISBN: 978-3-540-24988-7. DOI: 10.1007/3-540-27468-5_73.

[116] Lauer, K. 2019. (Optionally) Pure-Python 2D Game Physics, a Box2D Port. (2019). Retrieved Oct. 9, 2023 from https://github.com/pybox2d/pypybox2d.

[117] Lee, K. and Fanguy, M. 2022. Online Exam Proctoring Technologies: Educational Innovation or Deterioration? en. *British Journal of Educational Technology*, 53, 3, (May 2022), 475–490. Publisher: Wiley. DOI: 10.1111/bjet.13182.

[118] Levin, G. and Brain, T. 2021. *Code as Creative Medium: A Handbook for Computational Art and Design*. The MIT Press, Cambridge, Massachusets; London, England. ISBN: 978-0-262-54204-3.

[119] Lim, T. L. and Lee, A. S. H. 2021. Extended TAM and TTF Model: A Framework for the 21st Century Teaching and Learning. In *2021 International Conference on Computer & Information Sciences (ICCOINS)*. IEEE, Kuching, Malaysia, (July 2021), 339–334. ISBN: 978-1-7281-7153-1. DOI: 10.1109/ICCOINS49721.2021.9497216.

[120] Ling, H.-C., Hsiao, K.-L., and Hsu, W.-C. 2021. Can Students' Computer Programming Learning Motivation and Effectiveness Be Enhanced by Learning Python Language? A Multi-Group Analysis. en. *Frontiers in Psychology*, 11, (Jan. 2021), 600814. DOI: 10.3389/fpsyg.2020.600814.

[121] Liu, K., Yao, J., Tao, D., and Yang, T. 2023. Influence of Individual-technology-task-environment Fit on University Student Online Learning Performance: The Mediating Role of Behavioral, Emotional, and Cognitive Engagement. en. *Education and Information Technologies*, 28, 12, (Dec. 2023), 15949–15968. DOI: 10.1007/s10639-023-11833-2.

[122] Long, D., Mathieu, D., Pham, T. V., Xu, X., and Jin, W. 2023. From Coding to Creativity: Using Dancing Sphero Robots to Inspire STEM Learning. en. In *The 24th Annual Conference on Information Technology Education*. ACM, Marietta GA USA, (Oct. 2023), 180–181. DOI: 10.1145/3585059.3611419.

[123] Lorusso, S. 2023. Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative. eng. In *Graphic Design in the Post-Digital Age: A Survey of Practices Fueled by Creative Coding*. Number 15 in Set Margins'. (Second edition ed.). D. Conrad and R. v. Leijsen, editors. Set Margins', Eindhoven, 25–31. ISBN: 978-90-832706-9-2.

[124] Ma, Y. and Tilevich, E. 2021. "You Have Said Too Much": Java-Like Verbosity Anti-Patterns in Python Codebases. en. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E*. ACM, Chicago IL USA, (Oct. 2021), 13–18. ISBN: 978-1-4503-9089-7. DOI: 10.1145/3484272.3484960.

[125] Mahon, J., Mac Namee, B., and Becker, B. A. 2024. Guidelines for the Evolving Role of Generative AI in Introductory Programming Based on Emerging Practice. en. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ISBN: 979-8-4007-0600-4. ACM, Milan Italy, (July 2024), 10–16. DOI: 10.1145/3649217.3653602.

[126] Marikyan, D. and Papagiannidis, S. 2023. Task-Technology Fit: A Review. In *TheoryHub*. Newcastle University, Newcastle upon Tyne, (Sept. 2023), 18. ISBN: 978-1-7396044-0-0. Retrieved Mar. 3, 2025 from https://open.ncl.ac.uk/theories/3/task-technology-fit.

[127] Marji, M. 2014. *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math*. eng. No Starch Press, San Francisco. ISBN: 978-1-59327-543-3.

[128] Matthes, E. 2023. *Python Crash Course: A Hands-on, Project-Based Introduction to Programming*. eng. (3rd ed.). No Starch Press, San Francisco. ISBN: 978-1-7185-0270-3.

[129] Maurer, U. and Vogelzang, M. 2024. Popular Awesome Lists on GitHub. (2024). Retrieved Dec. 2, 2024 from https://awesome.facts.dev.

[130] McCarthy, L., Reas, C., and Fry, B. 2015. *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. Make Community, LLC. ISBN: 978-1-4571-8673-8. https://www.oreilly.com/library/view/getting-started-with/9781457186769.

[131] McDanel, B. and Novak, E. 2025. Designing LLM-Resistant Programming Assignments: Insights and Strategies for CS Educators. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Pittsburgh PA USA, (Feb. 2025), 756–762. DOI: 10.1145/3641554.3701872.

[132] McDonald, N., Johri, A., Ali, A., and Collier, A. H. 2025. Generative Artificial Intelligence in Higher Education: Evidence from an Analysis of Institutional Policies and Guidelines. en. *Computers in Human Behavior: Artificial Humans*, 3, (Mar. 2025), 100121. Publisher: Elsevier BV. DOI: 10.1016/j.chbah.2025.100121.

[133] McGregor, S. E. 2021. *Practical Python Data Wrangling and Data Quality*. (First ed.). O'Reilly Media. Inc, Sebastopol, CA. ISBN: 978-1-4920-9150-9.

[134] McKinney, W. 2022. *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter*. (Third ed.). O'Reilly, Beijing. ISBN: 978-1-0981-0403-0.

[135] McNutt, A., Outkine, A., and Chugh, R. 2023. A Study of Editor Features in a Creative Coding Classroom. en. (Jan. 2023). Retrieved July 12, 2024 from http://arxiv.org/abs/2301.13302.

[136] Melzer, P. 2019. A Conceptual Framework for Task and Tool Personalisation in IS Education. en. In *A Conceptual Framework for Personalised Learning*. Springer Fachmedien Wiesbaden, Wiesbaden, 47–76. ISBN: 978-3-658-23094-4. DOI: 10.1007/978-3-658-23095-1_3.

[137] Menard, S. and Nell, L. 2018. JPype 1.5.2.devo Documentation. English. (2018). Retrieved Aug. 15, 2022 from https://jpype.readthedocs.io.

[138] MIT Media Lab. 2024. Scratch - Imagine, Program, Share. (2024). https://scratch.mit.edu.

[139] Mlambo, S., Rambe, P., and Schlebusch, L. 2020. Effects of Gauteng Province's Educators' ICT Self-Efficacy on Their Pedagogical Use of ICTS in Classrooms. en. *Heliyon*, 6, 4, (Apr. 2020), e03730. DOI: 10.1016/j.heliyon.2020.e03730.

[140] Moore, A. D. 2021. *Python GUI Programming with Tkinter: Design and Build Functional and User-Friendly GUI Applications*. en. (Second ed.). Packt, Birmingham Mumbai. ISBN: 978-1-80181-592-5.

[141] Moreno, C. 2024. Fringe Game Craft: An Exploration of Fantasy Consoles and Exotic Game Tools. In *Abstract Proceedings of DiGRA 2024 Conference: Playgrounds*. DiGRA DL, Guadalajara Mexico. DOI: 10.26503/dl.v2024i2.2385.

[142] Mosunmola, A., Mayowa, A., Okuboyejo, S., and Adeniji, C. 2018. Adoption and Use of Mobile Learning in Higher Education: The UTAUT Model. en. In *Proceedings of the 9th International Conference on E-Education, E-Business, E-Management and E-Learning*. ACM, San Diego California, (Jan. 2018), 20–25. ISBN: 978-1-4503-5485-1. DOI: 10.1145/3183586.3183595.

[143] Nake, F. 2012. Construction and Intuition: Creativity in Early Computer Art. en. In *Computers and Creativity*. J. McCormack and M. d'Inverno, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 61–94. ISBN: 978-3-642-31727-9. DOI: 10.1007/978-3-642-31727-9_3.

[144] Nanavati, A., Owens, A., and Stehlik, M. 2020. Pythons and Martians and Finches, Oh My! Lessons Learned from a Mandatory 8th Grade Python Class. en. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, Portland OR USA, (Feb. 2020), 811–817. ISBN: 978-1-4503-6793-6. DOI: 10.1145/3328778.3366906.

[145] National Endowment for the Arts. 2021. Tech as Art. en. Tech. rep. National Endowment for the Arts, Washington, DC, USA, (June 2021), 126. https://www.arts.gov/sites/default/files/Tech-as-Art-081821.pdf.

[146] Nees, G. 1968. Schotter (Gravel Stones). (1968). https://spalterdigital.com/artworks/shotter-gravel-stones.

[147] Ng, M., Carter-Arlt, M., and Mozaffari Nezhad, A. 2023. *Collaborative VR Essentials: A Guidebook and Toolkit for Developing 3D Applications in Unity and Unreal for the Immersion Studio*. en. Toronto Metropolitan University Pressbooks, Toronto, Canada, (May 2023). https://pressbooks.library.torontomu.ca/immersionstudioguide.

[148] Nielsen, T. 2024. Awesome Creative Coding: Generative Art, Data Visualization, Interaction Design, Resources. (2024). Retrieved Jan. 10, 2024 from https://github.com/terkelg/awesome-creative-coding.

[149] No Starch Press. 2025. About. (2025). Retrieved Aug. 30, 2025 from https://nostarch.com/about.

[150] No Starch Press. 2025. Sales and Distribution. (2025). Retrieved Aug. 30, 2025 from https://nostarch.com/distribution.htm.

[151] Noble, J. 2009. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks*. O'Reilly Media, Newton, Massachusetts, USA. ISBN: 978-1-4493-7919-3. https://www.oreilly.com/library/view/programming-interactivity-2nd/9781449321482.

[152] O'Dea, X. 2024. Generative AI: Is It a Paradigm Shift for Higher Education? en. *Studies in Higher Education*, 49, 5, (May 2024), 811–816. DOI: 10.1080/03075079.2024.2332944.

[153] Ohio State University Library. 2018. *Choosing & Using Sources: A Guide to Academic Research*. en. (3.0 ed.). Pressbooks, Ohio, USA, (July 2018). Retrieved Mar. 3, 2025 from https://ohiostate.pressbooks.pub/choosingsources/.

[154] OpenAI, Inc. 2025. ChatGPT 4.x. (2025). https://chat.openai.com.

[155] Otto, D. and Kerres, M. 2022. Increasing Sustainability in Open Learning: Prospects of a Distributed Learning Ecosystem for Open Educational Resources. *Frontiers in Education*, 7, (May 2022), 866917. DOI: 10.3389/feduc.2022.866917.

[156] p5.js Contributors. 2024. P5.js Reference: 3D. (2024). Retrieved Oct. 2, 2024 from https://p5js.org/reference/#3D.

[157] Pal, A. 2020. Release Notes — p5 0.8.1. (2020). Retrieved Oct. 2, 2024 from https://p5.readthedocs.io/en/latest/releasenotes.

[158] Papadopoulos, P., Ilia, P., Polychronakis, M., Markatos, E. P., Ioannidis, S., and Vasiliadis, G. 2018. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation. (2018). DOI: 10.48550/ARXIV.1810.00464.

[159] Park, C. 2019. Exploring a New Determinant of Task Technology Fit: Content Characteristics. en. *Journal of International Technology and Information Management*, 27, 3, (Jan. 2019), 100–118. DOI: 10.58729/1941-6679.1385.

[160] Parlante, N. 2017. Codingbat. (2017). Retrieved Aug. 9, 2025 from https://codingbat.com/python.

[161] Parrish, A., Fry, B., and Reas, C. 2016. *Getting Started with Processing.py: Making Interactive Graphics with Processing's Python Mode*. Make Community, LLC, Santa Rosa, CA, USA. ISBN: 978-1-4571-8679-0. https://www.oreilly.com/library/view/getting-started-with/9781457186820.

[162] Paul, C. 2002. CODeDOC. (2002). Retrieved Oct. 9, 2023 from https://whitney.org/exhibitions/codedoc.

[163] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. 2007. A Survey of Literature on the Teaching of Introductory Programming. *SIGCSE Bulletin*, 39, 4, (Dec. 2007), 204–223. DOI: 10.1145/1345375.1345441.

[164] Pearson, M. 2011. *Generative Art: A Practical Guide Using Processing*. eng. Manning Publications Co. LLC, New York. ISBN: 978-1-935182-62-7.

[165] Polgár, T. 2008. *Freax: The Brief History of the Computer Demoscene. Vol. 1*. eng. (2nd ed.). Vol. 1. CSW-Verl, Winnenden. ISBN: 978-3-9810494-0-4.

[166] Prather, J., Reeves, B., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B., Kimmel, B., Wright, J., and Briggs, B. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. (May 2024). DOI: 10.48550/arXiv.2405.17739.

[167] Processing Foundation. 2022. A Look Back at the History of Processing. (2022). https://medium.com/processing-foundation.

[168] Processing Foundation. 2024. Books / Processing.org. (2024). Retrieved Jan. 15, 2024 from https://processing.org/books.

[169] Processing Foundation. 2022. Google Summer of Code 2022 Wrap-Up Post. English. (Oct. 2022). Retrieved Oct. 17, 2023 from https://medium.com/@ProcessingOrg/google-summer-of-code-2022-wrap-up-post-cb64caa840f0#f12a.

[170] Processing Foundation. 2022. Overview / Processing.org. (Sept. 2022). Retrieved Oct. 13, 2024 from https://processing.org/overview.

[171] Processing Foundation. 2024. Processing Foundation. (2024). Retrieved Dec. 1, 2024 from https://processingfoundation.org.

[172] Processing Foundation. 2024. Shader() / Reference / Processing.org. (2024). Retrieved Dec. 1, 2024 from https://processing.org/reference/shader_.html.

[173] Purvis, J. 2024. Visor. (2024). https://www.visor.live.

[174] Purvis, J. V. 2019. *CJing: Combining Live Coding and VJing for Live Visual Performance*. PhD Thesis. Open Access Te Herenga Waka-Victoria University of Wellington, (Jan. 2019). DOI: 10.26686/wgtn.17138669.

[175] Puthumanaillam, G., Bretl, T., and Ornik, M. 2025. The Lazy Student's Dream: ChatGPT Passing an Engineering Course on Its Own. (May 2025). DOI: 10.48550/arXiv.2503.05760.

[176] Python Software Foundation. 2024. Alternative Python Implementations. (2024). Retrieved Oct. 2, 2024 from https://www.python.org/download/alternatives.

[177] Python Software Foundation. 2024. PyPI Docs. (2024). Retrieved Oct. 2, 2024 from https://docs.pypi.org.

[178] Python Software Foundation. 1999. Python 1.5.2. (Apr. 1999). Retrieved Aug. 1, 2025 from https://www.python.org/download/releases/1.5.

[179] Quentel, P. 2024. Brython 3.13.0 Performance Compared to CPython 3.13.0. English. (Oct. 2024). Retrieved Dec. 8, 2022 from https://brython.info/speed_results.html.

[180] Quentel, P. 2024. Brython Documentation. English. (Oct. 2024). Retrieved Dec. 8, 2022 from https://brython.info/static_doc.

[181] Rahe, C. and Maalej, W. 2025. How Do Programming Students Use Generative AI? en. *Proceedings of the ACM on Software Engineering*, 2, FSE, (June 2025), 978–1000. DOI: 10.1145/3715762.

[182] Rambe, P. 2016. The Role of Educational Technology in Design and Delivery of Curricula Programmes: A Case of STEPS at a University of Technology. en. *The African Journal of Information Systems*, 8, 2, (Apr. 2016), 86–113.

[183] Raschka, S., Patterson, J., and Nolet, C. 2020. Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. en. *Information*, 11, 4, (Apr. 2020), 193. DOI: 10.3390/info11040193.

[184] Rashid, T. 2016. *Make Your Own Neural Network*. eng. CreateSpace Independent Publishing Platform, London, UK. ISBN: 978-1-5308-2660-5.

[185] Rathert, N. A. 2005. Knowledge Visualization Using Dynamic SVG Charts. en. In *Visualizing Information Using SVG and X3D*. V. Geroimenko and C. Chen, editors. Springer-Verlag, London, 245–255. ISBN: 978-1-85233-790-2. DOI: 10.1007/1-84628-084-2_11.

[186] Reas, C. and Fry, B. 2014. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, Cambridge, Massachusetts, USA. ISBN: 978-0-262-32185-3. https://mitpress.mit.edu/9780262028288/processing.

[187] Reas, C. and Fry, B. 2018. A Modern Prometheus. English. (May 2018). Retrieved May 20, 2020 from https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85.

[188] Reas, C. and Fry, B. 2006. Processing: Programming for the Media Arts. *AI & SOCIETY*, 20, 4, (Sept. 2006), 526–538. DOI: 10.1007/s00146-006-0050-9.

[189] Replit, Inc. 2025. Replit AI – Turn Natural Language into Apps and Websites. (2025). Retrieved May 13, 2025 from https://replit.com/ai.

[190] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. 2009. Scratch: Programming for All. en. *Communications of the ACM*, 52, 11, (Nov. 2009), 60–67. DOI: 10.1145/1592761.1592779.

[191] Riedl, M. O. 2015. A Python Engine for Teaching Artificial Intelligence in Games. en. (Nov. 2015). Retrieved Sept. 5, 2022 from http://arxiv.org/abs/1511.07714.

[192] Robins, K. and Kanowski, P. 2008. PhD by Publication: A Student's Perspective. en. *Journal of Research Practice*, 4, 2. http://jrp.icaap.org/index.php/jrp/article/view/136/154.

[193] Rostocki, M. 2021. PICO-8 Tweetcart Studies. (2021). https://demobasics.pixienop.net/tweetcarts.

[194] Rubinovitz, J. 2024. How It's Made - Exploring AI x Learning through ShiffBot, an AI Experiment Powered by the Gemini API. (Jan. 2024). Retrieved Dec. 1, 2024 from https://developers.googleblog.com/en/how-its-made-exploring-ai-x-learning-through-shiffbot-an-ai-experiment-powered-by-the-gemini-api.

[195] Ruf, A., Mühling, A., and Hubwieser, P. 2014. Scratch vs. Karel: Impact on Learning Outcomes and Motivation. en. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*. ACM, Berlin Germany, (Nov. 2014), 50–59. ISBN: 978-1-4503-3250-7. DOI: 10.1145/2670757.2670772.

[196] Ryu, S., Cho, C., and Choi, W. 2025. Hybrid-Based Learning Approach from Block Coding to Text Coding. en. In *Intelligent Data Engineering and Analytics*. Vol. 420. V. Bhateja, P. Patel, and M. Simic, editors. Springer Nature Singapore, Singapore, 185–197. ISBN: 978-981-96-0138-7. Retrieved Oct. 3, 2025 from https://link.springer.com/10.1007/978-981-96-0139-4_15.

[197] Salter, A. 2025. DGAH 200 | Creative Coding and Generative AI. (2025). Retrieved June 6, 2025 from http://anastasiasalter.net/Creative-Coding.

[198] Santo, A. 2024. WAB.com. (2024). https://www.wab.com.

[199] Savrun-Yeniçeri, G., Zhang, W., Zhang, H., Seckler, E., Li, C., Brunthaler, S., Larsen, P., and Franz, M. 2014. Efficient Hosted Interpreters on the JVM. en. *ACM Transactions on Architecture and Code Optimization*, 11, 1, (Feb. 2014), 1–24. DOI: 10.1145/2532642.

[200] Scherer, D., Dubois, P., and Sherwood, B. 2000. VPython: 3D Interactive Scientific Graphics for Students. *Computing in Science & Engineering*, 2, 5, (Oct. 2000), 56–62. DOI: 10.1109/5992.877397.

[201] Schlegel, A. 2015. ControlP5 (Documentation). (2015). Retrieved Oct. 9, 2023 from https://sojamo.de/libraries/controlP5.

[202] Schmitz, J. 2021. Welcome to py5! (Oct. 2021). https://py5coding.org.

[203] Shahzad, M. F., Xu, S., Hanif, S., and Akram, S. 2025. What Factors Affect the Intention to Use Information and Communication Technology? Perspectives of Chinese University Students. en. *Humanities and Social Sciences Communications*, 12, 1, (June 2025), 847. DOI: 10.1057/s41599-025-05156-5.

[204] Shannon, T. 2017. *Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings*. eng. Addison-Wesley. ISBN: 978-0-13-468076-7.

[205] Shiffman, D. 2008. *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 0-12-373602-1.

[206] Shiffman, D. 2024. *The Nature of Code: Simulating Natural Systems with JavaScript*. eng. No Starch Press, New York. ISBN: 978-1-7185-0370-0.

[207] Siek, M. and Wijaya, I. 2022. Investigating Cloud-Based Educational Technology Adoption in Advancing Learning Performance. In *2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*. IEEE, Prapat, Indonesia, (Oct. 2022), 1–8. ISBN: 978-1-6654-5395-0. DOI: 10.1109/ICORIS56080.2022.10031577.

[208] SIGCSE. 2025. Nifty Assignments. (2025). Retrieved Sept. 21, 2024 from http://nifty.stanford.edu.

[209] Singh, S. 2024. Empowering Learning: A Review of Online Programming Tools and Their Evolving Landscape. *International Journal for Research in Applied Science and Engineering Technology*, 12, 4, (Apr. 2024), 3238–3242. DOI: 10.22214/ijraset.2024.60415.

[210] Škorić, I., Orehovački, T., and Ivašić-Kos, M. 2021. Exploring the Acceptance of the Web-Based Coding Tool in an Introductory Programming Course: A Pilot Study. en. In *Human Interaction, Emerging Technologies and Future Applications III*. Vol. 1253. T. Ahram, R. Taiar, K. Langlois, and A. Choplin, editors. Springer International Publishing, Cham, 42–48. ISBN: 978-3-030-55307-4. DOI: 10.1007/978-3-030-55307-4_7.

[211] Škorić, I., Orehovački, T., and Ivašić-Kos, M. 2021. Task-Technology Fit and Continuance of Use of Web-Based Programming Tool: A Pilot Study. en. In *Human Systems Engineering and Design III*. Advances in Intelligent Systems and Computing. Vol. 1269. W. Karwowski, T. Ahram, D. Etinger, N. Tanković, and R. Taiar, editors. Springer International Publishing, Cham, 57–62. ISBN: 978-3-030-58281-4. Retrieved Aug. 10, 2025 from http://link.springer.com/10.1007/978-3-030-58282-1_10.

[212] H. Smith and R. T. Dean, eds. 2009. *Practice-Led Research, Research-Led Practice in the Creative Arts*. eng. *Research Methods for the Arts and Humanities*. Edinburgh University Press, Edinburgh. ISBN: 978-0-7486-3629-7.

[213] SMU Meadows School of the Arts. 2023. Differences Between Creative Computing and Creative Technology. (Mar. 2023). Retrieved June 1, 2024 from https://www.smu.edu/meadows/newsandevents/news/2023/differences-between-creative-computing-and-creative-technology.

[214] Sobota, B. and Pietriková, E. 2023. The Role of Game Engines in Game Development and Teaching. In *Computer Science for Game Development and Game Development for Computer Science*. B. Sobota and E. Pietriková, editors. IntechOpen, Rijeka. DOI: 10.5772/intechopen.1002257.

[215] Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., Papert, A., and Silverman, B. 2020. History of Logo. en. *Proceedings of the ACM on Programming Languages*, 4, HOPL, (June 2020), 1–66. DOI: 10.1145/3386329.

[216] Sonabend, R., Gruson, H., Wolansky, L., Kiragga, A., and Katz, D. S. 2024. FAIR-USE4OS: Guidelines for Creating Impactful Open-Source Software. en. *PLOS Computational Biology*, 20, 5, (May 2024), e1012045. J. A. Papin, editor. DOI: 10.1371/journal.pcbi.1012045.

[217] Sorhus, S. 2021. Awesome Lists About All Kinds of Interesting Topics. (2021). Retrieved Oct. 1, 2021 from https://github.com/sindresorhus/awesome.

[218] Stinson, L. 2021. Processing: The Software That Shaped Creative Coding. (2021). Retrieved Oct. 9, 2022 from https://eyeondesign.aiga.org/processing-the-software-that-shaped-creative-coding.

[219] Strive Math. 2025. Editor | Strive Learning Platform. (2025). https://code.strivemath.com.

[220] Suh, S. 2023. Cheat Sheet for Teaching Programming with Comics: Through the Lens of Concept-Language-Procedure Framework. Version Number: 1. (2023). DOI: 10.48550/ARXIV.2306.00464.

[221] Tabet, N., Gedawy, H., Alshikhabobakr, H., and Razak, S. 2016. From Alice to Python. Introducing Text-Based Programming in Middle Schools. en. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Arequipa Peru, (July 2016), 124–129. ISBN: 978-1-4503-4231-5. DOI: 10.1145/2899415.2899462.

[222] tabreturn. 2020. Processing.py in Ten Lessons. (2020). https://github.com/tabreturn/processing.py-tabreturn.github.io.

[223] Talton, J. O. and Fitzpatrick, D. 2007. Teaching Graphics with the OpenGL Shading Language. en. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM, Covington Kentucky USA, (Mar. 2007), 259–263. ISBN: 978-1-59593-361-4. DOI: 10.1145/1227310.1227402.

[224] Tamilmani, K., Rana, N. P., Wamba, S. F., and Dwivedi, R. 2021. The Extended Unified Theory of Acceptance and Use of Technology (UTAUT2): A Systematic Literature Review and Theory Evaluation. en. *International Journal of Information Management*, 57, (Apr. 2021), 102269. DOI: 10.1016/j.ijinfomgt.2020.102269.

[225] Terroso, T. and Pinto, M. 2022. Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education. en. *OASIcs, Volume 102, ICPEC 2022*, 102, 13:1–13:8. DOI: 10.4230/OASICS.ICPEC.2022.13.

[226] Tezos Foundation. 2024. Tezos Foundation Activity Report: July 2023 - March 2024. English. Tech. rep. Tezos Foundation, Zug, Switzerland. https://tezos.foundation/wp-content/uploads/2024/07/Tezos_Foundation_Activity_Report_July_2023_March_2024.pdf.

[227] Thabet, Z., Albashtawi, S., Ansari, H., Al-Emran, M., Al-Sharafi, M. A., and AlQudah, A. A. 2024. Exploring the Factors Affecting Telemedicine Adoption by Integrating UTAUT2 and IS Success Model: A Hybrid SEM–ANN Approach. *IEEE Transactions on Engineering Management*, 71, 8938–8950. DOI: 10.1109/TEM.2023.3296132.

[228] Thomas, D. and Hunt, A. 2019. *The Pragmatic Programmer, 20th Anniversary Edition: Journey to Mastery*. (Second ed.). Addison-Wesley, Boston. ISBN: 978-0-13-595705-9.

[229] Tzeng, S.-Y., Lin, K.-Y., and Lee, C.-Y. 2022. Predicting College Students' Adoption of Technology for Self-Directed Learning: A Model Based on the Theory of Planned Behavior With Self-Evaluation as an Intermediate Variable. *Frontiers in Psychology*, 13, (May 2022), 865803. DOI: 10.3389/fpsyg.2022.865803.

[230] Urban Arts. 2023. Creative Coders: Middle School CS Pathways Through Game Design. en. Proposal. Urban Arts, New York, USA, 26. https://www.ed.gov/sites/ed/files/2023/12/S411C230153_Urban-Arts-Partnership-Narrative_Redacted_508_QC.pdf.

[231] Vahid, F., Areizaga, L., and Pang, A. 2023. ChatGPT and Cheat Detection in CS1 Using a Program Autograding System. en. Whitepaper V1. Dept. of Computer Science, University of California, Riverside, (Feb. 2023). Retrieved June 3, 2025 from https://www.zybooks.com/chatgpt-and-cheat-detection-in-cs1-using-a-program-autograding-system.

[232] Vaidyanathan, S. 2019. *Creative Coding in Python: 30+ Programming Projects in Art, Games, and More*. eng. Quarto Publishing Group USA, Inc, Beverly, MA, USA. ISBN: 978-1-63159-581-3.

[233] Van der Zee, P. 2019. About JS1k. (2019). Retrieved Sept. 9, 2023 from https://js1k.com/about.

[234] Van Gumster, J. and Lampel, J. 2022. Procedural Modeling with Blender's Geometry Nodes: A Workshop on Taking Advantage of the Geometry Nodes Feature in Blender for Procedural Modeling. en. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Labs*. ACM, Vancouver BC Canada, (Aug. 2022), 1–2. ISBN: 978-1-4503-9367-6. DOI: 10.1145/3532725.3538516.

[235] Van Rossum, J. and van Blokland, E. 2023. Variables — DrawBot 3.130. (Feb. 2023). Retrieved Oct. 14, 2024 from https://www.drawbot.com/content/variables.html.

[236] Vaughan, L. 2023. *Python Tools for Scientists: An Introduction to Using Anaconda, JupyterLab, and Python's Scientific Libraries*. No Starch Press, San Francisco. ISBN: 978-1-7185-0266-6.

[237] Venkatesh, V., Morris, M., Davis, G., and Davis, F. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly*, 27, 3, 425. DOI: 10.2307/30036540.

[238] Victoria University of Wellington. 2025. DSDN 142 Creative Coding and AI I. en. (2025). Retrieved June 16, 2025 from https://www.wgtn.ac.nz/courses/dsdn/142/2025?crn=17154.

[239] Vidal Duarte, E., Castro Gutierrez, E., and Aedo, M. 2017. When the Robot Meets the Turtle: A Gentle Introduction to Algorithms and Functions. en. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Bologna Italy, (June 2017), 78–79. ISBN: 978-1-4503-4704-4. DOI: 10.1145/3059009.3072974.

[240] Villares, A. 2022. Online Course - Designing with Python: Programming for a Visual Context (Alexandre B A Villares). (Jan. 2022). Retrieved Oct. 9, 2022 from https://www.domestika.org/en/courses/4307-designing-with-python-programming-for-a-visual-context.

[241] Villares, A. 2022. Resources for Teaching Programming for Artists, Designers and Architects. (2022). Retrieved Apr. 2, 2022 from abav.lugaralgum.com/Resources-for-teaching-programming.

[242] Villares, A. B. d. A. and Moreira, D. d. C. 2017. Python on the Landscape of Programming Tools for Design and Architectural Education. en. In *Blucher Design Proceedings*. Editora Blucher, Concepción, Chile, (Nov. 2017), 207–211. DOI: 10.5151/sigradi2017-033.

[243] Walsh, M. J., Crowder, A., and Smith, M. 2022. Crafting to Code: Applying Creative Pedagogy to Promote Inclusion and Access in Learning to Code. In *Advances in Educational Technologies and Instructional Design*. IGI Global, 88–106. ISBN: 978-1-7998-8289-3. DOI: 10.4018/978-1-7998-8287-9.ch005.

[244] Wang, W.-T. and Kartika Sari, M. 2024. Examining the Effect of the Task-Technology Fit of Game Mechanisms on Learning Outcomes in Online Gamification Platforms. en. *Journal of Educational Computing Research*, 61, 8, (Jan. 2024), 1568–1595. DOI: 10.1177/07356331231187285.

[245] Wikipedia. 2025. Processing - Wikipedia. (May 2025). Retrieved June 16, 2025 from https://en.wikipedia.org/wiki/Processing.

[246] Wikipedia. 2025. SIGGRAPH - Wikipedia. (July 2025). Retrieved July 30, 2025 from https://en.wikipedia.org/wiki/SIGGRAPH#SIGGRAPH.

[247] Wing, J. M. 2006. Computational Thinking. en. *Communications of the ACM*, 49, 3, (Mar. 2006), 33–35. DOI: 10.1145/1118178.1118215.

[248] World Intellectual Property Organization. 2024. Generative Artificial Intelligence – Patent Landscape Report. en. Tech. rep. WIPO, Geneva, Switzerland, 116. https://www.wipo.int/web-publications/patent-landscape-report-generative-artificial-intelligence-genai/assets/62504/Generative%20AI%20-%20PLR%20EN_WEB2.pdf.

[249] Xie, Y., Wu, S., and Chakravarty, S. 2023. AI Meets AI: Artificial Intelligence and Academic Integrity - a Survey on Mitigating AI-Assisted Cheating in Computing Education. en. In *The 24th Annual Conference on Information Technology Education*. ACM, Marietta GA USA, (Oct. 2023), 79–83. DOI: 10.1145/3585059.3611449.

[250] Yaakop, A. Y., Mahadi, N., Ariffin, Z. Z., Abu Hasan, Z. R., and Harun, M. 2020. Examining Students' Continuance Usage Intention for Web-Based Educational Tools: A Developed Integrated Structural Model Approach. *Asian Academy of Management Journal*, 25, 1, (June 2020). DOI: 10.21315/aamj2020.25.1.2.

[251] Yadegaridehkordi, E., Iahad, N. A., and Ahmad, N. 2016. Task-Technology Fit Assessment of Cloud-Based Collaborative Learning Technologies: ng. *International Journal of Information Systems in the Service Sector*, 8, 3, (July 2016), 58–73. DOI: 10.4018/IJISSS.2016070104.

[252] Yang, A. C., Lin, J.-Y., Lin, C.-Y., and Ogata, H. 2024. Enhancing Python Learning with PyTutor: Efficacy of a ChatGPT-Based Intelligent Tutoring System in Programming Education. en. *Computers and Education: Artificial Intelligence*, 7, (Dec. 2024), 100309. DOI: 10.1016/j.caeai.2024.100309.

[253] Yang, A., Li, Z., and Li, J. 2024. Advancing GenAI Assisted Programming–a Comparative Study on Prompt Efficiency and Code Quality Between GPT-4 and GLM-4. (Feb. 2024). DOI: 10.48550/arXiv.2402.12782.

[254] Yee-King, M., McCallum, L., Llano, M. T., Ruzicka, V., d'Inverno, M., and Grierson, M. 2020. Examining Student Coding Behaviours in Creative Computing Lessons Using Abstract Syntax Trees and Vocabulary Analysis. en. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Trondheim Norway, (June 2020), 273–279. ISBN: 978-1-4503-6874-2. DOI: 10.1145/3341525.3387408.

[255] S. A. Yoon and K. J. Baker-Doyle, eds. 2018. *Networked by Design: Interventions for Teachers to Develop Social Capital*. Routledge,Taylor & Francis Group, New York. ISBN: 978-1-138-56534-0.

[256] Zabala, E. and Narman, H. S. 2024. Development and Evaluation of an AI-Enhanced Python Programming Education System. In *2024 IEEE 15th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. ISBN: 979-8-3315-4090-6. IEEE, Yorktown Heights, NY, USA, (Oct. 2024), 787–792. DOI: 10.1109/UEMCON62879.2024.10754661.

[257] Zastudil, C., Rogalska, M., Kapp, C., Vaughn, J., and MacNeil, S. 2023. Generative AI in Computing Education: Perspectives of Students and Instructors. (Aug. 2023). DOI: 10.48550/arXiv.2308.04309.

[258] Zelle, J. M. 2024. *Python Programming: An Introduction to Computer Science*. eng. (Fourth ed.). OCLC: 1426299777. Franklin, Beedle & Associates Inc., Portland. ISBN: 978-1-59028-297-7.

Appendices

Note on Embedded Documents

This appendix contains scanned or embedded documents that have been scaled to fit the page layout. At standard zoom, some text or visual elements may appear small. However, all files are embedded in high resolution, allowing readers to zoom in for finer detail without loss of clarity.

A Ethics & Paperwork

A.1 TUA · Ethics Approval

TORRENS UNIVERSITY AUSTRALIA

Dr Aaron Bere, Principal Investigator (HDR Supervisor)

Mr Tristan Bunn, Investigator (HDR Candidate)

E-mail: ethics@torrens.edu.au

31/10/2024

Dear Aaron and Tristan

Human Research Ethics Approval Notification
Torrens University Australia Human Research Ethics Committee (HREC): Low & Negligible Risk (LNR) Panel

Ethics Application 0394 Evaluating the Impact of Python Programming Education Through Creative Computing Environments
E2.1 Low Risk (LNR)

Thank you for submitting 0394 for ethical review.

This research project was reviewed by the Low & Negligible Risk (LNR) Panel, as delegated by the Torrens University Australia Human Research Ethics Committee (HREC).

I am pleased to advise you that 0394 meets the requirements of the National Statement on Ethical Conduct in Human Research (2023) and ethical approval for this research project has been granted.

In this research project the methodology and participating site(s) are as stated in the application. If additional sites are engaged or withdrawn, the Principal Investigator is required to notify the Human Research Ethics Office.

The approved documents:

Document Type	File Name	Date	Version
CV	2. Curriculum Vitae (CV) - Aaron Bere	23/07/2024	1.0
CV	2. Curriculum Vitae (CV) - Michelle Douglas	23/07/2024	1.0
CV	2. Curriculum Vitae (CV) - Tristan Bunn	23/07/2024	1.0
Project Description	0. Project Description	13/09/2024	1.1
Participant Recruitment	3. Participant Recruitment Flyer	13/09/2024	1.1
PICF	4. Participant Information & Consent Form (PICF)	13/09/2024	1.1
Data Collection	5. Data Collection	13/09/2024	1.1
Research Endorsement	1. TUA Research Endorsement	15/10/2024	1.2
Response Summary	Ethics Application 0394 Response Summary	29/10/2024	1.0

Approval of this research project is valid from 31/10/2024 for a period of three (3) years, subject to the following conditions:

- The Principal Investigator will immediately report anything that might warrant review of ethical approval of the project.
- The Principal Investigator will notify the HREC of any event that requires an amendment to the originally approved ethics application. An Amendment Request is to be submitted, and any change must be approved prior to implementation.
- The Principal Investigator will notify the HREC of any plan to extend the duration of the project past the approval period. An Extension Request is to be submitted, and any change to the approval period must be approved prior to implementation.

Page 1 of 2

- The Principal Investigator will notify the HREC of any complaint received regarding ethical aspects of the project.
- The Principal Investigator will notify the HREC of any adverse or serious events that may impact participant welfare, in a timely manner.
- The Principal Investigator will provide an Annual Report / Final Report to the HREC and notify the HREC when the project is completed at all sites.
- The Principal Investigator will notify the HREC if the project is discontinued at a participating site before the expected completion date.
- The Principal Investigator will notify the HREC of their inability to continue as Principal Investigator including the name of and contact information for a replacement.
- The Principal Investigator will comply with audit requests or other forms of monitoring by the HREC, as required.

This letter constitutes ethical approval only. This research project cannot proceed at any site until site specific authorisation, or relevant permission, has been obtained from the appropriate authority.

Should you have any queries about the review of your research project please contact the Human Research Ethics Office, ethics@torrens.edu.au.

The Torrens University Australia HREC wishes you every success in your research.

Regards

Torrens University Australia HREC

The Torrens University Australia HREC is constituted and operates in accordance with the National Statement on Ethical Conduct in Human Research (2023)

Page 2 of 2

A.2 VUW · PhD Candidature

VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

Office of the Associate Dean (Postgraduate Research)
FACULTY OF ENGINEERING Te Wāhanga Ahunui Pūkaha
VICTORIA UNIVERSITY OF WELLINGTON, PO Box 600, Wellington 6140, New Zealand
Email: nukuhaka.sibanda@vuw.ac.nz **Web**: wgtn.ac.nz/engineering

Tuesday, 12 December 2023

To whom it may concern,

This is to confirm that Tristan Bunn has been enrolled part-time from 01-Oct-2019 until now in a Doctor of Philosophy program at Victoria University of Wellington. Tristan is doing a thesis in Computer Science (COMP692) supervised by Dr Craig Anslow.

Tristan had passed the confirmation of candidature and was transferred to full registration on 28-Feb-2022.

A PhD at Victoria Universtiy of Wellington is a 3-4 year full-time equivalent degree, barring any periods of leave of absence (suspension). Tristan's registration will have reached 43 months part-time on 31-Dec-2023, equivalent to total 1.792 EFTS.

Please refer to Tristan's thesis registration history on the second page for further information and contact me for any further information if required.

Kind Regards,

Associate Dean (Postgraduate Research)
Faculty of Engineering

1

Term	CRN	Course	EFTS	Start Date	End Date	Reason	Act Date	Comment	Workload FT/PT

2

B Data & Instruments

B.1 Thonny-py5mode · Assessment Brief

Assessment 2 Brief	
Subject Code and Title	ITP122 Introduction to Programming
Assessment	Intermediate Programming Tasks
Individual/Group	Group
Length	Source code and comments
Learning Outcomes	The Subject Learning Outcomes demonstrated by successful completion of the task below include: a) Select appropriate programming development tools and methodologies to meet the software requirements. b) Apply coding, debugging and testing skills in software development using a suitable IDE (integrated development environment) platform. c) Validate and verify computer programs to meet the software requirements.
Submission	Due by 11:55pm AEST/AEDT, Sunday, end of Module 08
Weighting	45%
Total Marks	100 marks

Assessment Task

You must complete **programming tasks** that demonstrate your understanding of decision logic, including (`if-else` statements) and loops (`for` and `while`), based on the concepts covered in Modules 2 to 8. Each program should include clear and concise documentation in the form of comments explaining the logic and functionality of the code.

Context

This assessment focuses on writing Python scripts that generate graphic output using the **Thonny Python IDE** with the **Thonny-py5mode plugin**. To set up your development environment for the assessment, watch the relevant installation guide video:

- **View Windows video guide** (software: https://github.com/villares/thonny-portable-with-py5/releases/latest)
- **View macOS & Linux video guide** (software: https://thonny.org)

In Assessment 1, you applied and explained the use of variables, expressions, basic and intermediate conditionals, and other fundamental programming concepts. In this assessment, you'll expand upon this knowledge by implementing intermediate-level decision logic and loops, marking an important step toward developing problem-solving skills and confidence in devising algorithmic solutions.

Key Objectives:

You're required to submit all the programming tasks for this Assessment:

- Implementing `if-else` statements for decision-making
- Using `for` and `while` loops to manage repetition
- Generating **graphic output** using the **Thonny-py5mode plugin**
- Providing well-structured comments to explain your code

Instructions

Each task begins with some code to get you started. The goal is to complete the script so that your output matches each graphic provided. Here's a summary of the different functions you'll need:

Fills & Strokes

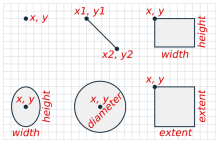
```
background(color)  # set bg color
fill(color)  # set fill color
no_fill()  # disable the fill
stroke(color)  # set stroke color
stroke_weight()  # stroke width in px
no_stroke()  # disable the stroke
```

A red fill using three different color values:

```
fill('#FF0000')  # hexadecimal
fill(255, 0, 0)  # red, green, blue
# HSB (Hue Saturation brightness)
color_mode(HSB, 360, 100, 100)
fill(0, 100, 100)
```

2D Primitives

```
point(x, y)
line(x1, y1, x2, y2)
rect(x, y, width, height)
ellipse(x, y, width, height)
circle(x, y, diameter)
square(x, y, extent)
```



Note that y-coordinates increase as you move downward, while x-coordinates increase as you move rightward. In other words, origin -- the coordinate (0, 0) -- is the top-left of the display window.

You'll add Python `if`, `else`, `for`, and other statements to those above to successfully complete each task.

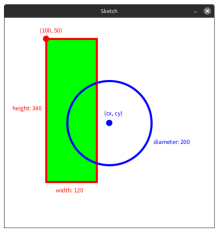
A Quick Demonstration

Using Thonny-py5mode, you can draw lines and shapes in various colours using different functions. Here's a basic, commented example:

```
# setup
size(500, 500)            # canvas size
background('#FFFFFF')     # white background colour
cx = width / 2            # canvas horizontal centre
cy = height / 2           # canvas vertical centre
stroke_weight(5)          # set outline to 5-pixels-wide

# draw rectangle
stroke('#FF0000')         # set outline to red
fill('#00FF00')           # set fill to green
rect(100, 50, 120, 340)   # draw rectangle

# draw circle
no_fill()                 # set fill to none
stroke('#0000FF')         # set outline to blue
circle(cx, cy, 200)       # draw circle
```

Note that commands like `stroke()` and `fill()` remain in effect until you override their behaviour, like using a paint brush -- when you dip the brush in a red pot, it'll paint in red shapes, until you dip it in blue. You'll notice that colours are specified using *hexadecimal* values. You can access a hexadecimal mixer using **py5 > Color selector** in the Thonny interface.

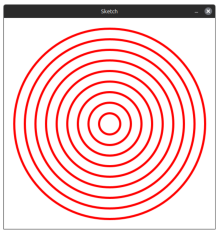
Here's the result, labelled so you can tell how the dimensions correspond to function arguments:



Task 1

Task 1.1

Replicate this result as closely as you can using different Python techniques and drawing functions:



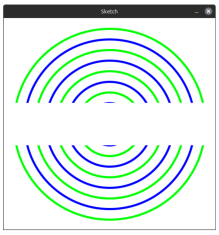
Module 6 (*Simple Loops*) covers the techniques you'll need to employ here.

Starter code:

```
# setup
size(500, 500)          # canvas size
background('#FFFFFF')   # white background colour
cx = width / 2          # canvas horizontal centre
cy = height / 2         # canvas vertical centre

i = 1

while i < 10:
    no_fill()
    stroke_weight(5)
    # ... INSERT MISSING CODE HERE ...
    i += 1
```

Task 1.2

Replicate this result as closely as you can using different Python techniques and drawing functions:



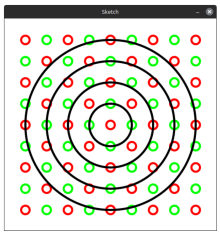
Consider how you might nest conditional statements within a loop (see *Module 8: Intermediate Loops*), and how you can visually mask lines by drawing shapes over them (filled the same colour as the background).

Starter code:

```
# setup
size(500, 500)          # canvas size
background('#FFFFFF')   # white background colour
cx = width / 2          # canvas horizontal centre
cy = height / 2         # canvas vertical centre
no_fill()               # set fill to none
stroke_weight(5)        # set outline to 5-pixels-wide

# ... INSERT MISSING CODE HERE ...
```


Task 1.3

Replicate this result as closely as you can using different Python techniques and drawing functions:



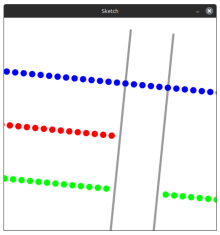
Consider how you might use loop within a loop to achieve a grid-like arrangement of shapes. Modules 5 and 8 cover *Nested Decision Logics* and *Intermediate Loops*, respectively.

Starter code:

```
size(500, 500)
background('#FFFFFF')

# ... INSERT MISSING CODE HERE ...
```


Task 2

Replicate this result as closely as you can using different Python techniques and drawing functions:



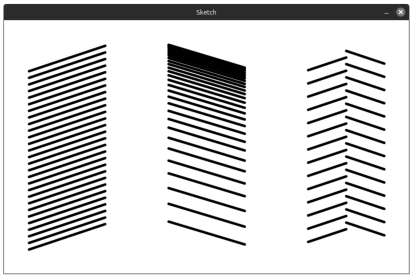
Use (`for`?) loops to draw the rows of blue, red, and green dots. You must use `break` and/or `continue` statements (see *Module 8: Intermediate Loops*) to control the interruptions of the red and green dots. **TIP:** Consider nesting `if` statements within your loops and how you might utilise the `line_1_x` and `line_2_x` variables.

Starter code:

```
size(500, 500)
background('#FFFFFF')
rotate(0.1)  # rotates the entire drawing space clockwise by ~6°

line_1_x = 300
line_2_x = 400

# draw two grey lines
stroke_weight(5)
stroke('#999999') # grey
line(300, 0, line_1_x, height)
line(400, 0, line_2_x, height)
no_stroke()

# loop for blue dots
# loop for red dots
# loop for green dots
```


Task 3

Replicate this result as closely as you can using different Python techniques and drawing functions:



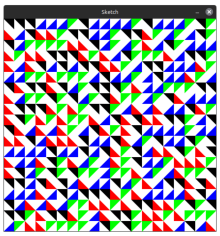
Draw all three patterns in a single canvas, using a separate loop for each. You'll need to use the `line()` function. You may find the `translate()` function useful: https://py.processing.org/reference/translate (although this isn't required).

Starter code:

```
size(800, 500)
background('#FFFFFF')
stroke('#000000')
stroke_weight(5)

# ... INSERT MISSING CODE HERE ...
```

Task 4

This task involves **Truchet tiles** (https://en.wikipedia.org/wiki/Truchet_tiles) -- a set of four contrasting tiles randomly arranged in a grid formation to make interesting patterns.



Here are the four tiles, which you can draw using the `triangle()` function:

Starter code:

```
size(500, 500)
background('#FFFFFF')
no_stroke()
tile_size = 25

# single blue tile demonstration
fill('#0000FF')
triangle(
    tile_size, 0,          # point 1 x-y coord
    tile_size, tile_size,  # point 2 x-y coord
    0, tile_size           # point 3 x-y coord
)
```

(continued on next page)

You'll need to use the `random()` function, which returns a random value between `0.0` and `0.1`

```
# ten random values between 0.0 and 1.0
for i in range(10):
    print(random())
```

As an example, the above code might produce:

```
0.12961569776802295
0.520657584472319
0.20049211567441561
0.7942788165062895
0.35182071797307446
0.4607723717260174
0.5828424497881715
0.911555329559598
0.8145882527706059
0.030076611396973485
```


Submission

Prepare your files

- Ensure you have written your student ID and name as a comment at the top of each file
- Make a **zip** folder containing all your files (namely, a Python file for each of the following tasks: 1.1, 1.2, 1.3, 2, 3, and 4)
- Name your zip file to match this convention: *ITP122_LastnameFirstname_A2.zip*

Credit any sources

As many online programming resources provide solutions to programming tasks along with documentation, if any such source code is acquired (reference works, documentation, help and tutorial sites, etc), it must be preceded by a code comment that lists the original site/creator and followed by a comment that declares the end of the acquired code. Acquisitions should be kept to a few lines or less and solve single problems (i.e., changing the range of a randomly generated number, using additional drawing functions you may have researched, and so forth).

Submit your files

Your submission will contain the zip archive of your project; submit this via the **Assessment 2 - Submission** link, accessible via the **Briefs & Submissions** link within main navigation menu of the *ITP122 Introduction to Programming* MyLearn portal.

Your Lecturer(s) will provide grades and feedback via MyLearn.

Before you submit your assessment, please ensure you have read and understood Torrens Academic Integrity policies: https://library.torrens.edu.au/academic_integrity. If you are unsure about anything, please reach out to your lecturer.

Assessment Attributes	Fail (Yet to achieve minimum) 0-49%	Pass (Functional) 50-64%	Credit (Proficient) 65-74%	Distinction (Advanced) 75-84%	High Distinction (Exceptional) 85-100%
Programming Task 1: Demonstrated understanding of the task and applied knowledge in programming the given task/problem. ***Percentage for this criterion = 25%***	Coding/implementation of the task produces large differences between the original and coded graphic. Poor code layout, commenting, and variable naming. Generally, a poor effort, and the program fails to run.	Coding/implementation of the task produces easily discernible differences between the original and coded graphic. Acceptable code layout, commenting, and meaningful variable names. Program runs, but with obvious room for code improvements.	Coding/implementation of the task produces some discernible differences between the original and coded graphic. Sensible code layout, commenting, and meaningful variable names. Program runs, but there is some room for improvement to enhance the accuracy of the replicated graphic.	Coding/implementation of the task produces minor discernible differences between the original and coded graphic. Good code layout, commenting, and meaningful variable names. Program runs well, exhibiting a solid grasp of Python programming concepts.	Coding/implementation of the task produces a highly accurate recreation of the original graphic. Code demonstrates best practices regarding layout, comments, variable names. Program runs very well, exhibiting a solid grasp of coding techniques in line with- and slightly beyond those covered in classes.
Programming Task 2: Demonstrated understanding of the task and applied knowledge in programming the given task/problem. ***Percentage for this criterion = 25%***	Coding/implementation of the task produces large differences between the original and coded graphic. Poor code layout, commenting, and variable naming. Generally, a poor effort, and the program fails to run.	Coding/implementation of the task produces easily discernible differences between the original and coded graphic. Acceptable code layout, commenting, and meaningful variable names. Program runs, but with obvious room for code improvements.	Coding/implementation of the task produces some discernible differences between the original and coded graphic. Sensible code layout, commenting, and meaningful variable names. Program runs, but there is some room for improvement to enhance the accuracy of the replicated graphic.	Coding/implementation of the task produces minor discernible differences between the original and coded graphic. Good code layout, commenting, and meaningful variable names. Program runs well, exhibiting a solid grasp of Python programming concepts.	Coding/implementation of the task produces a highly accurate recreation of the original graphic. Code demonstrates best practices regarding layout, comments, variable names. Program runs very well, exhibiting a solid grasp of coding techniques in line with- and slightly beyond those covered in classes.
Programming Task 3: Demonstrated understanding of the task and applied knowledge in programming the given task/problem. ***Percentage for this criterion = 25%***	Coding/implementation of the task produces large differences between the original and coded graphic. Poor code layout, commenting, and variable naming. Generally, a poor effort, and the program fails to run.	Coding/implementation of the task produces easily discernible differences between the original and coded graphic. Acceptable code layout, commenting, and meaningful variable names. Program runs, but with obvious room for code improvements.	Coding/implementation of the task produces some discernible differences between the original and coded graphic. Sensible code layout, commenting, and meaningful variable names. Program runs, but there is some room for improvement to enhance the accuracy of the replicated graphic.	Coding/implementation of the task produces minor discernible differences between the original and coded graphic. Good code layout, commenting, and meaningful variable names. Program runs well, exhibiting a solid grasp of Python programming concepts.	Coding/implementation of the task produces a highly accurate recreation of the original graphic. Code demonstrates best practices regarding layout, comments, variable names. Program runs very well, exhibiting a solid grasp of coding techniques in line with- and slightly beyond those covered in classes.
Programming Task 4: Demonstrated understanding of the task and applied knowledge in programming the given task/problem. ***Percentage for this criterion = 25%***	Coding/implementation of the task produces large differences between the original and coded graphic. Poor code layout, commenting, and variable naming. Generally, a poor effort, and the program fails to run.	Coding/implementation of the task produces easily discernible differences between the original and coded graphic. Acceptable code layout, commenting, and meaningful variable names. Program runs, but with obvious room for code improvements.	Coding/implementation of the task produces some discernible differences between the original and coded graphic. Sensible code layout, commenting, and meaningful variable names. Program runs, but there is some room for improvement to enhance the accuracy of the replicated graphic.	Coding/implementation of the task produces minor discernible differences between the original and coded graphic. Good code layout, commenting, and meaningful variable names. Program runs well, exhibiting a solid grasp of Python programming concepts.	Coding/implementation of the task produces a highly accurate recreation of the original graphic. Code demonstrates best practices regarding layout, comments, variable names. Program runs very well, exhibiting a solid grasp of coding techniques in line with- and slightly beyond those covered in classes.

C Publications Evidence

C.1 JISE · Acceptance Letter

Outlook

Major Revision – JISE #2508137

[redacted]

2 attachments (60 KB)
JISE_ReviewForm-#2508137-A.docx; JISE_ReviewForm-#2508137-B.docx;

EXTERNAL EMAIL: Please do not click on any link or open any attachment unless you trust the sender and know the content is safe.

Tristan,

Thank you for submitting "Evaluating Task-Technology Fit in Thonny-py5mode: Supporting Graphical Output in Introductory Python Education" (JISE #2508137) to the *Journal of Information Systems Education* (JISE). My reading of the reviewers' comments (attached) leads me to suggest that the paper is a good fit but a major revision of the paper must be done before we can consider publishing it.

I invite you to revise and resubmit this paper. Please use the reviewers' comments to guide your revision. Your revision will be sent through the review process again, with a different set of reviewers, before a decision will be made on publishing it, and therefore a Document of Revisions is NOT required at this time.

If you choose to revise and resubmit, the revised paper should be sent by email to jise-editor@una.edu.

Thank you for the opportunity to see your work. We hope you will continue to consider JISE as a primary outlet for your IS educational research in the future.

Sincerely,

Paul

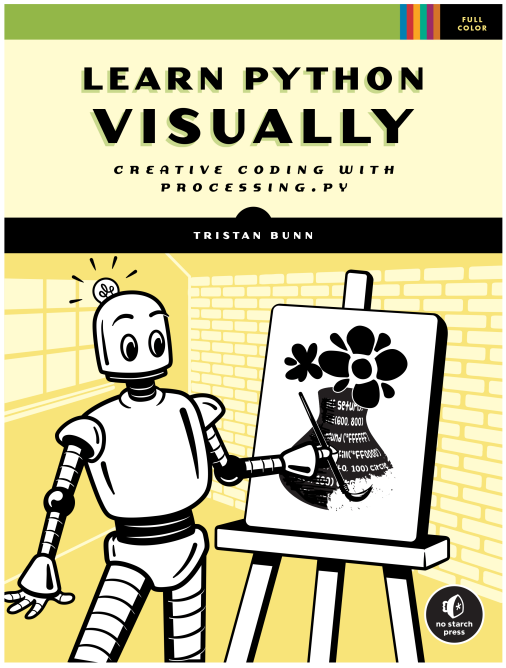
[redacted]

Board of Directors, ISCAP (Information Systems & Computing Academic Professionals)
Web: https://iscap.us/org/

Professor, Computer Information Systems

[redacted]

C.2 No Starch · Retail Book Cover



C.3 No Starch · Royalty Statement

no starch press

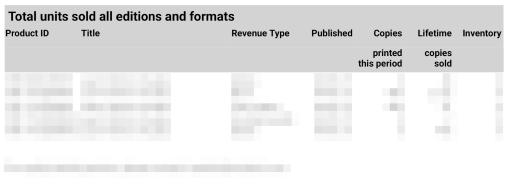
Tristan Bunn

No Starch Press
245 8th Street
San Francisco, CA 94103
USA

ROYALTY STATEMENT

Royalty Period: 01/01/2025 - 06/30/2025
Statement Frequency: Semi-annually

Page 1 of 2

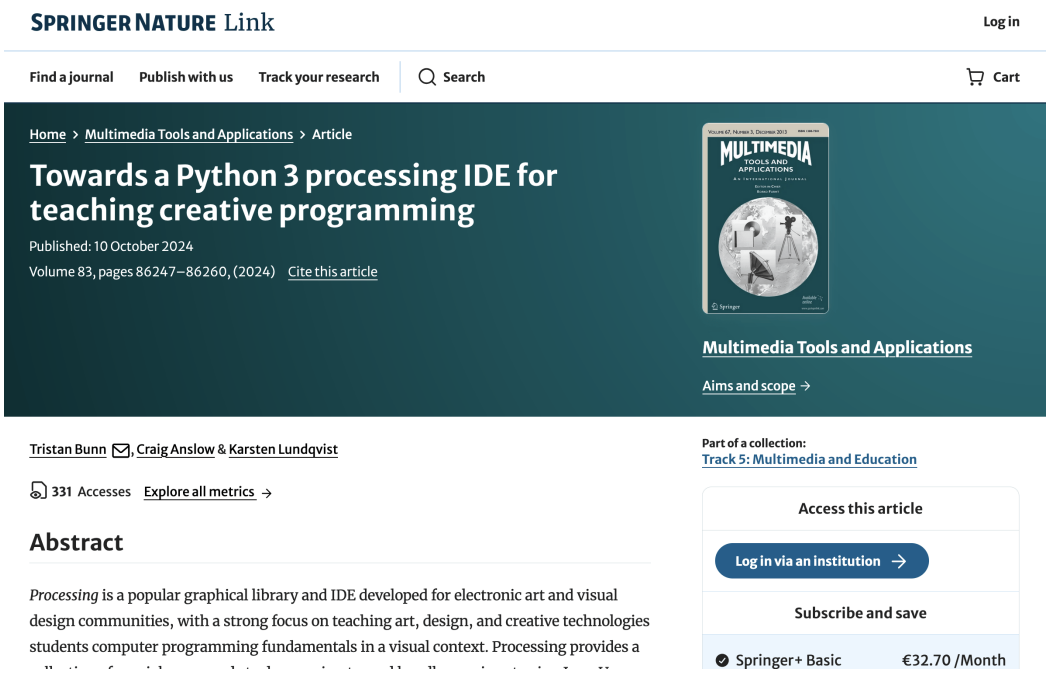


Total units sold all editions and formats

Product ID	Title	Revenue Type	Published	Copies printed this period	Lifetime copies sold	Inventory

Page 2 of 2

C.4 SpringerLink · MTAP Article



SPRINGER NATURE Link

Log in

Find a journal | Publish with us | Track your research | Search | Cart

Home > Multimedia Tools and Applications > Article

Towards a Python 3 processing IDE for teaching creative programming

Published: 10 October 2024

Volume 83, pages 86247–86260, (2024) Cite this article

Multimedia Tools and Applications

Aims and scope →

Tristan Bunn, Craig Anslow & Karsten Lundqvist

331 Accesses Explore all metrics →

Part of a collection:
Track 5: Multimedia and Education

Access this article

Log in via an institution →

Subscribe and save

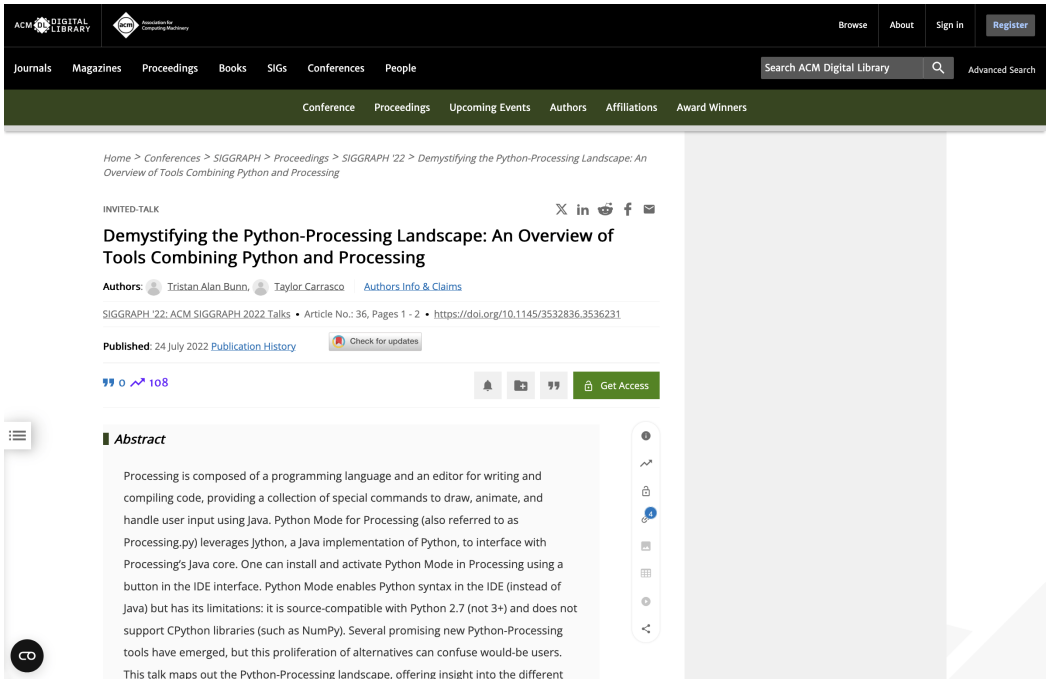
Springer+ Basic €32.70 /Month

Abstract

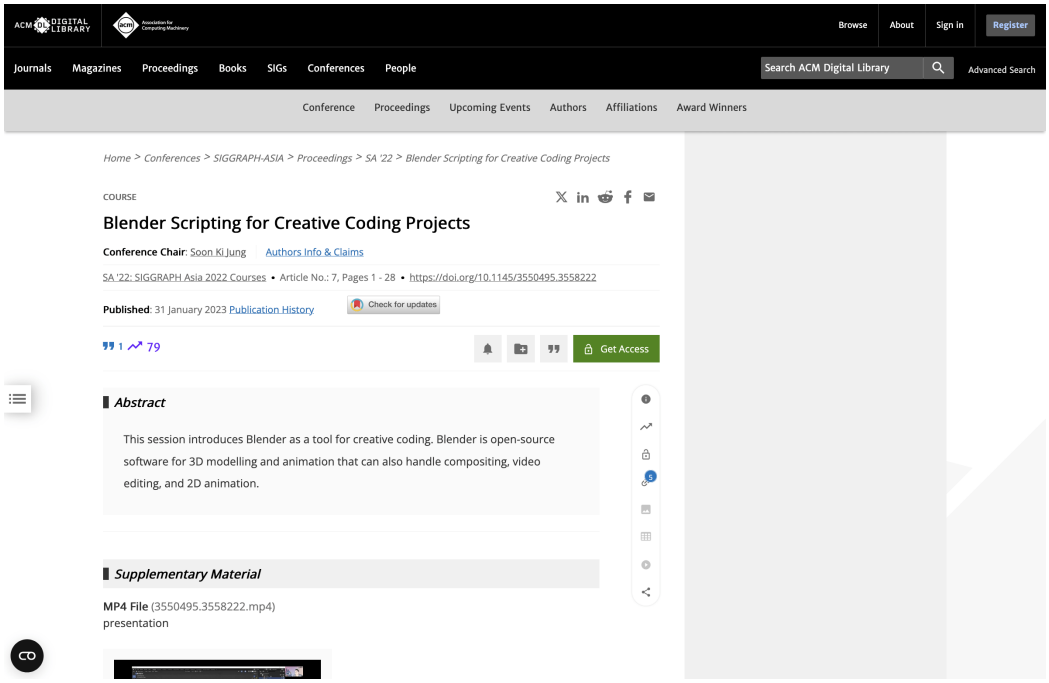
Processing is a popular graphical library and IDE developed for electronic art and visual design communities, with a strong focus on teaching art, design, and creative technologies students computer programming fundamentals in a visual context. Processing provides a

D Presentations Evidence

D.1 ACM Digital Library · SIGRRAPH '22 Proceedings Article

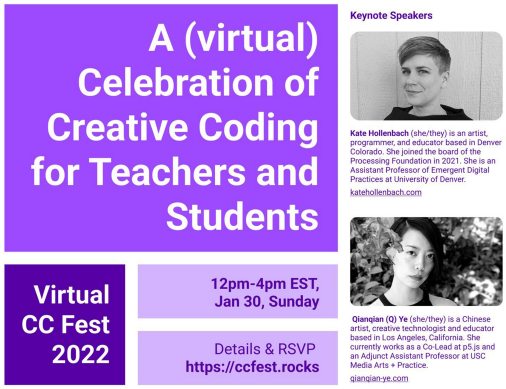


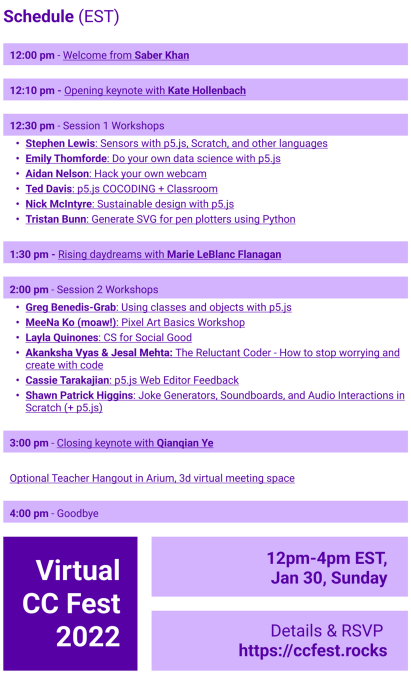
D.2 ACM Digital Library · SIGRRAPH Asia '22 Proceedings Article



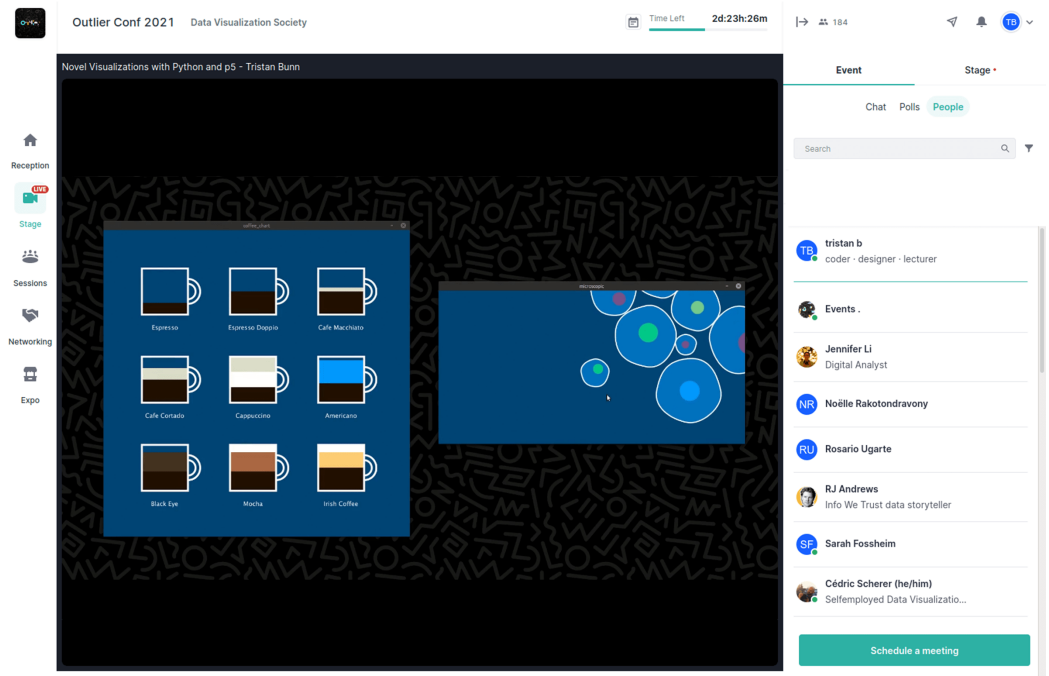
D.3 CC Fest 2021 · Event Programme

D.4 CC Fest 2022 · Event Programme

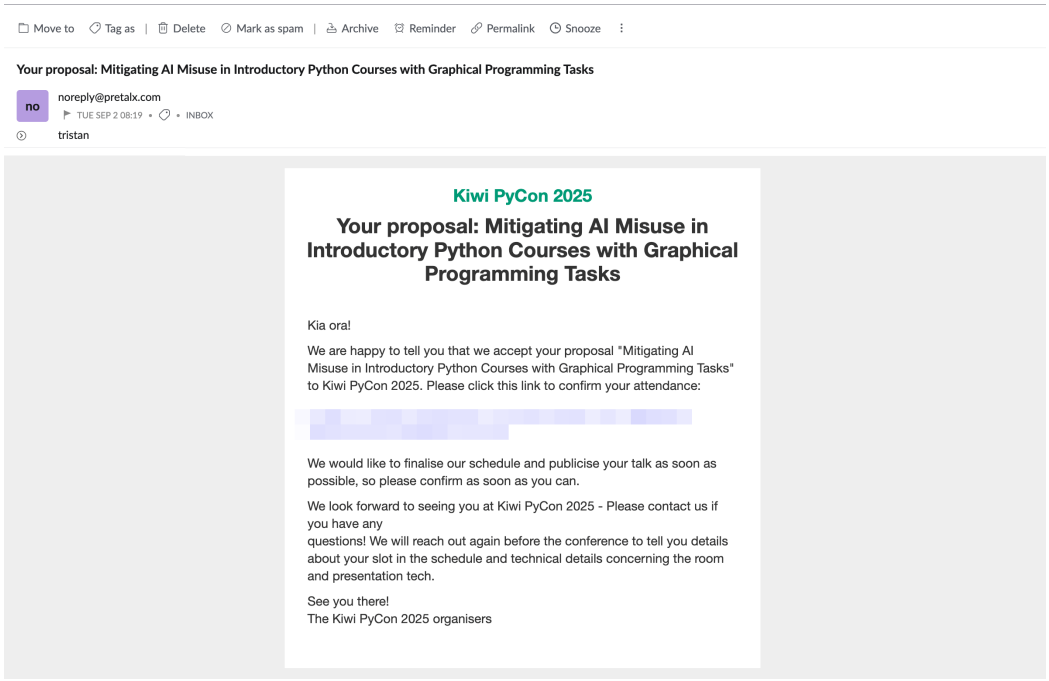




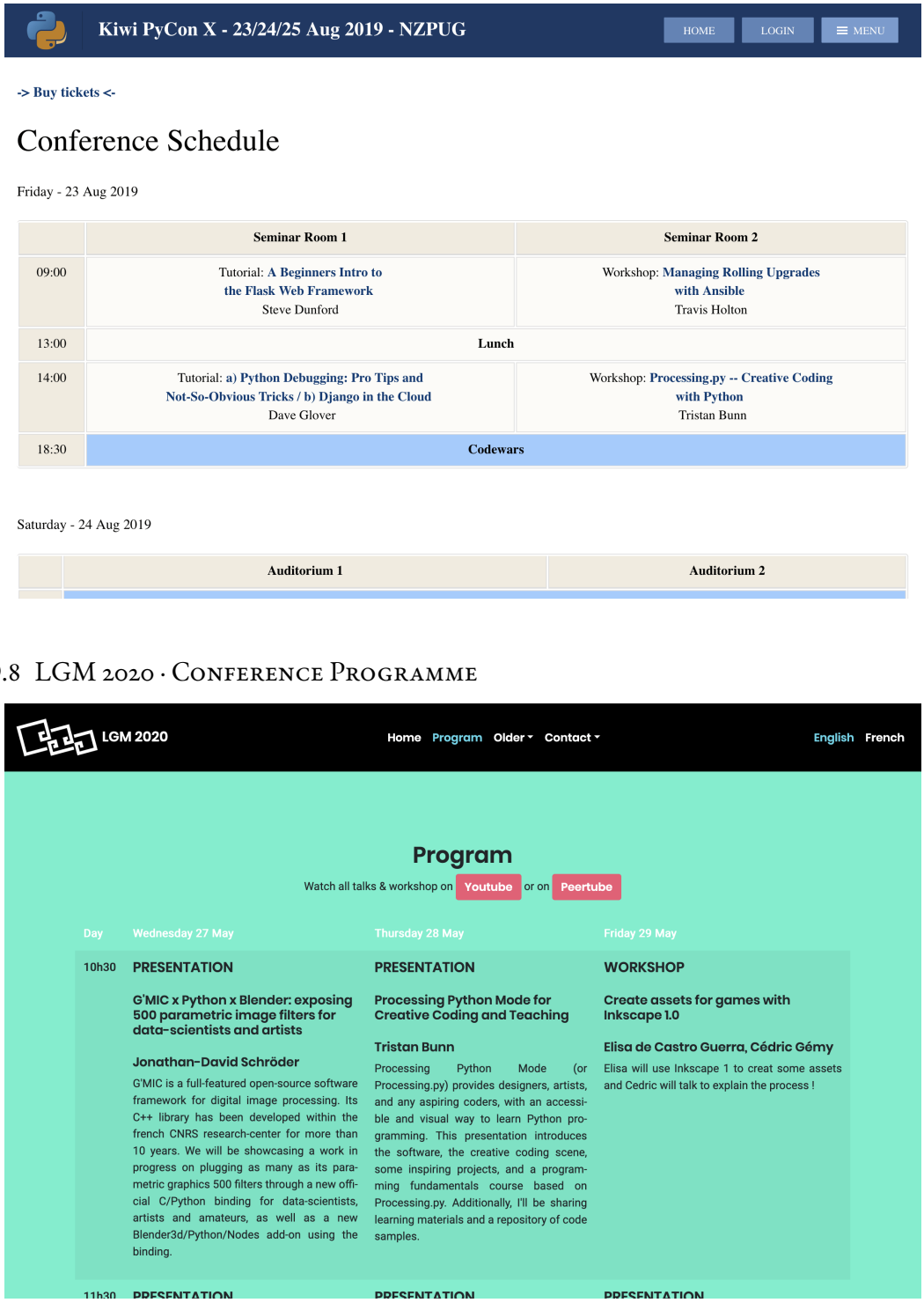
D.5 DVS Outlier · Event Stream



D.6 Kiwi PyCon 2025 · Acceptance Email



D.7 Kiwi PyCon X (2019) · Conference Programme



Kiwi PyCon X - 23/24/25 Aug 2019 - NZPUG

HOME LOGIN MENU

-> Buy tickets <-

Conference Schedule

Friday - 23 Aug 2019

	Seminar Room 1	Seminar Room 2
09:00	Tutorial: **A Beginners Intro to the Flask Web Framework** Steve Dunford	Workshop: **Managing Rolling Upgrades with Ansible** Travis Holton
13:00	**Lunch**	
14:00	Tutorial: **a) Python Debugging: Pro Tips and Not-So-Obvious Tricks / b) Django in the Cloud** Dave Glover	Workshop: **Processing.py -- Creative Coding with Python** Tristan Bunn
18:30	**Codewars**	

Saturday - 24 Aug 2019

	Auditorium 1	Auditorium 2

D.8 LGM 2020 · Conference Programme

LGM 2020

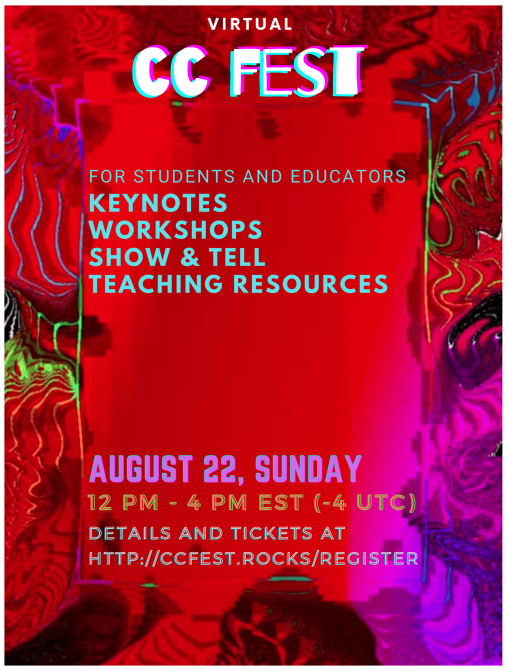
Home Program Older Contact English French

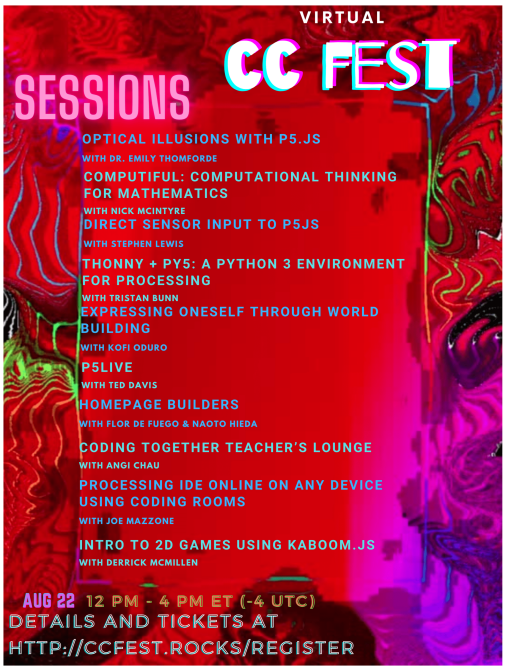
Program

Watch all talks & workshop on Youtube or on Peertube

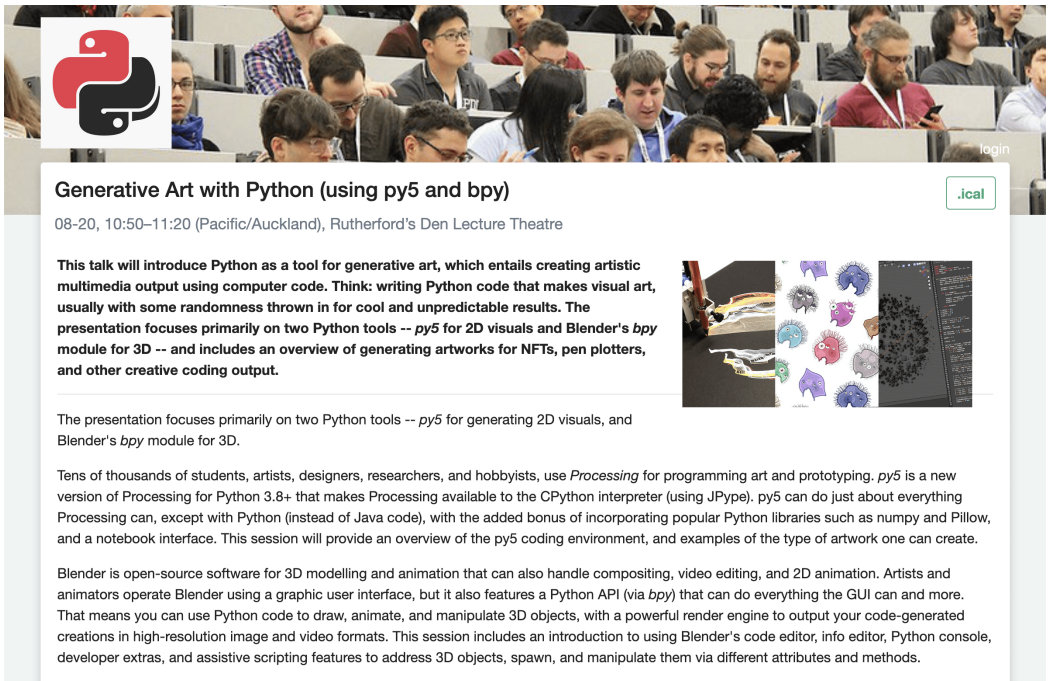
Day	Wednesday 27 May	Thursday 28 May	Friday 29 May
10h30	**PRESENTATION** **G'MIC x Python x Blender: exposing 500 parametric image filters for data-scientists and artists** **Jonathan-David Schröder** G'MIC is a full-featured open-source software framework for digital image processing. Its C++ library has been developed within the french CNRS research-center for more than 10 years. We will be showcasing a work in progress on plugging as many as its parametric graphics 500 filters through a new official C/Python binding for data-scientists, artists and amateurs, as well as a new Blender3d/Python/Nodes add-on using the binding.	**PRESENTATION** **Processing Python Mode for Creative Coding and Teaching** **Tristan Bunn** Processing Python Mode (or Processing.py) provides designers, artists, and any aspiring coders, with an accessible and visual way to learn Python programming. This presentation introduces the software, the creative coding scene, some inspiring projects, and a programming fundamentals course based on Processing.py. Additionally, I'll be sharing learning materials and a repository of code samples.	**WORKSHOP** **Create assets for games with Inkscape 1.0** **Elisa de Castro Guerra, Cédric Gémy** Elisa will use Inkscape 1 to creat some assets and Cedric will talk to explain the process !
11h30	**PRESENTATION**	**PRESENTATION**	**PRESENTATION**

D.9 PCD · 2021 Event Programme



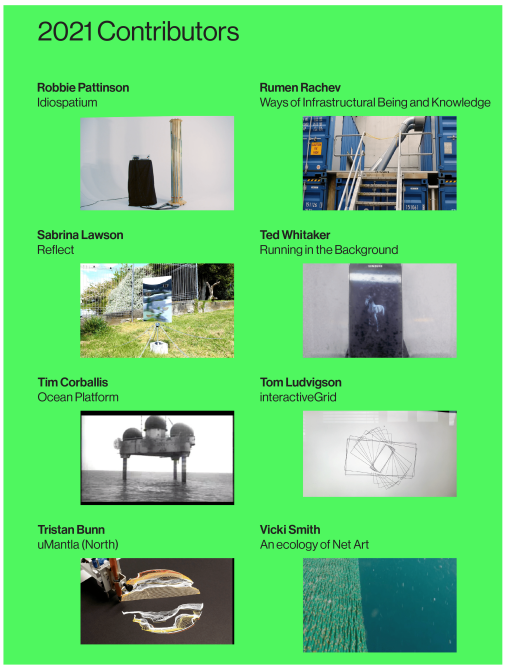


D.10 PyCon XI (2022) · Conference Programme



E Creative Works Evidence

E.1 ADA 2021 · Event Programme



E.2 {Between} · Exhibition Programme

PCD@COIMBRA 2021

{BETWEEN} — AN INVENTORY OF ANACHRONIC PRACTICES
08.12.2021 – 10.12.2021
NEST COLLECTIVE DOWNTOWN

reference manual shows it obeys computing paradigms of the 1970-80s, (e.g. 6 bit bytes, hand-made assembly) but nonetheless boasts time-travel assisted computing capabilities.

THE END OF RANDOM_SEED (INT (MD5 (B'KEPLER 186F').HEXDIGEST(), 16))
Tristan Bunn (tabreturn) (NZ)

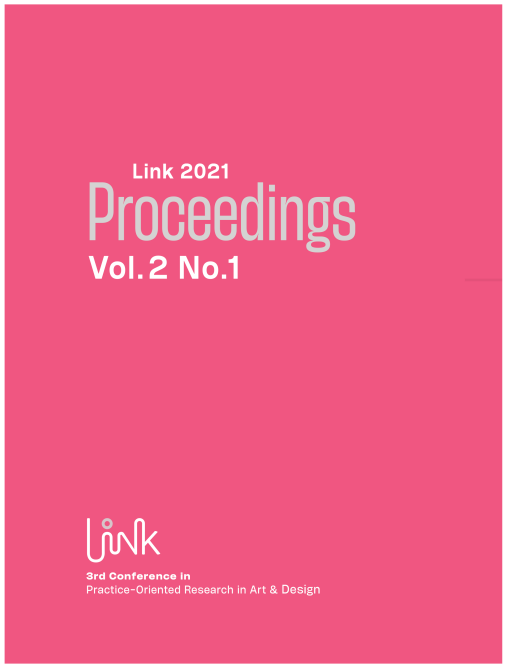
Intelligent life likely exists elsewhere in our universe, and we are unremarkable technologically and societally—an anachronism in the universe's grand, evolutionary timeline. Humanity may precariously teeter a single technological leap away from signalling Earth as a nascent interstellar threat, to be abruptly ended at the whim of some alien race. This poster is one of a series of generative artworks, each a variation depicting the annihilation of a planet—in this case, Kepler 186f.

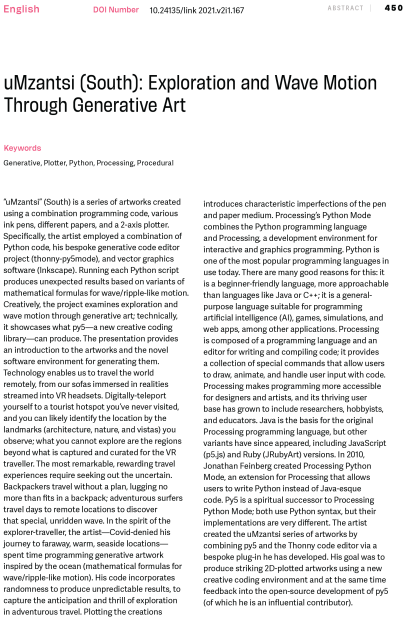
VARIOUS LIFE FORMS INVESTIGATING THE OLD ROAD IN THE POLISH MOUNTAINS
Bart Pilarczyk (PL/CA)

This piece explores the ideas of our presence and differences. Although we are all human, we are all different There are similarities that help us to plan, create groups, and work together. On

PROGRAM
ABOUT IG FB TW

E.3 LINK 2021 · Conference Proceedings





English DOI Number 10.24135/link 2021.v2i1.167 ABSTRACT | 450

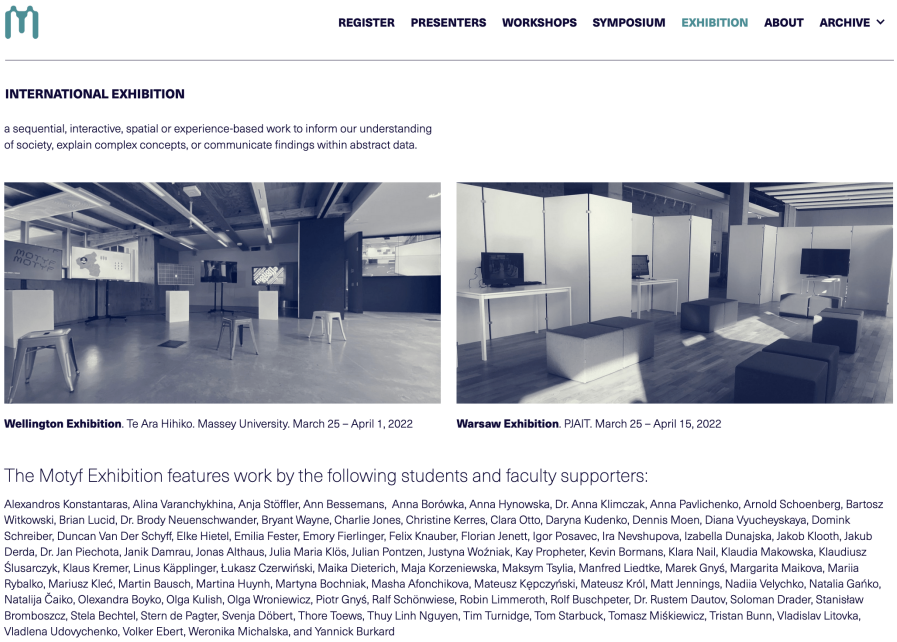
uMzantsi (South): Exploration and Wave Motion Through Generative Art

Keywords

Generative, Plotter, Python, Processing, Procedural

"uMzantsi" (South) is a series of artworks created using a combination programming code, various ink pens, different papers, and a 2-axis plotter. Specifically, the artist employed a combination of Python code, his bespoke generative code editor project (thonny-py5mode), and vector graphics software (Inkscape). Running each Python script produces unexpected results based on variants of mathematical formulas for wave/ripple-like motion. Creatively, the project examines exploration and wave motion through generative art; technically, it showcases what py5—a new creative coding library—can produce. The presentation provides an introduction to the artworks and the novel software environment for generating them. Technology enables us to travel the world remotely, from our sofas immersed in realities streamed into VR headsets. Digitally-teleport yourself to a tourist hotspot you've never visited, and you can likely identify the location by the landmarks (architecture, nature, and vistas) you observe; what you cannot explore are the regions beyond what is captured and curated for the VR traveller. The most remarkable, rewarding travel experiences require seeking out the uncertain. Backpackers travel without a plan, lugging no more than fits in a backpack; adventurous surfers travel days to remote locations to discover that special, unridden wave. In the spirit of the explorer-traveller, the artist—Covid-denied his journey to faraway, warm, seaside locations—spent time programming generative artwork inspired by the ocean (mathematical formulas for wave/ripple-like motion). His code incorporates randomness to produce unpredictable results, to capture the anticipation and thrill of exploration in adventurous travel. Plotting the creations introduces characteristic imperfections of the pen and paper medium. Processing's Python Mode combines the Python programming language and Processing, a development environment for interactive and graphics programming. Python is one of the most popular programming languages in use today. There are many good reasons for this: it is a beginner-friendly language, more approachable than languages like Java or C++; it is a general-purpose language suitable for programming artificial intelligence (AI), games, simulations, and web apps, among other applications. Processing is composed of a programming language and an editor for writing and compiling code; it provides a collection of special commands that allow users to draw, animate, and handle user input with code. Processing makes programming more accessible for designers and artists, and its thriving user base has grown to include researchers, hobbyists, and educators. Java is the basis for the original Processing programming language, but other variants have since appeared, including JavaScript (p5.js) and Ruby (JRubyArt) versions. In 2010, Jonathan Feinberg created Processing Python Mode, an extension for Processing that allows users to write Python instead of Java-esque code. Py5 is a spiritual successor to Processing Python Mode; both use Python syntax, but their implementations are very different. The artist created the uMzantsi series of artworks by combining py5 and the Thonny code editor via a bespoke plug-in he has developed. His goal was to produce striking 2D-plotted artworks using a new creative coding environment and at the same time feedback into the open-source development of py5 (of which he is an influential contributor).

E.4 Motyf 2021/2022 · Exhibition Website



REGISTER PRESENTERS WORKSHOPS SYMPOSIUM EXHIBITION ABOUT ARCHIVE

INTERNATIONAL EXHIBITION

a sequential, interactive, spatial or experience-based work to inform our understanding of society, explain complex concepts, or communicate findings within abstract data.

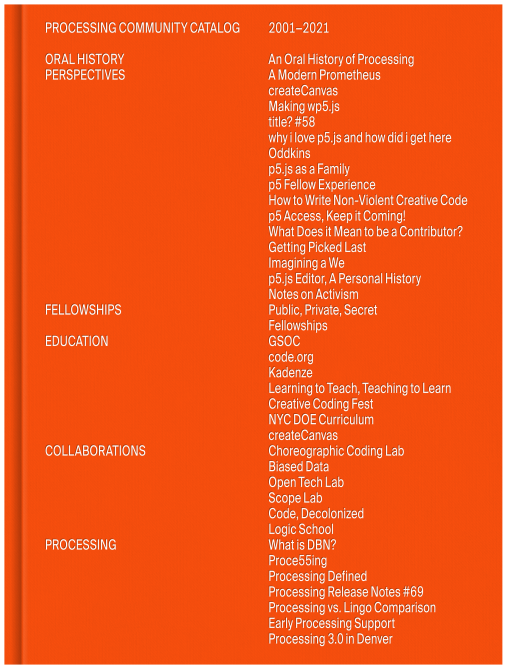
Wellington Exhibition. Te Ara Hihiko. Massey University. March 25 – April 1, 2022

Warsaw Exhibition. PJAIT. March 25 – April 15, 2022

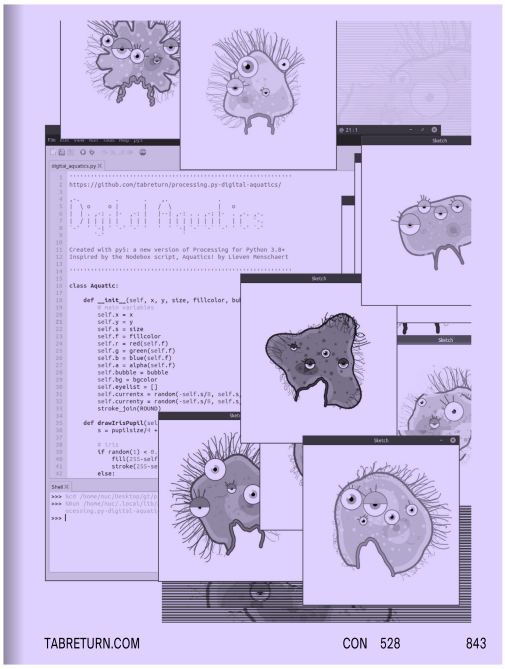
The Motyf Exhibition features work by the following students and faculty supporters:

Alexandros Konstantaras, Alina Varanchykhina, Anja Stöffler, Ann Bessemans, Anna Borówka, Anna Hynowska, Dr. Anna Klimczak, Anna Pavlichenko, Arnold Schoenberg, Bartosz Witkowski, Brian Lucid, Dr. Brody Neuenschwander, Bryant Wayne, Charlie Jones, Christine Kerres, Clara Otto, Daryna Kudenko, Dennis Moen, Diana Vyucheyskaya, Domink Schreiber, Duncan Van Der Schyff, Elke Hietel, Emilia Fester, Emory Fierlinger, Felix Knauber, Florian Jenett, Igor Posavec, Ira Nevshupova, Izabella Dunajska, Jakob Klooth, Jakub Derda, Dr. Jan Piechota, Janik Damrau, Jonas Althaus, Julia Maria Klös, Julian Pontzen, Justyna Woźniak, Kay Propheter, Kevin Bormans, Klara Nail, Klaudia Makowska, Klaudiusz Ślusarczyk, Klaus Kremer, Linus Käpplinger, Łukasz Czerwiński, Maika Dieterich, Maja Korzeniewska, Maksym Tsylia, Manfred Liedtke, Marek Gryś, Margarita Maikova, Mariia Rybalko, Mariusz Kleć, Martin Bausch, Martina Huynh, Martyna Bochniak, Masha Afonchikova, Mateusz Kępczyński, Mateusz Król, Matt Jennings, Nadiia Velychko, Natalia Gańko, Natalija Čaiko, Olexandra Boyko, Olga Kulish, Olga Wroniewicz, Piotr Gryś, Ralf Schönwiese, Robin Limmeroth, Rolf Buschpeter, Dr. Rustem Dautov, Soloman Drader, Stanisław Bromboszcz, Stela Bechtel, Stern de Pagter, Svenja Döbert, Thore Toews, Thuy Linh Nguyen, Tim Turnidge, Tom Starbuck, Tomasz Miśkiewicz, Tristan Bunn, Vladislav Litovka, Vladlena Udovychenko, Volker Ebert, Weronika Michalska, and Yannick Burkard

E.5 Processing Community Catalog 2001-2021 (2023)



PROCESSING COMMUNITY CATALOG	2001–2021
ORAL HISTORY	An Oral History of Processing
PERSPECTIVES	A Modern Prometheus
	createCanvas
	Making wp5.js
	title? #58
	why i love p5.js and how did i get here
	Oddkins
	p5.js as a Family
	p5 Fellow Experience
	How to Write Non-Violent Creative Code
	p5 Access, Keep it Coming!
	What Does it Mean to be a Contributor?
	Getting Picked Last
	Imagining a We
	p5.js Editor, A Personal History
	Notes on Activism
FELLOWSHIPS	Public, Private, Secret
	Fellowships
EDUCATION	GSOC
	code.org
	Kadenze
	Learning to Teach, Teaching to Learn
	Creative Coding Fest
	NYC DOE Curriculum
	createCanvas
COLLABORATIONS	Choreographic Coding Lab
	Biased Data
	Open Tech Lab
	Scope Lab
	Code, Decolonized
	Logic School
PROCESSING	What is DBN?
	Proce55ing
	Processing Defined
	Processing Release Notes #69
	Processing vs. Lingo Comparison
	Early Processing Support
	Processing 3.0 in Denver



F Statements of Authorship

TORRENS UNIVERSITY AUSTRALIA

Statement of Authorship

Publication/output:
Evaluating TTF in Thonny-py5mode: Supporting Graphical Output in Introductory Python Education
Authors: (1) Tristan Bunn, (2) Aaron Bere, (3) Michelle Douglas

This research activity forms part of Tristan Bunn's PhD by folio/publication. As a condition for inclusion in the PhD portfolio, Tristan Bunn must be listed as the first author; he must have contributed at least 50% to the research, development, and authorship of the output. He also serves as the primary contact for all co-authors and is responsible for managing the submission, revision, and preparation of all associated materials.

Record of author activity

Activity	Author 1	Author 2	Author 3
Research concept, questions, and warrant	90%	5%	5%
Human ethics approval	90%	5%	5%
Literature search	70%	30%	
Research design	40%	60%	
Data collection	50%	50%	
Data analysis	10%	90%	
Manuscript writing	60%	30%	10%

Total contributions

	65%	**30%**	**5%**

Acknowledgement of authorship
By signing below, authors acknowledge this agreement as a true record of the contribution of each author.

Author name	Signature	Date
[illegible]	[illegible]	[illegible]

TORRENS UNIVERSITY AUSTRALIA

Statement of Authorship

Publication/output:
Towards a Python 3 Processing IDE for Teaching Creative Programming
Authors: (1) Tristan Bunn, (2) Craig Anslow, (3) Karsten Lundqvist
https://doi.org/10.1007/s11042-024-20345-1

This research activity forms part of Tristan Bunn's PhD by folio/publication. As a condition for inclusion in the PhD portfolio, Tristan Bunn must be listed as the first author; he must have contributed at least 50% to the research, development, and authorship of the output. He also serves as the primary contact for all co-authors and is responsible for managing the submission, revision, and preparation of all associated materials.

Record of author activity

Activity	Author 1	Author 2	Author 3
Research concept, questions, and warrant	70%	20%	10%
Literature search	80%	10%	10%
Software development	100%		
Data analysis	80%	10%	10%
Manuscript writing	80%	10%	10%

Total contributions

	75%	**15%**	**10%**

Acknowledgement of authorship
By signing below, authors acknowledge this agreement as a true record of the contribution of each author.

Author name	Signature	Date
[illegible]	[illegible]	[illegible]

TORRENS UNIVERSITY AUSTRALIA

Statement of Authorship

Publication/output:
Generative Art with Python (using py5 and bpy)
Authors: (1) Tristan Bunn, (2) Taylor Carrasco
https://www.youtube.com/playlist?list=PLBGI1tVyiWQT0Gq9NwrA2Rb7m16Sb_CRu

This research activity forms part of Tristan Bunn's PhD by folio/publication. As a condition for inclusion in the PhD portfolio, Tristan Bunn must be listed as the first author; he must have contributed at least 50% to the research, development, and authorship of the output. He also serves as the primary contact for all co-authors and is responsible for managing the submission, revision, and preparation of all associated materials.

Record of author activity

Activity	Author 1	Author 2	Author 3
Research concept, questions, and warrant	60%	40%	
Literature search	50%	50%	
Research design	60%	40%	
Materials development	70%	30%	
Presentation	50%	50%	

Total contributions

	60%	**40%**	

Acknowledgement of authorship
By signing below, authors acknowledge this agreement as a true record of the contribution of each author.

Author name	Signature	Date
[illegible]	[illegible]	[illegible]

TORRENS UNIVERSITY AUSTRALIA

Statement of Authorship

Publication/output:
Demystifying the Python-Processing Landscape:
An Overview of Tools Combining Python and Processing
Authors: (1) Tristan Bunn, (2) Taylor Carrasco
https://doi.org/10.1145/3532836.3536231

This research activity forms part of Tristan Bunn's PhD by folio/publication. As a condition for inclusion in the PhD portfolio, Tristan Bunn must be listed as the first author; he must have contributed at least 50% to the research, development, and authorship of the output. He also serves as the primary contact for all co-authors and is responsible for managing the submission, revision, and preparation of all associated materials.

Record of author activity

Activity	Author 1	Author 2	Author 3
Research concept, questions, and warrant	70%	30%	
Literature search	70%	30%	
Research design	70%	30%	
Materials development	70%	30%	
Presentation	80%	20%	

Total contributions

	75%	**25%**	

Acknowledgement of authorship
By signing below, authors acknowledge this agreement as a true record of the contribution of each author.

Author name	Signature	Date
[illegible]	[illegible]	[illegible]

TORRENS UNIVERSITY AUSTRALIA

Statement of Authorship

Publication/output:

Blender Scripting for Creative Coding Projects

Authors: (1) Tristan Bunn, (2) Taylor Carrasco

https://doi.org/10.1145/3550495.3558222

This research activity forms part of Tristan Bunn's PhD by folio/publication. As a condition for inclusion in the PhD portfolio, Tristan Bunn must be listed as the first author; he must have contributed at least 50% to the research, development, and authorship of the output. He also serves as the primary contact for all co-authors and is responsible for managing the submission, revision, and preparation of all associated materials.

Record of author activity

Activity	Author 1	Author 2	Author 3
Research concept, questions, and warrant	60%	40%	
Literature search	60%	40%	
Research design	70%	30%	
Materials development	70%	30%	
Presentation		100%	

Total contributions

	60%	**40%**	

Acknowledgement of authorship

By signing below, authors acknowledge this agreement as a true record of the contribution of each author.

Author name	Signature	Date
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]