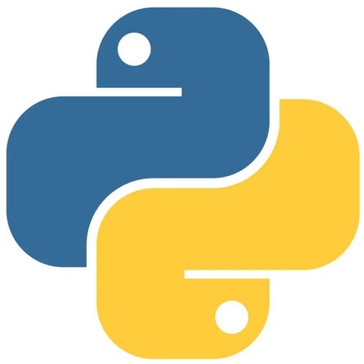
'Squashing Bugs, Not Snakes': Co-creating a Python toolkit to develop students' programming proficiency

Muhie Al Haimus, Yash Vaghela, Ilanthiraiyan Sivagnanamoorthy and Dr. Rehan Shah

Queen Mary University of London

INTRODUCTION & MOTIVATION

Feedback from first-year engineering students at QMUL revealed that they found programming **rather difficult,** particularly those without prior experience of it from their early education. Moreover, they felt that programming was **taught very sparsely** to engineers, leading to a lack of conceptual understanding, as it is not explained from the foundations of how a programming language works, making it **hard to conceptualise**.



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 handbook
  - Practice questions booklet
  - Video resources
- **Piloted** in a first-year undergraduate applied mathematics module
- 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:

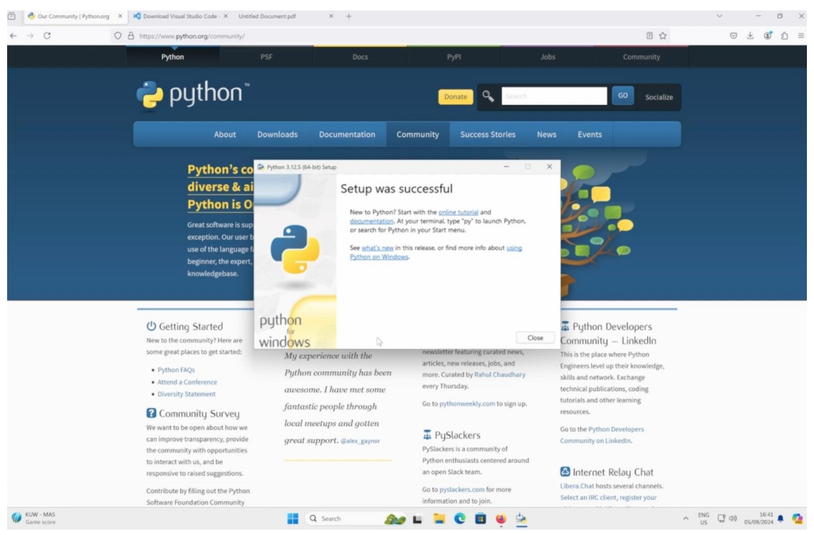
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}$$

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:

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



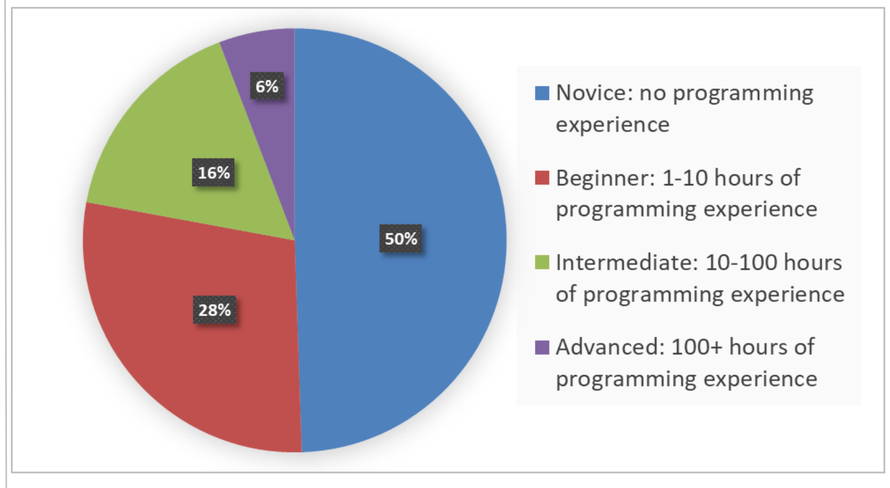
Practice Questions Toolkit

Video Playlists

Common Errors Toolkit

Module Survey (190 responses)

How would you describe your previous Python programming experience?



STUDENT FEEDBACK

*"The **toolkit has lots of non-obvious but also obvious faults and mistake**, which can be difficult to spot as a beginner and so it has been **useful for me to see what I have been making mistakes on**."*

*"I like that there are questions asking you to **identify where the errors are** as this is always a **good skill** to have when coding in general."*

*"It gives **clear and detailed notes** related to the mathematical topics which are covered in the course"*

KEY FINDINGS OF STUDENT FEEDBACK

Student feedback was positively commented on how the **toolkit design** started from a **foundational approach** by addressing **frequently made errors first**.

Many of the respondents also highlighted that they would like **additional resources** covering different python libraries covered in the module such as **numpy**, **matplotlib** and **pandas**.

CONCLUSION AND FUTURE WORK

- **Incorporate feedback and make necessary adjustments**, including additional snippets and examples for future iterations
- **Include more video/interactive resources** to assist students who prefer different teaching styles
- **Incorporate machine learning elements** to cater for the second-year applied mathematics and data science module
- **Future dissemination** at QMUL Festival of Education and UK EERN conference in 2025

evasys

School of Engineering and Materials Science