UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DANIEL KELLING BRUM

PanScript – A free platform for teaching programming in many human languages

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Lucas Mello Schnorr

Porto Alegre
November 2021

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ANTLR	ANother Tool for Language Recognition
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AST	Abstract Syntax Tree
CJK	Chinese, Japanese, and Korean
CPU	Central Processing Unit
CS	Computer Science
CSS	Cascading Style Sheets
CT	Computational Thinking
ELF	Executable and Linking Format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IT	Information Technology
JS	JavaScript
JSON	JavaScript Object Notation
OOP	Object-oriented Programming
PE	Portable Executable
PEG	Parsing Expression Grammar
RTL	Right-to-left
SOV	Subject-object-verb
SVO	Subject-verb-object
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
XML	Extensible Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Learning to program can be a difficult task. It requires understanding new paradigms and abstractions, logic and computational thinking, and the programming language's syntax itself. That task becomes even more challenging when the student lacks basic English understanding, which is the case for millions of children in many developing countries. The main programming languages of today are all English-based: their keywords, the names of their functions, and most of the code available on the Internet are in English. For teachers, it can be hard to find adequate resources to teach basic programming concepts in the student's native language, which is something we believe could simplify the learning process. The purpose of PanScript is to help solve this problem by introducing an open-source platform to aggregate many text-based programming languages that are localized and simplified, along with a web-based beginner-friendly localized programming environment. The platform's design allows contributors to easily add new programming language dialects by translating tokens, error messages, and code samples. In this way, PanScript aims to help more people have their first contact with programming.

Keywords: Panscript. localized. programming. language. educational. novice. teaching. algorithm.

PanScript – Uma plataforma livre para ensinar programação em vários idiomas

ABSTRACT

Aprender a programar pode ser uma tarefa difícil. Requer o entendimento de novos paradigmas e abstrações, pensamento lógico e computacional, e a própria sintaxe da linguagem de programação. Esta tarefa se torna ainda mais desafiadora quando o estudante não possui entendimento básico da língua inglesa, que é o caso de milhões de crianças em vários países em desenvolvimento. As principais linguagens de programação de hoje são todas baseadas no inglês: suas palavras-chave, os nomes de suas funções, e a maior parte do código disponível na Internet estão em inglês. Para professores, pode ser difícil encontrar recursos adequados para ensinar conceitos básicos de programação no idioma nativo do estudante, algo que acreditamos poder simplificar o processo de aprendizagem. O propósito do PanScript é ajudar a resolver este problema introduzindo uma plataforma de código aberto para agregar várias linguagens de programação textuais, localizadas e simplificadas, bem como um ambiente de programação baseado em web, localizado e amigável. O desenho da plataforma permite que contribuintes adicionem novos dialetos da linguagem de programação com facilidade, traduzindo símbolos, mensagens de erro, e exemplos de código. Desta forma, PanScript visa ajudar mais pessoas a terem seu primeiro contato com programação.

Keywords: panscript, localizado, linguagem, programação, educativa, ensino, básico, algoritmo.

1 INTRODUCTION

English is currently the most taught foreign language in the world (CRYSTAL, 2003). Although estimates show more than 6000 human languages exist today (ANDERSON, 2010), the last century has seen English alone become a native or auxiliary language to more than one billion people (ETHNOLOGUE, 2019). Nowadays, students can benefit from the language's role as a global lingua franca for business, science, and technology (KIRKPATRICK, 2011). By learning a single additional language, a Brazilian student gains access to up to 50% of the knowledge and content on the Internet (SOLTANO, 2021). The language's ability to reach a much wider audience is precisely why this work is in English instead of Portuguese.

While some countries such as the Netherlands, Norway, Sweden, and Denmark may have more than 80% of the population understanding English (EUROBAROMETER, 2012), others such as India, Mexico, and Russia have closer to 11% (INDIA, 2011; IMCO, 2015; CENTER, 2014). In Brazil, Data Popular (2013) estimated that only 5% of the population has *some* knowledge of English, with almost half of those claiming to have just a basic level of understanding. Although these numbers seem higher among the younger generations, they are still close to only 10%. In other words, for some developing countries, even a basic level of English understanding may be hard to achieve, and expecting it from students in unrelated tasks could hinder their learning compared to students that can learn in their native languages (DASGUPTA; HILL, 2017).

Since the beginning of modern computer programming, developers write their code primarily in English. In the 1950s, the first high-level programming languages already used English keywords and function names. That was the case for languages such as Fortran, Lisp, ALGOL, and COBOL, to name a few (SEBESTA, 2012). For professional programmers, the benefits of learning English are undeniable. Most software documentation, programming examples, and instructional materials are in English (GUO, 2018). Official style guides advise programmers to use English when naming variables, functions, classes, and even when commenting code (ROSSUM; WARSAW; COGHLAN, 2001). The largest programming-related online forum, Stack Overflow, has an English-only policy, and although a few localized versions of the site *do* exist, they have fewer users and are thus less convenient (ATWOOD, 2009).

As for beginner programmers, supposedly a basic level of English understanding should be enough. Students only need to internalize a few keywords, such as `if`, `else`,

`print`, and `return`, as well as a few function names. Even so, Guo (2018) has found that close to one in every six non-native English students face problems learning to program while also learning English. Considering that 94% of the world does not have English as their first language (ETHNOLOGUE, 2019), the issue reveals itself more prevalent. For Anido (2015), perhaps the most significant barrier for Brazilian students is that the target demographics of existing tools have English as their first language. And while some of those tools have means of internationalization, most programming environments still require *some* knowledge of English. If done right, though, Guo (2018) also suggests that students might feel more motivated to improve their English skills due to a newfound interest in programming, in which case the barrier could become an opportunity.

Researching this topic, we have come across several projects relevant to the issue of teaching programming to non-English speakers. We have found projects aimed at localizing programming to a specific human language, such as Portugol Studio (ESTEVES et al., 2014), PSeInt (NOVARA, 2003), and Kati (كاتی) (KATI, c. 2019); projects aimed at supporting multiple human languages at once, such as Babylscript (IU, 2011a), Citrine (MOOIJ, 2014), and WLanguage (PC SOFT, 1992); and projects aimed at teaching young students how to program visually, such as Scratch (RESNICK et al., 2007) and Blockly (GOOGLE; MIT, 2012). We have looked at many such projects trying to find a localized solution that eases students into a mainstream programming language after learning the basics of programming in their native languages. While localized programming solutions could accomplish that goal, the existing ones are either not educational or are limited to a single human language, such that adding support to a different human language might not be an easy task, if at all possible. Moreover, having several localized projects without compatibility creates the risk of fragmenting student communities, while our objective is to integrate these students into the English-speaking community. *Block programming* environments could also achieve that objective, given that they can provide multiple localized textual representations for a single *code block*. However, we believe that transitioning from visual programming to a mainstream programming language feels jarring and that a simple localized text-based language could help bridge that gap (PERERA et al., 2021).

Given this scenario, this work proposes creating a free online platform to provide localized simplified programming languages, collectively known as **PanScript**. Teachers will have the option of using the platform in computing class laboratories, as it mainly targets students aged 11 to 16 in their first contacts with programming. To best accommodate such a target demographic, PanScript's design has objectives such as *localization*,

user-friendliness, *accessibility*, *code simplicity*, *frictionless UX*, *extensibility*, and *usefulness*. These are defined more precisely in Chapter 4. Following these objectives, we intend to provide teachers with a tool for teaching the basics of programming to young students anywhere in the world in their native language. After that, PanScript may be used in its canonical English version to teach English keywords before classes transition to a mainstream programming language such as C, Java, Python, or others.

Chapter 2 introduces basic concepts that are relevant for understanding implementation details. Chapter 3 explores a vast array of related work in localized and educational programming languages and platforms. Chapter 4 describes PanScript's design and implementation in detail. Chapter 5 specifies this work's evaluation methodology, which involves self-evaluation and a survey with a prototype of the platform. Chapter 6 discusses the results of both evaluations, and Chapter 7 presents conclusions and future work.

2 BASIC CONCEPTS

This chapter briefly introduces a few concepts of programming languages, software development, and code execution. These concepts enable a better understanding of the implementation details in Chapter 4.

2.1 Programming languages

A programming language is a notation for writing computer programs by permitting the specification of instructions, computations, or algorithms to control a computer's operation (SIGPLAN, 2003). It also assists the programmer in areas such as program design, documentation, and debugging (HOARE, 1973). Programming languages range from low-level (which are more similar to machine code, such as Assembly) to high-level (which enable higher levels of abstraction, such as C++ and Java) (SEBESTA, 2012). Low-level languages tend to be machine-oriented, while high-level languages are more user-oriented. We can classify programming languages using several criteria, including their supported programming paradigms and their typing systems.

Programming paradigms are classifications based on the main concepts and supported features of a programming language (WATT, 2004). The most well-known categories include imperative (containing both procedural and object-oriented paradigms), functional, and logic (SEBESTA, 2012). Nowadays, most mainstream languages are multi-paradigm, supporting a combination of constructs from functional, procedural, and object-oriented programming.

A type system is a set of types and associated rules governing their use in programs, as defined by a programming language (SEBESTA, 2012). They are a means to group values into known categories, allowing the programmer to express their intent more clearly, and allowing a type-checking routine to prevent the program from violating the language's semantic rules. An example of one such rule is disallowing multiplication between a number and a *boolean* (or logic) value (WATT, 2004). We often categorize type systems as either static or dynamic. In a statically typed language, all of the variables and expressions always have well-defined fixed types, such that programs can be fully type-checked during compile-time (WATT, 2004). Some languages support both static and dynamic typing as well. For example, the type *dynamic* in C# allows an object to bypass static type-checking (MICROSOFT, 2015).

2.2 Software development

Writing code typically enjoys the help of an Integrated Development Environment (IDE). The IDE is a program that provides functionality such as code editing, syntax highlighting, code completion, refactoring, and debugging, among others (WATT, 2004). Notable examples include Visual Studio (MICROSOFT, 1997), IntelliJ (JETBRAINS, 2001), Xcode (APPLE, 2003), and PyCharm (JETBRAINS, 2010).

Code reusability is the ability for programmers to reuse existing code when developing new applications, avoiding the need to recreate the same solution many times (SEBESTA, 2012). Programmers reuse code in several ways, such as using published libraries, packages, and modules; *forking* a repository to create a private version of an open-source solution; and sharing code snippets on online forums.

Web development is writing programs to run on the web, typically in a client-server architecture. Using a web browser, the user connects to a web server that hosts the web application, and the server returns files and data in response to interactions. Some applications provide an offline version of the website, such that basic functionality remains available even without Internet connectivity. A web application that never requires an Internet connection can be released as a purely client-side application.

Modern web applications contain several independent parts, known as components (REACT, 2017), packages, modules (NPM, 2019), or plug-ins (GULP.JS, 2014). These components are self-contained, reusable applications that provide specific functionality. Examples include a date picker shaped like a calendar or a theme-able code editor. Open-source developers create components such as these, which other developers then reuse in many different applications.

Web applications also typically make use of Application Programming Interfaces (APIs) when interacting with data. An API is an interface that defines the many operations a service provides, including which data formats to use when issuing a specific request (PEZOA et al., 2016). In web development, APIs commonly make meaningful use of HTTP verbs, such as GET and POST, in conjunction with either JSON or XML-encoded objects to send and receive data to and from a server.

2.3 Code execution

A program executes on Central Processing Units (CPUs), which require specific machine instructions instead of textual source code. There are many options for converting source code into machine code. The most common are compilation and interpretation (AHO; SETHI; ULLMAN, 1986); however transpilation is also relevant. *Compilation* is a process that transforms the source code into a binary object format, containing the low-level instructions for the target processor. Object files can be linked together to form an executable file, such as ELF (for Unix) or PE (for Windows). *Interpretation* is a process that parses source code and executes it in runtime, generally without the need for binary object files. In Just-In-Time (JIT) compilers, an intermediary bytecode representation may allow optimizations before code execution. When using interpretation without static-checking, programming mistakes such as invalid code can remain unnoticed until the affected code section runs. Finally, while compilation and interpretation are means to turn source code into machine code for execution, *transpilation* is a process that translates source code from one programming language to another (ANDRÉS; PÉREZ, 2017).

For all previous cases, concepts such as grammars, lexers, and parsers are helpful for language specification, lexical analysis, and syntactic analysis (AHO; SETHI; ULLMAN, 1986). A grammar describes all the terminal symbols (tokens) in a language and all possible combinations of such symbols in syntactically correct structures (programs). Several different grammar notations exist. They typically consist of a set of "head = body" production rules, in which "head" is a non-terminal symbol and "body" may contain one or more terminal and non-terminal symbols. Recursion occurs when the same non-terminal appears on both sides of a production rule, either directly or indirectly.

A lexer transforms the source code into a list of tokens based on the terminal symbols of the grammar. These tokens are matched against the grammar's productions using a parsing algorithm to generate a syntax tree, which in turn may be optimized and eventually translated to machine code or to another programming language (AHO; SETHI; ULLMAN, 1986). Although less common, some scannerless parsers also exist, in which both tokenization and parsing occur in the same step (VISSER, 1999). Manually developing parsers is complex and error-prone. A popular alternative is to use a tool that interprets the grammar definition and generates a lexer/parser program that transforms the input text into a syntax tree for further processing. Tools such as Flex/Bison and ANTLR are examples of parser generators (AHO; SETHI; ULLMAN, 1986).

3 RELATED WORK

While researching related work for this project, we have noticed that teaching programming to non-English speakers lies in the intersection between programming education and multilingualism. That is why we have divided the related work into two main groups: localized programming languages and educational programming platforms. This chapter defines these two concepts, explores existing solutions, and points out gaps that can prevent these solutions from teaching programming to non-English speakers.

3.1 Localized programming languages

We here define a "localized programming language" as a programming language, general-purpose or not, that uses localized keywords in their programs. Therefore, either the programming language uses non-English tokens, or it supports tokens in multiple human languages. We have found more than 90 examples of such programming languages on the Internet. While many more are bound to exist, we have selected 19 of them to describe in detail in this section, starting with Portugol.

Portugol consists of a simplified programming language using Portuguese keywords without accents (e.g., "senao" instead of "senão"). A few implementations exist, including VISUALG (SOUZA; NICOLODI, c. 2003) and Portugol Studio (ESTEVES et al., 2014). While VISUALG's design has remained the same for more than a decade, Portugol Studio still sees active development by its maintainers. Portugol Studio includes code samples and documentation written in Portuguese, as well as localized error messages. The language's syntax is similar to C, Java, and PHP (NOSCHANG et al., 2014).

Portugol Studio is one of the few localized programming projects with articles published about them. Noschang et al. (2014) describe the IDE and the motivations for teaching Brazilian students using Portugol. Junior and França (2017) compare several educational programming projects and state that, since Portugol Studio is a desktop application, this makes it harder for students to practice at home. Esteves et al. (2019) describe the advancements made after five years of the IDE's initial release, most notably mentioning Portugol Webstudio, which is a web-based version of Portugol Studio. Other projects with an academic background include PSeInt (NOVARA, 2003), Potigol (LUCENA et al., 2011), Babylscript (IU, 2011a), and CodeInternational (PIECH; ABU-EL-HAIJA, 2020a), which we describe in the following paragraphs.

PSeInt is perhaps one of the most successful localized programming projects to date. Initially created in Argentina in 2003, it was proven helpful in introductory programming classes in Panama (SÃNCHEZ; BAHAMONDEZ; CLUNIE, 2020), Peru (ARO, 2016), Mexico (CRUZ-BARRAGÁN; MARTÍN; LULE-PERALTA, 2019), and Cuba (CAÑETE; ENRIQUE; RICARDO, 2019). Cañete, Enrique and Ricardo (2019) have found that the usage of PSeInt may favor the development of algorithmic thinking when learning Linear Algebra. Aro (2016) has found a positive correlation between the use of PSeInt and the levels of cognitive learning on students of a "Fundamentals of Algorithms" course in Peru. Huerta and González-Bañales (2018) have observed a significant reduction in student failure rates on a "Fundamentals of Algorithms" course in Mexico. At the time of writing, PSeInt's development remains active 18 years after its initial release.

Other recent open-source projects for Portuguese and Spanish include Potigol (LUCENA et al., 2011) and Latino (MONTERO et al., 2015), respectively. According to Lucena and Lucena (2016), Potigol is a multi-paradigm programming language with syntax similar to Ruby and Python. Potigol is designed for beginners and encourages a functional programming style. Meanwhile, Latino is a procedural programming language with syntax inspired by Lua and Python. Latino too prioritizes educational purposes. Both projects were first released in 2015 and remain active to this day.

Babylscript is a modified JavaScript interpreter that supports 19 dialects. One curious feature of the project is the ability for programmers to switch between two or more languages in the same source code file, which is something we believe could create some confusion for non-polyglot developers. The project's repository was last updated in 2019 with minor translation adjustments. Most recently, the author has uploaded a YouTube video with motivations for using Babylscript (IU, 2020). In his article about the project, Iu (2011b) mentions that supporting the automated translation of code from one vocabulary to another is the holy grail of multilingual programming. CodeInternational accomplishes something similar to that objective.

According to Piech and Abu-El-Haija (2020b), CodeInternational is a tool to make programming more accessible to non-English speakers while still using typical introductory Computer Science programming languages, such as Python and Java. Their approach is to leverage the Google Translate API to automatically translate (and possibly transliterate) comments, string literals, class names, function names, and even variable names from one human language to another. However, the tool does not translate keywords because that would require a custom compiler or interpreter. The authors report that their work is

in use in at least four classes around the world.

Yet another effort for localized programming can be seen in both Excel (Microsoft, 1987) and WLanguage (PC SOFT, 1992). At least since Excel 97, the program includes localization for the function names used in formulas on the spreadsheet's cells (SOUTO; LIVI, 1999). Today, according to documentation for the official "Functions Translator" add-in, Excel supports more than 80 human languages (MICROSOFT, 2018), with at least 20 unique translations (where multiple function names differ from all the other translations). As for WLanguage, the creators of WINDEV have localized their programming language to support coding in French, English, and Chinese (PC SOFT, 2014). Users may automatically translate their code between localizations, and similar to Babylscript, multiple languages may appear in the same source code file.

Citrine (MOOIJ, 2014) is a project that goes even further in its localization efforts. Aiming to make the language more abstract by using pictographic symbols instead of just words, Citrine's syntax incorporates mandatory Unicode characters to represent actions such as "declare" (☞) and "write" (✎). The project includes automatic translations for more than a hundred human languages. However, the quality of these translations varies. For Brazilian Portuguese, the `list` type appears mistranslated as the verb `listar`, and the `stop` keyword appears localized as `punto`, which is an Italian word.

While Citrine uses Unicode symbols to make its syntax more abstract, Tampio (HAUHIO, 2017) goes in the opposite direction: it tries to resemble the Finnish language as much as possible, including several cases of word inflection and some forms of verb conjugation. Code written in Tampio looks like regular written text, except for the indentation. The project's repository was last updated in 2018, with almost all the commits written by a single person.

Other language-specific localized programming languages include Al-Khawarizm (الخوارزم) (AL-KHAWARIZM, 2018), Alif (ألف) (DRAGA, 2018), Ammoria (عمّوريا) (AMMOURI, 2006), Qalb (قلب) (NASSER, 2012), and Kati (کاتی) (KATI, c. 2019). These projects have in common the support for right-to-left (RTL) languages: Arabic and Persian. Not only does text flow from right to left, but the UI elements also have their positions shifted. For example, horizontal menu entries usually appear in the opposite order compared to their English counterparts. It is also noteworthy how Qalb explores the use of Arabic "kashida," which are extensions of arbitrary length between characters, as a creative way to align source code.

One finds an entirely different set of challenges in the CJK (Chinese, Japanese,

Korean) language family. Nadesiko (なでしこ) (Whale Flight Desk, 2004) and Produire (プロデル) (YUTO, 2007) are two examples for Japanese, and Wenyan (文言) (HUANG, 2019) is an esoteric example for Classical Chinese. These languages typically do not require spaces in their sentences. Programming languages attempting to support Japanese and Chinese must either require the usage of spaces or use a language-specific lexer to split the words, either using a dictionary or some heuristic based on auxiliary particles and inflections. Another notable characteristic of Japanese, Persian, and many other languages, is their subject-object-verb (SOV) sentence structure, different than English's subject-verb-object (SVO) structure. This difference may influence the programming language's design, as is the case for both Produire and Kati.

Finally, we could mention many other projects that are either abandoned or that are much more limited in scope and popularity. Wikipedia (2021b) lists more than 100 of the so-called "Non-English-based programming languages." Hopefully, this project will not end up as yet another unknown entry on the list.

3.2 Novice programming environments

We here define a "novice programming environment" as a platform where the primary goals are to teach beginners how to think computationally and how to create computer programs. Although some examples from the previous section also qualify as "novice programming environments," such as Portugol Studio, this section focuses on projects with greater notoriety, such as Scratch and Blockly, to further explore the existing literature about existing solutions. Section 3.3 later presents other classifications including purpose, platform, and paradigm.

Scratch (RESNICK et al., 2007) is a visual programming environment that allows users to learn computer programming in a self-directed way through exploration and collaboration (MALONEY et al., 2010). Rather than providing a text editor and some help articles, as some of the works in the previous section do, Scratch gives its users a collection of small code blocks that they can combine like LEGO bricks to create event-driven visual programs. These code blocks are then connected to form complex imperative logic without much risk of syntax errors (MALONEY et al., 2008). Scratch programs typically involve manipulating 2D object sprites on the screen to create a game, an animation, or an interactive story (ZAMIN et al., 2018).

A very distinguishing feature of Scratch is the social nature of the platform (MAL-

ONEY et al., 2010). The more than 73 million registered Scratch users have access to rich galleries of projects created by the community in many different countries (SCRATCH, 2021). Users may inspect these projects and "remix" them with ease, as human language is mostly not a barrier when reusing Scratch code. A program created in Scratch is purely a JSON file and a collection of assets (images, sounds, etc.) inside of a Zip archive. Since the code is not textual, code blocks may have their labels localized independently of the author's native language (DASGUPTA; HILL, 2017). Scratch does not automatically translate variable names, although it could, similar to what CodeInternational does.

The past decade has seen a vast number of studies on the usage of Scratch for teaching young students about programming and computational thinking. Researchers have verified that Scratch and similar solutions may help students recognize and assimilate computing-related concepts (SÁEZ-LÓPEZ; ROMÁN-GONZÁLEZ; VÁZQUEZ-CANO, 2016; AIVALOGLOU; HERMANS, 2016; ZHANG; NOURI, 2019). They may also make students more inclined to pursue Computer Science courses (OUAHBI et al., 2015; WEINTROP; WILENSKY, 2017; BALA; ALACAPINAR, 2021). More often than not, students report positive attitudes towards Scratch (ÖZORAN; CAGILTAY; TOPALLI, 2012; WILSON; MOFFAT, 2012; REZENDE; BISPO, 2018), even though a significant improvement is not noticeable in their final grades (ARMONI; MEERBAUM-SALANT; BEN-ARI, 2015; HERMANS; AIVALOGLOU, 2017).

Wing (2006) defines computational thinking (CT) as "using abstraction and decomposition when attacking a large complex task or designing large complex systems." According to Pérez-Marín et al. (2020), Scratch has a positive impact on both CT and general knowledge of computing. These are reasons why many authors have recommended the use of Scratch in the school environment for both middle school and high school students (SÁEZ-LÓPEZ; ROMÁN-GONZÁLEZ; VÁZQUEZ-CANO, 2016; OUAHBI et al., 2015; ZAMIN et al., 2018). Maloney et al. (2008) also noted that children learn Scratch even in the absence of experienced mentors.

One recurring concern is whether Scratch can teach good programming habits. Meerbaum-Salant, Armoni and Ben-Ari (2011) found that Scratch users picked up bad habits such as complete bottom-up development and excessively fine-grained programming. Aivaloglou and Hermans (2016) found procedures and conditional loops to be seen infrequently, while code duplication and dead code were much too common. Hermans and Aivaloglou (2017) mention the frequent use of infinite loops in the projects, indicating the concept is perhaps not fully understood. Armoni, Meerbaum-Salant and Ben-Ari

(2015) noted the same phenomenon, where one student stated during class that "loops are forever." To paraphrase Kalelioglu and Gulbahar (2014), just providing the learning environment will not fulfill the need for effective teaching.

Scratch is one of several *block programming* projects. Other examples include Blockly (GOOGLE; MIT, 2012), Alice (PAUSCH, 1998), Tynker (TYNKER, 2013), and App Inventor (ABELSON; FRIEDMAN, 2010), among others. According to Zamin et al. (2018), Blockly is Google's version of MIT's Scratch. It is mostly used as a tool to build other educational projects, such as App Inventor and even Scratch 3.0 itself. Alice is a 3D tool designed by Carnegie Mellon that teaches concepts of object-oriented programming (OOP). Alice does not provide a web version of its platform, which causes students to have some difficulties setting it up at home (DURAK, 2020). Tynker is a freemium platform offering online self-paced courses that teach block programming, Python, JavaScript, Java, HTML, and CSS. App Inventor is a visual platform for developing mobile apps, similar to Android Studio. App Inventor has a more complicated setup process that requires a mobile device for testing the app. Several other visual programming tools are available (PEREIRA; SEABRA; SOUZA, 2020).

Another common concern echoed by Fraser (2015) is encouraging students to move on from Scratch to learn fully-fledged programming languages such as Python, Java, and C#. Although Blockly presents JavaScript code alongside the Blockly editor, students may not pay much attention to that code (SERAJ et al., 2019). Judging by the findings in Hermans and Aivaloglou (2017), Weintrop and Wilensky (2017), and Rezende and Bispo (2018), we believe it could be beneficial to introduce visual programming tools along with simplified text-based programming. Such a process could give students something interesting to experiment with (ARMONI; MEERBAUM-SALANT; BEN-ARI, 2015) while at the same time demonstrating what else to expect from the field, bridging the gap between block programming and "real" programming languages (MARIMUTHU; GOVENDER, 2018; PERERA et al., 2021).

Finally, yet another recent solution in this category is Grasshopper (GOOGLE, 2019): a platform created for adults that teaches JavaScript using *gamification*. Like other visual programming solutions, Grasshopper also represents code internally as Abstract Syntax Trees (ASTs) rather than text. The difference is that, instead of using blocks like Scratch and Blockly, Grasshopper presents the code very similarly to formatted JavaScript code in a text-based code editor (MALYSHEVA, 2017). We believe that Grasshopper's code editing functionality would also work well on an app targeted at younger students.

3.3 Comparisons between related work and PanScript

This section compares the works we have discussed in this chapter. We classify the projects in terms of purpose, supported platforms, localization, paradigm, input mode, and approximate year of release. Table 3.1 presents an overview of these comparisons.

We can observe that most of the selected text-based projects have an educational purpose and a single localization option is provided. That is the case for VISUALG, Portugol Studio, Potigol, PSeInt, Latino, Al-Khawarizm, Alif, Ammoria, Kati, and Nadesiko. The exceptions are a few languages that aspire to be general-purpose (e.g., Babylscript, Citrine, and Produire), a few esoteric projects (Tampio, Qalb, and Wenyan), and a couple of business-oriented products (WLanguage and Excel Formulas).

Moreover, we notice almost half of the educational projects support a purely web-based environment: Portugol Studio, Kati, Nadesiko, Scratch, Blockly, Tynker, and Grasshopper. We did not count App Inventor because it requires a combination of web and mobile platforms, as the code runs on the developer's smartphone.

Considering paradigms, the most common are imperative and object-oriented programming (OOP). Potigol is the only educational initiative that supports functional programming constructs as well as OOP. Excel and Qalb are strictly functional; however, these are not educational projects. As usual, block programming initiatives rely on the event-driven paradigm. The exception is Tynker, which also offers programming courses in Python, JavaScript, Java, etc.

Considering the input mode, Grasshopper is the most unique project. Its puzzles provide a few tokens that the user can add to their answer. The source code appears onscreen as if it had been typed in a code editor. Later lessons then teach the student how to use an actual code editor to type their code instead of selecting tokens one by one.

Based on these criteria, PanScript is different in the sense that it offers text-based programming localized to many human languages with an educational purpose. At the moment its paradigm is merely imperative; however it could be expanded to support OOP in the future if educators believe it is necessary. For the time being, we have preferred to keep the language small and simple.

Table 3.1 – Comparisons between related work and PanScript.

Project	**Purpose**	**Platform**	**Localization**	**Paradigm**	**Input Mode**	**Year (approx.)**
VISUALG	Educational	Windows	Portuguese	Imperative	Textual	2003
Portugol Studio	Educational	Windows, Linux, Mac, Web	Portuguese	Imperative	Textual	2014
Potigol	Educational	Many (Java)	Portuguese	Multi	Textual	2011
PSeInt	Educational	Windows, Linux, Mac	Spanish	Imperative	Textual	2003
Latino	Educational	Windows, Linux, Mac	Spanish	Imperative	Textual	2015
Babylscript	General	Many (Java)	Many	OOP	Textual	2011
CodeInternational	Educational	Many (Python)	Many	N/A	N/A	2020
Excel Formulas	Business	Windows, Mac	Many	Functional	Textual	1987
WLanguage	Business	Windows, Linux, Mac	French, English, Chinese	OOP	Textual	1992
Citrine	General	Windows, Linux	Many	OOP	Textual	2014
Tampio	Esoteric	Many (Python)	Finnish	OOP	Textual	2017
Al-Khawarizm	Educational	Windows, Linux	Arabic	OOP	Many	2018
Alif	Educational	Windows, Linux, Mac	Arabic	OOP	Textual	2018
Ammoria	Educational	Windows	Arabic	Imperative	Textual	2007
Qalb	Esoteric	Web	Arabic	Functional	Textual	2012
Kati	Educational	Web	Persian	Imperative	Textual	2019

Continued on next page.

Continued from previous page.

Project	**Purpose**	**Platform**	**Localization**	**Paradigm**	**Input Mode**	**Year (approx.)**
Nadesiko	Educational	Windows, Web	Japanese	Imperative	Textual	2004
Produire	General	Windows	Japanese	OOP	Textual	2007
Wenyan	Esoteric	Web, Node.js	Classical Chinese	OOP	Textual	2019
Scratch	Educational	Windows, Mac, Web, Mobile	Many	Event-driven	Block	2007
Blockly	Educational	Web	Many	Event-driven	Block	2012
Alice	Educational	Windows, Linux, Mac	Many	Event-driven	Block	1998
Tynker	Educational	Web, Mobile	English	Many	Block	2013
App Inventor	Educational	Web + Mobile (together)	Many	Event-driven	Block	2010
Grasshopper	Educational	Web, Mobile	English, Spanish, Portuguese	Imperative	Hybrid	2019
PanScript	Educational	Web	Many	Imperative	Textual	2021

Source: The Author

4 THE PANSCRIPT PLATFORM

This chapter explores the solution design and implementation details of PanScript. Section 4.1 specifies the solution's target demographics, main objectives, architecture, programming environment, and the features of the simplified programming language. Section 4.2 elaborates on PanScript's dependencies, code structure, canonical grammar, and support for different human languages.

4.1 Solution design

This section introduces PanScript's target demographics, the project's main objectives, the platform itself, the canonical programming language, and the localized dialects.

4.1.1 Target demographics

PanScript focuses on three target demographics: non-English speakers aged 11 to 16 who are learning computer programming, teachers responsible for these students, and developers who wish to contribute to the project. We proceed to describe these demographics using three generic personas: student, teacher, and developer.

The student persona represents a student learning computer programming at middle school, high school, or summer camp. The student is not a native English speaker, but they may understand a few English words thanks to mobile phones, computers, and games. The student may have previous experience with Scratch or a similar tool, although they may not have much interest in programming yet. They wish to complete their assignments and find out if programming is an enjoyable hobby or career path.

The teacher persona represents a teacher who educates young students about basic computer programming and algorithms. They may not have a Computer Science background, and they may have learned to program later in their career. The teacher intends to prepare students for the digital era by providing them with the necessary skills in a low-risk environment. They have at least a working knowledge of English and can teach students how to write their programs in English PanScript. They also have at least basic knowledge of Python or another mainstream programming language and can teach students how to write programs in that language.

The developer persona represents an open-source developer who enjoys collaborating to free educational projects. They understand that PanScript does not aim to become a mainstream programming language, and they appreciate how the platform facilitates the creation of simple localized programming languages. If they are a native speaker of a language other than English, they can contribute a new PanScript dialect. If not, they can help improving interface design, user experience, accessibility features, etc.

4.1.2 Main objectives

As mentioned in Chapter 1, to best accommodate the needs of its target demographics, this project has a few key objectives: *localization*, *user-friendliness*, *accessibility*, *code simplicity*, *frictionless UX*, *extensibility*, *static checking*, and *usefulness*, as well as being *free* and *open-source*. We define these concepts more precisely below.

For PanScript, *localization* refers to adapting the programming language's keywords and function names to different languages, adjusting the user interface as a whole, and translating any error message produced during the transpilation or execution of the code. For example, when supporting Arabic, the user interface should switch to an entirely right-to-left (RTL) layout. *User-friendliness* means the solution should be intuitive and easy to use by teachers and students. *Accessibility* means it must support screen-readers for the visually impaired and keyboard-only input for people with motor disabilities. It should also provide font-size adjustment and editor theme selection, with at least one high contrast option. *Code simplicity* manifests itself as the reduced use of symbols, such that the code is easier to read, the same way Python is perceived to be easier to read than C++. It also implies the PanScript language itself should employ simple concepts to avoid being overwhelming. *Frictionless UX* means the platform should provide a smooth user experience (UX), avoiding requirements such as software installation or specific platforms or plug-ins. It should also support mobile devices such as smartphones and tablets. *Extensibility* refers to supporting more human languages in the future. It must be relatively easy to introduce new localized PanScript dialects, including ones that use right-to-left writing and Unicode characters. *Static checking* means the code must preferably fail in compile-time rather than runtime, making errors easier to spot. The language should use static type-checking to avoid bugs caused by implicit conversions. *Usefulness* means that students must interact with relevant programming concepts and keywords that they can use in mainstream programming languages. Our goal is for students to learn PanScript be-

fore the introduction of a general-purpose programming language. *Free* and *open-source* means the final solution should be readily available, free of charge, for anyone interested in using, learning, copying, sharing, modifying, or contributing to its expansion.

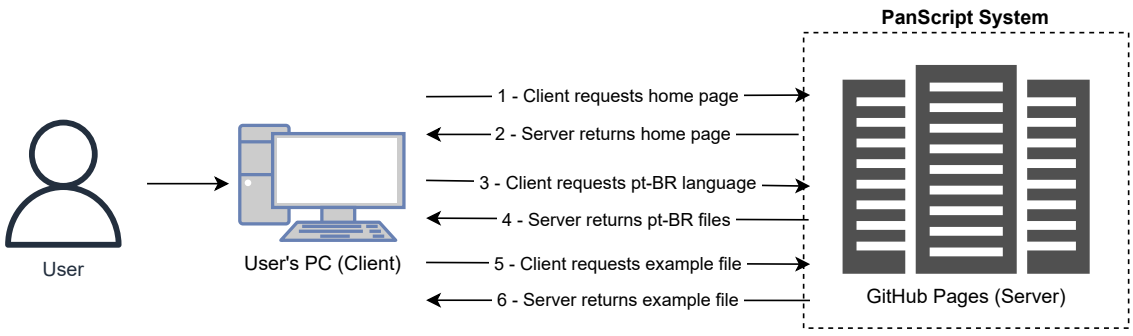
As we show in this chapter, these key objectives have influenced all major design decisions in the project.

4.1.3 Architecture and user interface

PanScript's web-based design supports any operating system that can run modern web browsers like Mozilla Firefox, Google Chrome, and Microsoft Edge. This design decision is in line with the objective of *frictionless UX*, since avoiding a complicated setup process allows students to more easily access the platform from their home computers or their smartphones.

Figure 4.1 presents PanScript's client-server architecture. New requests are made by the client every time the user selects a language they have not previously selected. To avoid web hosting costs, PanScript is currently a static web application hosted on GitHub Pages. As such, everything runs on the client-side: syntax highlighting, code validation, code transpilation, and code execution all occur in the user's web browser.

Figure 4.1 – PanScript's client-server architecture, also depicting requests made by the client.

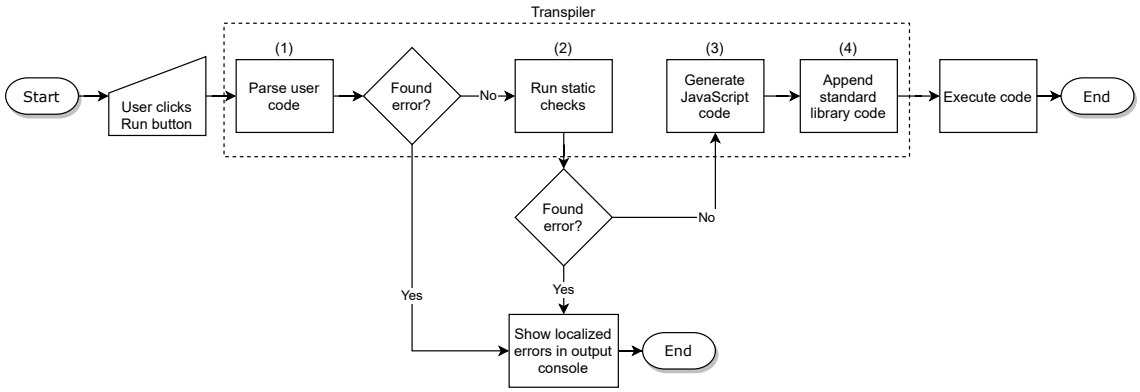


Source: The Author

Being a web-based application, PanScript contains several internal components. A *code editor* allows users to write their code with syntax highlighting directly in the browser. A *virtual console* displays any output or error message related to the transpilation or execution of the user's code. A *menu bar* allows users to select a different human language, save and load PanScript code files, and run the code in the editor. Users can also view code samples for the current language and choose a theme for the code editor.

Figure 4.2 presents an overview of what happens when the user clicks the Run button. Within the dotted box, a *transpiler* must (1) parse the localized PanScript code, (2) run static checks to verify the declaration of all identifiers and the adherence to all type rules, and (3) generate valid JavaScript code that the browser can execute, equivalent to the user's code. Finally, (4) the transpiler concatenates the standard library code and the generated JavaScript code.

Figure 4.2 – Flowchart of the code execution process.



Source: The Author

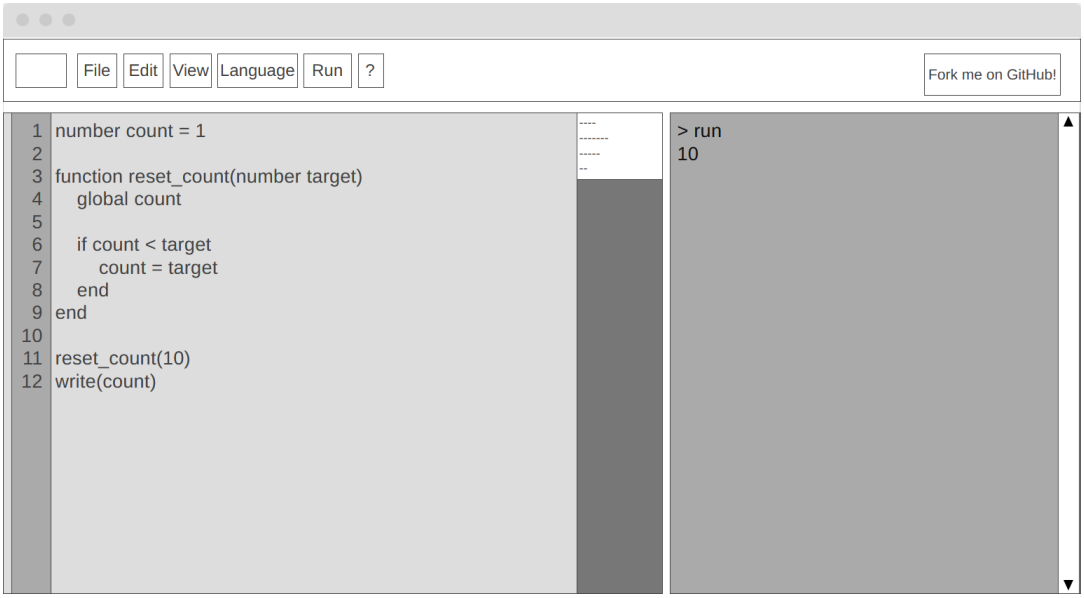
Most of the internal components are present in PanScript's user interface (UI). Similar to Scratch's design philosophy, PanScript's UI is a single page with all relevant controls visible at all times. Figure 4.3 presents an initial wireframe of the user interface. It includes the *code editor*, the *virtual console*, and several *menu* buttons for interacting with the platform.

An example of user interaction with the PanScript platform is as follows. The user navigates to the PanScript website using a web browser. They choose the appropriate localization language in the menu. They view the code samples for that language and use them as a reference to write their own code. They click a button to run the code and verify if the output console displays any compilation error. If there is an error, they modify and re-run their code. Finally, they check the output generated in the console.

4.1.4 Programming language and dialects

PanScript's language design has gone through several iterations since the project started, back in December 2019. Initially, its syntax would more closely resemble natural language, such as using the word `equals` instead of `==`. However, the principle of

Figure 4.3 – Early wireframe of PanScript's user interface.



Source: The Author

usefulness has helped steer the syntax towards inspirations such as Ruby, Python, PHP, and C. Objectives such as *code simplicity* and *static checking* have also informed other fundamental aspects of the design, such as the typing system. *Code simplicity* has led us to deliberately avoid using a large number of symbols such as semicolons (`;`), colons (`:`), and braces (`{}`) in code blocks. It has also induced us to prefer a smaller set of features and focus the standard library to a core minimum, as shown in Appendix B.

PanScript uses an imperative paradigm, mainly because command sequences tend to give a reasonably accurate picture of how computers work. Explaining algorithms as sets of instructions analogous to the steps of a recipe can make the concept more intuitive for first-time learners. Initial drafts included object-orientation, but we have dropped it to keep the language small and simple. As such, PanScript deliberately lacks support for more complex concepts such as objects, namespaces, and first-class functions.

Static checking has influenced us to adopt a strong, static, and explicit typing system. In PanScript, every variable declaration must have a type. The same is true for function declarations containing a non-empty `return` statement. PanScript currently provides the following primitive types: `text`, `number`, and `logical`. To avoid accidental null values, PanScript requires initialization for all variable declarations. At this moment, there is no distinction between integers and real numbers, although these subtypes are under consideration for future versions of the platform.

Similar to C, constants and variables in PanScript can be declared either globally or locally. To avoid name shadowing, PanScript requires the redeclaration of global variables using the `global` keyword before they are accessed, similar to Python and PHP. Names of identifiers may contain non-ASCII characters if the dialect allows them. Functions may have no type declared, in which case their internal type is `none`. Recursive functions are supported, but much like ANSI C, function declarations cannot appear nested inside another function's body. PanScript uses static scoping, as it is more common (SEBESTA, 2012) and more intuitive than dynamic scoping. PanScript does not include any language that requires right-to-left writing at this time, but that too will be supported.

PanScript represents its operators using symbols. The following operations are supported: assignment (`=`), addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), remainder (`%`), text concatenation (`+`), and composite assignment operators (e.g., `+=` and `-=`). We have chosen not to include bitwise operators (`>>`, `<<`, `|`, `&`, `^`), increment operators (`++x`, `x++`), and decrement operators (`--x`, `x--`). As for comparison operators, PanScript includes logical and (`&&`), or (`||`), and not (`!`) operators, equality (`==`), difference (`!=`), and the standard numeric comparison operators such as greater-than (`>`) and less-than (`<`). PanScript also supports PHP-like text interpolation, in which text literals may contain variable names or expressions surrounded by braces.

Contrary to Python, users cannot separate multiple statements in the same line of code using semicolons (`;`). Furthermore, indentation is not meaningful in PanScript. As in Ruby and Lua, users close blocks of code using the `end` keyword. PanScript does not contain context-specific versions of the `end` keyword (e.g., `endIf` or `endFor`). Supported control flow statements include `if/else`, `while` loops, `forever` loops (similar to Scratch), and `for-from-to` loops (similar to ALGOL). As for comments, PanScript supports C-style multi-line comments, and both C-style and Python-style inline comments.

The canonical language's standard library is a reduced version of JavaScript's standard library, with some modifications. The functions that manipulate the output console include `write`, `write_inline`, and `clear`. Text functions include `pad_left`, `length`, `repeat`, `left`, `right`, `reverse_text`, and `trim`. Numeric functions include `absolute`, `power`, `round`, `floor`, `ceiling`, `minimum`, and `maximum`. The complete list of standard library functions is much longer, and is available in Appendix B. We have also considered providing functions to manipulate date/time types or interact with the web through HTTP. However, we believe such features would rarely be useful and would only amount to unneeded complexity. The standard library is written in JavaScript, which is the same pro-

gramming language that the transpiler generates. During execution, the transpiler loads all function definitions together with the user's code, evaluating both simultaneously. PanScript dialects provide localized names for the standard library functions.

Having described the PanScript platform and canonical language, we can discuss PanScript's support for localized dialects. Figure 4.4 presents code samples for PanScript in English and Portuguese. The figure also showcases text interpolation, with expressions appearing inside text literals, surrounded by braces. The localized versions of PanScript allow students to develop computational thinking and learn general programming concepts in their native language. PanScript can also support languages that are not based on the canonical language, that is, languages that do not implement the same lexer, parser, and standard library. This way, the platform may host localized programming languages that include an entirely different set of features than PanScript, as long as they can be transpiled to JavaScript.

Figure 4.4 – Code samples for PanScript in English and Portuguese.

```
function say_hello(text title, text name)
    if length(title) > 0
        write("Hello, {title} {name}!")
    else
        write("Hello, {name}!")
    end
end

text title = "Mr."
text name = "George"
say_hello(title, name)
```

(a) English (US)

```
funcao diga_olá(texto pronome, texto nome)
    se comprimento(pronome) > 0
        escreva("Olá, {pronome} {nome}!")
    senao
        escreva("Olá, {nome}!")
    fim
fim

texto pronome = "Sr."
texto nome = "Jorge"
diga_olá(pronome, nome)
```

(b) Portuguese (Brazil)

Source: The Author

4.2 Implementation

This section details PanScript's current implementation. It describes the organization of the code, the technologies and dependencies in use, the support for multiple dialects, and the canonical grammar.

4.2.1 Technology stack

Following the project's objectives, PanScript's technology stack consists of web-based components developed with HTML, CSS, and JavaScript. Aiming to modernize the project as much as possible, we have adopted TypeScript (MICROSOFT, 2012) as the primary programming language and Node.js (DAHL, 2009) as the back-end runtime. TypeScript is a superset of JavaScript with support for static type-checking, which makes it ideal for avoiding bugs related to wrong variable types. For the front-end, we have chosen React.js (WALKE, 2013) due to its current popularity, hopefully facilitating adoption by open-source developers.

Within the Node.js and TypeScript realms, we have considered dozens of components for package management, code bundling, unit testing, code formatting, code linting, and build task orchestration. We have also tested several options for incorporating the parser, the code editor, and the output terminal as components readily available rather than developing them from scratch. The current version of the platform leverages technologies such as npm (SCHLUETER, 2010), webpack (KOPPERS; LARKIN et al., 2014), Testing Library (DODDS et al., 2018), Prettier (CHEDEAU; LONG et al., 2017), ESLint (ZAKAS, 2013), gulp.js (SCHOFFSTALL, 2013), antlr4ts (HARRIS; HARWELL, 2016), CodeMirror (HAVERBEKE, 2007), Xterm.js (KASIDIARIS; IMMS et al., 2016), and Ant Design (TEAM, 2016). The following paragraphs discuss why we have opted for some of these components instead of alternatives.

The first major technical decision was which programming language to use for the project. JavaScript and WebAssembly are the only programming languages that run natively on all major web browsers, with JavaScript currently being far more common. Alternatively, developers can write their code in TypeScript and have that code be transpiled to JavaScript during the build process. Since we believe static checking is beneficial for writing correct source code, we have chosen TypeScript instead of JavaScript.

The next decision concerns front-end libraries or frameworks. According to StackOverflow's Developer Survey (STACKOVERFLOW, 2020), the most popular front-end technologies for JavaScript as of 2020 are React.js, Angular (GOOGLE, 2016), Vue.js (YOU, 2014), and Gatsby (MATHEWS, 2015). Of these, React and Vue are the most "loved" and the most "wanted," according to the same survey. We have chosen React for this project because it has extensive documentation and many components readily available through the package manager npm.

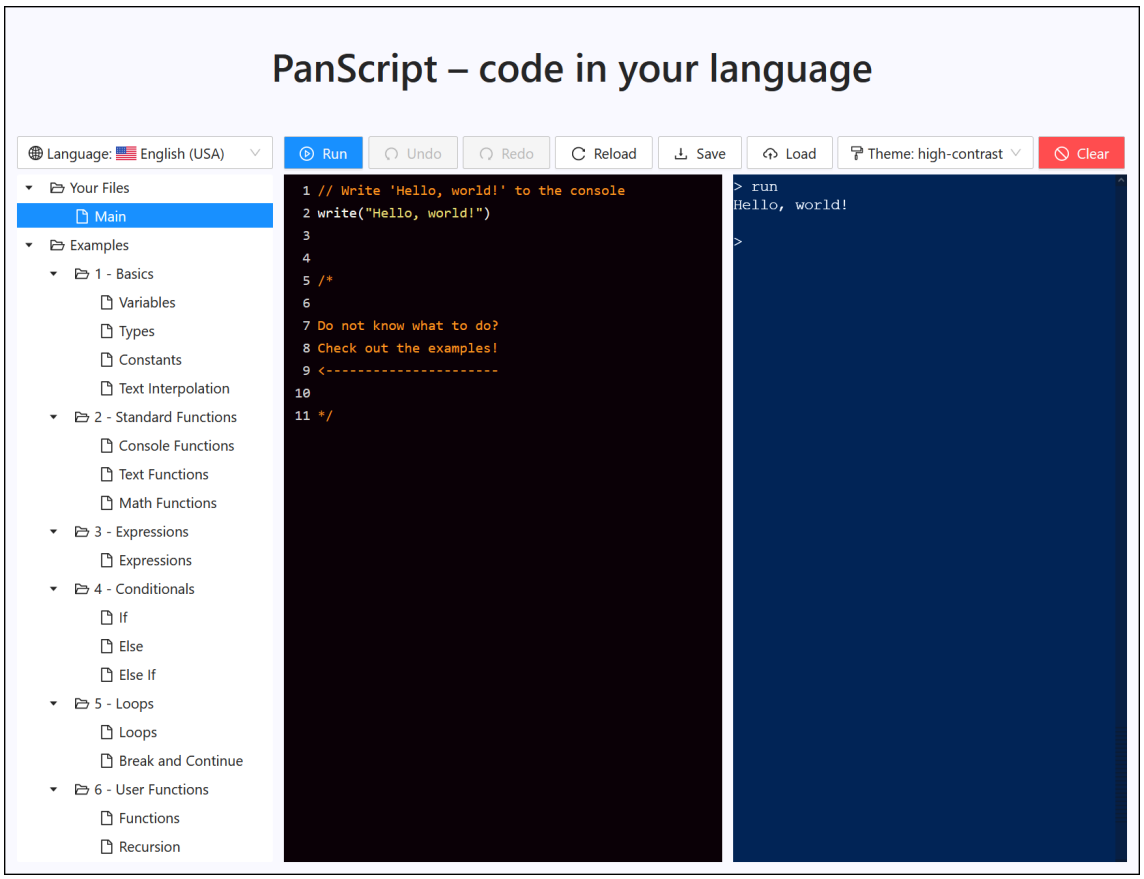
After settling on TypeScript and React.js, the next decision needed for a proof-of-concept was which parser library to use. Wikipedia (2021a) lists more than 150 parser generators, dozens of which generate JavaScript code, including ANTLR (PARR; HARWELL; FISHER, 1992), JavaCC (ORACLE, 2000), Canopy (COGLAN, 2010), Waxeye (HILL, 2008), PEG.js (MAJDA; RYUU, 2010), and nearley (CHANDRA, 2014). One feature we were actively seeking was the ability to replace the tokens in the lexer grammar while preserving both the parser grammar and the abstract syntax tree (AST) code. Because of this, we have opted for a parser generator that would allow separating grammar definitions from code. Furthermore, we wanted the grammar notation to be intuitive enough to be easily understood by new contributors. The result was the elimination of all PEG-based parsers since their grammars can quickly become polluted with whitespace tokens. The preference for native TypeScript support has pushed us towards ANTLR, for which a forked version, named antlr4ts, generates TypeScript code instead of JavaScript. ANTLR was also a good candidate due to its use of custom visitors to manually traverse the AST using object-oriented code. The custom visitor code is kept separate from both the lexer and parser grammar files, improving grammar readability. Additionally, ANTLR grammars support left recursion and use adaptive parsing strategies to choose the most appropriate derivation rule in runtime, improving readability even further (PARR; HARWELL; FISHER, 2014).

Finally, the last component choice worth mentioning was between the Monaco Editor (MICROSOFT, 2016), CodeMirror (HAVERBEKE, 2007), and Ace (AJAX.ORG, 2010). We have developed the proof-of-concept using Monaco – the same code editor used in Visual Studio Code and several other online platforms. One considerable limitation we have found was that Monaco does not yet support mobile devices. As such, students would be unable to use the platform on their smartphones. Due to the objective of *frictionless UX*, the choice was between CodeMirror and Ace. We have chosen CodeMirror because of its prevalence in several web-based platforms that contain code editors, such as phpMyAdmin (PHPMYADMIN, 1998), pgAdmin (PAGE et al., c. 2000), CodePen (VAZQUEZ; SABAT; COYIER, 2012), LeetCode (LEETCODE, 2011), and Jupyter (JUPYTER, 2015).

Figure 4.5 shows the latest version of the UI using these components. The menu buttons and the file explorer on the left are React components from Ant Design. The code editor at the center is a React component based on CodeMirror, and the output terminal on the right is a React component based on Xterm.js. During operation, the contents of

the files are periodically saved to the browser's local storage, meaning the user's code remains available even if they close the page and return a few days later.

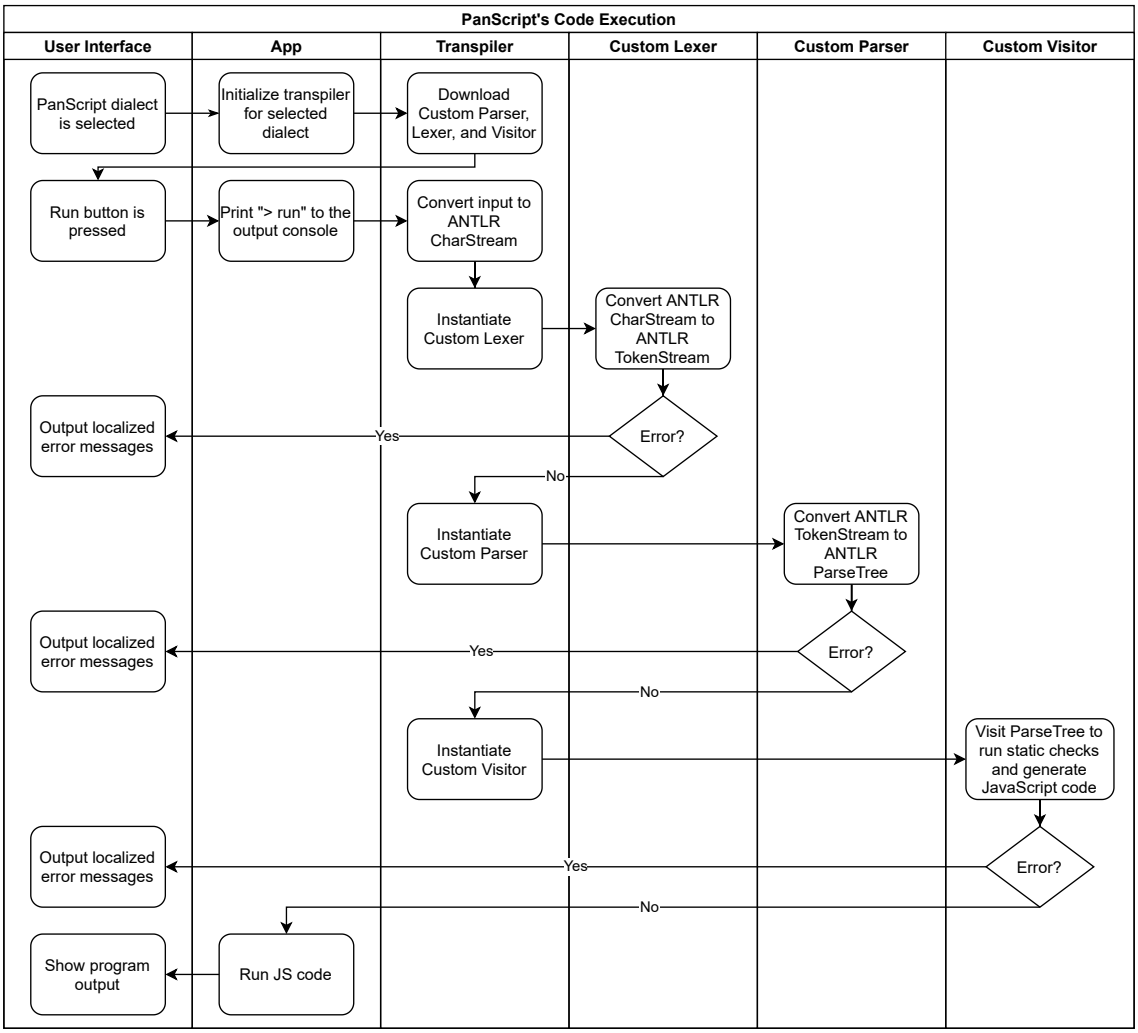
Figure 4.5 – PanScript's user interface



Source: The Author

Figure 4.6 presents a diagram illustrating what happens internally when the user presses the Run button. It is a more detailed version of Figure 4.2, showing the different parts of the transpiler in action, along with the localization artifacts. In runtime, the transpiler imports the Custom Lexer, Custom Parser, and Custom Visitor for the language that is currently selected. With them, it transforms the input code into a sequence of tokens. It then parses the tokens to generate an ANTLR parse tree. In a separate parsing step, the transpiler traverses the parse tree to generate JavaScript code while validating declarations and variable types. When it finishes, PanScript evaluates both the user's transpiled JavaScript code and the standard library code. The `> run` text appears in the output console, followed by any output generated by the user's program.

Figure 4.6 – PanScript's code execution with implementation details.



Source: The Author

4.2.2 Code structure and reusability

Figure 4.7 presents the file structure of PanScript's source code, with both a collapsed and an expanded view. The npm package manager maintains the `node_modules` directory. When the developer runs the `npm install` command, npm downloads all of the project's dependencies to `node_modules`. The `src` directory is where the project's source files live. The `src/components` directory contains all the TypeScript/React (.tsx) code files. The `src/languages` directory holds the `common` subdirectory and one additional subdirectory per PanScript dialect. The `src/languages/common` directory contains the standard definitions used in all PanScript languages, including the Common Lexer, Common Parser, and Common Visitor. Other language-specific directories contain spe-

cialized versions of these files, such as the Custom Lexer and Custom Parser. The files in `src/languages/en_us` mostly import and re-export the Common code files, while the files in `src/languages/pt_br` redefine the keywords and messages with ones in Brazilian Portuguese. The `src/static` directory contains static files such as themes for the editor.

Figure 4.7 – Directory and file structure of PanScript's source code.

Only directories

```
/
├─ node_modules/
│  └─ ...
└─ src/
   ├─ components/
   ├─ languages/
   │  ├─ common/
   │  ├─ en_us/
   │  │  └─ code_samples/
   │  ├─ fr_fr/
   │  │  └─ code_samples/
   │  └─ pt_br/
   │     └─ code_samples/
   ├─ static/
   │  └─ themes/
   └─ transpiler/
```

Directories and files

```
/
├─ node_modules/
│  └─ ...
├─ src/
│  ├─ components/
│  │  ├─ App.tsx
│  │  ├─ Console.tsx
│  │  ├─ Editor.tsx
│  │  └─ ...
│  ├─ languages/
│  │  ├─ common/
│  │  │  ├─ CommonLexer.g4
│  │  │  ├─ CommonParser.g4
│  │  │  ├─ CommonVisitor.ts
│  │  │  └─ ...
│  │  ├─ en_us/
│  │  │  ├─ code_samples/
│  │  │  │  ├─ variables.en_us.pan
│  │  │  │  └─ ...
│  │  │  ├─ CustomLexer.g4
│  │  │  ├─ CustomParser.g4
│  │  │  ├─ CustomVisitor.ts
│  │  │  └─ ...
│  │  ├─ pt_br/
│  │  │  ├─ code_samples/
│  │  │  │  ├─ variables.pt_br.pan
│  │  │  │  └─ ...
│  │  │  ├─ CustomLexer.g4
│  │  │  ├─ CustomParser.g4
│  │  │  ├─ CustomVisitor.ts
│  │  │  └─ ...
│  │  ├─ Transpiler.ts
│  │  └─ ...
│  ├─ static/
│  │  ├─ themes/
│  │  │  ├─ monokai.css
│  │  │  └─ ...
│  │  └─ index.html
│  ├─ index.tsx
│  └─ ...
├─ package.json
└─ ...
```

Source: The Author

The project includes two types of code reusability. External reusability is how we refer to the reuse of code created by other open-source developers that we have incorpo-

rated into the PanScript code base. It encompasses all of the packages mentioned earlier, as well as the dependencies of those packages. Because of the nature of dependencies having other dependencies, PanScript indirectly includes more than 900 npm packages. Internal reusability is the reuse of PanScript code to develop new PanScript dialects. PanScript dialects may inherit most of the grammar and code defined by the canonical language. Contributing new dialects typically requires nothing more than copying an existing localized version, such as pt_br, and defining new translations for keywords and messages in the `CustomLocalizedStrings.ts` file. The custom parser grammar can also redefine statements, allowing developers to reorder the tokens or use different ones altogether. In this way, PanScript could host programming languages that are much more complex than its canonical language.

4.2.3 Canonical grammar

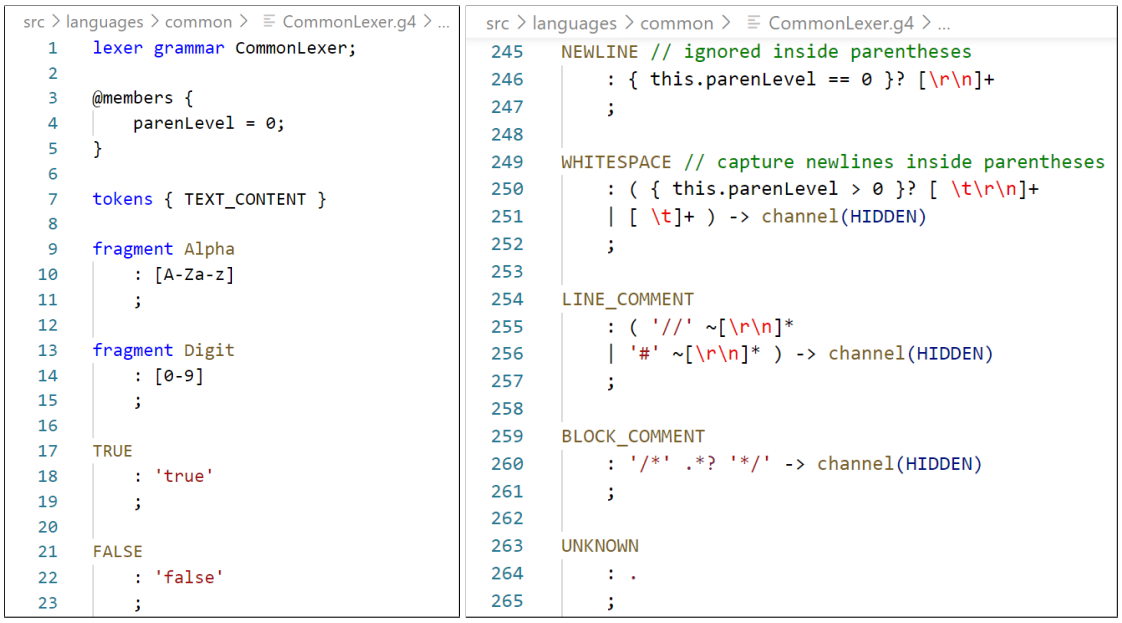
PanScript's canonical grammar consists of a lexer grammar and a parser grammar. Appendix C presents both of these grammars in full.

Figure 4.8 presents samples of the canonical lexer grammar. It recognizes the following reserved words: `and`, `break`, `constant`, `continue`, `else`, `end`, `false`, `for`, `forever`, `from`, `function`, `global`, `if`, `logical`, `not`, `number`, `or`, `return`, `returns`, `text`, `to`, `true`, and `while`. PanScript dialects usually localize these keywords into different words.

Figure 4.9 presents a localized lexer grammar in which all keywords are in Brazilian Portuguese. This localized lexer grammar recognizes the following reserved words: `e`, `interrompa`, `constante`, `continue`, `senao`, `fim`, `falso`, `para`, `para sempre`, `de`, `funcao`, `global`, `se`, `logico`, `nao`, `numero`, `ou`, `retorne`, `retorna`, `texto`, `ate`, `verdadeiro`, and `enquanto`. Initially we had used accented keywords such as `número`. However, user feedback was mostly negative in that regard, which is why accented characters are no longer included in keywords and standard library function names. They can still appear in user-created variable and function names, if the dialect allows it.

Figure 4.10 presents samples of the canonical parser grammar, which defines a `program` as a sequence of zero or more `topStatement`. A `topStatement` may be either a `functionDeclaration` or a `statement`. The `eos` shown in the figure is an end-of-statement, which may be either one or more newlines or an end-of-file marker. A `statement` may be a `variableDeclaration`, a `variableAssignment`, an `ifStatement`, a `forFromToStatement`, a `whileStatement`, a `foreverStatement` (for infinite loops), or a `functionCall`.

Figure 4.8 – PanScript's canonical lexer grammar.



src > languages > common > CommonLexer.g4 > ...

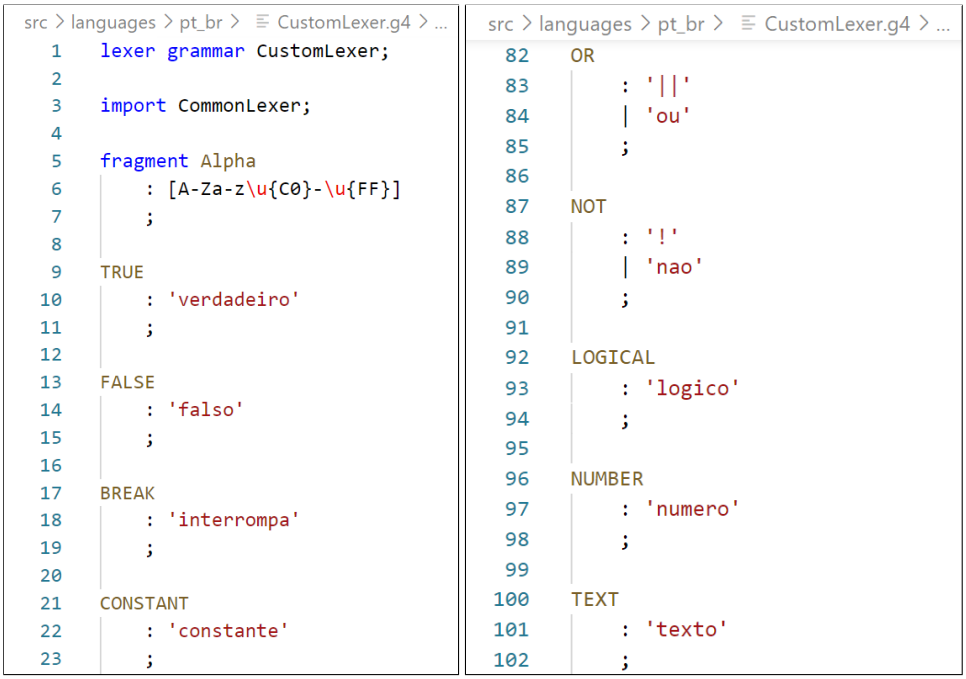
```
lexer grammar CommonLexer;

@members {
    parenLevel = 0;
}

tokens { TEXT_CONTENT }

fragment Alpha
    : [A-Za-z]
    ;

fragment Digit
    : [0-9]
    ;

TRUE
    : 'true'
    ;

FALSE
    : 'false'
    ;
```

src > languages > common > CommonLexer.g4 > ...

```
NEWLINE // ignored inside parentheses
    : { this.parenLevel == 0 }? [\r\n]+
    ;

WHITESPACE // capture newlines inside parentheses
    : ( { this.parenLevel > 0 }? [ \t\r\n]+
    | [ \t]+ ) -> channel(HIDDEN)
    ;

LINE_COMMENT
    : ( '//' ~[\r\n]*
    | '#' ~[\r\n]* ) -> channel(HIDDEN)
    ;

BLOCK_COMMENT
    : '/*' .*? '*/' -> channel(HIDDEN)
    ;

UNKNOWN
    : .
    ;
```

Source: The Author

Figure 4.9 – PanScript's lexer grammar for Brazilian Portuguese.



src > languages > pt_br > CustomLexer.g4 > ...

```
lexer grammar CustomLexer;

import CommonLexer;

fragment Alpha
    : [A-Za-z\u{C0}-\u{FF}]
    ;

TRUE
    : 'verdadeiro'
    ;

FALSE
    : 'falso'
    ;

BREAK
    : 'interrompa'
    ;

CONSTANT
    : 'constante'
    ;
```

src > languages > pt_br > CustomLexer.g4 > ...

```
OR
    : '||'
    | 'ou'
    ;

NOT
    : '!'
    | 'nao'
    ;

LOGICAL
    : 'logico'
    ;

NUMBER
    : 'numero'
    ;

TEXT
    : 'texto'
    ;
```

Source: The Author

Figure 4.10 – PanScript grammar with program and statements.

```
program
  : NEWLINE* topStatement*
  ;

topStatement
  : functionDeclaration eos
  | statement
  ;

statement
  : variableDeclaration eos
  | variableAssignment eos
  | ifStatement eos
  | forFromToStatement eos
  | whileStatement eos
  | foreverStatement eos
  | functionCall eos
  ;
```

Source: The Author

As shown in Figure 4.11, `functionDeclaration` consists of the keyword `function` followed be the function's name, an optional list of parameters surrounded by parentheses, an optional return type after the keyword `returns`, and a set of zero or more `innerStatement` before the keyword `end`. `innerStatement` may be a `statement`, as previously defined, a `globalStatement` (used to bring an existing global variable into the local scope), a `breakStatement`, a `continueStatement`, or a `returnStatement`. As previously mentioned, `type` may be `logical`, `number`, or `text`.

Finally, Figure 4.12 presents the `expression` and `atom` rules. An `expression` can be either a `parenthesisExpression`, used to manually specify the order of operations; a unary `plusExpression` (`+`), `minusExpression` (`-`), and `notExpression` (`!` or `not`); a right-associative binary `powerExpression` (`^`); the standard mathematical operations of multiplication (`*`), division (`/`), remainder (`%`), addition (`+`), and subtraction (`-`); the standard comparison operators less-than (`<`), less-than-or-equal-to (`<=`); greater-than (`>`), greater-than-or-equal-to (`>=`), equal-to (`==`), and different-than (`!=`); the logical `andExpression` (`&&` or `and`), and `orExpression` (`||` or `or`); and the `atomExpression`. An `atomExpression` is either `true`, `false`, a `number`, a `text`, a `functionCall`, or an identifier.

Figure 4.11 – PanScript grammar with function declaration.

```
functionDeclaration
  : FUNCTION IDENTIFIER OPEN_PARENTHESIS parameterList? CLOSE_PARENTHESIS
    (RETURNS type)? NEWLINE+ innerStatement* END
  ;

innerStatement
  : globalStatement eos
  | breakStatement eos
  | continueStatement eos
  | returnStatement eos
  | statement
  ;

globalStatement
  : GLOBAL IDENTIFIER
  ;

parameterList
  : type IDENTIFIER (COMMA type IDENTIFIER)*
  ;
```

Source: The Author

Figure 4.12 – PanScript grammar with expression and atom.

```
expression
  : OPEN_PARENTHESIS expression CLOSE_PARENTHESIS  #parenthesisExpression
  | ADD expression                                 #plusExpression
  | SUBTRACT expression                            #minusExpression
  | NOT expression                                 #notExpression
  | <assoc=right> expression POWER expression      #powerExpression
  | expression MULTIPLY expression                 #multiplyExpression
  | expression DIVIDE expression                   #divideExpression
  | expression REMAINDER expression                #remainderExpression
  | expression ADD expression                      #addExpression
  | expression SUBTRACT expression                 #subtractExpression
  | expression LESS expression                     #lessExpression
  | expression LESS_OR_EQUAL expression            #lessEqualExpression
  | expression GREATER expression                  #greaterExpression
  | expression GREATER_OR_EQUAL expression         #greaterEqualExpression
  | expression EQUAL expression                    #equalExpression
  | expression DIFFERENT expression                #differentExpression
  | expression AND expression                      #andExpression
  | expression OR expression                       #orExpression
  | atom                                           #atomExpression
  ;

atom
  : TRUE           #trueAtom
  | FALSE          #falseAtom
  | numberLiteral  #numberAtom
  | textLiteral    #textAtom
  | functionCall   #functionCallAtom
  | IDENTIFIER     #identifierAtom
  ;
```

Source: The Author

5 EVALUATION METHODOLOGY

This chapter specifies how we evaluate the solution's prototype. We have chosen to use a self-evaluation against the project's long-term goals, as well as an online survey. We use these two separate evaluations because, together, they provide both an inside and outside view of the project's accomplishments thus far.

5.1 Self-evaluation

The self-evaluation is a strict review of the PanScript prototype to identify whether it adheres to the long-term goals we have set out to accomplish. These goals are the same project objectives previously described in Chapter 4. From the beginning, we have expected that the prototype might not offer all the desired functionality of the complete solution. The self-evaluation is a means to highlight what this work effectively achieved and what may remain as future work.

Within the objective of *localization*, we expected the solution to provide at least two localized dialects other than English. All the text visible in the user interface should appear in the user's selected language. The error messages that the transpiler generates should be localized, including the names of the types in a type-mismatch error. When writing logical values to the console, the keywords `true` and `false` should be localized, too. If the prototype includes a language with right-to-left writing, the user interface should reorganize itself to show the file selector on the right and the output console on the left, and all components should have their text flow from right to left. Localizations should support accented characters and Unicode characters.

Within *accessibility*, we expected the interface to support keyboard-only navigation, making it clear whenever the user needs to use some key combination to deselect a given component (e.g., when deselecting the code editor). The solution should also support screen readers, although we admit code editors can have severely low usability for blind people even with screen reader support. At the very least, the solution should provide means to increase all font sizes and enable a high-contrast mode, such that it accommodates people with low vision.

For *code simplicity*, we expect the solution to require a small number of symbols, making its syntax resemble a simplified Python rather than C or Java. The language should not require a semicolon (`;`) at the end of every statement, and it should not require

a colon (`:`) at the start of code blocks. However, function arguments should appear surrounded by parentheses, and expression operators should have symbolic representations, such as `+` for addition and `||` for logical disjunction. We expect the language to provide a small set of functions with intuitive names in its standard library.

Regarding the objective of *frictionless UX*, PanScript should be a fast website that works in all major web browsers (Mozilla Firefox, Google Chrome, Microsoft Edge, and Safari) in both desktop and mobile platforms. The user should not have to install browser plug-ins or other apps to use the solution.

In *extensibility*, PanScript should allow the inclusion of new localizations with relative ease. Contributors should not need to know how to write ANTLR grammars from scratch. Instead, dialect contributors should localize keywords, error messages, and code samples in very few source code files. Additionally, contributors providing languages with right-to-left (RTL) writing should be able to easily configure their language with a flag that enables such behavior throughout the user interface.

For *static checking*, we expect the entirety of the user's code to be type-checked before execution begins. The transpiler must verify that all declarations, attributions, expressions, and function call arguments have types that are equivalent to the ones required. The static checker must issue an error if the user's code references a variable before its declaration. In a function's body, accessing a global variable without bringing it into scope with the `global` keyword should also lead to an error. In case of type mismatches, the error message in the console should indicate the line in which the error occurred, the type that the static checker expected, and the actual type it has found. Finally, the error messages should provide hints for how to fix common errors whenever possible.

Concerning *usefulness*, the language should still resemble well-known programming languages such as Python, Ruby, PHP, and C. To verify this, we plan to compare simple implementations of programs such as FizzBuzz, Factorial, and Fibonacci among different programming languages. The PanScript implementations should resemble the other ones in the number of lines, keywords, and functions used.

As for being *free* and *open-source*, we intend to have PanScript's code available online in a public repository under the MIT license (LIN et al., 2006). The website must have a link to the source code for all to find. The repository should contain manifests with the adopted code of conduct and guides for contributing to the PanScript project. The repository's initial page should acknowledge all contributors ever involved in the project.

User-friendliness is left out of self-evaluation, as the online survey provides us

with better means to gauge it. The following section describes the methodology used in said survey.

5.2 Online survey

The objective of our online survey is to gather feedback on PanScript's prototype from students, developers, researchers, and teachers. The survey is conducted only in Portuguese due to difficulties in localizing it to other non-English languages. We asked participants to access PanScript using a desktop computer since the prototype did not support mobile devices when we conducted the survey.

We start by asking the respondents a few questions about themselves. **Q1.** *Do you agree to participate in this survey?* **Q2.** *How old are you?* **Q3.** *Do you know how to read and write in English?* **Q4.** *At what age could you already express yourself in English?* **Q5.** *What resources did you use when learning English? E.g., games, movies, TV series, songs, dictionary, school, courses, online forums...* **Q6.** *Do you know how to program?* **Q7.** *At what age could you already write computer programs?* **Q8.** *In which programming languages did you learn to program? E.g., Portugol, Assembly, BASIC, C, Delphi, Fortran, Pascal, PHP, Java, Python...* **Q9.** *What resources did you use when learning to program? E.g., books, magazines, school, technical program, online forums...* **Q10.** *What difficulties did you face while learning to program? If you have learned programming before learning English, did that cause any specific issue?*

All age-related questions (Q2, Q4, and Q7) provide alternatives bucketed as follows: *a) 15 years or less*; *b) 16-20*; *c) 21-25*; . . . ; *h) 46-50*; and *i) 50 years or more*. Q3, which asks if the respondent knows English, offers the following alternatives: *a) I know how to read and write in English very well*; *b) I know how to read and write in English reasonably well*; *c) I do not know how to read or write in English, but I am learning it*; and *d) I do not know how to read or write in English and I am not learning it*. Q6 offers similar options when asking the respondent if they know how to program: *a) I know how to program very well*; *b) I know how to program reasonably well*; *c) I do not know how to program, but I am learning it*; and *d) I do not know how to program and I am not learning it*. Q5, Q8, Q9, and Q10 are open-ended questions.

In the next section of the survey, we ask respondents to access the PanScript prototype at the URL <https://panscript.github.io/>. We again ask them to use a desktop computer to answer the remaining questions since the prototype does not currently sup-

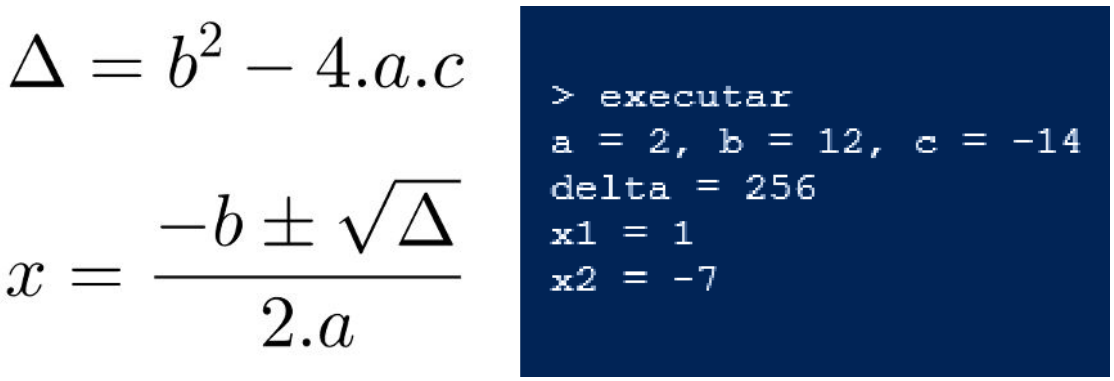
port mobile devices. **Q11.** *Were you able to access the website?* If the respondent states that they could not access the website, they will skip questions Q12 through Q20 and go to the final open-ended questions to explain whichever issues they have faced.

Having already accessed the PanScript prototype, we ask respondents to familiarize themselves with the interface, observing the different elements and controls at their own pace. We proceed to instruct the participant in using a few of these elements before attempting a programming exercise. **Q12.** *Could you get used to the interface?* **Q13.** *Change the language of the tool to Portuguese (Brazil).* **Q14.** *Change the editor's theme to the one you prefer.* **Q15.** *View the Basic example that teaches Variables.* **Q16.** *Run the Basic example that teaches Variables.* Q12 is a simple *Yes* and *No* question. Q13 through Q16 provide the following alternatives: *a) I have completed the task without difficulties*; *b) I have completed the task with difficulties*; and *c) I could not complete the task.*

Our programming exercise asks the participant to write localized PanScript code to calculate the real roots of a quadratic equation for a given set of coefficients. We have provided instructions on what the quadratic formula looks like, which standard library functions we expected them to use, and the results they should obtain. **Q17.** *Based on the various examples available, write a simple program to calculate the roots of a quadratic equation given the coefficients a = 2, b = 12, c = -14. You should only need the functions "square_root" and "write." You* ***do not*** *need to create a function. Calculating the results and writing them onscreen is enough. If you cannot complete this task, there is no problem. Just indicate so and continue answering the survey. Formulas and expected results are in the images below. More information is at the following link: <https://brasilescola.uol.com.br/matematica/formula-bhaskara.htm>*. Figure 5.1 shows the images presented below this question and before its alternatives. Note that "executar" is Portuguese for "run." The following alternatives are available: *a) I have completed the task without difficulties*; *b) I have completed the task with difficulties*; *c) I could not complete the task*; and *d) I did not feel like completing the task.* We then ask the respondent to provide their source code for further analysis. **Q18.** *Copy and paste your code below*.

The subsequent section contains a series of qualitative questions about PanScript. Q19 is a Likert-type scale questionnaire (LIKERT, 1932) with 17 items in total. The following alternatives are available for each item: *a) Strongly disagree*; *b) Partially disagree*; *c) Indifferent*; *d) Partially agree*; *e) Strongly agree*; *f) I do not know*. The Likert items are as follows: **i.** *I think the tool is limited*; **ii.** *I think the code's syntax is easy to understand*; **iii.** *I think the website is fast*; **iv.** *I think the tool can help in teaching programming*; **v.**

Figure 5.1 – Images shown as reference in Q17, the programming exercise.



Source: The Author

I like the tool; **vi.** *I think the tool is intuitive*; **vii.** *I think the examples are incomplete*; **viii.** *I think better tools already exist*; **ix.** *I think the examples are easy to understand*; **x.** *I think the layout is disorganized*; **xi.** *I think the tool is useful*; **xii.** *I think the interface is ugly*; **xiii.** *I think the error messages are complex*; **xiv.** *I do not like the code's syntax*; **xv.** *I think the tool is easy to use*; **xvi.** *I do not like the tool*; and **xvii.** *I think the tool is frustrating*. There are nine negative Likert items (e.g., I think the interface is ugly) and eight positive ones (e.g., I think the website is fast). We have randomly sorted the Likert items; however, all participants receive them in this order.

Q20 is a Net Promoter Score (NPS) question (REICHHELD, 2003). It asks participants to rate PanScript on a scale of zero to ten. These numbers will help us understand how well respondents accepted the tool. **Q20.** *On a scale of zero to ten, how much would you recommend PanScript as a tool to help in the basic teaching of programming to students that do not understand English?*

The final section of the survey includes five open-ended questions to gather the participant's feedback on PanScript and uncover any problems they might have faced. **Q21.** *What did you consider good in PanScript?* **Q22.** *What would you change in PanScript?* **Q23.** *Describe any issues you have faced while using PanScript.* **Q24.** *Did you miss any features in the tool? Which one(s)?* **Q25.** *This space is for your additional comments.*

6 RESULTS OBTAINED

In this chapter, we discuss the results of our self-evaluation and the online survey. We have characterized both of these evaluation criteria in full in Chapter 5.

6.1 Self-evaluation results

We discuss the results of the self-evaluation below. Table 6.1 presents an overview of the completion status of each of the project's objectives. As previously mentioned, our self-evaluation is not concerned with *user-friendliness* because that is best measured by the online survey in the following section.

Table 6.1 – Self-evaluation of the PanScript project

Objective	**Status**
Localization	Needs Attention
User-friendliness	N/A
Accessibility	Needs Attention
Code simplicity	OK
Frictionless UX	Almost Done
Extensibility	OK
Static checking	OK
Usefulness	OK
Free and Open-Source	OK

Source: The Author

Regarding *localization*, the PanScript prototype succeeded in localizing all the visible elements of the user interface, including the heading, the labels of all the buttons, all messages written in the output console, and the localized programming languages themselves. Currently, the project offers both Portuguese and French localization. We did not yet have the opportunity to include a language that requires right-to-left (RTL) writing. Almost all of the components we use today support RTL modes, including Ant Design (DESIGN, 2020) and CodeMirror (CODEMIRROR, 2017). The exception is Xterm.js, which has an open issue about RTL support since 2017 (XTERM.JS, 2017). Therefore, significant changes are needed to support Arabic, Persian, Hebrew, and other RTL languages. Portuguese already supports accented characters in variables and function names.

As for *accessibility*, the prototype currently lacks most of the features that the complete solution should have. Keyboard-only navigation is not entirely possible yet.

We have included a feature to enable deselecting the editor by pressing the escape key; however, this does not allow the user to select the output console using only a keyboard. Even if they could, they would be unable to scroll it using only the keyboard, hindering their user experience. Another problem with keyboard-only input is that the Directory Tree element from Ant Design had a bug that prevented keyboard navigation. We have contributed a fix to that bug such that the next release will not have the same problem (BRUM, 2021). Support for screen readers in PanScript is also sub-optimal. Upon testing the solution with the Screen Reader extension for Google Chrome, we have found that the language selector is being read out loud as "auto-completion list," and the tree of examples is read as "Edit text." These problems prevent a blind person from selecting code samples. Furthermore, the prototype does not yet provide the option to increase font sizes, although several editor themes are available, including a high contrast version of the Monokai theme.

We believe the prototype firmly achieves the goal of *code simplicity*. While dedicating multiple weeks to planning language syntax and features, we have conducted several small surveys with friends and colleagues to gather feedback on design decisions along the way. The result is a small language, a clean syntax, and a standard library with function names that we believe are easy to read and understand. As intended, PanScript code uses a reduced number of symbols. If we ignore expression operators, the only symbols required are the equals sign (`=`) in variable initialization, double and single quotes (`"` and `'`) in text literals, parentheses and commas (`,`) in function arguments, and braces for interpolating variables within text literals.

Frictionless UX is another objective only partially met. PanScript is accessible through a fast static website hosted in GitHub Pages. However, mobile support is currently unavailable, and we were unable to test whether the application works on the newest versions of the Safari browser. It reportedly does not work at the moment. Other modern desktop-based web browsers are supported, and no browser plug-in is required.

In terms of *extensibility*, PanScript's design allows easy integration of new dialects into the platform. For example, adding support for Spanish requires only a few code changes. First, the contributor should clone the localization folder of another language, such as pt_br for Brazilian Portuguese, renaming it with an appropriate language code, such as es_es. They should also rename the files in the `code_samples` directory with the new language code. After that, the contributor should review the `CustomLexer.g4` and `CustomParser.g4` grammar files, translating all token values as necessary. The file

`LocalizedStrings.ts` contains tokens, function names, and error messages that require translation as well. The next step is to update the `LanguageOptions.ts` file to display the new language in the menu. Having completed these changes, running `npm run dev` allows the contributor to test if everything is working as expected. They can then use the platform to translate the code samples for the new language, saving the changes back to the `code_samples` folder. The final step is to run `npm run qa` to invoke the code formatter, the code linter, and the platform's unit tests before submitting the changes for review.

PanScript's *static checking* is working as planned. The current implementation is correctly keeping track of new variables and functions as the user declares them, along with their types. Expressions and attributions are always type-checked as expected. The arguments of function calls are also compared with the function's signature to make sure they match in number and types.

Concerning *usefulness*, code syntax of all PanScript dialects retains the language's inspiration in Python, Ruby, PHP, and C. Figure 6.1 compares a Fibonacci implementation in PanScript and Python. Figure 6.2 compares Factorial between PanScript and PHP. Figure 6.3 compares FizzBuzz implemented in both PanScript and Ruby. The PanScript language provides concepts relevant to the development of computational thinking and allows a swift transition to mainstream programming languages.

Finally, PanScript meets the goal of being *free* and *open-source*, having adopted the MIT license since the beginning. We have chosen to host the website using GitHub Pages, making it available at the following URL: <https://panscript.github.io/>. Anyone interested can access the project's source code at the following public git repository: <https://github.com/panscript/code>.

6.2 Online survey results

We have conducted our online survey to evaluate the PanScript prototype between September 22nd and October 2nd, 2021. We have shared a link to the Microsoft Forms on mailing lists and social networks, aimed primarily at IT students, teachers, analysts, and researchers from Brazil. We have gathered 62 responses in total. However, one of the respondents has answered Q1 refusing to participate in the survey, and two respondents answered Q11 indicating they did not have access to desktop computers to try PanScript. Therefore, we proceed to analyze the results in this section considering 59 answers.

Figure 6.1 – Fibonacci in PanScript and Python.

Panscript

```
function fib(number n) returns number
    if n == 0
        return 0
    else if n == 1
        return 1
    end

    return fib(n - 1) + fib(n - 2)
end

write(fib(7))
```

Python

```
def fib(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1

    return fib(n - 1) + fib(n - 2)


print(fib(7))
```

Source: The Author

Figure 6.2 – Factorial in PanScript and PHP.

Panscript

```
function factorial(number n) returns number
    if n == 0
        return 1
    end

    return n * factorial(n - 1)
end

write(factorial(5))
```

PHP

```
function factorial(int $n): int {
    if ($n == 0) {
        return 1;
    }

    return $n * factorial($n - 1);
}

echo factorial(5);
```

Source: The Author

Figure 6.3 – FizzBuzz in PanScript and Ruby.

Panscript

```
function fizzbuzz(number n)
    for number i from 1 to n
        if i % 15 == 0
            write("FizzBuzz")
        else if i % 3 == 0
            write("Fizz")
        else if i % 5 == 0
            write("Buzz")
        else
            write(i)
        end
    end
end

fizzbuzz(20)
```

Ruby

```
def fizzbuzz(n)
    for i in 1..n
        if i % 15 == 0
            puts "FizzBuzz"
        elsif i % 3 == 0
            puts "Fizz"
        elsif i % 5 == 0
            puts "Buzz"
        else
            puts i
        end
    end
end

fizzbuzz(20)
```

Source: The Author

6.2.1 Questions about the participants

This subsection groups questions Q2 through Q10. These questions asked participants about their age, proficiency in English and programming, the materials they have learned from, and the barriers encountered while learning.

Q2 attempted to characterize participants in terms of age. Table 6.2 presents the age distribution of the 59 respondents. We can see that almost 70% are aged 21 to 30, which is not in line with PanScript's target age demographic. However, we believe this survey still holds value as a peer-review of our work, even though it can neither prove nor disprove the adequacy of the tool in teaching young students.

Table 6.2 – Q2. How old are you?

Age	**# Respondents**
15 years or less	0
16-20	4
21-25	22
26-30	19
31-35	9
36-40	3
41-45	0
46-50	2
50 years or more	0

Source: The Author

Q3 asked participants to rate their English proficiency considering both reading and writing. Table 6.3 presents the aggregated results, in which almost 95% of participants claim to read and write in English sufficiently well. This number is much higher than the one observed by Data Popular (2013), indicating that our sample might be heavily biased towards people who have had the time and resources to learn English better than most of the Brazilian population. Once again, this is not in line with PanScript's target demographic, which focuses on students that do not understand English yet. Our sample also seems to suffer from a survivorship bias (BROWN et al., 1992), as we are surveying students from a prestigious university (UFRGS), professionals working in IT, teachers, and researchers, instead of average Brazilian students. These observations should discourage the generalization of these results to other parts of the population.

Table 6.3 – Q3. Do you know how to read and write in English?

English Proficiency	**# Respondents**
I know how to read and write in English very well	29
I know how to read and write in English reasonably well	27
I do not know how to read or write in English, but I am learning it	3
I do not know how to read or write in English and I am not learning it	0

Source: The Author

We have also asked participants at what age they could already express themselves in English. Table 6.4 presents the responses we have collected for Q4. More than a third of participants indicated being able to express themselves in English when they were 15 or younger. Another third of participants had achieved the same level before turning 21. Three respondents have not answered this question, having previously indicated that they cannot read or write in English yet.

Table 6.4 – Q4. At what age could you already express yourself in English?

Age for English	**# Respondents**
15 years or less	21
16-20	20
21-25	7
26-30	8
31-35	0
36-40	0
41-45	0
46-50	0
50 years or more	0

Source: The Author

Q5 is the last question about English proficiency. It asked participants which resources helped them learn English. Since this was an open-ended question, we have manually reviewed the responses and grouped them into the following categories: Apps, Books, Courses, Dictionary, Friends, Games, Internet, Movies, School, Self-paced Studies, Songs, Technical Materials, Travel, TV Series, and Work-related Tools. Table 6.5 presents the aggregated results from all participants. Responses could mention more than one resource, which is why the sum of all answers adds up to more than 59. From our understanding, the high proportion of respondents mentioning English courses helps corroborate our hypothesis about survivorship bias. English courses can be very pricey in Brazil, and a large part of the population does not have enough money to pay for them.

Table 6.5 – Q5. What resources did you use when learning English? E.g., games, movies, TV series, songs, dictionary, school, courses, online forums...

English Resource	# Respondents
Courses	37
Movies	33
TV Series	32
Songs	30
Games	29
School	23
Internet	22
Dictionary	9
Technical Materials	7
Travel	6
Books	5
Apps	4
Friends	3
Self-paced Studies	3
Work-related Tools	3

Source: The Author

On the subject of programming proficiency, Q6 asked participants whether they already knew how to program. Table 6.6 presents the results. We can observe that more than 90% of respondents claim they know how to program. Again, this is not in line with PanScript's target demographics, which should consist predominantly of students who are still learning to program. Still, we maintain that the results can serve as a peer-review and that it could still invalidate our prototype depending on the feedback received.

Table 6.6 – Q6. Do you know how to program?

Programming Proficiency	# Respondents
I know how to program very well	36
I know how to program reasonably well	18
I do not know how to program, but I am learning it	5
I do not know how to program and I am not learning it	0

Source: The Author

Similar to Q4, Q7 asked participants at what age they could already create their own computer programs. Table 6.7 contains the results. We observe that close to 47% of respondents have learned to write their own programs at the age range of 16 to 20. Our goal is for PanScript to enable students even younger than that to experiment with basic programming concepts while still experiencing the regular "look-and-feel" of writing code. Five respondents have not answered this question, as they have previously indicated that they cannot program yet.

Table 6.7 – Q7. At what age could you already write computer programs?

Age for Programming	# Respondents
15 years or less	8
16-20	28
21-25	15
26-30	2
31-35	1
36-40	0
41-45	0
46-50	0
50 years or more	0

Source: The Author

We compare the answers for Q4 and Q7 in Table 6.8, which cross-references the age at which participants learned English and the age at which they learned programming. We have omitted rows and columns that would contain only zeroes. There are a total of 52 respondents who indicated knowing both English and programming. We can see that 34% of those respondents have learned both English and programming in the same five-year range, and another 21% have learned programming before learning English. It is precisely these students that PanScript aims to help, allowing them to learn programming logic without the need for English words.

Table 6.8 – # Respondents by Ages for English and Programming

Age for English	Age for Programming				
	15 years or less	16-20	21-25	26-30	31-35
15 years or less	3	13	3	0	0
16-20	2	12	4	1	0
21-25	1	1	3	1	0
26-30	2	1	4	0	1

Source: The Author

Q8 asked respondents which programming languages taught them how to program. Table 6.9 presents the results. C was by far the most frequently mentioned programming language, followed by Python and Java. The high spot for C could be due to the language's general popularity and also because it is present in first-semester classes of Computer Science and Computer Engineering at UFRGS. It is worth noting how Portugol appeared in 13% of all answers, indicating some programming courses in Brazil have adopted it as an educational tool. PanScript aims to be similar to a globalized version of Portugol, available in dozens of languages to help teach students of all cultures and backgrounds. Finally, the considerably high number of mentions for R could be because some

respondents work with Data Analysis and Data Science. We have omitted responses with less than three mentions.

Table 6.9 – Q8. In which programming languages did you learn to program? E.g., Portugol, Assembly, BASIC, C, Delphi, Fortran, Pascal, PHP, Java, Python...

Programming Language	**# Respondents**
C	43
Python	34
Java	17
C++	13
JavaScript	12
R	9
Portugol	8
Pascal	8
PHP	8
Assembly	7
Delphi	4
SQL	3
Ruby	3

Source: The Author

Similar to Q5, Q9 proceeded to ask participants which resources have helped them learn to program. Table 6.10 presents the aggregated results. Here, too, we have manually grouped responses into several categories: Apps and Platforms, Books, Colleagues, Courses, Family, Internet, School, Technical Materials, and University. Once again, we can see how prevalent university students (or former students) are in our sample since more than two-thirds of respondents have mentioned university.

Table 6.10 – Q9. What resources did you use when learning to program? E.g., books, magazines, school, technical program, online forums...

Programming Resource	**# Respondents**
University	40
Internet	38
Courses	36
Books	18
School	6
Technical Materials	4
Apps and Platforms	2
Colleagues	2
Family	1

Source: The Author

As the last question about participants themselves, Q10 asked respondents to list difficulties they have faced while learning to program. We have also asked if they have faced any particular issue related to English understanding. Table 6.11 presents the results of our manual categorization of the answers. We have used the following categories: Computing Concepts, Communities, Documentation, English, Errors, Best Practices, Materials, Motivation, Paradigms, Practical Use, Programming Languages, and Programming Logic. Most respondents have mentioned programming languages (syntax, semantics, and standard libraries) and programming logic as difficulties. With C in particular, it is notable how some abbreviated function names can seem meaningless even for English speakers: `scanf`, `malloc`, `strtok`, `snprintf`, to name a few. All these names have meaning; however, that meaning is only evident after reading some documentation that mentions that `strtok` is a string tokenizer, for example. Close to 20% of respondents considered not knowing English as being an issue. A few more respondents stated they probably did not have this issue because they already knew English before learning to program. The full set of responses for this question is available in Appendix D.

Table 6.11 – Q10. What difficulties did you face while learning to program? If you have learned programming before learning English, did that cause any specific issue?

Difficulty	**# Respondents**
Programming Logic	15
Programming Languages	15
Computing Concepts	12
English	12
Materials	9
Paradigms	6
Practical Use	4
Documentation	3
Communities	3
Best Practices	3
Errors	2
Motivation	1

Source: The Author

The majority of respondents have mentioned that programming logic was their most significant barrier. We proceed to highlight some of the answers we have received. One participant has noted that: *"When I started programming, I already knew English well, and I believe that has greatly facilitated learning. Initially, I had trouble mentally visualizing the steps that the algorithm would follow in the code that I was writing; I often forgot to type colons, semicolons, etc.; and understanding the usage of so many*

functions [...]." A second participant stated that: *"As I was already fluent in English before programming, that aspect has not caused any issues. The hardest part [...] was remembering the specific programming language's syntax to express my logic.*" Another respondent said that *"Programming logic was the greatest difficulty; however, knowing English before learning to program was certainly fundamental.*" Yet another answer puts it similarly: *"The largest difficulty was the development of programming logic; yes, I learned programming before learning English more in-depth, so that has brought issues in interpreting what I was programming [...].*"

On subjects other than logic, one participant stated: *"I had practically no English proficiency when I started with programming; however, understanding structured code was easy because it involved few words such as main, printf, and some strange mnemonics such as malloc. What I would say was the hardest part was understanding the abstraction of memory through variables [...].*" A second respondent told us that: *"The biggest issue was understanding abstract concepts, especially in object-oriented programming. Learning programming before learning English has created difficulties when looking for help on the Internet, as most documentation and forums in this subject area are in English.*" A third participant has said that: *"The greatest difficulty was always understanding error messages, even as an English speaker [...].*" Finally, one participant has mentioned learning programming with Portugol, an educational programming language localized to Portuguese: *"Yes, it has brought difficulties with English terms; however, using Portugol, there was no problem related to language.*"

6.2.2 Multiple-choice questions about PanScript

This subsection groups questions Q12 through Q20. These are multiple-choice questions about PanScript and the several tasks we have asked participants to perform using the prototype.

Questions Q12 through Q16 asked participants to access the PanScript prototype and perform small tasks to familiarize themselves with the user interface. All of these tasks were completed without difficulty by at least 98% of respondents. Only one participant has reported an issue while changing the language to Portuguese, and another had some problem changing the editor's theme. Although these respondents did not elaborate on these issues in the open-ended questions, we assume they were related to the placement or visibility of menu items.

Q17 was the programming exercise in which we asked participants to write simple code to calculate the real roots of a second-degree polynomial using the quadratic formula. Table 6.12 presents the results. We can observe that close to 88% of respondents have completed the task, although almost 36% had difficulties. Of the nearly 12% of respondents who did not finish the exercise, only two participants said they could not do it, with the other five indicating they merely preferred to move on with the survey.

Table 6.12 – Q17. Based on the various examples available, write a simple program to calculate the roots of a quadratic equation given the coefficients...

Response	**# Respondents**
I have completed the task without difficulties	38
I have completed the task with difficulties	14
I could not complete the task	2
I did not feel like completing the task	5

Source: The Author

Q18 asked participants to copy and paste the code they had written for the previous question, allowing us to verify if the solutions were indeed correct and whether any unexpected coding pattern had appeared. Table 6.13 presents the results of our observations. A total of 49 respondents provided their code for analysis. We have found that 39 of these respondents did produce an entirely correct solution to the problem. Two of them have opted to write their code using English PanScript, while others have used Brazilian Portuguese as per our instructions. Figure 6.4 presents one such solution. Of the nine respondents who wrote incorrect code, six of them had mistakes in the quadratic equation. These mistakes included missing parentheses in the denominator, forgetting to change a sign to calculate the second root, etc. The other three respondents seemed to have abandoned their solutions. Finally, one respondent was seemingly confused by our instructions and thought they were not allowed to use variables. However, even in that case, they had successfully calculated the real roots of the given polynomial.

Table 6.13 – Q18. Copy and paste your code below

Code provided?	**Code correct?**	**Remarks**	**# Respondents**
Yes	Yes	All correct	37
		Used English PanScript	2
		Did not use variables	1
	No	Error in the quadratic formula	6
		Incomplete code	3
No	N/A	Code not available	10

Source: The Author

Figure 6.4 – Sample solution for the exercise about the quadratic formula.

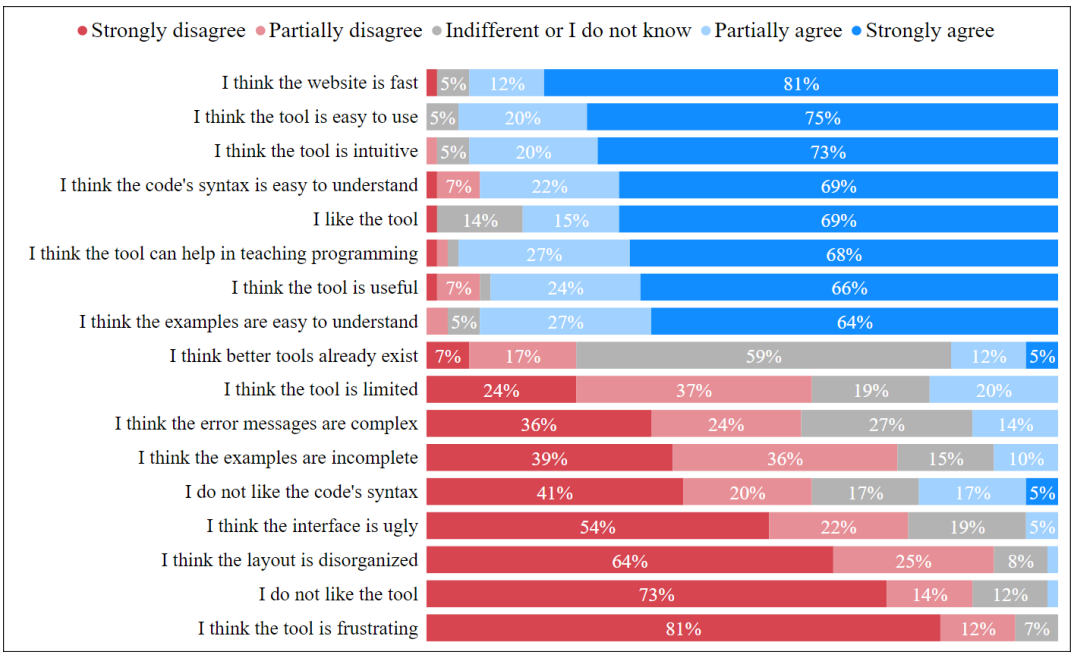
```
numero a = 2
numero b = 12
numero c = -14

numero delta = b*b-4*a*c
numero x1 = (-b+raiz_quadrada(delta))/(2*a)
numero x2 = (-b-raiz_quadrada(delta))/(2*a)

escreva(delta)
escreva(x1)
escreva(x2)
```

Source: The Author

Q19 contained 17 Likert-type items (LIKERT, 1932) in total. Respondents should indicate whether and how much they agree or disagree with each statement. Figure 6.5 presents the aggregated results for all the items. We can see that participants show a favorable attitude towards PanScript. They agree with many positive statements and disagree with many negative ones. Two topics seem particularly contentious: whether better tools exist and whether the code's syntax is good. For the first of these topics, most participants do not agree nor disagree with the statement. It is likely that, even if they know tools like Scratch and Portugol Studio, they might not know them well enough to compare them with PanScript. As for the second topic, code syntax was quite controversial. 22% of respondents did not like the code's syntax, and more than 8% did not think it was easy to understand. As we will see in the open-ended questions, the vast majority of complaints were related to the use of accented characters in Portuguese keywords such as `número` and `lógico`. We have since revised the Brazilian Portuguese dialect such that it no longer contains obligatory accented characters. Users may include accented characters in identifiers, but they do not appear in keywords and standard library function names.

Q20 is a Net Promoter Score (NPS) type question (REICHHELD, 2003), in which we asked participants how likely they were to recommend PanScript as a tool for teaching elementary programming concepts. Table 6.14 presents the breakdown of responses. When applying the standard NPS methodology, we obtain 59% of promoters (those who voted 9 or 10), 25% of passives (voting 7 or 8), and 15% of detractors (who voted less than 7). We calculate the final NPS score by subtracting the percentage of detractors from the percentage of promoters. For PanScript, the final result was 44, which is considered good, although it shows room for improvement.

Figure 6.5 – Results for each item in the Likert-type questionnaire.



Source: The Author

Table 6.14 – Q20. On a scale of zero to ten, how much would you recommend PanScript as a tool to help in the basic teaching of programming to students that do not understand English?

Response	**# Respondents**
0	0
1	0
2	1
3	1
4	0
5	4
6	3
7	6
8	9
9	12
10	23

Source: The Author

6.2.3 Open-ended questions

This subsection groups Q21 through Q25, which are open-ended questions. Participants who could not access the PanScript platform could still answer these questions, allowing them to describe any issues they had. Below we discuss the general sentiments expressed by respondents and their suggestions on ways to improve the tool. The full set of responses for these questions is available in Appendix D.

Q21 asked what participants liked about PanScript. Table 6.15 presents the most frequent topics mentioned. Most of the positive reactions are related to the user interface being intuitive, simple, friendly, and uncluttered. The second most frequently mentioned positive aspect of PanScript is its simple and intuitive code, followed by its localization and potential support for more human languages. Respondents were also favorable of our code samples, stating that they were clear, objective, and plentiful. Finally, other aspects mentioned include the tool's general ease of use, the fact that it is entirely web-based (not requiring installation), and the fast feedback when interpreting the user's code.

Table 6.15 – Q21. What did you consider good in PanScript?

Mentions	**# Respondents**
UI/UX	30
Code syntax	19
Localization	16
Code samples	12
Ease of use	7
Educational	6
Web-based	5
Fast feedback	2

Source: The Author

Q22 asked participants for constructive criticism, prompting them to list things they would change in PanScript. Q23 also asked participants if they had any issues using PanScript. Additionally, Q24 asked participants if they had missed any feature while using PanScript. We have analyzed the answers to all of these questions to create a list of topics that can inform future work in the project. Table 6.16 presents the most prominent subjects. The most frequently mentioned aspect was a distaste for the accented keywords in the Portuguese dialect, shared by more than a third of respondents. Upon receiving this feedback, we opted to remove mandatory accents from keywords and standard library function names. The second most relevant issue was the lack of layout responsiveness since the solution did not work well on smaller resolutions and scaled environments, such as notebooks. We have since made several changes to PanScript's layout to accommodate smaller resolutions.

We proceed to list other items mentioned by multiple respondents. Some participants have pointed out missing features such as an autocomplete functionality, missing data structures such as lists and tuples, and the lack of a visual programming component (e.g., providing some code visualization such as the control flow diagrams generated by PSeInt). The input function is also not yet present because it would require type conver-

sions and error handling functionality to try and parse numbers. Additionally, there is room for improving code samples, such as making them shorter and more kid-friendly. We have omitted from the table several additional improvement opportunities mentioned only once. We list all relevant entries as future work in Appendix A.

Table 6.16 – Q22. What would you change in PanScript?

Mentions	**# Respondents**
Remove accents	22
Layout responsiveness	6
Autocomplete in the editor	5
Data structures (such as lists and tuples)	4
Visual programming	4
Improve code samples (shorter and more kid-friendly)	3
Input function in the standard library	3
Shorter keywords	3
Transpilation to other programming languages	2
Text search in the editor	2
Improve error messages (more kid-friendly)	2
Additional documentation similar to Python's	2

Source: The Author

Particularly about Q23, a few participants have mentioned issues with missing features or existing bugs that we have since fixed in PanScript. These included a bug that could cause the code sample to load with the wrong language, the undo button being able to clear the code editor, and a lack of type-checking for function arguments. These are all fixed. Others have mentioned bugs that may still be present in PanScript. These include the tool being incompatible with the Safari browser and an error message indicating the wrong line of code as the origin of an error. We do not have access to a Mac to test the current Safari browser, and we could not reproduce the issue with the error message.

Finally, answers for Q25 mostly congratulated our work. A few participants were keen to show the tool to their friends or students and help them learn programming with it. They have felt PanScript could help teach elementary school students and high school students alike. However, at least three respondents have mentioned the importance of testing the tool with its actual target audience first. As previously stated, even though the present survey could have invalidated our hypotheses from a peer-review standpoint, it would not prove our assumptions since it did not reach PanScript's target demographic. Only two participants have remained skeptical of the tool, stating that it is preferable to teach using mainstream programming languages; and pointing out that students themselves might not be interested in learning through an educational programming language.

7 CONCLUSION AND FUTURE WORK

In this work, we have sought to understand and help mitigate the issue of young students trying to learn programming without knowing the English words that appear in most programming languages. We have noted how learning English is beneficial for developers, computer scientists, and computer engineers alike. English is currently a global lingua franca for many fields, while at the same time, many developing countries still display low proficiency levels in it. This imposes additional cognitive loads for non-English speakers, which English speakers do not face. We have looked at similar works that attempted to address these issues and the gaps that are still present in them.

Given this scenario, this work proposed the introduction of PanScript: a free web-based educational platform providing simple text-based programming languages that are localized to many human languages. PanScript's target demographics include young students aged 11 to 16, teachers that can use it in class, and open-source developers who wish to contribute. The main goals of the project are *localization*, *user-friendliness*, *accessibility*, *code simplicity*, *frictionless UX*, *extensibility*, *static checking*, *usefulness*, and being *free* and *open-source*. We have developed a prototype following these objectives using several technologies, such as TypeScript, React, ANTLR, CodeMirror, Xterm.js, and Ant Design. The prototype is available at the URL <https://panscript.github.io/>.

We have evaluated PanScript using both self-evaluations against the project's goals and an online survey conducted with 59 students, professionals, teachers, and researchers in IT. Results for the self-evaluation show room for improvement in terms of *localization* for right-to-left languages, *accessibility*, and *frictionless UX*. Results for the online survey are significantly positive; however, they cannot prove our hypotheses because the respondents we have reached are not part of the tool's target demographic. Instead, the survey functioned more as a peer-review of our work. Furthermore, the sample is not representative of the general population, at least due to its level of education.

Through the online survey, we have received many suggestions for improving PanScript. We have added most of them to our product backlog, including improvements to the platform's layout, making code samples and error messages more kid-friendly, and providing features such as an input function and basic data structures. Appendix A presents the current technical backlog for the project in full. Additionally, further research is needed to test PanScript with elementary and high school students, observing whether they enjoy learning programming with the tool or prefer alternatives such as Scratch.

REFERENCES

ABELSON, H.; FRIEDMAN, M. **MIT App Inventor**. 2010. Accessed on 2021-07-01; Archived under <https://archive.is/kRtnL>. Available from Internet: <https://appinventor.mit.edu/>.

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compilers: Principles, Techniques, and Tools**. USA: Addison-Wesley Longman Publishing Co., Inc., 1986. ISBN 0201100886.

AIVALOGLOU, E.; HERMANS, F. How kids code and how we know: An exploratory study on the scratch repository. In: **Proceedings of the 2016 ACM Conference on International Computing Education Research**. New York, NY, USA: Association for Computing Machinery, 2016. (ICER '16), p. 53–61. ISBN 9781450344494. Available from Internet: <https://doi.org/10.1145/2960310.2960325>.

AJAX.ORG. **Ace**. 2010. Accessed on 2021-07-26; Archived under <https://archive.is/B5pz7>. Available from Internet: <https://ace.c9.io/>.

AL-KHAWARIZM. **Al-Khawarizm**. 2018. Accessed on 2021-05-23; Archived under <https://archive.is/uLLI9>. Available from Internet: <https://alkhawarizm.org/>.

AMMOURI, A. A. A. **Ammoria**. 2006. Accessed on 2021-05-23; Archived under <https://archive.is/5ik2F>. Available from Internet: <http://ammoria.sourceforge.net/ar/ar_index.html>.

ANDERSON, S. R. How many languages are there in the world. **Linguistic Society of America**, 2010.

ANDRÉS, B. F.; PÉREZ, M. Transpiler-based architecture for multi-platform web applications. In: **2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)**. [S.l.: s.n.], 2017. p. 1–6.

ANIDO, R. Saci – ainda outro ambiente para o ensino de programação. In: **Anais do XXIII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2015. p. 226–235. ISSN 2595-6175. Available from Internet: <https://sol.sbc.org.br/index.php/wei/article/view/10239>.

APPLE. **Xcode**. 2003. Accessed on 2021-06-16; Archived under <https://archive.ph/ejPs4>. Available from Internet: <https://developer.apple.com/xcode/>.

ARMONI, M.; MEERBAUM-SALANT, O.; BEN-ARI, M. From scratch to "real" programming. **ACM Trans. Comput. Educ.**, Association for Computing Machinery, New York, NY, USA, v. 14, n. 4, feb. 2015. Available from Internet: <https://doi.org/10.1145/2677087>.

ARO, W. M. E. Software pseint en los niveles cognitivos en estudiantes del curso principios de algoritmos de la universidad tecnológica del perú-lima. Universidad César Vallejo, 2016.

ATWOOD, J. **Non-English Question Policy**. [S.l.]: Stack Overflow Blog, 2009. <https://stackoverflow.blog/2009/07/23/non-english-question-policy/>. Accessed on 2021-04-04; Archived under <https://archive.is/1UyZf>.

BALA, R. B.; ALACAPINAR, F. G. Scratch in teaching programming: Effect on problem solving skill and attitude. **International Journal of Quality in Education**, Abdülkadir KABADAYI, v. 5, p. 63 – 81, 2021.

BROWN, S. J. et al. Survivorship bias in performance studies. **The Review of Financial Studies**, Oxford University Press, v. 5, n. 4, p. 553–580, 1992.

BRUM, D. **fix: DirectoryTree keyboard error**. 2021. Accessed on 2021-11-02; Archived under <https://archive.is/izk50>. Available from Internet: <https://github.com/ant-design/ant-design/pull/32551>.

CAÑETE, B.; ENRIQUE, J.; RICARDO, A. V. La introducción de la herramienta didáctica pseint en el proceso de enseñanza aprendizaje: una propuesta para álgebra lineal. **Transformación**, Universidad de Camagüey, v. 15, n. 1, p. 147–157, 2019.

CENTER, L. **Foreign Language Proficiency**. [S.l.]: Leveda Center, 2014. <https://www.levada.ru/2014/05/28/vladenie-inostrannymi-yazykami/>. Accessed on 2021-04-08; Archived under <https://archive.is/BL3cK>.

CHANDRA, K. **nearley**. 2014. Accessed on 2021-07-26; Archived under <https://archive.is/imC1c>. Available from Internet: <https://nearley.js.org/>.

CHEDEAU, C.; LONG, J. et al. **Prettier**. 2017. Accessed on 2021-07-26; Archived under <https://archive.is/S2ewW>. Available from Internet: <https://prettier.io/>.

CODEMIRROR. **Bi-directional Text Demo**. 2017. Accessed on 2021-11-02; Archived under <https://archive.is/4N0nD>. Available from Internet: <https://codemirror.net/demo/bidi.html>.

COGLAN, J. **Canopy**. 2010. Accessed on 2021-07-26; Archived under <https://archive.is/y6J8u>. Available from Internet: <http://canopy.jcoglan.com/>.

CRUZ-BARRAGÁN, A.; MARTÍN, A.; LULE-PERALTA, A. Pseint technological tool to develop logical-mathematical intelligence in structured computer programming. **Journal of Technology and Innovation**, p. 22–30, 12 2019.

CRYSTAL, D. **English as a Global Language**. [S.l.: s.n.], 2003. ISBN 9780521530323.

DAHL, R. **Node.js**. 2009. Accessed on 2021-07-26; Archived under <https://archive.is/lhoeX>. Available from Internet: <https://nodejs.org/>.

DASGUPTA, S.; HILL, B. M. Learning to code in localized programming languages. In: **Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale**. New York, NY, USA: Association for Computing Machinery, 2017. (L@S '17), p. 33–39. ISBN 9781450344500. Available from Internet: <https://doi.org/10.1145/3051457.3051464>.

Data Popular. **Learning English in Brazil**. [S.l.]: British Council, 2013. <https://www.britishcouncil.org.br/sites/default/files/learning_english_in_brazil.pdf>. Accessed on 2021-04-04; Archived under <https://archive.is/m0DNq>.

DESIGN, A. **ConfigProvider**. 2020. Accessed on 2021-11-02; Archived under <https://archive.is/0b9sS>. Available from Internet: <https://ant.design/components/config-provider/>.

DODDS, K. C. et al. **Testing Library**. 2018. Accessed on 2021-07-26; Archived under <https://archive.is/kU8ep>. Available from Internet: <https://testing-library.com/>.

DRAGA, H. **Alif**. 2018. Accessed on 2021-05-23; Archived under <https://archive.is/Ts9sl>. Available from Internet: <https://www.aliflang.org/>.

DURAK, H. Y. The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. **Technology, Knowledge and Learning**, v. 25, 03 2020.

ESTEVES, A. et al. **Portugol Studio**. 2014. Accessed on 2021-05-22; Archived under <https://archive.ph/K11eT>. Available from Internet: <http://lite.acad.univali.br/portugol/>.

ESTEVES, A. et al. Portugol studio: Em direção a uma comunidade aberta para pesquisa sobre o aprendizado de programação. In: SBC. **Anais do XXVII Workshop sobre Educação em Computação**. [S.l.], 2019. p. 513–522.

ETHNOLOGUE. **English | Ethnologue**. 2019. <https://www.ethnologue.com/language/eng>. Accessed on 2021-04-05; Archived under <https://web.archive.org/web/20190926183242/https://www.ethnologue.com/language/eng>.

EUROBAROMETER. **Europeans and Their Languages**. [S.l.]: European Comission, 2012. <https://ec.europa.eu/commfrontoffice/publicopinion/archives/ebs/ebs_386_en.pdf>. Accessed on 2021-04-08; Archived under <https://web.archive.org/web/20210401070700/https://ec.europa.eu/commfrontoffice/publicopinion/archives/ebs/ebs_386_en.pdf>.

FRASER, N. Ten things we've learned from blockly. In: **2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)**. [S.l.: s.n.], 2015. p. 49–50.

GOOGLE. **Angular**. 2016. Accessed on 2021-07-26; Archived under <https://archive.is/bQYzU>. Available from Internet: <https://angular.io/>.

GOOGLE. **Google Grasshopper**. 2019. Accessed on 2021-07-01; Archived under <https://archive.is/YHIE9>. Available from Internet: <https://grasshopper.app/>.

GOOGLE; MIT. **Blockly**. 2012. Accessed on 2021-06-17; Archived under <https://archive.is/Bsor9>. Available from Internet: <https://developers.google.com/blockly>.

GULP.JS. **Popular plugins**. 2014. Accessed on 2021-11-01; Archived under <https://archive.is/OQuCe>. Available from Internet: <https://gulpjs.com/plugins/>.

GUO, P. J. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In: **Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2018. (CHI '18), p. 1–14. ISBN 9781450356206. Available from Internet: <https://doi.org/10.1145/3173574.3173970>.

HARRIS, B.; HARWELL, S. **antlr4ts**. 2016. Accessed on 2021-07-26; Archived under <https://archive.is/UqN0k>. Available from Internet: <https://github.com/tunnelvisionlabs/antlr4ts>.

HAUHIO, I. **Tampio**. 2017. Accessed on 2021-05-23; Archived under <https://archive.is/A5Chm>. Available from Internet: <https://github.com/fergusq/tampio>.

HAVERBEKE, M. **CodeMirror**. 2007. Accessed on 2021-07-26; Archived under <https://archive.is/byA3p>. Available from Internet: <https://codemirror.net/>.

HERMANS, F.; AIVALOGLOU, E. To scratch or not to scratch? a controlled experiment comparing plugged first and unplugged first programming lessons. In: **Proceedings of the 12th Workshop on Primary and Secondary Computing Education**. New York, NY, USA: Association for Computing Machinery, 2017. (WiPSCE '17), p. 49–56. ISBN 9781450354288. Available from Internet: <https://doi.org/10.1145/3137065.3137072>.

HILL, O. **Waxeye**. 2008. Accessed on 2021-07-26; Archived under <https://archive.is/ZjLds>. Available from Internet: <https://waxeye.org/>.

HOARE, C. A. R. **Hints on Programming Language Design**. Stanford, CA, USA: Stanford University, Department of Computer Science, 1973.

HUANG, L. **Wenyan**. 2019. Accessed on 2021-05-23; Archived under <https://archive.is/1FgEy>. Available from Internet: <https://wy-lang.org/>.

HUERTA, J. A. A.; GONZÁLEZ-BAÑALES, D. L. Pseint como herramienta para mejorar el proceso de enseñanza aprendizaje de algoritmos, pseudocódigo y diagramas de flujo. **Tecnologías de la Información en Educación: Sistematización de experiencias docentes**, p. 91, 2018.

IMCO. **Ingles es posible**. [S.l.]: Instituto Mexicano para la Competitividad, 2015. <https://imco.org.mx/wp-content/uploads/2015/04/2015_Documento_completo_Ingles_es_posible.pdf>. Accessed on 2021-04-08; Archived under <https://web.archive.org/web/20210117115841/https://imco.org.mx/wp-content/uploads/2015/04/2015_Documento_completo_Ingles_es_posible.pdf>.

INDIA, G. of. **Population by Bilingualism and Trilingualism**. [S.l.]: Office of the Registrar General & Census Commissioner, 2011. <https://www.censusindia.gov.in/2011census/C-17.html>. Accessed on 2021-04-08; Archived under <https://web.archive.org/web/20201019143407/https://www.censusindia.gov.in/2011census/C-17/DDW-C17-0000.XLSX>.

IU, M.-Y. C. **Babylscript**. 2011. Accessed on 2021-05-22; Archived under <https://archive.ph/A2BZW>. Available from Internet: <http://www.babylscript.com/>.

IU, M.-Y. C. Babylscript: multilingual javascript. In: **Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion**. [S.l.: s.n.], 2011. p. 197–198.

IU, M.-Y. C. **Babylscript: Why Are Programming Languages Always in English?** 2020. Accessed on 2021-11-02; Archived under <https://web.archive.org/web/20210115100325/https://www.youtube.com/watch?v=_-PvtTVeunQ>. Available from Internet: <https://www.youtube.com/watch?v=_-PvtTVeunQ>.

JETBRAINS. **IntelliJ**. 2001. Accessed on 2021-06-16; Archived under <https://archive.ph/krjJB>. Available from Internet: <https://www.jetbrains.com/idea/>.

JETBRAINS. **PyCharm**. 2010. Accessed on 2021-06-16; Archived under <https://archive.ph/UDJ3w>. Available from Internet: <https://www.jetbrains.com/pycharm/>.

JUNIOR, S. M. da S.; FRANÇA, S. V. A. Programação para todos: Análise comparativa de ferramentas utilizadas no ensino de programação. In: SBC. **Anais do XXV Workshop sobre Educação em Computação**. [S.l.], 2017.

JUPYTER. **Jupyter**. 2015. Accessed on 2021-07-26; Archived under <https://archive.is/WwgIP>. Available from Internet: <https://jupyter.org/>.

KALELIOGLU, F.; GULBAHAR, Y. The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. **Informatics in Education**, v. 13, p. 33–50, 04 2014.

KASIDIARIS, P.; IMMS, D. et al. **Xterm.js**. 2016. Accessed on 2021-07-26; Archived under <https://archive.is/KizpT>. Available from Internet: <https://xtermjs.org/>.

KATI. **Kati**. c. 2019. Accessed on 2021-05-23; Archived under <https://archive.is/M7xq2>. Available from Internet: <https://www.scanf.ir/?page=kati>.

KIRKPATRICK, A. Internationalization or englishization: Medium of instruction in today's universities. **Centre for Governance and Citizenship, The Hong Kong Institute of Education**, 01 2011.

KOPPERS, T.; LARKIN, S. et al. **webpack**. 2014. Accessed on 2021-07-26; Archived under <https://archive.is/oY3WE>. Available from Internet: <https://webpack.js.org/>.

LEETCODE. **LeetCode**. 2011. Accessed on 2021-07-26; Archived under <https://archive.is/b60uT>. Available from Internet: <https://leetcode.com/>.

LIKERT, R. A technique for the measurement of attitudes. **Archives of psychology**, 1932.

LIN, Y.-H. et al. Open source licenses and the creative commons framework: License selection and comparison. **J. Inf. Sci. Eng.**, v. 22, p. 1–17, 01 2006.

LUCENA, L. R.; LUCENA, M. Potigol, a programming language for beginners. In: **Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education**. [S.l.: s.n.], 2016. p. 368–368.

LUCENA, L. R. et al. **Potigol**. 2011. Accessed on 2021-05-22; Archived under <https://archive.ph/Zkykf>. Available from Internet: <https://potigol.github.io/>.

MAJDA, D.; RYUU, F.-z. **PEG.js**. 2010. Accessed on 2021-07-26; Archived under <https://archive.is/UxE1Y>. Available from Internet: <https://pegjs.org/>.

MALONEY, J. et al. The scratch programming language and environment. Association for Computing Machinery, New York, NY, USA, v. 10, n. 4, nov. 2010. Available from Internet: <https://doi.org/10.1145/1868358.1868363>.

MALONEY, J. H. et al. Programming by choice: Urban youth learning programming with scratch. **SIGCSE Bull.**, Association for Computing Machinery, New York, NY, USA, v. 40, n. 1, p. 367–371, mar. 2008. ISSN 0097-8418. Available from Internet: <https://doi.org/10.1145/1352322.1352260>.

MALYSHEVA, Y. An ast-based interface for composing and editing javascript on the phone. In: **2017 IEEE Blocks and Beyond Workshop (B B)**. [S.l.: s.n.], 2017. p. 9–16.

MARIMUTHU, M.; GOVENDER, P. Perceptions of scratch programming among secondary school students in kwazulu-natal, south africa. **The African Journal of Information and Communication**, v. 21, p. 51–80, 11 2018.

MATHEWS, K. A. **Gatsby**. 2015. Accessed on 2021-07-26; Archived under <https://archive.is/CrFAF>. Available from Internet: <https://www.gatsbyjs.com/>.

MEERBAUM-SALANT, O.; ARMONI, M.; BEN-ARI, M. Habits of programming in scratch. In: **Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2011. (ITiCSE '11), p. 168–172. ISBN 9781450306973. Available from Internet: <https://doi.org/10.1145/1999747.1999796>.

Microsoft. **Excel**. 1987. Accessed on 2021-05-24; Archived under <https://archive.is/M6hWJ>. Available from Internet: <https://www.microsoft.com/en-us/microsoft-365/excel>.

MICROSOFT. **Visual Studio**. 1997. Accessed on 2021-06-16; Archived under <https://archive.is/6L0f3>. Available from Internet: <https://visualstudio.microsoft.com/>.

MICROSOFT. **TypeScript**. 2012. Accessed on 2021-07-26; Archived under <https://archive.is/Ns00M>. Available from Internet: <https://www.typescriptlang.org/>.

MICROSOFT. **Using type dynamic (C# Programming Guide)**. [S.l.]: Microsoft, 2015. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/using-type-dynamic>. Accessed on 2021-04-17; Archived under <https://archive.is/ILSda>.

MICROSOFT. **Monaco Editor**. 2016. Accessed on 2021-07-26; Archived under <https://archive.is/U0hPc>. Available from Internet: <https://microsoft.github.io/monaco-editor/>.

MICROSOFT. **Excel Functions Translator**. [S.l.]: Microsoft, 2018. <https://support.microsoft.com/en-us/office/excel-functions-translator-f262d0c0-991c-485b-89b6-32cc8d326889>. Accessed on 2021-05-24; Archived under <https://archive.ph/Yzdwv>.

MONTERO, P. R. et al. **Latino**. 2015. Accessed on 2021-06-13; Archived under <https://archive.is/CJL11>. Available from Internet: <https://www.lenguajelatino.org/>.

MOOIJ, G. de. **Citrine**. 2014. Accessed on 2021-05-23; Archived under <https://archive.is/Qu0Fu>. Available from Internet: <https://citrine-lang.org/>.

NASSER, R. **Qalb**. 2012. Accessed on 2021-05-23; Archived under <https://archive.is/nd1lK>. Available from Internet: <http://nas.sr/%D9%82%D9%84%D8%A8/>.

NOSCHANG, L. F. et al. Portugol studio: Uma ide para iniciantes em programaçao. **Anais do CSBC/WEI**, p. 535–545, 2014.

NOVARA, P. **PSeInt**. 2003. Accessed on 2021-05-22; Archived under <https://archive.ph/f08vl>. Available from Internet: <http://pseint.sourceforge.net>.

NPM. **About packages and modules**. 2019. Accessed on 2021-11-01; Archived under <https://archive.is/OgNWC>. Available from Internet: <https://docs.npmjs.com/about-packages-and-modules>.

ORACLE. **JavaCC**. 2000. Accessed on 2021-07-26; Archived under <https://archive.is/7o1n1>. Available from Internet: <https://javacc.github.io/javacc/>.

OUAHBI, I. et al. Learning basic programming concepts by creating games with scratch programming environment. **Procedia - Social and Behavioral Sciences**, v. 191, p. 1479–1482, 2015. ISSN 1877-0428. The Proceedings of 6th World Conference on educational Sciences. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1877042815024842>.

PAGE, D. et al. **pgAdmin**. c. 2000. Accessed on 2021-07-26; Archived under <https://archive.is/Hhrps>. Available from Internet: <https://www.pgadmin.org/>.

PARR, T.; HARWELL, S.; FISHER, K. **ANTLR**. 1992. Accessed on 2021-07-26; Archived under <https://archive.is/pLubC>. Available from Internet: <https://www.antlr.org/>.

PARR, T.; HARWELL, S.; FISHER, K. Adaptive ll(*) parsing: The power of dynamic analysis. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 10, p. 579–598, oct. 2014. ISSN 0362-1340. Available from Internet: <https://doi.org/10.1145/2714064.2660202>.

PAUSCH, R. **Alice**. 1998. Accessed on 2021-07-01; Archived under <https://archive.is/XUU8v>. Available from Internet: <https://www.alice.org/>.

PC SOFT. **WLanguage**. 1992. Accessed on 2021-05-24; Archived under <https://archive.is/M6hWJ>. Available from Internet: <https://pcsoft.fr/wlangage.htm>.

PC SOFT. **WINDEV Nouvelle Version 26**. [S.l.]: PC SOFT, 2014. <https://pcsoft.fr/windev/ebook/56/index.html>. Accessed on 2021-05-24; Archived under <https://archive.is/PgdhH>.

PEREIRA, D. E. F.; SEABRA, R. D.; SOUZA, A. D. de. Ferramentas de apoio ao ensino introdutório de programação: um mapeamento sistemático. **RENOTE**, v. 18, n. 2, p. 491–500, 2020.

PERERA, P. et al. A systematic mapping of introductory programming languages for novice learners. **IEEE Access**, v. 9, p. 88121–88136, 2021.

PÉREZ-MARÍN, D. et al. Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children? **Computers in Human Behavior**, v. 105, p. 105849, 2020. ISSN 0747-5632. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0747563218306137>.

PEZOA, F. et al. Foundations of json schema. In: **Proceedings of the 25th International Conference on World Wide Web**. Republic and Canton of Geneva, CHE:

International World Wide Web Conferences Steering Committee, 2016. (WWW '16), p. 263–273. ISBN 9781450341431. Available from Internet: <https://doi.org/10.1145/2872427.2883029>.

PHPMYADMIN. 1998. Accessed on 2021-07-26; Archived under <https://archive.is/FZeIK>. Available from Internet: <https://www.phpmyadmin.net/>.

PIECH, C.; ABU-EL-HAIJA, S. **CodeInternational**. 2020. Accessed on 2021-05-22; Archived under <https://archive.ph/n3Hwx>. Available from Internet: <https://compedu.stanford.edu/codeInternational/docs/>.

PIECH, C.; ABU-EL-HAIJA, S. Human languages in source code: Auto-translation for localized instruction. In: **Proceedings of the Seventh ACM Conference on Learning@ Scale**. [S.l.: s.n.], 2020. p. 167–174.

REACT. **Components and Props**. 2017. Accessed on 2021-11-01; Archived under <https://archive.is/dT8lY>. Available from Internet: <https://reactjs.org/docs/components-and-props.html>.

REICHHELD, F. F. The one number you need to grow. **Harvard business review**, v. 81, n. 12, p. 46–55, 2003.

RESNICK, M. et al. **Scratch**. 2007. Accessed on 2021-06-16; Archived under <https://archive.is/IoEza>. Available from Internet: <https://scratch.mit.edu/>.

REZENDE, C. M. C.; BISPO, E. L. Comparison between the use of pseudocode and visual programming in programming teaching: An evaluation from scratch tool. In: **2018 13th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.: s.n.], 2018. p. 1–5.

ROSSUM, G. van; WARSAW, B.; COGHLAN, N. **Style Guide for Python Code**. [S.l.]: Python Software Foundation, 2001. <https://www.python.org/dev/peps/pep-0008/>. Accessed on 2021-04-05; Archived under <https://archive.is/4Tcxu>.

SÁEZ-LÓPEZ, J.-M.; ROMÁN-GONZÁLEZ, M.; VÁZQUEZ-CANO, E. Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. **Computers & Education**, v. 97, p. 129–141, 2016. ISSN 0360-1315. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0360131516300549>.

SÃNCHEZ, M.; BAHAMONDEZ, E. V.; CLUNIE, G. T. de. Use of pseint in teaching programming: A case study. In: **Proceedings of the 10th Euro-American Conference on Telematics and Information Systems**. New York, NY, USA: Association for Computing Machinery, 2020. (EATIS '20). ISBN 9781450377119. Available from Internet: <https://doi.org/10.1145/3401895.3402083>.

SCHLUETER, I. Z. **npm**. 2010. Accessed on 2021-07-26; Archived under <https://archive.is/Euvjq>. Available from Internet: <https://www.npmjs.com/>.

SCHOFFSTALL, E. **gulp.js**. 2013. Accessed on 2021-07-26; Archived under <https://archive.is/tU3Q0>. Available from Internet: <https://gulpjs.com/>.

SCRATCH. **Scratch Statistics**. [S.l.]: Scratch, 2021. <https://scratch.mit.edu/statistics/>. Accessed on 2021-07-01; Archived under <https://archive.ph/e7GMQ>.

SEBESTA, R. W. **Concepts of Programming Languages**. 10th. ed. [S.l.]: Pearson, 2012. ISBN 0273769103.

SERAJ, M. et al. Scratch and google blockly: How girls' programming skills and attitudes are influenced. In: **Proceedings of the 19th Koli Calling International Conference on Computing Education Research**. New York, NY, USA: Association for Computing Machinery, 2019. (Koli Calling '19). ISBN 9781450377157. Available from Internet: <https://doi.org/10.1145/3364510.3364515>.

SIGPLAN, A. **BYLAWS of the Special Interest Group on PROGRAMMING LANGUAGES of the Association for Computing Machinery**. [S.l.]: ACM, 2003. <http://www.acm.org/sigs/sigplan/sigplan_bylaws.htm>. Accessed on 2021-04-07; Archived under <https://web.archive.org/web/20060622110145/http://www.acm.org/sigs/sigplan/sigplan_bylaws.htm>.

SOLTANO, S. **Historical trends in the usage statistics of content languages for websites**. [S.l.]: W3techs.com, 2021. <https://w3techs.com/technologies/history_overview/content_language>. Accessed on 2021-04-04; Archived under <https://archive.is/we2IE>.

SOUTO, M. A. M.; LIVI, M. A. C. **INF01210 - INTRODUÇÃO À INFORMÁTICA**. [S.l.]: UFRGS - Instituto de Informática, 1999. <http://www.inf.ufrgs.br/~cidalivi/INF01210/excel1991.pdf>. Accessed on 2021-05-24; Archived under <https://archive.is/0Y15l>.

SOUZA, C. M. de; NICOLODI, A. C. **VISUALG 3.0**. c. 2003. Accessed on 2021-05-22; Archived under <https://archive.is/sf62p>. Available from Internet: <https://sourceforge.net/projects/visualg30/>.

STACKOVERFLOW. **Developer Survey 2020**. [S.l.]: Stack Overflow, 2020. <https://insights.stackoverflow.com/survey/>. Accessed on 2021-04-07; Archived under <https://web.archive.org/web/20210228225102/https://insights.stackoverflow.com/survey/>.

TEAM, A. D. **Ant Design**. 2016. Accessed on 2021-09-30; Archived under <https://archive.is/PZ5SZ>. Available from Internet: <https://ant.design/>.

TYNKER. **Tynker**. 2013. Accessed on 2021-07-01; Archived under <https://archive.is/hA0x3>. Available from Internet: <https://www.tynker.com/>.

VAZQUEZ, A.; SABAT, T.; COYIER, C. **CodePen**. 2012. Accessed on 2021-07-26; Archived under <https://archive.is/TdcDc>. Available from Internet: <https://codepen.io/>.

VISSER, E. Scannerless generalized-lr parsing. 04 1999.

WALKE, J. **React.js**. 2013. Accessed on 2021-07-26; Archived under <https://archive.is/jDg5y>. Available from Internet: <https://reactjs.org/>.

WATT, D. A. **Programming Language Design Concepts**. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004. ISBN 0470853204.

WEINTROP, D.; WILENSKY, U. Comparing block-based and text-based programming in high school computer science classrooms. Association for Computing Machinery, New York, NY, USA, v. 18, n. 1, oct. 2017. Available from Internet: <https://doi.org/10.1145/3089799>.

Whale Flight Desk. **Nadesiko**. 2004. Accessed on 2021-05-23; Archived under <https://archive.is/6NCQA>. Available from Internet: <https://nadesi.com/>.

Wikipedia. **Comparison of parser generators**. [S.l.]: Wikipedia, 2021. <https://en.wikipedia.org/wiki/Comparison_of_parser_generators>. Accessed on 2021-07-27; Archived under <https://archive.is/SHdkI>.

Wikipedia. **Non-English-based programming languages**. [S.l.]: Wikipedia, 2021. <https://en.wikipedia.org/wiki/Non-English-based_programming_languages>. Accessed on 2021-05-24; Archived under <https://archive.is/b5HEz>.

WILSON, A.; MOFFAT, D. Evaluating scratch to introduce younger schoolchildren to programming. **Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest group-PPIG2010, September 19-22, 2010**, 05 2012.

WING, J. Computational thinking. **Communications of the ACM**, v. 49, p. 33–35, 03 2006.

XTERM.JS. **Support RTL languages**. 2017. Accessed on 2021-11-02; Archived under <https://archive.is/hlmrc>. Available from Internet: <https://github.com/xtermjs/xterm.js/issues/701>.

YOU, E. **Vue.js**. 2014. Accessed on 2021-07-26; Archived under <https://archive.is/lF8x8>. Available from Internet: <https://vuejs.org/>.

YUTO. **Produire**. 2007. Accessed on 2021-05-23; Archived under <https://archive.is/pJMjN>. Available from Internet: <http://rdr.utopiat.net/>.

ZAKAS, N. C. **ESLint**. 2013. Accessed on 2021-07-26; Archived under <https://archive.is/R4wnG>. Available from Internet: <https://eslint.org/>.

ZAMIN, N. et al. Learning block programming using scratch among school children in malaysia and australia: An exploratory study. In: **2018 4th International Conference on Computer and Information Sciences (ICCOINS)**. [S.l.: s.n.], 2018. p. 1–6.

ZHANG, L.; NOURI, J. A systematic review of learning computational thinking through scratch in k-9. **Computers & Education**, https://authors.elsevier.com/a/1ZIfP1HucdHyVb, p. 103607, 06 2019.

ÖZORAN, D.; CAGILTAY, N.; TOPALLI, D. Using scratch in introduction to programming course for engineering students. In: . [S.l.: s.n.], 2012.

APPENDIX A — PANSCRIPT'S TECHNICAL BACKLOG

Current backlog for the PanScript project.

#	Summary	Description
1	Responsive layout	Have the UI adapt itself to smaller resolutions.
2	Resizable elements	Allow the user to collapse/resize the file explorer, the code editor, and the output console using vertical splitters.
3	Menu redesign	Make the Run button more prominent; perhaps group some menu items.
4	Detect language based on the user's browser	Automatically select a language for first-time visitors based on browser settings.
5	Code samples about how to fix common errors	Tell users the types of error messages that can appear, why they occur, and how to fix them.
6	Code samples about composite assignment operators	Add usage examples for `+=`, `-=`, `*=`, etc.
7	Improve code samples	Reorganize code samples, making them shorter and more kid-friendly; explain functions further; explain type compatibility.
8	Improve error messages	Make error messages more simple and kid-friendly; also add some color to them.
9	Input function	Add the ability to read a string from the standard input.
10	Automatic indentation in the code editor	Automatically indent code blocks for conditionals, loops, and function definitions.
11	Autocomplete suggestions in the code editor	Autocomplete for at least keywords and function names for the current dialect.
12	Tooltip showing the arguments a function expects	When editing a function call, display names and types of expected arguments.
13	Support for lists/arrays	Support for list types and related code samples.
14	Support the Safari browser	Fix whichever bug prevents PanScript from being used in Safari.

Continued on next page.

Continued from previous page.

#	Summary	Description
15	Add more languages	Add Spanish, Russian, Hindi, Tamil, Bengali, Marathi, Urdu, Hausa, Swahili, Yoruba, Amharic, Arabic, Persian, Tagalog, Vietnamese, Indonesian, Javanese, Malay, Thai, Italian, Greek...
16	Creation of new files in the file-tree	Add the ability to create new files in the file explorer; file upload should create a new file.
17	Output functions allowing multiple arguments	Allow write and write_inline functions to work with more than one argument, similar to Python and Ruby.
18	Better UI/UX design	Find designers to help improve PanScript's look and feel.
19	Mobile support	Support mobile devices using a different layout.
20	Accessibility features	Add font size controls; improve keyboard-only input; fix screen reader support.
21	Type casting	Add the ability to convert values between text, number, and logical types; have the means to test if conversion would be possible; throw error in case of invalid conversion.
22	Add optional/nullable types	Support for optional/nullable wrapper to safely represent missing values.
23	Try/catch	Ability to handle exceptions that occur during execution (e.g., conversion error).
24	Inform user when saving contents to Local Storage	Inform the user whenever PanScript stores the contents of the code editor in the browser's Local Storage.
25	Search in the code editor	Enable search in the code editor.
26	Textual documentation	Provide detailed textual documentation to accompany the code samples (similar to Python's).
27	Code visualization	Support generating control flow diagrams (similar to PSeInt's).

Continued on next page.

Continued from previous page.

#	Summary	Description
28	Code translation	Provide automatic translation of keywords and standard library function names between PanScript dialects.
29	Proposed exercises for each lesson	Invite students to solve problems using the concepts they learned (similar to Codecademy).
30	Support for tuples	Support for tuples and related code samples.
31	Support for dictionaries	Support for dictionary and related code samples.
32	Tooltips with translated keywords in the editor	Have the code editor show tooltips with translations when the user hovers over keywords.
33	Tour with usage instructions	Add a tour for first-time visitors presenting the UI and its controls.
34	Offline mode	Option to download the app for offline use.
35	Support for user-defined types	Support for creating and utilizing new user-defined types.
36	Support for importing other files	Ability to import code from other files.
37	Debug mode	Add a step-by-step execution mode for debugging.
38	Object-oriented programming concepts	Add support for OOP classes, objects, etc.
39	Option to show the JavaScript code	Option to display the JavaScript code generated from the user's PanScript code.
40	Code transpilation to Python	Ability to convert PanScript code to Python.

Source: The Author

APPENDIX B — THE PANSCRIPT STANDARD LIBRARY

Input/Output functions.

English name	Portuguese name	Description
write(1)	escreva(1)	Write a line of text to the output console.
write_inline(1)	escreva_na_linha(1)	Write text to the output console remaining in the same line.
new_line(1)	nova_linha(1)	Write a line break to the output console.
clear()	limpe()	Clear the output console.

Source: The Author

Text functions.

English name	Portuguese name	Description
to_text(1)	para_texto(1)	Convert a value of any type to text type.
pad_left(3)	preencha_esquerda(3)	Pad a text to the left using another text until a given length.
pad_right(3)	preencha_direita(3)	Pad a text to the right using another text until a given length.
length(1)	comprimento(1)	Number of characters in a text.
repeat(2)	repita(2)	Repeat a certain text n times.
upper_case(1)	maiusculas(1)	Return the text with all characters converted to uppercase.
lower_case(1)	minusculas(1)	Return the text with all characters converted to lowercase.
sentence_case(1)	sentenca(1)	Return a text with the first character converted to uppercase and the remaining characters converted to lowercase.
left(2)	esquerda(2)	Return up to n characters from the start of the text.
right(2)	direita(2)	Return up to n characters from the end of the text.

Continued on next page.

Continued from previous page.

English name	**Portuguese name**	**Description**
middle(3)	meio(3)	Return up to n characters from the text starting at the given position (0-indexed).
slice_text(3)	fatie_texto(3)	Return all characters between two positions of a text (0-indexed).
reverse_text(1)	inverta_texto(1)	Return the text with all characters in the inverse order.
in_text(2)	no_texto(2)	Return true if a text contains another text.
position(2)	posicao(2)	Return the index (inside a text) of the first occurrence of another text.
trim(1)	aparar(1)	Remove whitespace characters from the start and the end of a text.
trim_left(1)	aparar_esquerda(1)	Remove whitespace characters from the start of a text.
trim_right(1)	aparar_direita(1)	Remove whitespace characters from the end of a text.

Source: The Author

Math functions.

English name	**Portuguese name**	**Description**
pi()	pi()	Return an approximate value of the constant π.
e()	e()	Return an approximate value of the constant e.
absolute(1)	absoluto(1)	Return the absolute value (modulo) of a number.
power(2)	potencia(2)	Return a raised to the power b.
square_root(1)	raiz_quadrada(1)	Return the square root of a number.
sine(1)	seno(1)	Return the sine of a number. The number should be in radians.

Continued on next page.

Continued from previous page.

English name	**Portuguese name**	**Description**
cosine(1)	cosseno(1)	Return the cosine of a number. The number should be in radians.
tangent(1)	tangente(1)	Return the tangent of a number. The number should be in radians.
arc_sine(1)	arco_seno(1)	Return the arc sine (inverse of sine) of a number. The result is in radians.
arc_cosine(1)	arco_cosseno(1)	Return the arc cosine (inverse of cosine) of a number. The result is in radians.
arc_tangent(1)	arco_tangente(1)	Return the arc tangent (inverse of tangent) of a number. The result is in radians.
exponential(1)	exponencial(1)	Return e^x for a number x, where e is the Euler's number.
natural_logarithm(1)	logaritmo_natural(1)	Return the natural logarithm (inverse of exponential) of a number.
logarithm(2)	logaritmo(2)	Return the logarithm of a number in a given base.
floor(1)	piso(1)	Return the given number rounded down to the previous integer.
ceiling(1)	teto(1)	Return the given number rounded up to the next integer.
truncate(1)	trunque(1)	Return the given number without any fractional part.
minimum(2)	minimo(2)	Return the smallest of two numbers.
maximum(2)	maximo(2)	Return the largest of two numbers.
random_real(2)	real_aleatorio(2)	Return a random real number in the interval [a, b) using a uniform distribution.

Continued on next page.

Continued from previous page.

English name	**Portuguese name**	**Description**
random_integer(2)	inteiro_aleatorio(2)	Return a random integer in the interval [a, b) using a uniform distribution.
round(1)	arredonde(1)	Return the given number rounded to the nearest integer (midpoint rounded away from zero).
round_n_places(2)	arredonde_n_casas(2)	Return the given number rounded to n places (midpoint rounded away from zero).
truncate_n_places(2)	trunque_n_casas(2)	Return the given number truncated to n places.

Source: The Author

APPENDIX C — THE PANSCRIPT CANONICAL GRAMMARS

Canonical lexer grammar with minified formatting

```
lexer grammar CommonLexer;

@members { parenLevel = 0; }
tokens { TEXT_CONTENT }

fragment Alpha
  : [A-Za-z]
  ;

fragment Digit
  : [0-9]
  ;

TRUE: 'true';
FALSE: 'false';
BREAK: 'break';
CONSTANT: 'constant';
CONTINUE: 'continue';
ELSE: 'else';
END: 'end';
FOR: 'for';
FOREVER: 'forever';
FROM: 'from';
FUNCTION: 'function';
GLOBAL: 'global';
IF: 'if';
RETURN: 'return';
RETURNS: 'returns';
TO: 'to';
WHILE: 'while';
ASSIGN: '=';
ADD: '+';
SUBTRACT: '-';
MULTIPLY: '*';
DIVIDE: '/';
REMAINDER: '%';
POWER: '^';
```

```
ADD_ASSIGN: '+=';
SUBTRACT_ASSIGN: '-=';
MULTIPLY_ASSIGN: '*=';
DIVIDE_ASSIGN: '/=';
REMAINDER_ASSIGN: '%=';
POWER_ASSIGN: '^=';
LESS: '<';
LESS_OR_EQUAL: '<=';
GREATER: '>';
GREATER_OR_EQUAL: '>=';
EQUAL: '==';
DIFFERENT: '!=';
AND: '&&' | 'and';
OR: '||' | 'or';
NOT: '!' | 'not';
LOGICAL: 'logical';
NUMBER: 'number';
TEXT: 'text';
OPEN_PARENTHESIS: '(' { this.parenLevel += 1; };
CLOSE_PARENTHESIS: ')' { this.parenLevel -= 1; };
OPEN_BRACKET: '[';
CLOSE_BRACKET: ']';
OPEN_BRACE: '{';
CLOSE_BRACE: '}' -> popMode; // end text interpolation
DOT: '.';
COMMA: ',';

QUOTE_SINGLE
  : '\'' -> pushMode(SINGLE_QUOTE_TEXT) // start text
  ;

QUOTE_DOUBLE
  : '"' -> pushMode(DOUBLE_QUOTE_TEXT) // start text
  ;

IDENTIFIER
  : (Alpha | '_') (Alpha | Digit | '_')*
  ;

DECIMAL_NUMBER // allow numbers like 1, 1., .1 and 1.1
  : Digit+ '.'?
```

```
  | Digit* '.' Digit+
  ;

HEX_NUMBER
  : '0x' [0-9A-Fa-f]+
  ;

BINARY_NUMBER
  : '0b' [01]+
  ;

NEWLINE // ignored inside parentheses
  : { this.parenLevel == 0 }? [\r\n]+
  ;

WHITESPACE // capture newlines inside parentheses
  : ( { this.parenLevel > 0 }? [ \t\r\n]+
  | [ \t]+ ) -> channel(HIDDEN)
  ;

LINE_COMMENT
  : ( '//' ~[\r\n]*
  | '#' ~[\r\n]* ) -> channel(HIDDEN)
  ;

BLOCK_COMMENT
  : '/*' .*? '*/' -> channel(HIDDEN)
  ;

UNKNOWN
  : .
  ;


mode SINGLE_QUOTE_TEXT;

SINGLE_QUOTE_TEXT_QUOTE_SINGLE // end text
  : '\'' -> type(QUOTE_SINGLE), popMode
  ;

SINGLE_QUOTE_TEXT_CONTENT
```

```
120   : ( '\\' . // escaped character
121   | ~[\\\r\n']+ ) -> type(TEXT_CONTENT)
122   ;
123
124
125 mode DOUBLE_QUOTE_TEXT;
126
127 DOUBLE_QUOTE_TEXT_QUOTE_DOUBLE // end text
128   : '"' -> type(QUOTE_DOUBLE), popMode
129   ;
130
131 DOUBLE_QUOTE_TEXT_OPEN_BRACE // start text interpolation
132   : '{' -> type(OPEN_BRACE), pushMode(DEFAULT_MODE)
133   ;
134
135 DOUBLE_QUOTE_TEXT_CONTENT
136   : ( '\\' . // escaped character
137   | ~[\\\r\n"{]+ ) -> type(TEXT_CONTENT)
138   ;
```

Canonical parser grammar

```
parser grammar CommonParser;

options { tokenVocab=CommonLexer; }

program
  : NEWLINE* topStatement*
  ;

topStatement
  : functionDeclaration eos
  | statement
  ;

innerStatement
  : globalStatement eos
  | breakStatement eos
  | continueStatement eos
  | returnStatement eos
  | statement
  ;

statement
  : variableDeclaration eos
  | variableAssignment eos
  | ifStatement eos
  | forFromToStatement eos
  | whileStatement eos
  | foreverStatement eos
  | functionCall eos
  ;

globalStatement
  : GLOBAL IDENTIFIER
  ;

functionDeclaration
  : FUNCTION IDENTIFIER OPEN_PARENTHESIS parameterList? CLOSE_PARENTHESIS
    (RETURNS type)? NEWLINE+ innerStatement* END
  ;

```

```
41 parameterList
42   : type IDENTIFIER (COMMA type IDENTIFIER)*
43   ;
44
45 variableDeclaration
46   : CONSTANT? type IDENTIFIER ASSIGN expression
47   ;
48
49 type
50   : LOGICAL  #logicalType
51   | NUMBER   #numberType
52   | TEXT     #textType
53   ;
54
55 expression
56   : OPEN_PARENTHESIS expression CLOSE_PARENTHESIS  #parenthesisExpression
57   | ADD expression                                 #plusExpression
58   | SUBTRACT expression                            #minusExpression
59   | NOT expression                                 #notExpression
60   | <assoc=right> expression POWER expression      #powerExpression
61   | expression MULTIPLY expression                 #multiplyExpression
62   | expression DIVIDE expression                   #divideExpression
63   | expression REMAINDER expression                #remainderExpression
64   | expression ADD expression                      #addExpression
65   | expression SUBTRACT expression                 #subtractExpression
66   | expression LESS expression                     #lessExpression
67   | expression LESS_OR_EQUAL expression            #lessEqualExpression
68   | expression GREATER expression                  #greaterExpression
69   | expression GREATER_OR_EQUAL expression         #greaterEqualExpression
70   | expression EQUAL expression                    #equalExpression
71   | expression DIFFERENT expression                #differentExpression
72   | expression AND expression                      #andExpression
73   | expression OR expression                       #orExpression
74   | atom                                           #atomExpression
75   ;
76
77 atom
78   : TRUE           #trueAtom
79   | FALSE          #falseAtom
80   | numberLiteral  #numberAtom
81   | textLiteral    #textAtom
```

```
  | functionCall   #functionCallAtom
  | IDENTIFIER     #identifierAtom
  ;

numberLiteral
  : DECIMAL_NUMBER
  | HEX_NUMBER
  | BINARY_NUMBER
  ;

textLiteral
  : QUOTE_SINGLE simpleText* QUOTE_SINGLE        #simpleTextLiteral
  | QUOTE_DOUBLE interpolatedText* QUOTE_DOUBLE  #interpolatedTextLiteral
  ;

simpleText
  : TEXT_CONTENT
  ;

interpolatedText
  : simpleText                        #interpolatedSimpleText
  | OPEN_BRACE expression CLOSE_BRACE  #interpolatedExpressionText
  ;

functionCall
  : IDENTIFIER OPEN_PARENTHESIS argumentList? CLOSE_PARENTHESIS
  ;

argumentList
  : expression (COMMA expression)*
  ;

variableAssignment
  : IDENTIFIER ASSIGN expression           #assignment
  | IDENTIFIER ADD_ASSIGN expression       #addAssignment
  | IDENTIFIER SUBTRACT_ASSIGN expression  #subtractAssignment
  | IDENTIFIER MULTIPLY_ASSIGN expression  #multiplyAssignment
  | IDENTIFIER DIVIDE_ASSIGN expression    #divideAssignment
  | IDENTIFIER REMAINDER_ASSIGN expression  #remainderAssignment
  | IDENTIFIER POWER_ASSIGN expression     #powerAssignment
  ;
```

```
123
124 ifStatement
125   : IF expression NEWLINE+ innerStatement* elseIfPart* elsePart? END
126   ;
127
128 elseIfPart
129   : ELSE IF expression NEWLINE+ innerStatement*
130   ;
131
132 elsePart
133   : ELSE NEWLINE+ innerStatement*
134   ;
135
136 forFromToStatement
137   : FOR type IDENTIFIER FROM expression TO expression NEWLINE+ innerStatement* END
138   ;
139
140 whileStatement
141   : WHILE expression NEWLINE+ innerStatement* END
142   ;
143
144 foreverStatement
145   : FOREVER NEWLINE+ innerStatement* END
146   ;
147
148 breakStatement
149   : BREAK
150   ;
151
152 continueStatement
153   : CONTINUE
154   ;
155
156 returnStatement
157   : RETURN expression?
158   ;
159
160 eos
161   : NEWLINE+ EOF?
162   | EOF
163   ;
```

APPENDIX D — RESPONSES TO THE OPEN-ENDED QUESTIONS OF PANSCRIPT'S ONLINE SURVEY (IN BRAZILIAN PORTUGUESE)

Q10. Descreva brevemente as dificuldades que você enfrentou aprendendo programação. Se você aprendeu programação antes de aprender inglês, isso trouxe alguma dificuldade específica?

Id	Response
2	Encontrar material adequeado para o meu nível, motivação para me aprofundar mais. Aprendi inglês antes de programação.
3	Eu aprendi depois de aprender inglês. A principal dificuldade foi a inicial, de aprender o paradigma e os conceitos principais. Depois disso, foi coisas específicas de linguagens, e conceitos fundamentais de computação.
4	
5	Lidar com as diferentes sintaxes entre linguagens; lidar com diferentes paradigmas (OO x Procedural x Funcional); eventualmente lidar com linguagens muito verbosas (Java)
6	Eu já sabia inglês bem basico, então não acho que o idioma fez diferença.
7	Antes de ter mais confiança no Inglês tinha que procurar conteudo em português ou traduzir os que tinha a disposição.
8	Não lembro de ter tido grandes dificuldades para aprender programação. Acho que o mais difícil foi entender ponteiros em C. Quando comecei a aprender programação, eu já sabia o suficiente de inglês.
9	Sim, trouxe, principalmente no paradigma de classes porque existiam nomenclaturas diferentes e mais subjetivas.
10	Entender os conceitos da programação em si, trabalhar com matrizes, etc
11	Eu já sabia o básico do inglês, portanto, nesse quesito foi fácil. As maiores barreiras eram relacionadas a conceitos diretamente ligadas a programação (orientação a objetos, por exemplo).
12	Eu já sabia inglês quando comecei a programar, então não foi uma dificuldade.
13	Entender a lógica para montar as estruturas e o funcionamento delas. Como eu já tinha um nível de inglês, não senti dificuldades com relação aos comandos serem estarem no idioma.
15	Como aprendi programação junto com inglês, entender a documentação das bibliotecas e programas foi a maior dificuldade.

Continued on next page.

Continued from previous page.

Id	Response
16	A maior dificuldade sempre foi entender as mensagens de erro, mesmo falando ingles e lendo claramente o que a mensagem queria dizer, os compiladores parecem reclamar de um ";" faltando de uma maneira muito estranha.
17	Aprendi programação depois de aprender inglês. Maiores dificuldade giravam em torno de documentação incompleta, falta de algo que direcionasse os estudos (quando era o caso) e falta de fóruns de dúvidas mais básicas
18	Mesmo começando a programar sem saber inglês, isso não trouxe dificuldade para escrever o código em si. Porém, para conseguir auxílio de materiais sim, pois quase tudo, na época que comecei a programar, estava em inglês.
19	entendimento de hardware. inglês não foi um problema.
20	Lembrar as idiossincrasias da linguagem, traduzir a lógica pensada em lógica de programação
21	Quando comecei a programar, já sabia bem o inglês e acredito que isto tenha facilitado muito o aprendizado. No começo, tinha dificuldade de prever mentalmente os passos que o algoritmo iria seguir no código que eu estava escrevendo, eu esquecia com frequência de digitar dois pontos, ponto e vírgula etc. e entender o funcionamento das tantas funções que eu deveria usar. Não demorou muito para eu me adaptar, no entanto.
22	- Interpretar mensagens de erro da linguagem, especialmente de compiladores - Como organizar o código (diminuir acoplamento, aumentar coesão, garantir SRP) - Estruturas de dados tipo grafo e árvores
23	A maior dificuldade foi entender a parte técnica da linguagem. O idioma não foi um fator determinante.
24	Dificuldade relacionada às caracteristicas das linguagens, métodos de programação (orientada a objetos), modelagem de dados. Não, aprendia inglês primeiro
25	
26	Logica de programação foi a maior dificuldade, porém certamente saber inglês antes de saber a programar foi imprecindivel.
27	Sintaxe, semântica, erros de programação, bugs

Continued on next page.

Continued from previous page.

Id	Response
28	Não tinha muito bem onde aplicar os conhecimentos que estava adquirindo
29	Já possuía algum domínio do inglês. Dificuldades mais relacionadas a lógica.
30	Usar muito computador para diversas coisas e começar a automatizar tarefas
31	Nenhuma dificuldade relacionada a idioma
32	Sim, eu sou da area de dev frontend e ainda sinto um pouco de dificuldade no ingles.
33	A maior dificuldade é entender o problema a ser resolvido e transformá-lo em código. Além de escrever um código mais otimizado. Aprendi a programar sem ter muito conhecimento em inglês, e isso também dificultou um pouco.
34	
35	Sim, o inglês limita no aprendizado e.g no tempo que investe nas leituras e compreensão delas.
36	As principais dificuldades foram no aprendizado da lógica de programação. Já tinha domínio do inglês quando comecei a programar.
37	Sim, aprendi programação antes de aprender inglês. Entrei na universidade muito cedo e muito verde, demorei um tempo a entender que inglês seria essencial. Mas aprendi a programar no primeiro semestre com Pascal. Conceitos como Loop, palavras-chave como Then else demoraram a fazer sentido porque eu ficava buscando qual era tradução no meu dicionário mental. Lembro de gastar horas procurando tradução decente para NIL. São poucas palavras, parece que não vai influenciar, mas é um ledo engano. Atrapalha sim. Outra dificuldade relacionada que enfrentei era a escassez de títulos bibliográficos em português. Também tive a dificuldade com os exercícios. Na época, os principais repositórios de questões extras para treino eram a UVA e o topcoder, ambos em inglês. Enfim, não foi moleza.
38	A lógica quando apresentado com a linguagem ILA em português procedural ajudou no entendimento e depois isso virou uma tradução. Hoje em dia, a medida em que o entendimento do inglês é aprofundado, facilita ainda mais o entendimento de novos recursos e tecnologias.
39	Sintaxe, mas não relacionado ao inglês.

Continued on next page.

Continued from previous page.

Id	Response
40	A principal dificuldade foi entender conceitos mais abstratos, especialmente em programação orientada a objetos. Aprender programação antes de aprender inglês trouxe dificuldade na hora de buscar auxílio na Internet, pois a maior parte da documentação e fóruns da área está em inglês.
41	Algumas explicações disponíveis em livros eram excessivamente técnicas para a minha idade (10 anos na época) – não haviam livros de programação "para crianças". Para diversos conceitos, eram necessárias explicações por parte do meu pai, para depois entender o que o livro queria dizer (o livro que eu tinha, felizmente, era em português).
42	Na época achei difícil encontrar um material para iniciantes. Apesar de já saber um pouco de inglês na época, não era muito proficiente. Pensando agora isso pode ter dificuldado achar um material legal.
43	Ter aprendido inglês antes ajudou bastante.
44	Minha primeira experiência foi em uma disciplina da graduação, foi muito difícil porque eu não entendia a lógica, a dinâmica da coisa. Eu não estava acostumada a pensar dessa forma tão lógica e sequencial, e foi bem difícil no início, eu também não entendia bem o funcionamento do computador, e tinha dificuldades com a relação com o linux. Outra dificuldade foi entender que a dinâmica de programar envolve aprender copiando e errando, na época eu não me sentia confortável com isso, então acabava me frustrando muito.
45	Eu tinha praticamente nenhuma proficiência em inglês quando comecei com programação, mas entender código estruturado foi fácil pois envolvida poucas palavras como main, printf, e alguns mnemônicos estranhos como malloc. O que eu diria que foi mais difícil foi compreender a abstração de memória por variáveis, e com frequência eu confundia o nome da variável com o conteúdo da mesmo (p.ex. "int idade;")
46	Já sabia bem inglês quando comecei a aprender programação, mas, ainda assim, no início tive dificuldades com termos técnicos e demorei pra absorver com naturalidade o que cada comando significava dentro do programa.

Continued on next page.

Continued from previous page.

Id	Response
47	A falta de um professor mais acessível em cursos online gratuitos ou de plataforma é um grande obstáculo, nada se compara à faculdade, onde podemos conversar diretamente com os professores ou nossos colegas que provavelmente têm o mesmo nível de saber nosso. Perguntas em fóruns não deixam de ser úteis, mas podem dificultar um pouco o iniciante em programação, pois quem responderá não tem o mínimo de noção do que a pessoa que perguntou já sabe ou não; não é dinâmico.
48	Não me lembrod e ter dificuldades
49	A dificuldade maior é fazer rodar os programas de primeira (porque sempre esqueço alguma bobagem). Eu geralmente tenho muitas ideias, mas às vezes elas são ambiciosas demais para a minha habilidade. Acho o uso de matrizes difícil também.
50	Como eu já era fluente em inglês antes de programar, esse aspecto não trouxe nenhuma dificuldade. A parte mais difícil do aprendizado de programação foi lembrar as sintaxes específicas da linguagem de programação para expressar a minha lógica.
51	Assimilar a lógica de programação, principalmente. Já sabia inglês antes de começar a programar
52	Sim, aprendi a programar antes de aprender inglês, e o que aconteceu é que eu tive que acabar aprendendo inglês, pelo menos o básico das instruções (for, if, else...).
53	Aprendi inglês antes de programar. Dificuldades maiores no início foram entender os comandos, passar a lógica que montava na minha cabeça para a sintaxe da linguagem
54	Sim, aprendi, mas não foi uma dificuldade. Trouxe benefícios, pois aprendi inglês.
55	
56	A principal dificuldade foi o desenvolvimento da lógica de programação; e sim, eu aprendi programação antes de aprender inglês de forma mais aprofundada, então isso trouxe dificuldade quanto a intepretação do que eu estava programando; fazia e entendia os comandos, mas não o contexto da aplicação.
57	Sim, trouxe dificuldade em relação aos termos em inglês , mas no Portugol foi tranquilo a questão da língua

Continued on next page.

Continued from previous page.

Id	Response
58	Não houve dificuldades.
60	Não tive muitos problemas em aprender programação. Se incluir “pensar logicamente”, em que cada passo, mesmo que mínimo, precisa ser especificado, poderia ser um ponto de dificuldade.
61	dificuldade para estruturar o pensamento em linguagem de programação
62	Tive muitos problemas em entender como programas rodavam/compilavam, já que, nas cadeiras introdutórias, a maior parte do conteúdo é apresentada fora do contexto de plataformas UNIX. Além disso, tive dificuldades para aprender a organizar e a separar meu código em diversos arquivos fonte.

Source: The Author

Q21. O que você achou bom na ferramenta PanScript?

Id	Response
2	Interface simples, a possibilidade de poder criar a própria versão da sintaxe independente da língua
3	A clareza e a simplicidade. O site também é minimalista, o que é *excelente*, já que não afoga o usuário com opções que ele pode nem entender.
4	Praticidade, e a ferramenta é bem intuitiva
5	A função de troca de idiomas; o uso de acentos gráficos no código
6	Achei simples e sem grandes problemas de uso.
7	Fácil de usar, com exemplos legais
8	Achei a interface e a linguagem de programação bastante simples e intuitivas.
9	Acho uma ótima iniciativa.
10	Interface bem feita e (aparentemente) sem nenhum erro grave
11	O propósito de ensino de programação em língua portuguesa é muito legal. Também a interface é bem intuitiva e facilita o uso.
12	Fácil de usar, exemplos claros.
13	Simples e bem didática. Muito boa para quem irá iniciar os estudos de programação.
15	Simplicidade e vários exemplos inclusos.
16	Achei a sintaxe muito intuitiva. Passando rápido pelos exemplos deu pra escrever o programa e rodar de primeira, sem erros (o que é raro...). Não tive a experiencia de ter um erro pra saber o quão bom são as mensagens de erro.
17	Opções de comando totalmente em português, exemplos, ser totalmente web (tenho um cunhado que está tendo que aprender para uma matéria da Faculdade, mas a família da minha esposa é bem não tecnológica a ponto dele se quer conseguir instalar o Python no windows sem a minha ajuda)
18	Intuitiva e com exemplos fáceis. Apresenta o erro de forma clara permitindo ao estudante identificar a linha em que se encontra o erro.
19	Entendimento da lógica
20	A ideia parece interessante, mas é difícil julgar se atinge o objetivo já tendo experiência com programação.

Continued on next page.

Continued from previous page.

Id	Response
21	O fato da interface ser bem simplificada e permitir programar em português. A sintaxe não usa tantos caracteres, então é boa para o aprendizado de quem nunca programou. Me lembrou bastante o VisualG.
22	Rapidez e suporta a português
23	Intuitiva e ambiente “limpo” (sem poluição visual na tela).
24	Facilidade de testar e ter exemplos com explicação
25	Achei uma ideia muito boa e bem executada. Tem muito potencial.
26	Facilidade e objetividade
27	Eu achei que a sintaxe é muito didática, eu certamente recomendaria para alunes futuros. Tem aspectos muitos bons e simples na forma como a linguagem é apresentada.
28	* Ela ser traduzida para o português * Não focar em uma linguagem específica de programação mas no racional * Ser bem intuitiva
29	Bem organizada.
30	Pareceu bem interessante
31	Interface
32	Simplicidade.
33	Bastante intuitiva e fácil de usar.
34	Simplicidade
35	Está acorde ao padrão de editores online
36	Interface simples e intuitiva, funcionamento no navegador (não requer instalação), feedback rápido, possibilidades de customização
37	a proposta de programar em português, a disponibilidade de exemplos para explicar as diversas estruturas, os exemplos terem mais código que texto. a interface simples, direta, eficiente e amigável.
38	Uma ótima ferramenta que pode apoiar muitas pessoas,
39	Interface, exemplos
40	Exemplos bem organizados, explicações apresentadas de forma clara e direta.

Continued on next page.

Continued from previous page.

Id	Response
41	A quebra da barreira da língua Já lecionei em turmas onde raros eram aqueles que já tinham algum conhecimento prévio de inglês; algo simples como ‘`if (num > 5) { printf("blah"); }`‘ já esbarra no “o que é num” e “o que é print” – até pq as pessoas associam “print” com impressora
42	Achei bom que tem em português.
43	A facilidade no uso e na segmentação das paginas.
44	Eu gostei bastante dela. MAS, eu acho que ela é intuitiva para quem entende de programação, para quem já está acostumado. Ela serve de apoio, mas necessita de um professor/tutor para ajudar no início, eu não acho que ela serve para aprender sozinho. Para aprender sozinho talvez só se tivesse um arquivo de texto auxiliar que sugerisse atividades. Mas eu super vou recomendar a ferramenta
45	A ideia em si é boa, está muito bem executada.
46	É uma ferramenta simples com tudo que é necessário para quem está começando no mundo da programação. Extremamente intuitiva e a versão em português é muito prática. Já trabalhei em uma oficina ensinando programação com portugol, mas, para quem nunca desenvolveu nada e não está adaptado à uma IDE, essa ferramenta é muito útil!
47	Presença de comandos em português; não ser necessário baixar nenhum programa.
48	Ter uma sintaxe em português, não ter complexidades desnecessárias, o que é ótimo para quem está começando.
49	Design, intenção
50	Interface, opções de customização (língua e tema)
51	Sintaxe acessível, portabilidade para várias línguas, erros amigáveis, interpretação rápida.
52	Gostei da ideia de ter uma ferramenta inicial para códigos simples, quase pseudo-códigos.
53	A facilidade de usabilidade (poder rodar direto do navegador, sem ser necessária nenhuma instalação) e os exemplos para guia

Continued on next page.

Continued from previous page.

Id	Response
54	Interface, facilidade de uso, exemplos.
55	Linguagem legal, só apagar os resultados da execução quando eu quero, e ser online (não precisar instalar nenhum programa, ou seja, poder programar em qualquer computador é um ponto legal).
56	Ela é simples e intuitiva.
57	Achei intuitiva, com exemplos claros, mas ainda tenho dificuldade em programar
58	Interface.
60	Achei interessante por poder tem algo em PT-BR. Eu achei que pode ser muito valioso para ensinar para crianças do ensino fundamental sobre programação.
61	mais fácil de aprender a programar do que usando um compilador tradicional
62	O potencial de inclusão de novas linguagens e do futuro desenvolvimento open-source dessa ferramenta. Desconheço o design empregado para a criação das gramáticas da linguagem, mas imagino que seja possível distribuir de maneira modular cada língua implementada.

Source: The Author

Q22. O que você mudaria na ferramenta PanScript?

Id	Response
2	Exemplos menores e quebrados em mais arquivos
3	Quem sabe as mensagens de erro podiam ser um pouco mais básicas, quem sabe até “kid friendly”. “Não pude encontrar o identificador de nome m no escopo atual:” podia se tornar algo como “Não sei o que ’m’ é, tem certeza que ele foi declarado?” ou algo assim. (quem sabe até checar “erros clássicos”, como tentar chamar variáveis de outro escopo?)
4	Pessoalmente eu acho que é estranho usar acento ao programar, mas eu entendo o motivo de estar assim
5	No escopo atual, acredito que está em ótima condição, portanto não mudaria nada (além de funcionalidades adicionais como listas)
6	Nada
7	pelo que testei, não houve tradução automática de funções basicas da linguagem, achei que se mudasse de ingles pra portugues a função usada “square_root” mudaria para “raiz_quadrada”, mantendo os demais nomes no sistema como estavam, mas apenas carregou o outro arquivo do sistema pré-feito
8	Acho que nada.
9	Por um lado entendo que tenha a intenção de parecer uma IDE, mas poderia ser mais fora da caixa.
10	Não usaria acentos nas funções por padrão, ex: potência() -> potencia()
11	Só design. Achei meio quadrado demais.
12	Não consegui pensar em nada.
13	Não tive muito tempo para explorar a ferramenta e não sei se já tem essa possibilidade mas acharia interessante adicionar em paralelo a linguagem orientada a objetos.
15	Creio que já é um excelente produto mínimo viável e não necessite de mudanças por ora. Mas seria interessante criar uma trilha de aprendizado que fosse aos poucos substituindo o português pelo inglês para que a familiaridade com o idioma aumente.

Continued on next page.

Continued from previous page.

Id	Response
16	Adicionaria um exemplo com erros comuns :) Além disso, como existe um mapeamento 1-1 das funcões em cada língua (foi o que me parece ao menos) um incremento bem legal pro futuro seria um parser que transforma de uma lingua pra outra. Aka tu escreve em portugues e pode enviar o código em ingles e vice-versa. Claro que ainda tem o problema do nome das variaives/conteúdo das strings, mas ainda assim parece interessante. Um tipo nativo "tupla" seria bem interessante também :) pra poder retornar multiplas coisas ao mesmo tempo (eu tentei e não consegui ao menos)
17	Tiraria os acentos para quem está aprendendo já se acostumar e autocomplete de comandos do comando que está sendo digitado
18	Acho que não mudaria nada.
19	Nada
20	Colocaria em algum lugar a linguagem por trás da ferramenta e talvez uma explicação mais explícita de objetivos e o que fazer, talvez com uma tela inicial.
21	Não usei a ferramenta o suficiente para encontrar pontos nos quais eu faria mudanças.
22	Adicionaria suporte a contas de usuário para poder salvar meus códigos e acessá-los de qualquer lugar. Talvez poder gerar códigos em outras linguagens (ex: Python) a partir de um código .pan.
23	Não mudaria e sim manteria. Apenas as linguagens português e inglês. O português para melhor entendimento de quem está começando a programar e não conhece o idioma. O inglês por ser um idioma universal e quase todas as ferramentas de programação são no idioma inglês.
24	Responsividade para diferentes resoluções
25	Adicionaria mais idiomas e exemplos mais amigáveis para crianças, talvez não relacionados à matemática
26	copia e cola usar variáveis em cálculos

Continued on next page.

Continued from previous page.

Id	Response
27	Acho que precisaria mais exemplos de documentação, de como criar uma função, de como fazer retorno de múltiplos valores, exemplos diferentes além de coisas simples. Há limitações, como por exemplo, só se pode declarar funções antes de começar o código por inteiro.
28	* Como é uma ferramenta direcionada para estudantes, minha maior dificuldade era não ter a aplicação do que estava aprendendo. Minha sugestão seria criar alguns exercícios de fixação finais com respostas já pré-definidas para que o estudante aplique o que ele está aprendendo. Exemplo: ferramenta de R do Code Academy (https://www.codecademy.com/learn/learn-r)
29	Estratégias modernas de ensino de programação utilizam a visualização e outras metodologias lúdicas.
30	Talvez mais metodos gráficos para facilitar o entendimento
31	Evitar usar acentos em tipos de variáveis
32	Tiraria as palavras reservadas com acento. Ex: número
33	Alteraria o local onde está o botão de temas, já que ele altera o tema de onde você escreve o código, desta forma, estão em lados opostos.
34	Organização e tamanho dos arquivos de exemplo
35	Eu gostaria da sintaxis ser em inglês mesmo, por que até são poucas palavras reservadas por linguaguem. O que se seria um diferencial seria ter a documentação ao dar e.g. ctrl + enter (no idioma que a pessoa escolheu) isso para aumentar a produtividade do programador.
36	Melhorias na interface, possibilidade de instalar localmente para uso offline
37	As mensagens de erro eu destacaria em outras cores. Eu evitaria usar acentuação gráfica na sintaxe das palavras-
38	Facilidade para entendimento de lógica

Continued on next page.

Continued from previous page.

Id	Response
39	Alguns nomes ficaram compridos, por exemplo “verdadeiro”. Não sei se mudaria, mas talvez valha analisar se tem opções mais curtas. Também não sei se isso será um problema para quem está recém aprendendo. Permitir várias declarações de variáveis com o mesmo tipo. Ex: “número a = 2, b = 3”
40	Uma sugestão seria não utilizar acentuação em elementos da linguagem, como por exemplo em “lógico” ou “número”. Entendo que faz parte do idioma que escolhi (Português) e pode ser apenas um viés meu devido a experiência com outras linguagens, mas é algo que poderia ser avaliado com mais pessoas.
41	a regra `variable_attribution: TOK_TYPE TOK_NAME ‘=‘ TOK_LITERAL` em português, pra mim, soa estranho ‘`texto algo = "valor"`‘ Também acho que pode causar um pouco de frustração para os alunos iniciantes – “pq tenho que dizer que é texto? pq o computador não sabe que é texto se estou dando um texto como valor” – já vi isso em aula. Talvez eu utilizaria algo go-like (‘`variavel := literal`‘ , ou explicitamente ‘`variavel : tipo = literal`‘) Algo que poderia ser avaliado em uma pesquisa com alunos aprendendo programação, qual seria mais simples ;)
42	1. gostaria de uma documentação mais textual mesmo. Enquanto fazia o exercício da bhaskara achei chato ter que procurar as funções que precisava (raiz_quadrada etc) através de exemplos. Os exemplos são fundamentais e importantes, não estou sugerindo tirá-los ok 2. eu particularmente gosto de ter acesso às ferramentas por meios que não sejam o navegador. Mas acho que nesse contexto de ensino de gente mais nova o que faz mais sentido é ter acesso em um navegador mesmo, aí não precisa instalar nada nos PCs das escolas e tal.
43	Achei genial a ideia, nunca tinha visto nada do tipo. Com certeza vai ajudar quem está no processo de aprendizagem e não domina o ingles.

Continued on next page.

Continued from previous page.

Id	Response
44	Acho que uma sistema de busca, para achar as funções, e mensagens de erro mais didáticas. Por exemplo, coloque na mensagem de erro coisas como “A variável tal não foi achada, você lembrou de definir ela? o nome dela é o mesmo em todas as linhas?” “Algo de errado não está certo, verifique a sintaxe do seu texto”. Algo que ensine as pessoas a como fazer o debug.
45	Eu talvez colocaria algum mecanismo para clicar na saída do “console” e apontar qual linha do código fonte gerou tal saída. Além disso, não encontrei nenhum tipo de função para fazer leitura pelo terminal.
46	No meu computador a tela inteira ocupou mais espaço pros lados, o que fez com que, toda vez que queria ver a saída inteira, eu tivesse que mover pro lado (seria legal, talvez, tornar as caixas de escrita e saída redimensionáveis). A caixa da saída corta a linha (ás vezes, no meio das palavras) e, entendo que fica melhor de vizualizar, mas pode ser que a pessoa ache que tem algo pulando uma linha. Por fim, acredito que seja um vício meu já, mas, se tivesse uma opção para escrever os comandos sem acento ou cedilha, eu usaria haha
47	Além dos arquivos de exemplo, incluiria uma página de documentação com mais detalhamento sobre os aspectos da linguagem, a medida que o estudante evoluísse isso se tornaria necessário.
48	não colocaria acentos nas palavras reservadas, talvez algumas seriam abreviações ao invés da palavra inteira
49	Linguagem de programação é meio péssima porque você não sabe direito qual vai ser, explicitem que é para ensinar lógica de programação e não a programar.
50	Remover acentos do código (número -> num ou algum outro nome sem acento)
51	Inserção de alguma funcionalidade de entrada de dados por parte do usuário, para possibilitar pequenos programas interativos
52	Fiquei com dúvida em relação à acentuação, entendo que o objetivo seja utilizar a língua portuguesa, mas será que é necessário mesmo? Ex: número. T
53	Nada me vem a cabeça no momento
54	Talvez incluir algum sistema de ajuda, um fórum de debates ou de dúvidas.

Continued on next page.

Continued from previous page.

Id	Response
55	Deixar o usuário mexer em quanto da tela é ocupada pelo terminal e também deixar recolher o navegador de arquivos. O terminal também poderia seguir a cor do tema. Não encontrei uma forma de criar um novo arquivo para conseguir editar dois ao mesmo tempo, se ainda não for possível, acharia interessante fazer.
56	Poder armazenar os resultados de operações matemáticas diretamente em variáveis.
57	Esta muito bom o design, manteria assim.
58	O botão “executar” deveria ser mais chamativo (destacá-lo mais).
60	Por enquanto, não mudaria nada. Pela pesquisa e como ela foi conduzida, tudo fluiu super bem.
61	-
62	Eu alteraria o atual syntax highlighting para utilizar tree-sitter (ficando, assim, mais bonito).

Source: The Author

Q23. Descreva quaisquer problemas que você tenha encontrado usando o PanScript:

Id	Response
2	Nenhum problema
3	Nenhum!
4	
5	A limitação da minha inteligência
6	Não tive
7	O site podia se limitar ao tamanho da tela, acho que poderia deixar como opção minimizar a parte dos exemplos, porque a tela ficou “maior” que a minha e apareceu um scroll horizontal.
8	Às vezes o idioma do código de exemplo não muda quando eu troco no dropdown, daí fica invertido: aparece selecionado Português, mas o idioma do código está em inglês. No meu computador (notebook) a interface está um pouco mais larga que a largura da janela, daí aparece uma barra de rolagem horizontal. (Estou usando Google Chrome 93.0.4577.82 no Ubuntu 20.04.3) No exemplo do condicional “Se”, ele não faz nada se o resultado for 3. Não sei se era essa a intenção, mas se for, acho que deveria ter um comentário explicando.
9	Nenhum
10	Ao abrir um exemplo, pode-se utilizar o “desfazer” e apagar o exemplo
11	Ao executar a função desenvolvida no exercício anterior demorou um pouco para haver retorno. No caso eu usei “potência(x)”, invés de “potência(x,n)’, talvez isso tenha causado isso. Além disso, acho esquecer do “n” na função deveria subir um erro, invês de mostrar “NaN”.

12	No exemplo de constantes o código não roda. Aparece a seguinte mensagem:
	`Erro na linha 19:`
	`Não posso alterar o valor de nome porque foi declarado como "constante":`
	`nome = "Pedro" // comente esta linha para corrigir o erro`
	`^^^^`

13	
15	Nenhum problema.

Continued on next page.

Continued from previous page.

Id	Response
16	Por algum motivo quando eu foi fazer ‘`escreva("x1: ", para_texto(x1))`‘ no meu código eu usei uma vírgula ao invés da concatenação (+) (tentei fazer como faria mais “naturalmente”). Isso não gerou nenhum erro, só printou “x1: ” na tela :) depois percebi que tinha que trocar a vírgula por um +. não entendi qual a utilidade do segundo parametro do escreva (ou terceiro etc)
17	Falta de habituação com linguagem mais fortemente tipada (estou acostumado com Python)
18	Não encontrei nenhum problema
19	
20	Tive a impressão que algumas vezes o texto do código não se adequava à linguagem selecionada (os comentários iniciais apareciam em inglês estando a linguagem em português). Não sei dizer se é algum bugzinho, mas aconteceu comigo clicando aleatoriamente nos arquivos e na linguagem.
21	Não encontrei problemas.
22	Nenhum
23	Ao realizar testes, o erro que retorna não corresponde a linha do erro. Ex: alterada a linha 11, o erro apresentado mostra a linha 20.
24	
25	
26	
27	Problema ao ver exemplos mais complexos, como funções funcionam, tiver que ver a parte 6 e não teve essa instrução.
28	Não encontrei nenhum
29	Nenhum.
30	
31	
32	
33	
34	

Continued on next page.

Continued from previous page.

Id	Response
35	escrevi “numero” ao invés de “número”, então o tipos de dados são muito dependentes da gramatica. escrevi “escrever” ao invés de “escreva” mas para mim faz sentido escrever ao invés de escreva.
36	Não funcionou no primeiro navegador onde tentei (Safari), acabei usando no Chrome
37	não tive problemas
38	
39	Acento. Primeira vez escrevi “numero”. Possivelmente por costume de programar em inglês.
40	
41	
42	Senti falta de uma documentação mais textual, tipo como tem na documentação de python.
43	
44	Eu não achei a sintaxe para multiplicação e divisão, eu fiz por que imaginei que fosse o padrão.
45	A versão em português da ferramenta acaba tendo uns nomes mais longos. Além disso, trocar de idioma enquanto escreve código faz PanScript apagar o conteúdo do arquivo “Principal” sem ao menos avisar o usuário.
46	Não entendi só porque quando clico em “Desfazer”, estando em algum dos exemplos, ele apaga tudo, sendo que não foi uma alteração feita por mim. Não que seja ruim desse jeito, mas achei que essa ação fosse apenas para mudanças feitas pelo programador.
47	O código não fica salvo online quando o editamos, é preciso salvá-lo na nossa máquina, caso contrário, ele será perdido quando executarmos um outro código.
48	
49	
50	Nenhum
51	Nenhum problema

Continued on next page.

Continued from previous page.

Id	Response
52	
53	
54	nenhum
55	O layout ficou muito grande para a minha tela, não dava para ver que o botão vermelho era para limpar sem rolar a barra horizontal.
56	Não sei se porque a recomendação do exercício era para utilizar apenas o “escreva” e o “raiz quadrada”, mas foi frustrante ter que escrever tantas linhas de código para chegar a uma resposta de um problema tão simples, como de resolução de equação de segundo grau.
57	Não consegui calcular a formula de bhascara, mas devo ter errado em algum momento
58	
60	Nenhum problema encontrado.
61	-
62	Nenhum.

Source: The Author

Q24. Você sentiu falta de alguma funcionalidade na ferramenta? Qual(is)?

Id	Response
2	Não
3	Tentei declarar variáveis na mesma linha e não consegui: `número a = 3, b = 5, c = 4`
4	
5	Uso de listas
6	Para aprendizado, referências visuais são importantes. A possibilidade de desenhar ou de ver coisas mais parecidas com o que entendemos como um software é um estímulo importante.
7	
8	Entrada de texto; Estruturas de dados: listas, vetores.
9	Quick search
10	
11	Não senti falta.
12	Talvez uma explicação da parte conceitual junto dos exemplos. Eu já sei programar, então as coisas fazem sentido só de bater o olho, mas talvez alguém que nunca tenha contato com programação encontre dificuldade. Por exemplo, por que para imprimir uma variável do tipo número com um texto eu preciso utilizar a função “para_texto”, ou por que texto ficam entre “ ” e números não.
13	
15	Não
16	Tuplas !
17	Para aprendizado inicial está excelente, pensando em algo além do inicial talvez fosse interessante a possibilidade de importar funções que o próprio usuário crie em outros arquivos para que o aluno seja capaz de aprender questões de organização de código também. Pensando na possibilidade de fazer contas, creio que seja interessante a opção de manipulação de arrays e matrizes
18	
19	
20	Não, mas acredito que o público alvo possa dizer melhor.

Continued on next page.

Continued from previous page.

Id	Response
21	Acredito que seria muito válido acrescentar uma funcionalidade de depuração. Aprender a usar um depurador é elemental para quem está começando.
22	Autocomplete
23	Não
24	
25	
26	
27	Acho que se tivesse uma wiki além dos códigos exemplos, seria uma ótima opção. Acho que a linguagem tem tudo para expandir.
28	Exercícios de fixação.
29	operador ternário de comparação `x=(x==1?2:1)`
30	
31	
32	
33	
34	

35	`texto está_estudando = verdadeiro`
	`Erro na linha 20:`
	`Esperava que o tipo de verdadeiro fosse texto, mas era lógico:`
	`texto está_estudando = verdadeiro`
	`----------------------^^^^^^^^^^`

36	Funcionalidades encontradas em editores de código (indentação automática, fechamento automático de parênteses, sugestões ao digitar, etc)
37	Debug. Execução passo a passo, linha a linha. Pra quem está aprendendo é muito útil também.
38	Ensino a lógica com conceitos visuais
39	Permitir várias declarações de variáveis com o mesmo tipo. Ex: “`número a = 2, b = 3`”
40	

Continued on next page.

Continued from previous page.

Id	Response
41	Para introdução aos conceitos de programação, não senti falta de nada. Para avançar em alguns tópicos, daí senti falta de tipos estruturados e vetores
42	função que cálcula as soluções de polinômio de segundo grau. Brincs
43	
44	
45	É um recurso valioso para quem está iniciando em programação, mas poderia incluir alguns recursos como “autocompletar” com uma ajuda embutida em pop-up, por exemplo. Ainda assim, da minha experiência, esse tipo de recurso mais prejudica que ajuda o aprendizado de iniciantes de programação, então a abordagem com exemplos é mais adequada.
46	Uma coisa que senti foi que, ao fazer upload de um arquivo do meu computador, ele modificou o programa do exemplo onde eu estava, talvez fosse interessante, nesses casos, criar um novo arquivo em “Seus arquivos”.
47	
48	
49	
50	Interação com “usuário” (e.g. text input)
51	- Inserção de alguma funcionalidade de entrada de dados por parte do usuário, para possibilitar pequenos programas interativos - Alguma função inspirada em linguagens funcionais, para servir como um “extra” no aprendizado para aqueles que quisessem algo mais avançado
52	
53	Poder “apagar” o histórico do console (as vezes pode confundir, principalmente quando troca de script)
54	manipular arquivos. Tipos definidos pelo usuário, como enumerações e estruturas (ao menos).
55	

Continued on next page.

Continued from previous page.

Id	Response
56	Bem como disse, foi orientado a utilizar apenas as funções “escreva” e “ raiz quadrada”, mas o que mais me incomodou foi não poder alocar os resultados das operações diretamente em uma variável. Ademais, quando eu tentava inserir mais operações matemática na função escreva gerava um erro, tive que escrever em duas linhas uma função simples. O Python é uma linguagem tão simples quanto a proposta, e este permite uma otimização destes elementos em IDES tão intuitivos quanto esta, a única diferença é estar em português, mas também já há IDES que aceitam a programação em português.
57	
58	
60	Talvez um autocomplete seria interessante, ou pelo menos limitar autocomplete para recursos como funções e tipos nativos. Reescrever os nomes de variáveis e funções pode ser um bom exercício para o aluno entender que dar bons nomes é relevante durante o desenvolvimento de programas.
61	talvez seria bom ver o que os metodos esperam (tipo um control P pra ver os parametros)
62	Eu adicionaria uma interface de linha de comando para a linguagem (ou a tornaria explícita).

Source: The Author

Q25. Este espaço é livre para você fazer comentários adicionais:

Id	Response
2	
3	Excelente trabalho, parabéns!!!!!!!!!!!!!!!!!!!!!!!!!!!
4	
5	Eu achei muito TOP!
6	
7	Muito legal! Parabéns!
8	
9	Já quero saber como seriam classes.
10	Bom trabalho :)
11	Parabéns pelo projeto.
12	
13	
15	
16	Parabéns pelo trabalho Daniel, ficou muito legal :)
17	Está realmente muito bom. Eu consegui navegar bem e entender o que precisava. Porém talvez a minha crítica esteja enviesada por já saber programar e estar habituado a linguagens diferentes e gostar de fuçar as coisas. Por conta disso, aconselharia a testar a ferramenta com alunos que precisam aprender programação, mas que não são de cursos relativos. Testar em alunos de graduação de Física, Matemática, Ensino Médio e Fundamental, (quem sabe até Letras rs) etc.
18	Achei a ferramenta muito boa para iniciantes em programação, permitindo desenvolver a lógica que é o mais importante. Depois é só traduzir os comando para a sintaxe de qualquer linguagem de programação que se quer aprender.
19	Não entendi se ao escrever na lingua materna, o compilador conseguiria traduzir... Mas a questão é que o aprendizado de uma linguagem de programação é voltado a resolver problemas em um domínio específico. Não vejo com bons olhos uma linguagem que seria apenas conceitual e não adequada para usar em produção.
20	

Continued on next page.

Continued from previous page.

Id	Response
21	Embora a proposta principal da ferramenta seja propiciar o aprendizado de não falantes de inglês, acredito que o público-alvo da ferramenta poderia ser estendido para crianças ou adolescentes que possam ter interesse em programar, mas se sentem impedidos pela dificuldade de instalar programas ou outros motivos. Ter contato com programação desde cedo certamente é crucial para desenvolver as habilidades de desenvolvedor para o resto da vida, além de que geralmente este tipo de estímulo promove um aumento no desempenho escolar. Fica a dica.
22	
23	
24	
25	
26	
27	Eu queria que nós trocássemos uma ideia, pois achei muito legal o projeto e um TCC útil e com ação educacional.
28	Ótimo trabalho, Dani! Espero que teu TCC bombe demais!
29	Apenas para apontar que existem linguagens que não tem idiomas em sua sintaxe: J (<https://www.jsoftware.com/>), retina (<https://github.com/m-ender/retina/wiki/The-Language>), Jelly (<https://github.com/cairdcoinheringaahing/jellylanguage>)
30	
31	
32	
33	
34	
35	Parabéns pelo seu trabalho.
36	Parabéns à(s) pessoa(s) que desenvolveram a ferramenta.

Continued on next page.

Continued from previous page.

Id	Response
37	fico me perguntando porque eu apresentaria para os meus alunos mais uma ferramenta. Não tenho uma resposta clara. Já experimentei dar aula com portugol, e a reclamação mais recorrente (e também a mais cruel) é que foi um tempo inútil. Argumentam que apanharam igual pra compreender a lógica, mas diziam que após todo esse "sofrimento" não poderiam usar a linguagem útil. Bom seria se o sistema contasse com uma traduçaõ direta do código para outras linguagens mais badaladas.
38	Embora exista melhorias a ferramenta é uma ótima iniciativa e já quero compartilhar para todos que podem aprender
39	Sucesso! :)
40	Parabéns pelo trabalho! Proposta bem interessante e foi interessante utilizar a ferramenta desenvolvida.
41	
42	Gostei bastante e achei potencialmente útil!
43	
44	
45	
46	Amei o TCC e tenho interesse de usar futuramente em oficinas para estudantes do ensino médio e básico, se for possível! Parabéns pelo trabalho!
47	
48	
49	
50	
51	Gostei muito do projeto, acredito que poderia ser largamente utilizado no ensino de programação básica, inclusive introduzido no ensino público para fomentar mais profissionais nessa área.
52	Eu realmente achei interessante a ferramenta. Eu vou começar a lecionar a disciplina de programação para adolescentes e talvez esse seja um ótimo público alvo do PanScript. Parabéns pela pesquisa!

Continued on next page.

Continued from previous page.

Id	Response
53	Achei a iniciativa muito boa e que pode auxiliar muitos estudantes no início da vida de programação! Parabéns! :)
54	ótima iniciativa.
55	
56	
57	
58	
60	Gostei muito. Achei uma boa ferramenta de aprendizado inicial.
61	bacana, gostei da ideia, acho que pode ajudar estudantes aprendendo pois permite abstrair a parte do compilador, .c , .h, etc
62	

Source: The Author