The effect of automated error message feedback on undergraduate physics students learning Python: reducing anxiety and building confidence

Abstract

STEM fields, such as physics, increasingly rely on complex programs to analyse large datasets, thus teaching students the required programming skills is an important component of all STEM curricula. Since undergraduate students often have no prior coding experience, they are reliant on error messages as the primary diagnostic tool to identify and correct coding mistakes. However, such messages are notoriously cryptic and often undecipherable for novice programmers, presenting a significant learning hurdle that leads to frustration, discouragement, and ultimately a loss of confidence. Addressing this, we developed a tool to enhance error messages for the popular Python language, translating them into plain English to empower students to resolve the underlying error independently. We used a mixed methods approach to study the tool's effect on first-year physics students undertaking an introductory programming course. We find a broadly similar distribution of the most common error types to previous studies in other contexts. Our results show a statistically significant reduction in negative student emotions, such as frustration and anxiety, with the mean self-reported intensity of these emotions reducing by $73 \pm 12\%$ and $55 \pm 18\%$, respectively. This led to a corresponding decrease in discouragement and an increase in student confidence. We conclude that enhanced error messages provide an effective way to alleviate negative student emotions and promote confidence. However, further longer-term investigations are necessary to confirm if this translates into improved learning outcomes. To our knowledge, this is the first physics-specific investigation of the effect of Python error message enhancement on student learning.

Keywords: Python, novice programming, STEM, Physics, error enhancement, feedback

1 Introduction

As STEM fields increasingly rely upon generating large quantities of data as well as processing, analysing, and visualising this data, the ability to code has become a required skill for current and future generations of STEM degree graduates outside of computer science (CS), such as physics graduates. As such, teaching students computer programming has become a key component of the vast majority of STEM curricula (Cropper, 2018; Weiss, 2017; Wilson, 2006) and is an integral part of almost all physics degree programs.

Error messages are a vital ingredient of all computing languages, providing the details of any mistakes made when inputting the code. Programmers of all levels depend on the feedback error messages provide to understand, locate and ultimately fix issues with their programs. Such issues encompass both syntactic errors, where the user enters code that does not conform to the syntax rules defining the acceptable combinations of symbols for the language in question, or more general flaws in the design of the program that prevent correct execution, leading to run-time errors. Throughout this paper we refer to both collectively as 'bugs'.

Beginner programmers are no different. Encountering errors is an unavoidable part of any beginner programmer's learning experience and the lack of prior experience not only means that they are likely to make more mistakes and thus produce error messages, but they are also heavily reliant on these error messages as the primary diagnostic

tool (B.A. Becker et al., 2019; Marceau, Fisler, & Krishnamurthi, 2011). Consequently, being able to interpret these messages is a key ingredient in effectively teaching computer programming, allowing students to learn from their mistakes.

Unfortunately error messages can be difficult for beginners to decipher. Regardless of the specific programming language being used, the error messages have always been notoriously cryptic (B.A. Becker et al., 2019; Brown, 1983), especially to the uninitiated, often containing technical vocabulary or terminology that may be unfamiliar (Marceau et al., 2011). Perhaps unsurprisingly given the error messages are largely written with experienced programmers in mind, the lack of readability in error messages particularly affects novice programmers (Traver, 2010). Consequently, students often waste hours trying to correct or 'debug' even simple mistakes (e.g. Barik, Ford, Murphy-Hill, and Parnin (2018)), as supported by our results below.

Furthermore, the actual cause of the error is often hidden amongst a plethora of details (known as a traceback) about the how the program arrived at the point of error, resulting in off-putting detail that increases the cognitive load on the learner. According to Lahtinen, Ala-Mutka, and Järvinen (2005), when asked to grade the difficulty of various programming aspects and concepts, novice programmers responded that finding and fixing bugs is the greatest challenge.

Unexpected error messages can also be disheartening for students if they do not yet feel equipped with the knowledge and skills required to resolve the error. This experience can induce anxiety, which is a powerful hindrance in learning (Mandler & Sarason, 1952; Warr & Downing, 2010). The experience of positive emotions during learning (e.g. enjoyment, excitement, and pride) is widely-recognised as promoting confidence and engagement (Simon, Aulls, Dedic, Hubbard, & Hall, 2015; Sinatra, Heddy, & Lombardi, 2015; Villavicencio & Bernardo, 2016), with the influence on learning being less clear (Villavicencio & Bernardo, 2016). Despite the lack of evidence that positive emotions are a reliable indicator of learning, cognition or achievement (Ben-Eliyahu & Linnenbrink-Garcia, 2013; Finch, Peacock, Lazdowski, & Hwang, 2015; Villavicencio & Bernardo, 2016), it is clear that negative emotions, such as anxiety, boredom, or hopelessness, predict poorer academic achievement (Daniels et al., 2009; Mega, Ronconi, & de Beni. R., 2014; Pekrun, Lichtenfeld, Marsh, & Goetz, 2017) and leave students feeling discouraged (S. Murphy, MacDonald, Wang, & Danaia, 2019).

Without experience to draw on, or even enough context-specific knowledge to adequately search for a solution, often students' only course of action is to ask a demonstrator or a peer for help diagnosing the issue. Whilst peer-instruction is an effective learning strategy, it can be limiting if this is the only action a student can take with a broken piece of code. Ultimately, misunderstood error messages result in a "barrier to progress" (B.A. Becker, 2016a). In addition, the significant demonstrator time spent solving trivial problems, often repeating the explanation to individual students, has the knock-on effect of longer waiting times for assistance (Coull, 2008).

Large computer classes make it difficult to give detailed, individualised, formative feedback (Chow, Yacef, Koprinska, & Curran, 2017). Various systems have been invented to counteract this problem and these are described in the Related Work section of this paper.

According to Robins, Rountree, and Rountree (2003), students' attitudes to mistakes and errors matter. Students who become frustrated or are quick to negative emotional reactions are more likely to become what Perkins, Hancock, Hobbs, Martin, and Simmons (1989) refers to as "stoppers". Stoppers are students who, when encountering a problem where they do not see immediately how to proceed, will, as the name suggests, stop. Appearing to "abandon all hope of solving the problem on their own" (Perkins et al., 1989). The alternative category of novice programmer is a "mover". Movers are students who will attempt to modify their code, often experimenting through trial and error. Often movers use error feedback more effectively, allowing them to progress through a problem. In the case of "tinkerers" (a subset of movers), this is not always the case, since, as noted in Perkins, Hancock, Hobbs, Martin, and Simmons (1988), some tinkerers can alter their code non-systematically and have little understanding why the code is behaving how it is. Nevertheless, tinkerers like all movers, have a greater chance of solving the issue. This study attempts to help students use the error messages

and prevent students from falling into the "stopper" mindset.

In this paper, we focus explicitly on automating feedback related to Python error messages. Due to the relative ease of readability and the large standard library, the coding language Python has surged in popularity over recent years climbing in October 2021 to top position on the TIOBE Index, which measures the popularity of programming languages (TIOBE, 2021).

We present a tool called the Error Explainer, that hooks into Jupyter (Kluyver et al., 2016) notebooks to display a description of the error type in 'plain English' directly below the error message, and guides the student how to read and interpret the error message.

This new tool was introduced to students in the University of Liverpool Physics Department's first-year "Introduction to Computational Physics" undergraduate module. Through two surveys and a focus group we evaluated the effectiveness of the Error Explainer's automated feedback and the impact the tool has on reducing negative emotions brought about by error messages. We also analysed the types of errors encountered and the number of consecutive attempts required by the students to resolve the error.

2 Research Questions

Based on the observations above, our hypothesis is: a tool that provides automated feedback, enhancing Python error messages, will positively impact how students respond to error messages and aid in their learning.

This tool to enhance Python error messages, which is explained in detail in Sec. 4, was named the Error Explainer. The tool translates Python error message into plain English, referencing different parts of the Python error message.

To effectively evaluate the impact of the Error Explainer we first needed to understand two preparatory reseach questions:

RQ1 Do students properly utilise the provided error messages and are they aware of the information they contain?

RQ2 What is the distribution of error types encountered by these novice programmers?

To assess the perceived usefulness of the Error Explainer and its impact on students' emotions and confidence, we sought to answer the following primary research questions.

RQ3 Did the enhanced error messages provided by the Error Explainer tool help the students to solve errors and reduce the time spent doing so?

RQ4 Did the Error Explainer tool improve the students' confidence and reduce negative emotions surrounding Python errors?

3 Related Work

Given that programming error messages themselves show no signs of improving significantly, various approaches to ameliorate their inadequacies have been studied in the literature (e.g. Traver (2010) and B.A. Becker et al. (2019) for a summary).

One category of approach consists of trying to prevent errors occurring in the first place. Integrated development environments (IDEs) can be built to ease the process of writing code and examples exist for the majority of programming languages. Amongst these are several pedagogic IDEs that are aimed at novices, perhaps most notably BlueJ (Kölling, Quig, Patterson, & Rosenberg, 2003) for Java. Amongst other features, these often provide functionality such as templates for common coding constructs and code completion hints, which reduce the likelihood of syntactic errors. Such aids, however, risk users not actually learning the necessary syntax rules, thus causing issues further down the road.

Another method to reduce errors amongst students is to limit the language features available at any given time to a predefined subset, which increases as the student becomes more experienced. An example of this is the Java-based ProfessorJ (Gray & Flatt, 2003), with three increasing levels. By limiting the features available to those a student is more familiar with, the idea is that they are only likely to encounter error messages they

have more chance of understanding, but there is no guarantee this will actually be the case.

The other main approach is to provide automated code feedback, which can be further split into two main sub-areas. One focuses on parsing the user's code and comparing it to previous peer solutions to suggest fixes; the bank of prior solutions may either be crowd-sourced from the internet, such as in the HelpMeOut (Hartmann, MacDougall, Brandt, & Klemmer, 2010) system or, within the educational context, previous students' solutions to the task being performed (Marwan, Gao, Fisk, Price, & Barnes, 2020). Whilst rapid and scalable, this approach falters in its inability to provide individual feedback on where a student may have made a mistake and guide them towards a solution. It also has the potential disadvantage that students may blindly apply the solution, without learning the essential skill of how to resolve the issue using the information provided by the error message itself. An extension of this is Intelligent Tutor Systems (ITSs), which parse the student's code using machine learning to create personalised feedback (Mousavinasab et al., 2021). These systems have the advantage of providing immediate feedback, allowing students to edit their code and learn through doing (Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004; Paladines & Ramirez, 2020; Woolf, 2009), but they are typically time-intensive to develop and can be limited when applied to open-ended problems (Folsom-Kovarik, Schatz, & Nicholson, 2010).

The other, more common, method of providing feedback (B.A. Becker et al., 2019) is to enhance the error message to make it more understandable to inexperienced programmers. This may take the form of either replacing the original error message with a more educational one or keeping the original message and augmenting it with additional information to aid comprehension, the latter having the advantage that the students can connect the two pieces to gradually learn how to decode the native error message (Coull & Duncan, 2011). In both cases, the enhanced error messages often provide tips or suggestions for how to solve the generated error.

There has been a long history of error message enhancement, beginning with the Fortran language as far back as 1965 (Rosen, Spurgeon, & Donnelly, 1965). Since 2000, there has been rapid growth in this area, the landscape of which has recently been comprehensively studied and summarised by B.A. Becker et al. (2019). While many of the studies have focused on Java, leading to tools such as Gauntlet (Flowers, Carver, & Jackson, 2004), Snoopie (Coull, 2008) and Decaf (B. Becker et al., 2016; B.A. Becker, 2016b), various error enhancements have been developed for a wide variety of programming languages.

The results suggest that syntax errors make up a large fraction of novice programmer errors (Kohn, 2019). Consecutive repeated errors can be another useful data set, as Jadud (2006) found that consecutive repeated errors are often the "best indicator of how well (or poorly) a student is progressing".

Despite the myriad of tools designed to improve error messages, B.A. Becker (2016a) noted that, until recently, many of the studies "lack empiricism in determining if they make any difference, particularly to novices". Although the situation has improved in recent years, there is still debate surrounding the effectiveness of error enhancement (B.A. Becker et al., 2019), with some studies (e.g. B.A. Becker (2015)) finding a positive effect while others (e.g. the debated Denny, Luxton-Reilly, and Carpenter (2014)) suggesting they are "ineffectual". In addition, there is a lack of literature regarding the effect of the enhancement on the emotional state of the student.

There are several other important avenues of study that feed in to the development of error enhancement tools. One is whether programmers encountering errors actually read the error messages encountered and how their use of them affects the ability to resolve the underlying issue. A 2017 study by Barik et al. (2017) investigated this in the context of JAVA by tracking students eye movements when attempting to resolve commonly encountered issues. Their results showed not only that students read the error messages produced but that as much as 25% of their eye fixations were focused on the error messages, suggesting that reading error messages is "a cognitively demanding task". Comparing the probability of a correct solution with the number of times the student revisited the error message further showed that difficulty in reading the error messages is significantly anti-correlated with the ability to solve the problem at hand. This supports the potential of error enhancement to significantly improve learning outcomes.

Another is to better understand which issues students are particularly struggling with at a given time. This can be ascertained via specialised tools (e.g. Jadud (2006) for Java) that log students' keyboard inputs while coding, which can even prepare automatic summaries, along with associated recommendations, for students and teachers (C. Murphy, Kaiser, Loveland, & Hasan, 2009). The results of such studies provide important input into which sources of error are particularly problematic to resolve and whose error messages are thus primary candidates for enhancement.

Despite the rising popularity of the Python language, particularly in educational settings, relatively few error studies have focused on Python, with B. Becker et al. (2016) noting the need for more research as recently as 2016.

One of the earliest attempts was a tool called Cat-Soop in 2011 (Hartz, 2012), which automatically collected and assessed homework exercises, with an initial small-scale ($n = 25$) study in the context of MIT CS courses providing inconclusive results on its usefulness. While Cat-Soop contained a proof-of-concept add-on ("The Detective") to analyse errors and provide a simple plain-English explanation in addition to the original message, this was not rigorously tested.

In 2013, Guo (2013) developed an online Python tutor, whereby students could execute their Python program step-wise via a web browser while viewing the run-time state of the various structures, with anecdotal evidence that it helped "clarify concepts". More recently, Rivers and Koedinger (2017) developed a data-driven ITS for Python called ITAP (Intelligent Teaching Assistant for Programming), which automatically generates hints for individual students. While this showed impressive technical performance, improving over time as more data is added to provided more personalized suggestions, the effect of ITAP's feedback on students was not evaluated.

A study by Kohn (2019) investigated the cause of errors in programs collected from more than 4000 high-school students undertaking introductory Python courses. While the results showed that a large fraction of the errors are due to minor mistakes (e.g. misspellings), it was unable to reliably determine the nature of many of the errors due to not knowing the context and goals of the program.

Also in 2019, a plugin for the Sublime Text (Sublime HQ, 2021) IDE called Pycee (Python Compiler Error Enhancer) (Thiselton & Treude, 2019) was introduced, which uses data collected from the popular Stack Overflow question-and-answer webpage to augment error messages with additional information and suggestions for their resolution. A small-scale ($n = 16$) "think-aloud" study showed that the majority of students found Pycee helpful, particularly the code examples included and concrete solutions suggested.

Most recently, Zhou, Wang, and Qian (2021) investigated the most common Python errors encountered by middle-school students, developing an automated feedback tool called Mulberry to provide enhanced error messages for these. Statistical analysis, however, did not show any significant reduction in errors encountered nor improvement in students' ability to debug those errors.

Finally, during the course of our study we became aware of a new third-party Python module, Friendly (Roberge, 2021), that "replaces standard tracebacks by something easier to understand", including interactive buttons to query why an error occurred, what it means and where it happened, with augmentations available in English and French. While this seems useful for novice programmers, we are not aware of any studies investigating its effectiveness.

Much of the above research has, perhaps unsurprisingly, focused on the field of CS education (Traver, 2010). Even if they have no background in programming, CS students are likely to have more familiarity with computer systems and an interest in computing. In addition, learning to program will be a major focus of CS students' studies, with courses dedicated to specific aspects such as computer systems, data structures, algorithmic design etc. Hence, while such cohorts provide a large sample of introductory students encountering issues with error messages to study, they may not be representative of the increasing number of students learning computer programming in other disciplines. In particular, despite the rise in the use of programming languages such as Python for analysis within the wider STEM area, to the best of our knowledge there have been no dedicated studies on error message enhancement and logging within this domain. This work concentrates

on studying the effects on physics students and hence represents a first step in this direction.

4 The Error Explainer Tool

The Error Explainer tool was designed to be used with Jupyter notebooks, which are used in many STEM courses (J.W. Johnson, 2020). The aim of the tool is to assist the student in deciphering the Python error messages and give them agency to resolve the error, through translating the error message into 'plain English' (Cutts, 1996) and providing some common causes that can result in that error being produced.

By 'plain English', we mean writing that is easy to read. Whereever possible we avoided jargon, technical words, and nominalization. Where it was not possible to avoid technical words, we defined that word immediately and succinctly. For example, when describing error messages referring to a data structure, the words "data structure" were immediately followed in brackets by "(E.g. list/-tuple/string/array etc.)". Following the guidelines on plain English (Cutts, 1996), we used lists with bullet points, referred to the reader as "you", and aimed for short sentences averaging 10 words or less. We also used the same repeating format for all Error Explainer messages (see Fig. 1) to make it fast and easy to read. These design choices align with the comprehensive guidelines produced by B.A. Becker et al. (2019).

4.1 Components of Error Explainer output

Figure 1 shows an example of the Error Explainer output. Directly below the Python error (red shaded area), the additional Error Explainer feedback appears containing the following elements: 1) Title, 2) explanation of error in plain English, 3) "What to do" section, 4) "Common causes of this message" section, and 5) a final statement: "If you are still unsure, please ask one of the demonstrators". The contents of each of these sections is detailed in the following section, which describes the rationale behind what was included and how the information was presented.

4.2 Design Rationale

Several design choices were made when creating the Error Explainer. We will explain those choices in this subsection.

Firstly, the Error Explainer text appears in blue to distinguish it from the Python output or other Markdown text displayed in subsequent notebook cells. Like an error message, we wanted to ensure that the Error Explainer text stood out as different from the usual Jupyter notebook outputs.

Component 1: Title

Each Error Explainer output begins with the title 'PHYS105 help on "{ErrorType}" error message'. As this tool is intended for first-time learners of Python we wanted to ensure the students were aware that the blue Error Explainer text was not a standard Python output, and instead a tool introduced to them specifically for their PHYS105 class.

Component 2: Explanation

Below the title we included the explanation of the error in plain English. Table 1 contains the short explanations for various error types for which we provided explanations.

Component 3: What to do

As the aim of the Error Explainer is to help students become more confident interpreting and resolving Python error messages, instead of including the relevant line number in the Error Explainer output, we direct the students to look at the relevant part of the Python error message. In the "What to do" section the student is guided where to look in the error message, which usually involves looking where the caret (∧) symbol points to or the line number indicated in the error message. To make the connection clear, the colour of the words "line [number]" in the Error Explainer output matches the colour of the line number reference in the Python error message output.

Component 4: Common causes

Rather than telling the students how to fix the error, we provide a list of common causes of

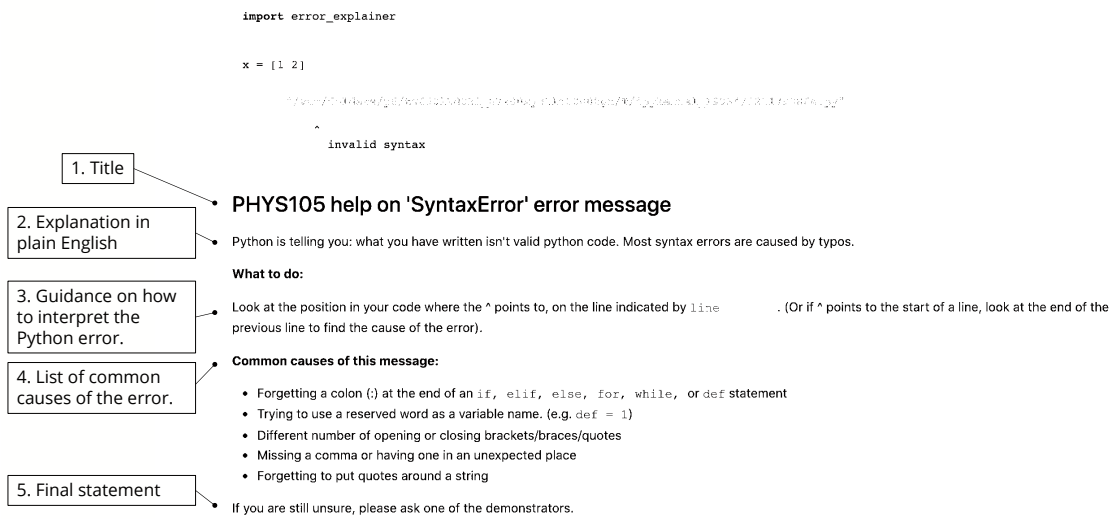


Fig. 1 Screenshot of a Jupyter notebook using the Error Explainer tool, with annotations identifying the key components. The Python error message (red shaded area) is immediately followed by the Error Explainer output (blue text).

that particular error type (see Fig. 1 for an example). Whilst we could have given students a troubleshooting checklist to work through, this approach encourages the students to consider common causes and decide whether any of these causes could be relevant to them, empowering the students to resolve the error themselves.

Component 5: Final statement

Finally, each Error Explainer output contains the same final line: "If you are still unsure, please ask one of the demonstrators", to express to the students that they can and should seek further help if needed.

It is worth noting that the Error Explainer abides by the majority of the requirements for effective support tools outlined in Coull and Duncan (2011):

- both the standard error and enhanced support appear concurrently,
- the tool embodies knowledge of key constructs needed to problem solve,
- the tool accommodates feedback for various error types and various causes leading to errors,
- use of the tool was voluntary on the part of the students, and
- the tool support does not promote dependence on the tool.

The only requirements that the Error Explainer does not meet was: linking to other teaching resources and disseminating knowledge in successive stages. Wanting to keep the Error Explainer output succinct, we opted to not link to other teaching resources and instead included the encouragement for students to seek help from demonstrators (who could point to other resources) if needed. Disseminating knowledge in stages could be considered for later releases of the Error Explainer, with expandable sections that offer further detail or extension.

4.3 Usage and Implementation

We have made the Error Explainer publicly available under a MIT licence for others to use or draw inspiration from. To use the Error Explainer simply download the Python file, error_explainer.py (available at https://github.com/carlgwilliam/error_explainer), and place it in

Table 1 Error Explainer short descriptions for various Python error types. These short description appear in location 2 in Fig. 1.

Python Error type	Error Explainer **short description**
AttributeError	Python is telling you: the variable or function you're asking for isn't provided by the object or module. E.g. trying to use .append() on a string, but strings don't support .append().
FileNotFoundError	Python is telling you: it can't open the file.
ImportError	Python is telling you: it can't find what you are trying to import.
IndentationError	Python is telling you: it has found too much or too little indentation (i.e. number of spaces or tabs).
IndexError	Python is telling you: you're trying to access an element in a data structure (e.g. list/tuple/string/array etc) that is bigger than the size of the structure. E.g. trying to access the 10th element of a list that is only 9 elements long.
KeyError	Python is telling you: you're trying to look up a key in a dictionary that doesn't exist.
ModuleNotFoundError	Python is telling you: it can't find the module you tried to import.
NameError	Python is telling you: the variable, function, or module you're trying to use can't be found. This is often due to a typo.
SyntaxError	Python is telling you: what you have written isn't valid Python code. Many syntax errors are caused by typos.
TypeError	Python is telling you: you're trying to use an operator on the wrong type of object, or you're trying to combine/compare variables that are of incompatible type.
ValueError	Python is telling you: the variable you're using has the correct type but the value isn't acceptable. E.g. trying sqrt(−1).
ZeroDivisionError	Python is telling you: you're trying to divide by zero.
when an unexpected exception was raised	We haven't implemented an explanation for this error message. Try: - Looking at the python error documentation - Googling the error (professional developers do this all the time!)

the same directory as the student's Jupyter notebook. Then activate the tool via the command: `import error_explainer` (see Fig. 1).

The Error Explainer tool catches when a Python exception (i.e. error) is raised using Jupyter's custom exception handler (`set_custom_exc`) and then, depending upon the

error type, will display explanatory HTML text directly below the Python error, as shown in Fig. 1.

For the purpose of this study, the possibility of logging the errors was incorporated into the Error Explainer, whereby the error type, related error message, and the date and time the error occurred were recorded.

5 Methods

5.1 Participants and Procedure

Since 2018, the University of Liverpool has embedded the widely-used Python programming language across their physics degree programme, with compulsory courses in the first two years of study followed by an optional third-year course on computer modelling. While this represents a significant investment in training physics students to program, it should be noted that these courses only represent a small fraction of each year's teaching (7.5/120 credits in the first year) with a lack of other courses directly supporting the learning outcomes.

The first-year "Introduction to Computational Physics" (PHYS105) course, led by one of the authors, aims to introduce physics majors to the Python programming language and its use in modelling physical systems. Since computing experience is not a prerequisite for entry onto a physics degree most of the students are novices, with 69% of students having no prior programming experience in any language and 27% having only a basic knowledge of simple computer programs. Given this, the first approximately half of the course focuses on introducing the basic Python building blocks needed for procedural programming (object-orientated programming is not covered until the second year), including the numerical Python package NumPy (Harris et al., 2020) and the Matplotlib (Hunter, 2007) visualisation package. In the second part the students practise what they have learnt, bringing together the various concepts to solve a series of physical problems both analytically and numerically. These include solving differential equations symbolically using SymPy (Meurer et al., 2017); numerical integration and differentiation, culminating in modelling the motion of a projectile taking into account air resistance; and fitting functional forms to laboratory data using the least-squares method via SciPy (Virtanen et al., 2020)).

The data utilised in the study presented were collected as part of PHYS105 course, consisting of 149 students, during the 2021–2022 academic year. The 12-week course ran over the first semester, with each week consisting of an online one-hour introductory lecture followed by in-person hands-on computing classes lasting two hours. For the latter the cohort was split into four similar-sized groups, each overseen by a member of academic staff with several postgraduate students on-hand to help. Each week the students were required to work through an interactive Jupyter notebook running Python 3.8.10[1], containing a series of formative and summative exercises, hosted via the CoCalc (Sagemath, Inc., 2020) collaborative online learning platform.

5.2 Data Collection and Analysis

The evaluation took place over a five-week period of the PHYS105 course covering weeks 4 to 8 of the semester, with data being collected from hands-on use of the Error Explainer tool by students during the computing classes in weeks 5 and 6. This period was specifically chosen to coincide with the students bringing together the basic Python building blocks learnt previously into more complex programs to solve physics problems for the first time. As such, they would be likely to experience a significant number of error messages, covering a wide variety of issues, during the completion of the two weekly assignments. While the students had encountered error messages in practice previously, this was prior to the formal explanation of the various Python error categories and how to understand the associated messages in the online lectures. The usage of the tool was entirely voluntary.

In order to address the research questions outlined above we utilised a mixed-methods approach (Creswell & Plano Clark, 2017; R. Johnson & Onwuegbuzie, 2004) to combine qualitative and quantitative data collected from a variety of sources to achieve a more global understanding. The data sources consisted of error logging, surveys and focus groups as described below. A timeline of

[1]This version of Python precedes several improvements to Python error messages undertaken in the 3.10 and 3.11 releases.

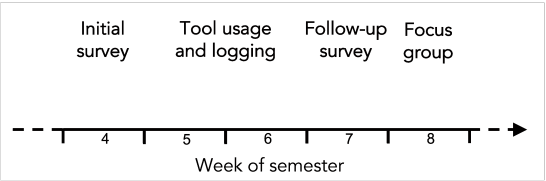


Fig. 2 Study timeline

these various components of the evaluation within the semester is presented in Figure 2.

5.2.1 Error logging

While using the Error Explainer tool, the students were asked to optionally consent to the tool anonymously logging information on the errors encountered (as described above) to see which errors are most commonly encountered and the distribution and frequency of reoccurring errors. Of the 149 students registered on the course, 105 agreed to their data being logged for at least one of the weeks.

We analysed the Repeated Error Density (RED) (B.A. Becker, 2016c) of the various error types encountered to identify the number of consecutive errors of the same type encountered by students within a 5 minute moving window.

5.2.2 Student Surveys

The students were invited to complete two anonymous surveys, one directly before (week 4) and one directly after (week 7) the two-week Error Explainer usage period. The survey questions consisted of a mixture of qualitative and quantitative questions designed to understand both their experience with Python errors and the helpfulness of the Error Explainer tool.

The initial survey mainly investigated the students' response to error messages, particularly in terms of emotions, how they approach solving them and how much time they spend on this. The follow-up survey concentrated on the student experience with the Error Explainer tool, including its usefulness and how understandable it was. To see if the tool had improved the students ability to resolve errors, in each survey the students were shown the same example error message and asked to comment on the cause and location of the error in the associated code.

Two main formats were used to collect the responses to the survey questions. Numerical data were collected on a sliding scale, chosen consistently to be $0 - 5$, except in the case of the usefulness question where the result could be either positive or negative and hence a scale of -3 to $+3$ was used. Non-numerical data were collected using multiple-choice questions in order to categorise the results and hence facilitate analysis. A limited number of free-text questions were included to draw out more details of the responses. The full set of survey questions, along with the associated answer format, is listed in Table A1 in the appendix.

Out of the 149 students enrolled on the course only a fraction engaged with the surveys, with the initial one being completed by 46 students (30%) and 21 students (14%) responding to the follow-up one. The resulting data were analysed graphically to visualise trends and numerical results were compared quantitatively using either a Wilcoxon signed-rank test (Wilcoxon, 1945) (within the same survey) or a Mann-Whitley U test (Mann & Whitney, 1947) (between surveys). In both cases, the SciPy (The SciPy Community, 2021) implementation of the tests was used for the calculations.

5.2.3 Demonstrator Focus Group

In the final week of the evaluation period (week 8) the demonstrators of the four computing classes, both the academic staff and the postgraduate helpers, were invited to a focus group. The goal of this was two-fold. Firstly, to understand how the demonstrators approach helping the students resolve errors and whether the Error Explainer tool aided this and was able to reduce the time spent on correcting (trivial) errors. Secondly, to obtain the demonstrators' view on how the students responded to both error messages and the Error Explainer tool, to compare to the students' own feedback.

One demonstrator from each computing class (three academics and one postgraduate helper) participated in the focus group, which lasted around 90 minutes and consisted primarily of open-ended questions to scaffold the discussion, allowing time for the participants to discuss around these topics.

A thematic analysis was performed on transcripts of the focus group recording and emerging themes across the focus group responses identified.

6 Results

In this section we report our findings, bringing together the various sources of data, in terms of each of the research questions outlined above. Since the focus of our study is on novice programmers, we exclude from both surveys the small number of students (5 and 6 students respectively) who identified themselves as good or excellent programmers, either in Python specifically or another language, prior to PHYS105. The resulting datasets consisted of 41 and 15 students for the two surveys respectively. Such distinction was not possible for the error logging data and hence logged data from all 105 students was included in the analysis in that case.

Table 2 Estimated proportion of workshop time spent attempting to resolve errors, before the Error Explainer was introduced.

Proportion of workshop time resolving errors	Number of responses
<10%	5 (12.2%)
10–25%	10 (24.4%)
25–50%	16 (39.0%)
50–75%	8 (19.5%)
75–90%	2 (4.8%)
>90%	0 (0.0%)

6.1 RQ1. Do students properly utilise the provided error messages and are they aware of the information they contain?

The experience of the demonstrators indicated that some students were unaware of the content of Python error messages, and unsure of how to read the error message. That is, the demonstrators believed that some students did not know which parts of the error message contained the important information to resolve their error.

To determine whether students were aware of the error message contents to be able to effectively utilise the error message, in the initial survey we asked the students to identify from a list, what information is contained in an error message. 91% of students selected that they were aware that error message includes the line number of the code that caused the error and 70% were aware that it includes the position of the problematic code within the line. 81% were aware that it includes the error type, while only 35% were aware that it includes "a summary of the steps in the code execution that led to the error location" (i.e. traceback).

We also investigated the time spent by students in the debugging process, since a lack of ability to efficiently decipher and resolve encountered errors would likely have an important baring on this. Based on the initial survey, the majority of PHYS105 students estimated they spent 25–50% of their class time on resolving errors, with a non-negligible fraction (4.8%) devoting in excess of 75% of their time to this (see Table 2). Consequently, if the clarity of error messages can be improved, such that students are able to resolve them more quickly, it can free up significant time to be allocated to other learning outcomes.

6.2 RQ2. What is the distribution of error types encountered by these novice programmers?

Alongside student approaches to error messages, we were interested in determining the most frequent error messages encountered and how many attempts it typically took for students to resolve the error with the Error Explainer tool in place. Table 3 shows the total number of occurrences of the most commonly encountered error types over a two week period. This data was collected through the error logging built into the Error Explainer tool for the 105 students that consented.

On average, students encountered 16.2 errors (with std. dev. = 13.4) per two-hour computer session. SyntaxError was the most common error, representing 29% of all errors, and 1.37 times more likely than the next most common error, TypeError.

The same data were analysed to extract the number of times a student generated the same error within a 5 minute moving window and the average number of repeated attempts required to resolve the error determined. Figure 3 summarises these results, with the error bars indicating one standard derivation. For the 8 most common error types, the average number of attempts to resolve the error was between 1 and 2.5 attempts. It is

Table 3 Most common Python errors types, including the most common error messages for SyntaxError and TypeError. This data analysed was the Python errors encountered by 105 students over a two week period, resulting in a total of 3219 errors logged.

Error type and message	Frequency
SyntaxError	**994 (29.33%)**
invalid syntax	635 (19.73%)
EOL while scanning string literal	105 (3.26%)
unexpected character after line continuation character	33 (1.03%)
cannot assign to function call	26 (0.81%)
f-string: empty expression not allowed	25 (0.78%)
unexpected EOF while parsing	24 (0.75%)
positional argument follows keyword argument	23 (0.71%)
invalid syntax (¡fstring¿, line 1)	14 (0.43%)
cannot assign to literal	11 (0.34%)
unmatched ')'	9 (0.28%)
TypeError	**722 (22.43%)**
unsupported operand type(s) for-: 'int' and 'str'	203 (6.31%)
unsupported operand type(s) for ** or pow(): 'str' and 'int'	32 (0.99%)
'str' object is not callable	31 (0.96%)
fit() missing 1 required positional argument: 'init_params'	25 (0.78%)
unsupported format string passed to list.__format__	20 (0.62%)
errorbar() missing 1 required positional argument: 'y'	16 (0.50%)
errorbar() missing 2 required positional arguments: 'x' and 'y'	16 (0.50%)
'int' object is not callable	15 (0.47%)
NameError	628 (19.51%)
ValueError	460 (14.29%)
IndentationError	176 (5.47%)
AttributeError	164 (5.09%)
ModuleNotFoundError	27 (0.84%)
IndexError	25 (0.78%)

interesting to note that no one error type took significantly more repeated attempts to resolve. The error type resolved with most ease was ModuleNotFoundError, where no students required more than one attempt to fix the error. The distributions for all other errors exhibited a long tail with some students requiring up to 16 attempts to resolve an error. Figure 4 shows the distribution of the number of attempts taken to resolve SyntaxErrors.

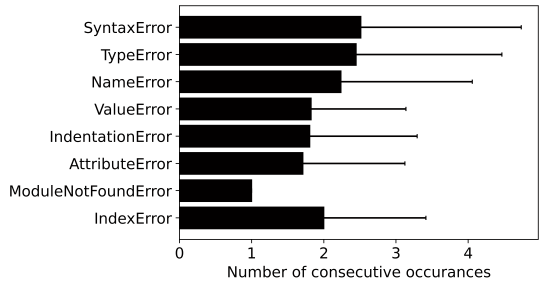


Fig. 3 The average number of repeated times a Python error message was generated within a 5 minute moving window, for the 8 most commonly encountered Python errors. Error bars indicate one standard deviation.

6.3 RQ3. Did the enhanced error messages provided by the Error Explainer tool help the students to solve errors and reduce the time spent doing so?

Prior to the introduction of the Error Explainer tool, the initial survey asked students if they felt a translation of Python error messages into plain

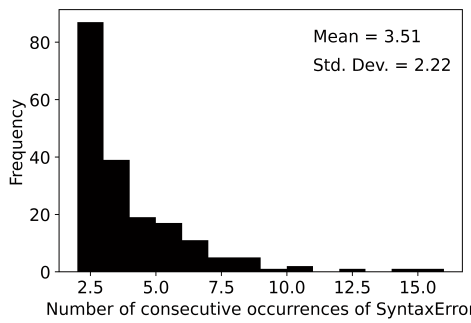
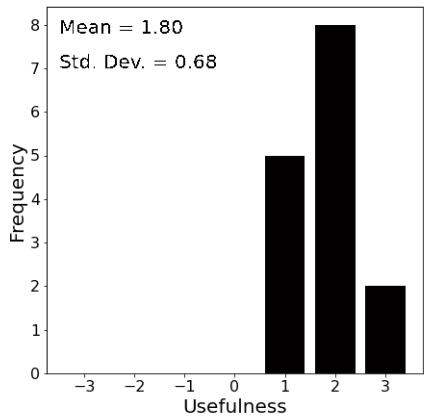


Fig. 4 Distribution of the number of attempts to resolve a SyntaxError, calculated by the number of times an error of the same error type was logged consecutively within a 5 minute moving window.



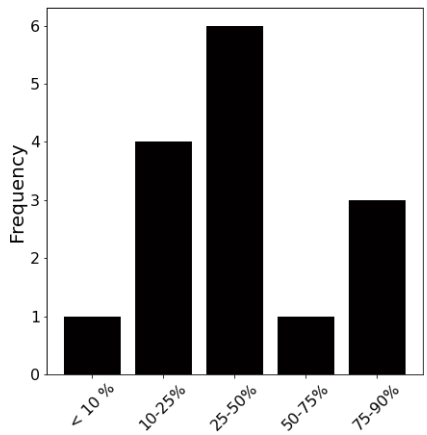


Fig. 5 Top: How useful the students found the tool on a scale from −3 (strongly hinders) to +3 (very helpful). Bottom: What fraction of the time the Error Explainer tool helped the students in resolving their error.

English, along with suggestions to resolve them, would help them resolve the errors encounter. The result was overwhelmingly positive, with 83% of the students answering "yes" and the remainder believing it might.

Following the evaluation period, the second survey sought to assess the usefulness of the Error Explainer tool in practice. The students were first asked to evaluate how useful the Error Explainer tool was in helping to understand the error messages and quantify how often it helped them to resolve the underlying error. The results are displayed in Figure 5, where it can be seen

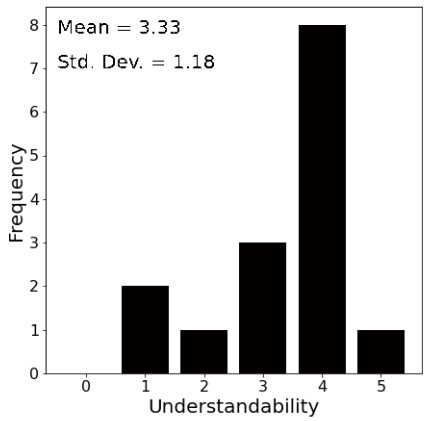


Fig. 6 How understandable the students found the tool's explanations on a scale from 0 (not easy to understand at all) to 5 (very easy to understand).

that students unanimously found the tool useful. Approximately two thirds of the students reported that the Error Explainer helped them resolve the error at least 25% of the time, with a quarter of students finding it helped them more than 50% of the time. Additionally, when asked their preferred initial port of call for help when coming across an error message they could not resolve, 73% of students opted for the Error Explainer tool. In particular, one student commented: "I [liked] having the Error Explainer appear as it allowed me to have a better go at solving the error myself without having to ask for extra help. It also allowed me to gain a better understanding of the error codes themselves".

In order to ascertain if the enhanced error messages provided were clear, we then asked students to rate how understandable they were. As can be seen in Figure 6, the majority of students found the explanations very easy to understand. This was reinforced by students' free-text comments on why they liked the tool.

While the students found the tool helpful in understanding and resolving errors, comparing the estimated time spent on resolving errors between the two surveys showed no significant change ($p = 0.679, Z = -0.41$) as determined using the two-sided Mann-Whitley U test. In addition, the test of the students' ability to identify the cause

and location of the error indicated by the example code and corresponding error message was inconclusive. Despite this, 86% of students surveyed reported that they would continue to use the Error Explainer tool, with the remainder considering doing so.

6.4 RQ4. Did the Error Explainer tool improve the students' confidence and reduce negative emotions surrounding Python errors?

We were particularly interested in the effect of the Error Explainer tool on the students' emotional response to encountering Python errors, since negative emotions are likely to reduce confidence and create a barrier to further learning. In the initial survey, students were asked to rank the various emotions they experienced when seeing Python error messages from 0 (not at all) to 5 (very intense). The emotions that ranked highest were frustration and confusion, with around 70% of students giving these a score of 3 or more. Over 40% of students also reported significant (2 or more) anxiety. This led to around 60% of students feeling significantly discouraged and many lacking motivation (mean = 1.8). This was corroborated by the focus group, where demonstrators reported that students appear to experience a range of emotions during the computing classes, with anxiety and frustration being the most obvious. They noted, in particular, that anxiety levels appeared higher in students who were working in isolation and not discussing with their peer group.

In order to asses the effect of the Error Explainer on these emotions, the follow-up survey asked the students to compare the level of these emotions while using the tool to that before. The data were quantitatively compared using the one-sided Wilcoxon signed-rank test to determine the probability (p) that negative emotions do not decrease or positive emotions do not increase. This is converted to a corresponding significance (Z) at which this hypothesis is ruled out. A selection of the results are presented in Figure 7. They show a significant reduction in both frustration ($p = 0.007, Z = -2.72$) and anxiety ($p = 0.008, Z = -2.64$), with the mean values reducing by $73 \pm 12\%$ and $55 \pm 18\%$, respectively. In particular, 80% of those students that had high anxiety (3 or more) initially, reported a decrease in the anxiety when using the Error Explainer tool. As a result students' discouragement decreased significantly ($p = 0.019, Z = -2.34$), perhaps leading to a hint of an increase in motivation ($p = 0.333, Z = -0.97$).

The overall effect was a increase in confidence reported by all students, as shown in Figure 8, with 73% of students ranking the improvement as 2 or more.

6.5 Insights from Demonstrator Focus Group

The primary theme that emerged from the demonstrator focus group was the role of feedback in the computer sessions; including the influence of the Error Explainer on how feedback was delivered and students' reception of the feedback. The demonstrators believed that the Error Explainer encouraged a focus on interpreting the Python error message output as opposed to immediately jumping to attempting to debug the cause of the error message.

The demonstrator focus group was asked about their usual approach to helping students understand the causes of error messages. All of the demonstrators said they typically articulate their thinking out loud as they try to determine the cause of the error, saying what they would look at and try, step-by-step. Essentially the demonstrators role-model problem solving and debugging techniques to the students.

When asked if the Error Explainer made discussing errors with students easier, more difficult, or no change, the group noted that the Error Explainer did serve as a reminder to explain what the Python error message means, rather than just jumping straight to an explanation of what caused the error. For example, rather than pointing out that a student forgot to add a colon (`:`) at the end of an `if` statement, the Error Explainer reminded the demonstrator to guide the student through interpreting the Python error message, highlighting the line number and the position the caret ($\wedge$) points to in the error message.

The focus group participants identified that students rarely ask "what does this error message mean?" and were more likely to ask "why isn't my code working?". This may reflect the students'

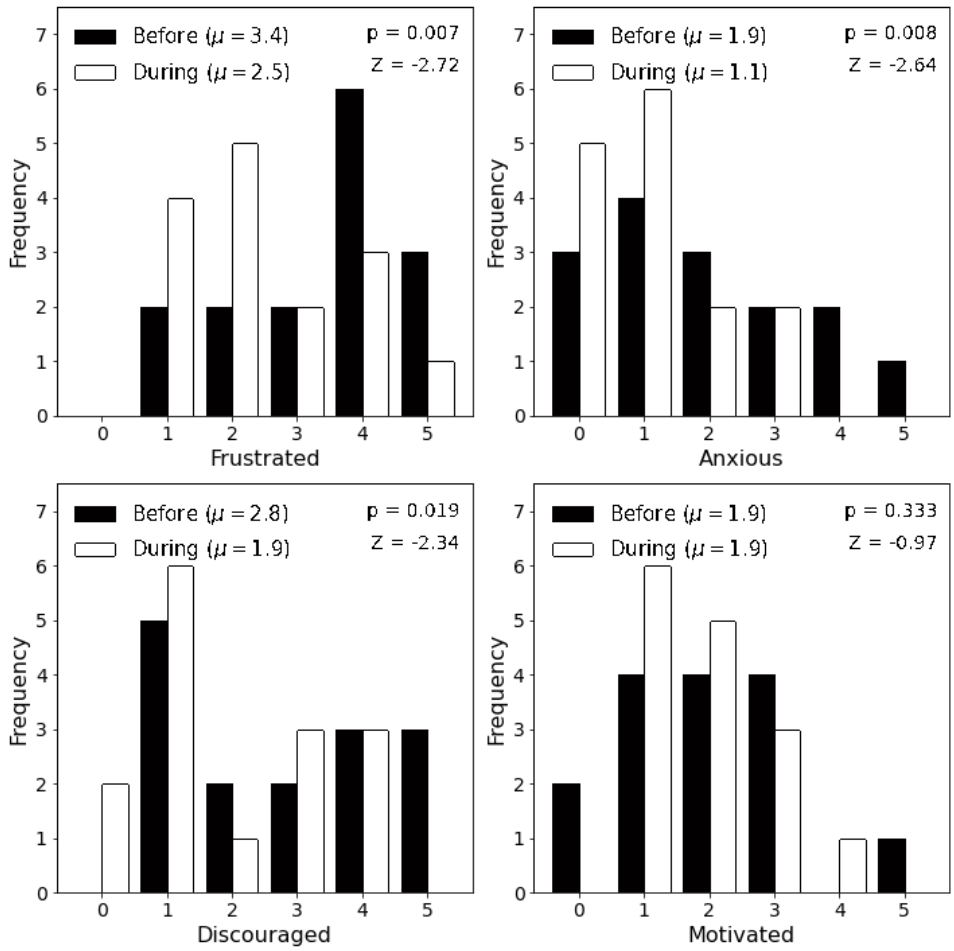


Fig. 7 The level which students experienced various emotions on encountering error messages before and during using the tool. The results are presented on a scale from 0 (not at all) to 5 (very intense). The mean value (μ) of both distributions is shown along with the probability and significance defined in the text.

focus on how to complete the current week's assessment rather than how to decipher error messages more generally.

Demonstrator interaction is one form of feedback, and the Error Explainer tool was designed to complement this with immediate, automated feedback. The second student survey included the question: "During all of your modules you will receive feedback in many different forms (e.g. talking with demonstrators, assignment feedback, etc.) Did you recognise the Error Explainer text as feedback?" The results showed that 53.3% responded that they did think of the Error Explainer output as feedback, 33.3% responded saying they hadn't thought of it as feedback but do now, and the remaining 13.3% did not see the Error Explainer output as feedback.

7 Discussion

In this section we begin by discussing the results of each of the research questions posed in Sec. 2 and the feedback from the demonstrator focus group. We then discuss potential limitations and threats to validity, before outlining possible future work.

7.1 Research Question Outcomes

RQ1. The results presented in Sec. 6.1 show that students utilise compiler error messages and even

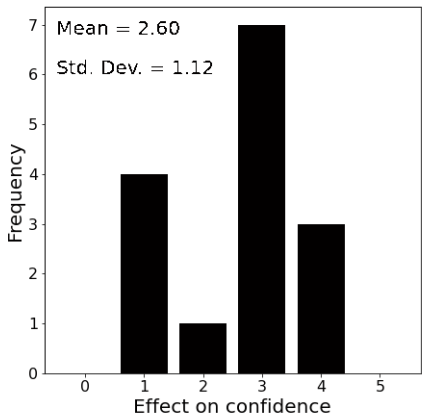


Fig. 8 The effect of the Error Explainer tool on student confidence on a scale from 0 (no influence) to 5 (a lot more confident).

before the introduction of the Error Explainer were aware of some of the information contained within Python error messages (e.g. line number, error type label, etc). Interestingly, this contrasts with the demonstrator focus group, who reported a strong impression that students often don't read the error message. It was acknowledged by the demonstrators that it is possible that the students who do read the error message ask for less help and a smaller number of students who don't read the error message can dominate the demonstrator's time. However, a more likely explanation is the students have looked at the error messages before (to know what it contains) but many were not yet equipped with the skills or vocabulary required to decipher and resolve the error message. It is in these cases that the Error Explainer tool may be particularly helpful.

RQ2. The finding that SyntaxError was the most commonly encountered error type is in agreement with other studies such as Zhou et al. (2021), Thiselton and Treude (2019), and Pritchard (2015). In fact the 5 most common errors identified were consistent across each of these studies, albeit with the ordering of places 2 through 5 varying slightly. Furthermore, Table 3 showed that the most common error message is the generic, "SyntaxError: invalid syntax", representing 19.7% of all error messages. A 2015 study by Pritchard (2015) which analysed 640,000 Python errors arising in code written by novice programmers submitted an online learning platform, "Computer Science Circles (CS Circles)", similarly found 28.1% of errors were caused by "SyntaxError: invalid syntax", followed by NameError which comprised 15.2% of errors. Although SyntaxError was the most common error type encountered, the fact that all error types took similar numbers of attempts to resolve suggests that all are worth of error enhancement. This data may be used to further improve the Error Explainer tool as outlined in Sec. 7.4.

RQ3. Similarly to many studies before this one (B.A. Becker et al., 2019), we were not able to show any quantitative improvement in students' ability to interpret error messages due to the Error Explainer. This maybe due to three reasons. Firstly, to mitigate the risk of survey fatigue, we limited the number of survey questions devoted to assessing the students' ability to decipher the error messages. Two questions were dedicated to this, one asking the students to identify the errors in a snippet of code and one asking about the time spent during computing classes trying to debug. The limited data did not indicate a significant improvement in either the students' ability to debug the code or show a reduction in time spent debugging. Secondly, the short time frame of 2 weeks over which the hands-on component of this study was conducted may have also reduced the measurable signal on student learning. More time spent using the Error Explainer may allow students the practice needed to consolidate their knowledge. Finally, comparing student performance before and after the introduction of the Error Explainer has the inherent challenge that the students were learning and practising more Python as the study progressed. Despite the lack of evidence that the Error Explainer tool helped the students better comprehend the Python error messages, the survey results clearly indicated that the students found the tool useful and that in most cases it helped them to resolve the error in question.

RQ4. The effect of emotions on learning to program is a relatively new area of research, with a recent synthesis of the available literature by Coto, Mora, Grass, and Murillo-Morera (2022) (published while the current work was ongoing) noting the sparsity of studies in this area. The results identify that the predominant emotions experienced by

students learning to program are overwhelmingly negative, with the most frequently experienced being frustration, followed by confusion and boredom. Our results from the initial survey agree with these findings. In this context, the reduction in frustration indicated by students in the follow-up survey is significant. In particular, it shows that the Error Explainer tool can help reduce prolonged frustration, which can avoid a loss of interest in learning (Leong, 2015). The corresponding reduction in discouragement reported by students supports this conclusion. Beyond this, since anxiety is known to inhibit students' ability to process information (Mandler & Sarason, 1952; Russell & Topham, 2012), the Error Explainer tool has the potential to improve learning outcomes although, as noted above, we were unable to demonstrate this concretely. Overall, as noted by Kinnunen and Simon (2012), introductory computing courses must "consider approaches that reverse the common overwhelming amount of negative feedback in the programming process (e.g. compiler or runtime errors)" and the Error Explainer tool can form part of an effective strategy in this direction.

Beyond the specific research question results, it is worth noting that in an informal discussion between one of the authors and a group of 8 students, the students commented that they initially found the Error Explainer very useful, however by the end of semester they no longer felt they needed it anymore. As described in the Methods section, the follow up survey was sent to the students after 2 weeks of using the Error Explainer, and at that stage the students all reported that the tool was useful. The student comments saying that they no longer needed the Error Explainer provide a reassuring indication that the tool, whilst helpful at first, does not promote dependence, which is a tenant of the requirements by Coull and Duncan (2011) for effective support tools. Instead the Error Explainer provides additional support where necessary but still forces the students to read the Python error message, not allowing to the students to rely solely on the Error Explainer. These findings suggest that the Error Explainer tool would be best introduced early in the corresponding course and could then be removed at a later stage. This could perhaps be done gradually by reducing the amount of explanation available as time progressed.

7.2 Demonstrator Feedback

Despite the rise in the use of programming languages such as Python for analysis within the wider STEM arena, the vast majority of the existing literature on error message enhancement has been applied to CS courses. The demonstrator focus group spoke of a mindset that may be unique to non-CS STEM students, namely that students often see the Python error messages as obstacles to their learning, rather than seeing learning to debug as a desirable skill in and of itself.

The focus group realisation that students rarely ask "what does this error message mean?" and more often ask "why isn't my code working?" reflect this mindset, suggesting that the students' focus is often on how to complete programming assignment, rather than how to decipher error messages more generally. This result may suggest that more emphasis needs to be placed on the idea that learning how to approach reading error messages and the problem-solving integral to debugging is an important skill worth developing. As a result, we suggest that deciphering error messages should be its own learning outcome of physics, and more generally STEM, programming modules.

Demonstrators also play an important role in encouraging a shift in students' mindset towards error messages. Instead of seeing error messages as an alarming warning that their work is wrong, they can express to the students that the error messages are there to help them solve the problem. Python is telling then what it knows and is giving them clues as to what to do next. By learning to interpret the error message (knowing where to look and what information is most pertinent) students are developing a skill required for all basic programming (not just resolving an error to completing a particular assignment.)

Finally, it was noted in the demonstrator focus group that sometimes the students were reluctant to try out different approaches, as they were worried they might break something (the server, their computer, Jupyter notebooks, etc), or that they would not be able to get back to a working version of their code. These thoughts and beliefs further exhibit diminished confidence, and could also lead to feelings of anxiety. However, such thoughts and

beliefs may be best addressed through reassurance from the demonstrators or lecturer (rather than a tool like the Error Explainer), using student focus groups to garner other common concerns. Indeed, some of the demonstrators mentioned that when helping students they often use the opportunity to normalise the student's experience (saying for example, "this is a very common problem..."). During these real-time exchanges, demonstrator comments could further alleviate some student anxiety.

7.3 Limitations and Threats to Validity

While no significant threats to validity were identified for the preparatory research questions (RQ1 and RQ2), there are several limitations related to the primary research questions (RQ3 and RQ4). As discussed in Sec 7.1, the main validity concern affecting RQ3 is the inherent improvement in the students coding ability as the course progresses, which cannot be easily untangled from the additional effect of introducing the Error Explainer tool. While the use of a control group could resolve this issue, it was not possible under the acquired ethical approval. This was compounded by the limited time during which the students used the Error Explainer tool and the few questions devoted to this in the surveys. A longer-term study, coupled with more in-depth surveys on this aspect would help to mitigate these concerns.

In terms of RQ4, one possible threat to the validity of could arise from the fact we asked students to self-report the intensity of various emotions, relying upon the students' recollection. Our results show that as the students used the tool their confidence grew. However, it is worth noting that the students' confidence may grow throughout semester naturally as they gain more experience. In an attempt to mitigate this concern and extract the impact of the Error Explainer on confidence, the students were asked explicitly "To what degree did the Error Explainer tool affect your confidence in your ability to resolve Python errors in the PHYS105 PC classes?" (see Fig. 8). Again, however, we relied upon students' recollection being accurate.

7.4 Further work

Future improvements to the Error Explainer could include different outputs not only for different error types but also for different error messages, to allow for more targeted feedback based on the context. The data gathered on which are the most common error messages (e.g. SyntaxErrors), can be used to guide those that would benefit most from having a more specific, targeted explanation. Further studies could also investigate the effectiveness of these more specific error messages by considering the Repeated Error Density, to inform which of the Error Explainer messages are more or least helpful to the students and why.

In addition to more targeted explanations, another feature that could be considered is to attempt to locate the position within the Jupyter notebook where an error message occurs in addition to the time at which it does so, in particular identifying if repeated error messages are generated from within in the same Jupyter notebook coding cell or a different one. This would allow us to more accurately determine which exercise of a problem set the student is working on and hence better investigate the rate of repeated errors within each exercise individually and correlate these with the exercise topic.

As alluded to earlier in the paper, a future version of the Error Explainer could include expandable sections that offer further detail or extension to allow knowledge to be disseminated in stages to accommodate diverse learners (Coull & Duncan, 2011). A reduction in the explanation provided as students become more comfortable with the native error messages could also be investigated.

Finally, the Error Explainer could potentially be extended to teach vocabulary. One reason novice programmers can struggle interpreting error messages is that they are still building up the technical vocabulary used in the error message. Making this more challenging is the finding by Marceau et al. (2011), that students pick up very little vocabulary through lectures. The Error Explainer could be extended to teach vocabulary in a just-in-time approach, allowing students to learn some of the technical terms as they need them and in context.

Future studies involving the Error Explainer or variations of the Error

Explainer could also be used to directly investigate the differences between Physics and CS student cohorts to allow direct comparisons between these two groups to be drawn.

8 Conclusion

Arguably the main stumbling block that novice programmers face is cryptic error messages that are inadequate to help them resolve issues encountered, leading to them feeling discouraged and ultimately presenting a significant hurdle to students learning to program. While several studies have explored enhancements to error messages, there is significant debate surrounding their effectiveness and little investigation into the effect on the student's emotional state. In particular, despite its rising popularity, there has been limited quantitative assessment of error enhancement in Python.

We developed a tool, called the Error Explainer, to provide enhanced Python error messages and subsequently studied its effect on first-year physics students during an introductory programming course using a mixed methods approach consisting of data from error logging, student surveys and a demonstrator focus group. To our knowledge this is the first such study specifically in the context of STEM curricula outside of computer science.

Analysing the most frequent error types showed a broadly similar distribution to previous studies, with the most common being SyntaxError, while no significant difference was observed in the average number of attempts students required to resolve errors of different types. While students reported that the tool helped them to resolve error messages in many cases, the data did not show any quantitative improvement in students' ability to understand and debug error messages. Although this is in line with several previous studies, the short time frame of the study make it difficult to draw any strong conclusions. In contrast, the data show a statistically significant reduction in negative emotions such as anxiety and frustration, along with a corresponding decrease in discouragement and increase in confidence. A future longer-term study is required to see if this translates into improved learning outcomes.

Acknowledgments

We wish to thank Dr. Eli Saetnan for her advice, including many helpful discussions, and reading and commenting on this manuscript. We also gratefully acknowledge the students and demonstrators who participated in the study.

Declarations

Funding

No funding was received for conducting this study.

Ethics approval

This study was granted ethical approval by the University of Liverpool research ethics committee (approval number 5402).

Conflicts of interest

The authors declare that they have no conflict of interest.

Informed Consent

Informed consent was obtained from all participants, both staff and students, for each data source analysed.

Availability of data and materials

The datasets generated and/or analysed during the current study are not publicly available due to the limitations of the current ethical approval but fully anonymised data are available from the corresponding author on reasonable request.

Code availability

The Error Explainer tool is publicly available under a MIT licence at https://github.com/carlgwilliam/error_explainer.

Appendix A Survey questions

Table A1 contains the questions, for both the surveys before and after the introduction of the Error Explainer tool.

Table A1 List of survey questions and the associated answer type.

	Question	Answer
Both	How would you describe your level of coding experience (in any programming language) prior to starting PHYS105?	choice
	How would you describe your level of Python coding experience prior to starting PHYS105?	choice
	Take a look at the following Python code snippet and associated error message and then answer the two questions below:	
	Which part of the code do you think requires correcting to resolve the error	choice
	What do you think is the cause of the error being reported?	choice
Before	Are you a native English speaker?	yes/no
	On a scale from 0 (not at all) to 5 (very intense), how strongly do you feel each of the following emotions when your code produces an error message?	scale (0–5)
	Please list any other terms that describe your feelings when encountering a Python error message?	free text
	What methods do you use to try to understand and correct the error? Tick all that apply.	multiple choice + free text
	When you come across an error message which of following information does it contain? Tick all that apply.	multiple choice + free text
	On average, what fraction of your time in PHYS105 PC classes is spent in trying to resolve Python errors?	choice
	Would a translation of the Python error message into plain English below the actual message, along with suggested tips to resolve it, help?	yes/no
After	Did you notice the Error Explainer tool appearing below Python error messages when using CoCalc in weeks 5 and 6?	yes/no/unsure
	How helpful/useful was the Error Explainer tool in helping understand the error (where -3 indicates a strong hindrance, 0 is neutral, and $+3$ is very helpful)?	scale (-3–3)
	How understandable were the Error Explainer descriptions of the errors (i.e. the blue text)?	scale (0–5)
	To what degree did the Error Explainer tool affect your confidence in your ability to resolve Python errors in the PHYS105 PC classes?	scale (0–5)
	When you come across a Python error message and you're not sure how to resolve it, what do you prefer as a first port of help?	choice
	During all of your modules you will receive feedback in many different forms (e.g. talking with demonstrators, assignment feedback, etc.) Did you recognise the Error Explainer text as feedback?	choice
	Did you like or not like having the Error Explainer (i.e. the blue text) appear? Please tell us why.	free text
	What percentage of the time did reading the Error Explainer message help you to solve the error?	choice
	Since using the Error Explainer tool, on average what fraction of your time in PHYS105 PC classes is spent in trying to resolve Python errors?	choice
	Reflecting back on PHYS105 weeks 1-4 (before we introduced the Error Explainer), on a scale from 0 (not at all) to 5 (very intense), how strongly did you feel each of the following emotions when your code produced an error message?	scale (0–5)
	When using the Error Explainer tool in weeks 5 and 6, on a scale from 0 (not at all) to 5 (very intense), how strongly do you feel each of the following emotions when your code produces an error message?	scale (0–5)
	Is there anything else that would have been helpful to include in the Error Explainer tool? E.g. other features	free text
	Would you like to continue to use the Error Explainer tool in the future?	yes/no/maybe

References

Barik, T., Ford, D., Murphy-Hill, E., Parnin, C. (2018). How should compilers explain problems to developers? *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (p. 633–643). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3236024.3236040

Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., Parnin, C. (2017). Do developers read compiler error messages? *2017 ieee/acm 39th international conference on software engineering (icse)* (p. 575-585). Retrieved from https://doi.org/10.1109/ICSE.2017.59

Becker, B., Glanville, G., Iwashima, R., Mcdonnell, C., Goslin, K., Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 1-28. Retrieved from http://doi.org/10.1080/08993408.2016.1225464

Becker, B.A. (2015). *An exploration of the effects of enhanced compiler error messages for computer programming novices* (Doctoral dissertation, Dublin Institute of Technology). Retrieved from http://doi.org/10.13140/RG.2.2.26637.13288

Becker, B.A. (2016a). An effective approach to enhancing compiler error messages. *Proceedings of the 47th acm technical symposium on computing science education* (p. 126–131). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2839509.2844584

Becker, B.A. (2016b). An effective approach to enhancing compiler error messages. *Proceedings of the 47th acm technical symposium on computing science education* (p. 126–131). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2839509.2844584

Becker, B.A. (2016c). A new metric to quantify repeated compiler errors for novice programmers. *Proceedings of the 2016 acm conference on innovation and technology in computer science education* (p. 296–301). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2899415.2899463

Becker, B.A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D.J., Harrington, B., ... Prather, J. (2019). Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the working group reports on innovation and technology in computer science education* (p. 177–210). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3344429.3372508

Ben-Eliyahu, A., & Linnenbrink-Garcia, L. (2013). Extending self-regulated learning to include self-regulated emotion strategies. *Motiv Emot*, *37*, 558–573. Retrieved from https://doi.org/10.1007/s11031-012-9332-3

Brown, P.J. (1983). Error messages: the neglected area of the man/machine interface. *Commun. ACM*, *26*, 246-249. Retrieved from https://doi.org/10.1145/2163.358083

Chow, S., Yacef, K., Koprinska, I., Curran, J. (2017). Automated Data-Driven Hints for Computer Programming Students. *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization* (pp. 5–10). ACM. Retrieved 2022-01-16, from http://doi.org/10.1145/3099023.3099065

Coto, M., Mora, S., Grass, B., Murillo-Morera, J. (2022). Emotions and programming learning: systematic mapping. *Computer Science Education*, *32*(1), 30-65. Retrieved from https://doi.org/10.1080/08993408.2021.1920816

Coull, N.J. (2008). *Snoopie : development of a learning support tool for novice programmers within a conceptual framework* (Doctoral dissertation, University of St Andrews, St Andrews, Scotland). Retrieved from https://research-repository.st-andrews.ac.uk/handle/10023/522

Coull, N.J., & Duncan, I.M.M. (2011). Emergent requirements for supporting introductory programming. *Innovation in Teaching and Learning in Information and Computer Sciences*, *10*(1), 78-85. Retrieved from https://doi.org/10.11120/ital.2011.10010078

Creswell, J.W., & Plano Clark, V.L. (2017). *Designing and conducting mixed methods research* (3rd ed.). Thousand Oaks, CA: SAGE Publications.

Cropper, C. (2018). Why should chemistry students code? : Are universities doing enough to build on a skill that is now taught at school and is needed for many areas of employment? *Education in Chemistry*, *55*. Retrieved from https://edu-rsc-org.liverpool.idm.oclc.org/opinion/why-should-chemistry-students-learn-to-code/3008177.article

Cutts, M. (1996). *The plain english guide*. Oxford University Press. Retrieved from https://books.google.co.uk/books?id=OWF5AAAAMAAJ

Daniels, L.M., Stupnisky, R.H., Pekrun, R., Haynes, T.L., Perry, R.P., Newall, N.E. (2009). A longitudinal analysis of achievement goals: From affective antecedents to emotional effects and achievement outcomes. *Journal of Educational Psychology*, *101*, 948 –963. Retrieved from https://doi.org/doi:10.1037/a0016096

Denny, P., Luxton-Reilly, A., Carpenter, D. (2014). Enhancing syntax error messages appears ineffectual. (p. 273–278). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2591708.2591748

Finch, D., Peacock, M., Lazdowski, D., Hwang, M. (2015). Managing emotions: A case study exploring the relationship between experiential learning, emotions, and student performance. *The International Journal of Management Education*, *13*(1), 23-36. Retrieved from https://doi.org/10.1016/j.ijme.2014.12.001

Flowers, T., Carver, C., Jackson, J. (2004). Empowering students and building confidence in novice programmers through gauntlet. *34th annual frontiers in education, 2004. fie 2004.* (pp. 433–436). Retrieved from http://doi.org/10.1109/FIE.2004.1408551

Folsom-Kovarik, J.T., Schatz, S., Nicholson, D. (2010). Plan Ahead: Pricing ITS Learner Models. *Proceedings of the 19th Conference on Behavior Representation in Modeling and Simulation* (p. 8).

Gray, K.E., & Flatt, M. (2003). Professorj: A gradual introduction to java through language levels. *Companion of the 18th annual acm sigplan conference on object-oriented programming, systems, languages, and applications* (p. 170–177). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/949344.949394

Guo, P.J. (2013). Online python tutor: Embeddable web-based program visualization for cs education. *Proceeding of the 44th acm technical symposium on computer science education* (p. 579–584). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2445196.2445368

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T.E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2

Hartmann, B., MacDougall, D., Brandt, J., Klemmer, S.R. (2010). What would other programmers do: Suggesting solutions to error messages. *Proceedings of the sigchi conference on human factors in computing systems* (p. 1019–1028). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1753326.1753478

Hartz, A. (2012). *Cat-soop : a tool for automatic collection and assessment of homework exercises* (Doctoral dissertation, Massachusetts Institute of Technology). Retrieved from https://dspace.mit.edu/handle/1721.1/77086

Hunter, J.D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. Retrieved from http://doi.org/10.1109/MCSE.2007.55

Jadud, M.C. (2006). Methods and tools for exploring novice compilation behaviour. *Proceedings of the second international workshop on computing education research* (p. 73–84). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1151588.1151600

Johnson, J.W. (2020). Benefits and pitfalls of jupyter notebooks in the classroom. *Proceedings of the 21st annual conference on information technology education* (p. 32–37). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3368308.3415397

Johnson, R., & Onwuegbuzie, A. (2004). Mixed methods research: A research paradigm whose time has come. *Educational researcher*, *33*, 14. Retrieved from http://doi.org/10.3102/0013189X033007014

Kinnunen, P., & Simon, B. (2012). My program is ok – am i? computing freshmen's experiences of doing programming assignments. *Computer Science Education*, *22*(1), 1-28. Retrieved from https://doi.org/10.1080/08993408.2012.655091

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., ... development team, J. (2016). Jupyter notebooks - a publishing format for reproducible computational workflows. F. Loizides & B. Scmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87–90). Netherlands: IOS Press. Retrieved from https://eprints.soton.ac.uk/403913/

Koedinger, K.R., Aleven, V., Heffernan, N., McLaren, B., Hockenberry, M. (2004). Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. J.C. Lester, R.M. Vicari, & F. Paraguaçu (Eds.), *Intelligent Tutoring Systems* (Vol. 3220, pp. 162–174). Springer Berlin Heidelberg. Retrieved 2022-01-27, from http://doi.org/10.1007/978-3-540-30139-4_16

Kohn, T. (2019). The Error Behind The Message: Finding the Cause of Error Messages in Python. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 524–530). ACM. Retrieved 2021-10-20, from https://dl.acm.org/doi/10.1145/3287324.3287381

Kölling, M., Quig, B., Patterson, A., Rosenberg, J. (2003). The bluej system and its pedagogy. *Computer Science Education*, *13*. Retrieved from http://doi.org/10.1076/csed.13.4.249.17496

Lahtinen, E., Ala-Mutka, K., Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *SIGCSE Bull.*, *37*(3), 14–18. Retrieved from http://doi.org/10.1145/1067445.1067453

Leong, F.H. (2015). Automatic detection of frustration of novice programmers from contextual and keystroke logs. *2015 10th international*

conference on computer science & education (iccse) (p. 373-377). Retrieved from https://doi.org/10.1109/ICCSE.2015.7250273

Mandler, G., & Sarason, S.B. (1952). A study of anxiety and learning. *The Journal of Abnormal and Social Psychology*, *47*(2), 166. Retrieved from https://doi.org/10.1037/h0062855

Mann, H.B., & Whitney, D.R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, *18*(1), 50 – 60. Retrieved from https://doi.org/10.1214/aoms/1177730491

Marceau, G., Fisler, K., Krishnamurthi, S. (2011). Mind your language: On novices' interactions with error messages. *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software - ONWARD '11* (p. 3). ACM Press. Retrieved 2021-10-21, from http://doi.org/10.1145/2048237.2048241

Marwan, S., Gao, G., Fisk, S., Price, T.W., Barnes, T. (2020). Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 194–203). ACM. Retrieved 2022-01-16, from https://dl.acm.org/doi/10.1145/3372782.3406264

Mega, C., Ronconi, L., de Beni. R. (2014). What makes a good student? how emotions, self-regulated learning, and motivation contribute to academic achievement. *Journal of Educational Psychology*, *101*(1), 121–131. Retrieved from https://doi.org/doi:10.1037/a0033546

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., ... Scopatz, A. (2017, January). Sympy: symbolic computing in python. *PeerJ Computer Science*, *3*, e103. Retrieved from https://doi.org/10.7717/peerj-cs.103

Mousavinasab, E., Zarifsanaiey, N., R. Niakan Kalhori, S., Rakhshan, M., Keikha, L., Ghazi Saeedi, M. (2021). Intelligent tutoring systems: A systematic review of characteristics, applications, and evaluation methods. *Interactive Learning Environments*, *29*(1), 142–163. Retrieved 2022-01-27, from http://doi.org/10.1080/10494820.2018.1558257

Murphy, C., Kaiser, G., Loveland, K., Hasan, S. (2009). Retina: Helping students and instructors based on observed programming activities. *Proceedings of the 40th acm technical symposium on computer science education* (p. 178–182). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1508865.1508929

Murphy, S., MacDonald, A., Wang, C., Danaia, L. (2019). Towards an understanding of stem engagement: a review of the literature on motivation and academic emotions. *Can. J. Sci. Math. Techn. Educ.*, *19*, 304–320. Retrieved from https://doi.org/10.1007/s42330-019-00054-w

Paladines, J., & Ramirez, J. (2020). A Systematic Literature Review of Intelligent Tutoring Systems With Dialogue in Natural Language. *IEEE Access*, *8*, 164246–164267. Retrieved 2022-01-27, from http://doi.org/10.1109/ACCESS.2020.3021383

Pekrun, R., Lichtenfeld, S., Marsh, K., H.W.and Murayama, Goetz, T. (2017). Achievement emotions and academic

performance: Longitudinal models of reciprocal effects. *Child Development*, *88*, 1653-1670. Retrieved from https://doi.org/10.1111/cdev.12704

Perkins, D., Hancock, C., Hobbs, R., Martin, F., Simmons, R. (1988). Conditions of learning in novice programmers. *Studying the Novice Programmer*.

Perkins, D., Hancock, C., Hobbs, R., Martin, F., Simmons, R. (1989). Conditions of learning in novice programming. *Studying the novice programmer* (p. 261—279). Lawrence Erlbaum Associates.

Pritchard, D. (2015). Frequency distribution of error messages. *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools* (pp. 1–8). ACM. Retrieved 2022-02-15, from http://doi.org/10.1145/2846680.2846681

Rivers, K., & Koedinger, K.R. (2017). Data-Driven Hint Generation in Vast Solution Spaces: A Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education*, *27*(1), 37–64. Retrieved 2022-01-27, from http://doi.org/10.1007/s40593-015-0070-z

Roberge, A. (2021). *Friendly 0.5.11.* Retrieved from https://pypi.org/project/friendly/ (Accessed: 31-01-2022)

Robins, A., Rountree, J., Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172. Retrieved from https://doi.org/10.1076/csed.13.2.137.14200

Rosen, S., Spurgeon, R.A., Donnelly, J.K. (1965). Pufft—the purdue university fast fortran translator. *Commun. ACM*, *8*(11), 661–666. Retrieved from https://doi.org/10.1145/365660.365671

Russell, G., & Topham, P. (2012). The impact of social anxiety on student learning and well-being in higher education. *Journal of Mental Health*, *21*(4), 375-385. Retrieved from https://doi.org/10.3109/09638237.2012.694505 (PMID: 22823093)

Sagemath, Inc. (2020). *Cocalc – collaborative calculation and data science.* Retrieved from https://cocalc.com (Accessed: 08-02-2022)

Simon, R.A., Aulls, M.W., Dedic, H., Hubbard, K., Hall, N. (2015). Exploring student persistence in stem programs: A motivational model. *Canadian Journal of Education/Revue canadienne de l'éducation*, *38*(1), 1–27. Retrieved from https://journals.sfu.ca/cje/index.php/cjerce/article/view/1729

Sinatra, G.M., Heddy, B.C., Lombardi, D. (2015). The challenges of defining and measuring student engagement in science. *Educational Psychologist*, *50*(1), 1-13. Retrieved from https://doi.org/10.1080/00461520.2014.1002924

Sublime HQ (2021). *Text editing, done right.* Retrieved from https://www.sublimetext.com (Accessed: 11-02-2022)

The SciPy Community (2021). *Statistical functions.* Retrieved from https://docs.scipy.org/doc/scipy/reference/stats.html (Accessed: 08-02-2022)

Thiselton, E., & Treude, C. (2019). Enhancing Python Compiler Error Messages via Stack Overflow. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1–12). IEEE. Retrieved 2022-01-25, from http://doi.org/10.1109/ESEM.2019.8870155

TIOBE (2021). *Tiobe index.* Retrieved from https://www.tiobe.com/tiobe-index/ (Accessed: 31-01-2022)

Traver, V.J. (2010). On compiler error messages: What they *say* and what they *mean*. *Adv. in Hum.-Comp. Int.*, *2010*. Retrieved from https://doi.org/10.1155/2010/602570

Villavicencio, F., & Bernardo, A. (2016). Beyond math anxiety: Positive emotions predict mathematics achievement, self-regulation, and self-efficacy. *Asia-Pacific Edu Res*, *25*, 415–422. Retrieved from https://doi.org/10.1007/s40299-015-0251-4

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., ... van Mulbregt, P.S. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. Retrieved from http://doi.org/10.1038/s41592-019-0686-2

Warr, P., & Downing, J. (2010). Learning strategies, learning anxiety and knowledge acquisition. *British Journal of Psychology*, *91*(3), 311–333. Retrieved from http://doi.org/10.1348/000712600161853

Weiss, C.J. (2017). Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *Journal of Chemical Education*, *94*(5), 592–597. Retrieved 2022-01-27, from http://doi.org/10.1021/acs.jchemed.7b00078

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80–83. Retrieved 2022-09-14, from https://doi.org/10.2307/3001968

Wilson, G. (2006). Software carpentry: getting scientists to write better code by making them more productive. *Comput. Sci. Eng.*, *8*, 768962. Retrieved from http://doi.org/10.1109/MCSE.2006.122

Woolf, B.P. (2009). *Building intelligent interactive tutors. student-centered strategies for revolutionizing e-learning.* Morgan Kaufmann Publishers/Elsevier.

Zhou, Z., Wang, S., Qian, Y. (2021). Learning From Errors: Exploring the Effectiveness of Enhanced Error Messages in Learning to Program. *Frontiers in Psychology*, *12*, 768962. Retrieved 2022-01-25, from http://doi.org/10.3389/fpsyg.2021.768962