'Squashing bugs not snakes': co-creating a Python toolkit to prepare engineers for a sustainable future

Muhie Al Haimus, Silvia Valls Santafe and Dr. Rehan Shah

Queen Mary University of London

THE LONDON STUDENT SUSTAINABILITY CONFERENCE 2026

INTRODUCTION

With **digital problem-solving** becoming central to tackling global sustainability challenges, programming has become a core skill for students to acquire. However, many students begin **university with little or no prior exposure to coding**, making their initial experience feel **unfamiliar and fragmented**. This can limit their sense of inclusion, reducing access to the digital skills needed to **thrive in today's technology-driven world.**

PROJECT AIMS

- To bridge the knowledge gap and create a **'level playing field'** between students with **varying levels of programming experience**
- **To help students** avoid frustration by **addressing common errors** that beginners often make
- **To build confidence** by delivering targeted practice questions that **reinforce fundamental concepts**
- **To enhance students' problem-solving skills** through a **practice-oriented learning approach**, in alignment with the QMUL Graduate Attributes framework

IMPLEMENTATION

- **Co-creation** (staff-student partnership) of teaching toolkit resources
- **Components of toolkit:**
  - Common Errors Guide
  - Practice Question Toolkit
  - Installation Videos
  - Weekly Summary Videos
- **Implemented** in a first-year undergraduate applied mathematics module (450 students) over 2-3 years
- Embedded as an **asynchronous**, **formative resource** for students to use alongside course content

```python
# Author: Muhie
# Date: 21/05/24
# Explanation of the program: A program that finds fractions
def get_valid_input(message, zero_check):
    valid_input = False
    while valid_input == False:
        user_input = int(input(message))
        if zero_check == False:
            return user_input
        if  user_input != 0:
            valid_input = True
            return user_input
        print("Invalid input, you cannot divide by zero, plese try again")

numerator = get_valid_input("Welcome to the fraction calculator, " +
                            "please enter the numerator", False)
denomenator = get_valid_input("Please enter the denomenator", True)
fraction = numerator/denomenator
print(fraction)
```

```
Welcome to the fraction calculator, please enter the numerator 10
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 0
Invalid input, you cannot divide by zero, plese try again
Please enter the denomenator 8
1.25
```

Figure 22: Code with input validation

5.3 Creating infinite loops by forgetting to add a stopping condition (base case)

The simplest case where a user could accidentally create an infinite loop is through the use of the 'while' loop functions. An example case consists of using 'while True:', this statement in conjunction with the boolean True, makes it so the while loop runs infinitely until the loop is manually broken by the user. As shown below, a while statement and the condition $a < b$ (which is always true), makes the loop run infinitely, since there is never a condition that causes the loop to break. For example:

2.6 Week 5: Getting familiar with numpy

1) Create a 3x2 matrix, initially filling it with zeros, then ask the user to input a value for each element in the matrix (the inputs should only be floats). You can make sure that the input type is correct by creating a separate method *validate_inputs* and rejects any rouge value. Finally, print out the matrix.

2) 3D rotations: Ask the user to input an XYZ point and then ask which axis the user would like to rotate around. Then output the rotated point to the screen.

For an anticlockwise rotation around the x-axis:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}$$

For an anticlockwise rotation around the y-axis:

$$\begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

For an anticlockwise rotation around the z-axis:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

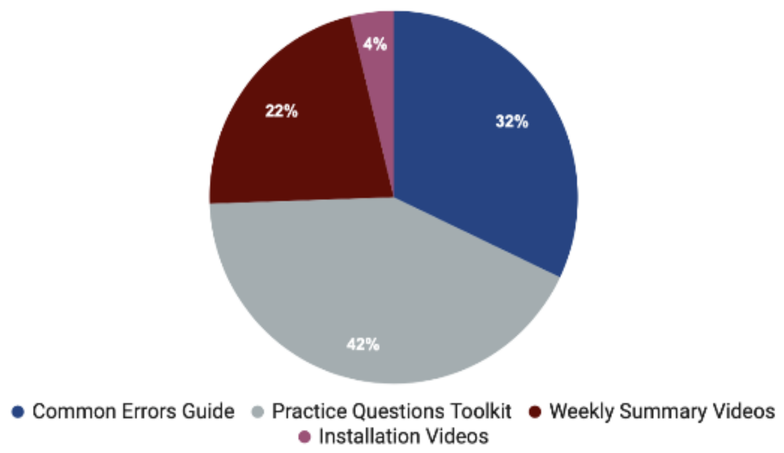
Video Playlists

Practice Questions Toolkit

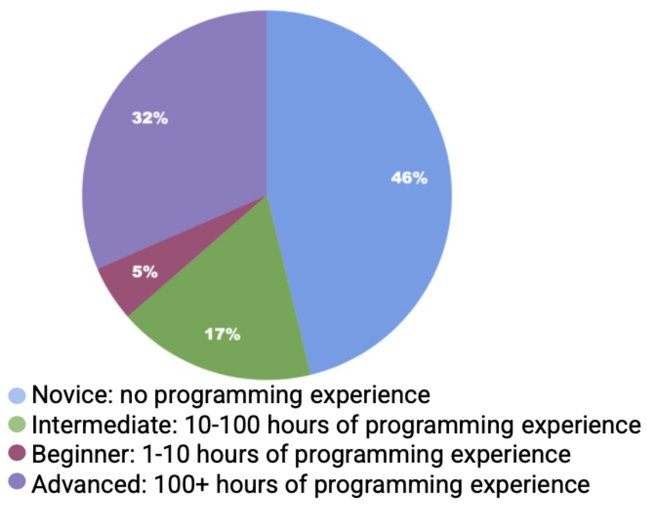
Common Errors Toolkit

Module Survey (184 responses)

Modes of the Python Toolkit most valued by students



How would you describe your previous Python programming experience?



STUDENT FEEDBACK

*"The common errors toolkit was without a doubt one of the best resources to **understand the mistakes I made**. I found myself wasting precious time trying to understand why my code was not working but after using the guide, **most issues were solved** within minutes."*

*"I liked the sections that explain the common syntax and runtime errors, since they show **real life examples** and the screenshots helped me deepen my understanding. The practice questions **reinforced my knowledge of the debugging process** in Python. Overall this toolkit has **helped me get better at Python**."*

KEY FINDINGS OF STUDENT FEEDBACK

Student feedback was **positive**, highlighting the toolkit as a practical resource that effectively supported the development of foundational programming skills and **contributed to a high-quality learning (SDG4) experience**.

Several respondents also suggested extending the resource with **additional practice questions** and further video-based support to help with more advanced content to reduce learning barriers.

CONCLUSION AND FUTURE WORK

- This project demonstrates that a Python learning toolkit **can improve student's skills and engagement**.
- The toolkit **successfully** addressed a key challenge in higher education in line with the UN SDGs.
- Future iterations will extend the toolkit by including **more video/interactive resources to cover machine learning and AI concepts.**