ABSTRACT

Title of Thesis:

DESIGN FOR JUST-IN-TIME: RESOURCE DESIGN FOR SELF-TEACHING COMPUTER SCIENCE AND ONLINE LEARNING

Carrie Lindeman, Human-Computer Interaction

Master of Science 2020

Thesis Directed By: Assistant Professor, Dr. David Weintrop, College of Information Studies and College of Education

The goal of this study is to investigate how just-in-time resources may support self-teaching for adult computer science learners who are new to coding. For people learning computer science on their own, just-in-time resources can be essential for solving problems. A popular online resource that computer scientists of all experience levels rely on is Stack Overflow, a forum that has a question and answer format. Resources like Stack Overflow can help new programmers problem-solve their code without consulting a teacher or professor. However, these resources may be creating

barriers in the learning experience that should prepare them for future computer science education. By observing learners using just-in-time resources and interviewing learners about their habits, this thesis provides guidance on potential design suggestions for better supporting users' future learning. Understanding how just-in-time materials currently support self-teaching for adult novice computer science learners will provide the foundation for these designs.

RESOURCE DESIGN FOR SELF-TEACHING COMPUTER SCIENCE

by

Carrie Lucille Lindeman

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Human-Computer
Interaction
2020

Advisory Committee:
Professor David Weintrop, Chair
Professor Mega M. Subramaniam
Professor Margaret Chmiel

Acknowledgements ii

I gratefully acknowledge the support of my advisor, David Weintrop. This study was dependent on his guidance and encouragement. I would also like to thank the participants of the study for their generosity in their time and experience. I have great appreciation for the contributions of my thesis committee, Mega Subramaniam and Margaret Chmiel. Lastly, thank you to my fellow Master's Thesis students, Kausalya Ganesh and Shannon Fitzgerald who provided comradery and additional support through the process of conducting this study.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction: Context of the Study of Self-Teaching in Computer Science

Introduction

The materials most often used by adult novice coders are not designed in a way that prepares them for computer science instruction. Adults who are new to programming may be consulting resources that include very little information about the foundations of the concepts they are using in their code, or equally as detrimental, bury the key contextual information in areas of the resource that are not utilized by this novice population.

The technology field is expected to grow in the next 8 to 10 years, requiring a workforce with diverse areas of technological expertise. The Bureau of Labor statistics found that "[e]mployment of computer and information technology occupations is projected to grow 12 percent from 2018 to 2028" which is likely to increase the number of adults becoming new coders (Bureau of Labor Statistics, 2019).

In terms of online education, the field of computer science experienced a 13% growth in online academic presence in 2018 according to the most recent Online Education Trends Report from BestColleges. (*2020 Online Education Trends Report*, 2020) This growth is expected to continue and is representative of how many individuals turn to online education in the field of computer science. However, the first materials used by novices may not be preparing them for the continued formal education they are seeking through such institutions.

Many of the first resources they encounter would be categorized as just-in-time learning resources. A just-in-time learning resource is one that provides the required information immediately when it is needed. An example of a just-in-time learning resource could be an online video that demonstrates how to fix a sink that can be accessed immediately after the problem is discovered. In a coding context, many forums and brief tutorials are used as just-in-time learning resources for novice coders who are debugging their early programs.

An important distinction to note for this study, is that I am using the phrases "computer science" and "coding" interchangeably. In the field of computer science, coding and computer science are determined to be two different subjects. Understandably, coding is seen as a component of executing the theories learned in computer science. However, since this study is focused on novice learners, I am using this phrase interchangeably. To a novice coder (specifically one who is self-teaching) the difference between coding and computer science is difficult to comprehend. Since most self-teaching novices are only introduced to the broader ideas of computer science through coding. However, I acknowledge that there is a greater distinction between the disciplines than is noted throughout the study by myself or the participants.

Through gaining further understanding of how novice coders use these resources, we can discover directed design changes to improve the learner experience. Much of the existing work in this area focuses on tutorial style online resources, as opposed to just-in-time learning resources.

Positionality Statement

In this section I present my own history with the content under investigation so as to make clear the lens I bring to this work and situate myself with respect to the work being done. My undergraduate education included numerous computer science courses and as a student I became aware of how heavily I relied on just-in-time computer science resources. The reliance on such materials was not as prevalent in my courses for other disciplines in my experience. My awareness of the importance of these materials increased when I served as a teaching assistant for introductory computer science courses. I observed how students leveraged these resources and the role these materials played in preparing students for further computer science education. My experiences as a computer science teaching assistant give me the insight to create a study about the just-in-time materials, but also introduces an element of potential bias. I may be more inclined to think students need formal computer science education since I used to be a part of formal teaching. Additionally, I have worked as an instructional designer for over 5 years for commercial businesses and higher education institutions. My experiences as an instructional designer give me insight into the processes used by the resource designers. However, this background could also bias me in the way I approach creating materials for learning.

Research Questions

My study addresses three key research questions:

1. What are the content differences between just-in-time coding resources and traditional computer science curriculum?
2. How do existing just-in-time learning solutions support self-teaching, adult computer science learners who are new to coding?
3. What are some interface design changes for just-in-time learning solutions that could better support the novice learner experience?

The purpose of Question 1, is to seek to understand the curriculum gap that may be present between traditional computer science and just-in-time coding solutions. In order to explore Question 1, I conducted a comparative analysis of content. I catalogued the content of one, just-in-time resource page and compared it to an aggregate list of topics from various university curricula. The findings from Question 1 were used to inform the designs I create later in the experiment which are dependent on existing just-in-time coding solutions.

The purpose of Question 2, is to seek to gain insight into how this population uses the existing resources leading to an understanding of their intentions and behaviors. I was also able to observe what components of the resource the users do and do not interact with, as well as what parts of the resources they leverage most effectively when self-teaching. The method chosen to collect this information was semi-structured interviews and observations of just-in-time resources in use through a coding exercise. The interview and exercise were recorded and categorized into themes. The outcomes of their coding exercises will also depend on how different the resources are that they choose to use during the exercise.

The purpose of Question 3 is to determine interface design decisions that can be made on a global, site-wide scale or by individual resource contributors that better support novice self-teachers. I conducted a second experiment to user-test the design. These design changes will be analyzed and potentially suggested with the goal of supporting learners in their self-teaching experience.

Outline

This thesis begins with a literature review found in **Chapter 1** that outlines some of the gaps in existing studies for adults novice coders.

Following the literature review, **Chapter 2** outlines the methods and findings of the content comparison between the curated materials by university professors and a Stack Overflow page. This work is a preliminary foundation for the second half of the study.

Chapter 3 begins this second part of the study by outlining the methods for the participant recruitment, the interviews, and the coding exercises conducted through the subsequent phases.

To answer the second research question, I observed adult novices using existing resources (phase 1). **Chapter 4** presents the themes and findings from phase 1.

In **Chapter 5** the resulting themes were used to inform a new design of an online resource (phase 2). This chapter features the process of ideation and the rationale for the final design.

To answer the third research question, a second set of participants was recruited to go through the same coding activity as the initial participants but were instead asked to use the new material (phase 3). **Chapter 6** explores the findings from the interviews and coding activities that were conducted in phase 3.

Lastly, **Chapter 7** concludes the study by delving into the potential for future research, the limitations of this thesis design, and the implications for the findings from this study.

Literature Review

This study spans the fields of online learning, self-teaching, and computer science education. Each of these subjects is a component of this study, culminating in a series of interface design recommendations. The intersection of the andragogical fields and interface design of educational materials is the foundation for this work.

Challenges in Learning Computer Science

An interesting study completed by researcher, Philip Guo focused on the population of adult learners who are largely outside of the workforce. He studied the motivations, behaviors, and frustrations of 504 survey-respondents with an average age of 65 years. Guo found that the motivations of the respondents were split into three categories: age related pressure, personal enrichment, and job-related growth (Guo, 2017). Guo suggested that the motivations of the respondents may or may not differ with new coders of differing age groups. Additionally, Guo did not make any conclusions about the participants' motivations being related to their self-teaching methods.

Another relevant aspect of Guo's study is the survey of learner frustrations. The top 5 non-age related frustrations with learning were bad pedagogy, syntax errors, software installation and configuration problems, programming language-specific features, and run-time errors (Guo, 2017). Guo postulated that these frustrations may be consistently troublesome for a wider age population as well.

Who is Learning Computer Science?

Adults learning computer science is an area of study that is of particular interest as working professionals pivot into technical careers. The Bureau of Labor statistics found that "[e]mployment of computer and information technology occupations is projected to grow 12 percent from 2018 to 2028" which will result in more individuals who are currently in the workforce, learning coding and computer science skills (Bureau of Labor Statistics, 2019). For a significant portion of adults learning computer science, they primarily rely on online platforms. In a post by Quincy Larson, the founder of freeCodeCamp.org, he presented the findings of a survey the company conducted targeted at new coders (Larson, 2017). Of the 31,000 respondents surveyed, 36.8% of the new coders had received a bachelor's degree. From this same survey, they found that the top three online resources used by these new coders are FreeCodeCamp, Stack Overflow, and Code Academy (Larson, 2017). The distribution of resources used by new coders may differ when data is collected by a source that is not also a coding resource, and by having participants generate the list of resources, instead of selecting from a list.

Effectiveness of Online Learning

Impactful work is being done to assure the effectiveness in online learning across all disciplines. A study by Antonis, Daradoumis, Papadakis, and Simos (Antonis et al.,

2011) evaluated the design of online computer science courses for adults. Their study focused on andragogical design of the materials and assessed the instructional design by learner performance, learner support, learner satisfaction. The evaluation in the study was based on a framework developed by Benigno and Trentin (2008) that bridged together the “learning process and the participant performance” (p. 259). The framework covered “five dimensions: participative, social, interactive,cognitive and metacognitive” with each dimension connected to an analytical model (Benigno & Trentin, 2008, p. 261). This framework was used by Antonis et al. (2011) to design their methods of evaluating student behaviors and performance in online courses.

When evaluating student struggles during the first half of their courses, Antonis et al. (2011) found that learners mostly dealt with, “technical problems with the distance-learning environment (missing passwords, losing connection), lack of administrative support, and the high scientific level of the courses” (p. 377). By the end of the course, learners described their main difficulties as a “lack of time” to work on the material of the course (Antonis et al., 2011, p. 377). In terms of learner performance, they measured through self-assessments. By the midterm of the courses, “53% of the learners judged that their performance was satisfactory” and by the end of the courses, “75% were positive about their overall performance during the courses” (Antonis et al., 2011, p. 377). However, of the initial number of students who enrolled in these courses, 48.16% of them dropped out of the courses over the semester (Antonis et al., 2011). So the participants who would have potentially self-assessed their performance as poor did not participate in the last survey.

Using Just-In-Time Resources

These methods and findings are very telling of the student experience with linear online learning. However, many novice coders will turn to just-in-time learning materials while self-teaching coding. A post from Ron Darby (2018) at Edge Learning Media defined just-in-time learning as, “a behavioural trait that has evolved through advances in technology and the Internet. Above all, it is a predisposition toward accessing the required knowledge when we need it” (para. 3). The behaviors and measurements of just-in-time learning materials may differ from those evaluating linear online learning experiences.

There have been numerous studies on how novice and expert coders use popular online just-in-time computer science resources. A study published in the *Journal of Systems and Software* by Chatterjee, Kong, and Pollock (2020) specifically surveyed novice software engineers about how they use the popular computer science forum, Stack Overflow. An important insight related to this study was that “novice programmers focus on 15–21% text and 27% code in a Stack Overflow post” (Chatterjee et al., 2020, p. 1). These researchers also narrowed down a pool of 400 Stack Overflow posts and annotated each component based on a framework they designed. The researchers then compared the components of the posts that novice software engineers interacted with and the prevalence of those components in the pool of posts. This study provided helpful context for a standard way to label the anatomy of a Stack Overflow page. Chatterjee et al.'s (2020) study was exploratory, providing a cursory window into the behaviors of novice computer science students using online resources.

Balancing Numerous Resources

The focus Chatterjee et al.'s study is specifically on resources used by individuals who are self-teaching. In the field of computer science there are vast numbers of resources available online. A study in interface design from Glassman and Russell at Google (Glassman & Russell, 2016) explored a design that helps synthesize the process of self-teaching from multiple sources. These researchers built a tool called DocMatrix which allowed users to view multiple documents "simultaneously in parallel" through a grid structure. A key limitation of the tool in its state at the time of the publication was that it only worked with "structured documents" that were accessible through Google Books API. They assessed their tool for the quality of the users' reading and ability to synthesize, as well as the amount they were able to read. The key finding from their work was that "[s]ubjects believed that with DocMatrix, they could more easily assess the usefulness of a book and synthesize information across multiple sources." (Glassman & Russell, 2016). This work explored the challenges of self-teaching from numerous online resources. The conclusions made about the tools interface advantages are not directly applicable to the challenges faced by self-teachers studying computer science. The interactivity of the online resources used by computer science students is not specifically accounted for.

Some areas of further study in the intersection of just-in-time learning resources and self-teaching computer science include further participant survey and interview, design solutions for the themes of frustration discovered, and personalization for self-teachers with different motivations for learning computer science.

Chapter 2: Content Comparison of Computer Science Curriculum and Online Resources

In order to understand the content differences between just-in-time coding solutions and formal computer science curricula, I completed a content comparison analysis between these types of educational materials. The purpose of comparing the online resources and computer science curricula was to determine how disparate these two methods are in their substance and content. In regards to my final research question, this comparison sets a baseline for how much of our analysis of design success could be affected by incongruent content between the two types of resources.

Methods

Collecting Materials

By sending emails through the Association for Computing Machinery's Special Interest Group on Computer Science Education (SIGCSE) list-serv and directly to university computer science professors, I was able to compile teaching resources from 10 professors that they use when teaching introductory computer science. In order to narrow the scope of the project, I chose to focus on one introductory topic, teaching functions. A total of ten professors provided materials on this topic. The materials included slide decks, videos, online classes, and more. I completed an in-depth review of all of the materials and catalogued the sub-topics covered in the materials. I read through or watched each material. I recorded each time a unique topic appeared in the materials. With an aggregated list of topics, I revisited each material and recorded if the topic was explicitly covered. If a topic was reiterated in the same material, the repetition was not recorded. In order for a topic to be explicitly covered, it had to be a part of the objectives of the material, involved in the activities prompted by the material, or the focused subject of a slide or section of the material. This eliminated topics that were only mentioned, since they were not deeply explored or were based on previous knowledge of the audience.

Selecting Resource for Comparison

From the existing studies on this population, I was aware that Stack Overflow is one of the most popular resources used by novice coders. This resource is also an interesting subject for the study because the content is not professionally curated. The site is a forum that has a voting system to promote preferred answers to coding questions.

In order to select a specific page from Stack Overflow, I searched "Python functions Stack Overflow" in the Google search engine. The first result is a question from the Stack Overflow forum that is titled “Basic explanation of python functions”. The question was asked on September 5th, 2015 and was answered by 3 unique users.

Findings

Research Question: What are the content differences between just-in-time coding resources and traditional computer science curriculum?

The materials sourced from the professors were distilled to a list of their key topics. The topics and their descriptions can be found in Table 1. I recorded how many times each discrete topic appeared in the aggregated 10 materials from professors and that total can be found in the third column of Table 1.

Table 1: Material Topics

Topic	Description	Count
Function Calls	Materials explain what it means to call a function. Materials demonstrate the syntax of a function call.	8
Parameter Passing	Materials explain what parameters are. Materials demonstrate how parameters define what values can be passed into a function.	8
What are functions?	Materials explain what functions are and how they work.	7
Function Syntax	Materials demonstrate function syntax.	7
Why divide code?	Material prompted students to consider why it may be useful to divide code.	6
Scope	Materials describe the concept of scope as it relates to functions.	4
Examples of Familiar Functions	Materials noted that students had already worked with some familiar, built-in functions such as print() and input().	3
Multiple Parameters	Material demonstrated a function that has multiple parameters.	3
Abstraction	Materials explained how functions can be used to introduce layers of abstraction in a program.	3
Returns	Materials demonstrated functions that had return statements.	3
Program Modularity Benefit	Materials described the benefits of having modular programs. Modular programs are ones that separate discrete tasks into independent modules so that they can be used individually and best strategically tested or patched.	2

Code Trace	Materials demonstrated a code trace/stack trace. They went line by line of a program and noted how the variables changed.	2
Functions Calling Other Functions	Materials demonstrated a function calling another function.	1
Documentation	Materials referenced or provided the language documentation for functions.	1

I used the topics that were aggregated from the formal computer science education materials and determined how often each one was mentioned in the entire thread of the Stack Overflow question.

The comparison between the computer science materials that were sourced from professors and the answers of the Stack Overflow forum are presented in Table 2.

Table 2: Content Comparison Topics Matrix

Topic	**CS Materials Count**	**Answer 1 Count**	**Answer 2 Count**	**Answer 3 Count**	**Total Stack Overflow Count**
Function Calls	8	2	2	1	**5**
Parameter Passing	8	1	3	1	**4**
What are functions?	7		1		**1**
Function Syntax	7	2	1		**3**
Why divide code?	6				**0**
Scope	4				**0**

Examples of Familiar Functions	3	1	1		**2**
Multiple Parameters	3	1	1		**2**
Abstraction	3				**0**
Returns	3				**0**
Program Modularity Benefit	2	1			**1**
Code Trace	2				**0**
Functions Calling Other Functions	1				**0**
Documentation	1	1			**1**

The topics of Function Calls, Passing Parameters, and Function Syntax were the most covered topics in the Stack Overflow forum. These topics were in the top five topics covered by the computer science materials. The most foundational aspects of teaching a function were covered by both materials. However, the computer science resources from professors more consistently covered the motivation for why a coder would want to use functions. This is evident through their inclusion of the topics “Why divide code?” and “Program Modularity Benefit”. Another disparity between the two types of materials is the lack of context provided in the Stack Overflow material versus the formal materials. There was no mention of scope in regards to functions and no explanation of abstraction in terms of functions. This missing expository information could make it difficult for a novice coder who is trying to recreate or adjust the code provided in the resource’s code samples.

It is important to note that this comparison does not consider the breadth or depth of the materials. This method only measures the presence of the topic. No claims can be made about the difference in effectiveness toward student learning of either computer science materials or the Stack Overflow forum.

The differences between the existing curriculum of introductory computer science courses and this singular Stack Overflow resource must be accounted for in the subsequent experiments. The subject areas that are absent in this Stack Overflow resource can impact the experience of the second round of the later code activity.

Summary

The aggregation of educational materials from professors on the topic of functions provided vital insight into the curricula used in a variety of university settings. As the materials were categorized into their subtopics, the differences across the materials were also made apparent. Since the presence of each topic was counted across the materials, the output was a list of the topics in order of priority by the individuals who contributed materials. Topics that were included in the majority of the materials are a higher priority to most of the contributing faculty.

The interest lies in how these materials compare to the topic distribution of the Stack Overflow resource. This content comparison did not measure the breadth or depth of the content in either the Stack Overflow page or the materials from professors. However, it exposed the differences in the topics present between materials from professors and the Stack Overflow page. Some key differences were that the materials from professors more commonly covered the motivations of why someone would use a function and the context for how functions fit within the greater scope of computer science.

These differences only reflect one page of Stack Overflow, but is representative of the information that is most discoverable by novices since it was the first result.

Chapter 3: Methods for Resource Observation and Evaluation of Design

In order to gain insight into how adult novice computer science learners who are self-teaching use the existing resources and the intentions behind their choices, I conducted a series of user tests. My goal was to observe what the users interacted with and what parts of the resources they leveraged when self-teaching. The goal of this approach was to inform my designs by observing the way individuals interacted with the existing materials. It was also important to gauge if and how learner behavior differs based on their goals for learning computer science.

Participant Recruitment

The qualifications for participants were that they must be 18 or older and that they considered themselves a novice coder or computer science learner who is self-teaching. I defined someone self-teaching as a person who is not currently enrolled in any formal face-to-face or online computer science classes. Individuals who were participating in self-paced online learning through tutorials or step-by-step guides were considered as valid participants for this experiment. I determined formal online classes to be anything that is offered by a university or higher education institution, or any online or face-to-face learning that would result in a degree or certification. I did exclude participants based on how long they had been self-teaching, so participants ranged from self-teaching for a few months to many years. However, if participants had taken a formal coding class in the last year, they were not included in the experiment. However, these qualifications still resulted in a variety of participant skill levels. Some learners had never taken any coding classes, some had taken courses many years ago, and some used coding regularly in their job.

The participants were recruited through personal and professional connections. I contacted potential participants via email and corresponded to schedule a time to engage in the experiment. The full text of the recruitment email can be found in Appendix A. All participants for the first set of interviews engaged remotely. They emailed the study consent form and the demographic survey before attending the online video call. The consent form can be found in Appendix B and the demographic survey in Appendix C. The experiments took place over a Google Hangouts video call. The participants were given the opportunity to read the paperwork on their own and ask any questions. After the consent process was completed the study began. Participants were filmed and/or screen-recorded. I used Open Broadcast Software to record the screen of the Google Hangouts video call, as well as the audio from both the device (incoming call) and the microphone. The recordings include video of the participants for those who opted to have their cameras on during the video call. The recordings were used to create research notes after the session. It was important to have the screens recorded so that I could deduce what areas of the screen the participants were interacting with when they made certain comments. Individuals who participated in person attended the session at an office on the University of Maryland College Park campus. They completed the survey and consent form and then performed the coding activity at a provided computer. The audio of the session was recorded by an external microphone and the screen activity was recorded by Open Broadcast software. I used the recordings of the sessions to take notes after the fact to code into themes.

Participant Profiles

There were two iterations of the observations, each with 6 participants. This number of participants was sufficient in informing my understanding of current practices from students. The participants varied in their backgrounds and goals, and patterns were able to emerge from this sample size. Secondly, the 6 participants for phase 3 were sufficiently able to provide insight into how the re-designed resource would be used. For the scope of the implications of this study, I deemed 12 total participants to be adequate. Table 3 demonstrates their age, gender, education level, and self-teaching methods.

Table 3: Participant Information

Phase	Participant	Age	Gender	Education Level	Self-Teach Methods
1	1	25	Female	Current PhD Student	Forums, online guides, Googling specific questions
1	2	29	Female	Bachelor's Degree	Online classes, textbook, forums, online guides, tutorial videos, Googling specific questions, free

					classes from universities
1	3	23	Female	Bachelor’s Degree	Forums, online guides, tutorial videos, Googling specific questions
1	4	23	Female	Current PhD Student	Online classes, online guides, free classes from universities
1	5	29	Male	Master’s Degree	Forums, online guides, Googling specific questions
1	6	25	Male	Bachelor’s Degree	Online classes, textbook, forums, online guides, tutorial videos, Googling specific questions
3	7	34	Male	Bachelor’s Degree	Online classes, online guides, tutorial videos, Googling specific questions
3	8	23	Male	Current Master’s Student	Forums, online guides, tutorial videos, Googling specific questions
3	9	23	Female	Current Master’s Student	Textbook, forums, online guides, Googling specific questions
3	10	24	Male	Bachelor’s Degree	Forums, tutorial videos, Googling specific questions
3	11	20	Female	Current Bachelor’s Student	Online classes, textbook, forums,

					tutorial videos, Googling specific questions
3	12	31	Female	Current Master's Student	Forums, tutorial videos, online guides, Googling specific questions

The participants were asked to complete a survey (Appendix C) prior to the experiment. The questions yielded the demographic information, educational experiences, and interest level in the computer science field. These questions included:

- What have you used to teach yourself computer science?
- What are your motivations for learning computer science?
- How do you plan to use your computer science knowledge?

These questions gave the context to help tailor the semi-structured interview questions used for the experiment.

Semi-Structured Pre-Interview

I began the sessions by conducting a semi-structured interview with each participant about their prior education, learning habits, methods of self-teaching, goals for their education in computer science, and any plans for formal education in computer science. See the full collection of questions in Appendix D. Some of the questions included:

- How have you approached learning computer science?
- What resources do you use?

- What are your goals for your computer science education?
- What is the most frustrating part about teaching yourself?
- What is the most rewarding part of learning computer science?

Experiment Design

After the pre-activity interview, I presented the individual with a coding challenge. For phase 1, participants were allowed to use any just-in-time learning resources to complete the activity. The participants were in full control of what resources they would use and when. For phase 3, the participants were only given the resource I created during the design phase of this study to use.

For the experimental part of the data collection, participants were asked to attempt a coding challenge while using the think-aloud procedure. Given the language options of Java, Python, and JavaScript, they were asked to choose the language that they had most recently been studying. The coding challenge focused on the computer science concept of functions. This was chosen because functions are an introductory concept that does not differ greatly between the three chosen languages.

The coding challenges asked participants to adjust an existing program so that it used a function to execute a series of print statements. The subsequent prompts added levels of complexity such as requiring values passed through parameters and values returned with return statements. By completing the coding exercise the participants will have written 4-5 functions.

For Prompt 0, participants were given a program that printed the lyrics to Happy Birthday. They were asked to answer the following questions:

- What does this program do?
- What do you think about this program's design?
- How would you change or improve this program? Why?
- Have you written programs like this one before?

In Prompt 1, they were asked to adjust the program so that the lyrics were printed in a function. Prompt 2 provided new code that got user input for a name to be the subject of the Happy Birthday song. The participants were asked to adjust the program so the input was gathered in its own function. Prompt 3 asks the participants to rewrite the program so that it uses functions to get the input and print the song lyrics. The suggested function names are "getBirthdayName" and "printLyrics". Lastly, Prompt 4 asks participants to explain a code trace of their final program. See the full coding activity in Appendix E. Table 4 provides the specific intentions of the prompts used for the activity.

Table 4: Code Activity Prompt Intentions

Prompt	**Intentions**
Prompt 0	This prompt serves the purpose of gauging how much the participant is able to read the code provided and deduce what the program will execute. It also lets the researcher know how comfortable the participant will be with changing and manipulating the existing code.
Prompt 1	Participants must define a new function and have the function execute the print statements. This task requires knowledge of the syntax for function definition, naming the function, and calling

	the function to execute the code.
Prompt 2	Participants must define a second new function and have the function receive input from the user. There are numerous methods to fulfil this prompt and it is up to the participants discretion which method they choose. The most straightforward execution of this prompt would include passing values by reference through parameters. Other solutions involve scope or creating global variables, or defining functions within other functions.
Prompt 3	Participants must define two discrete functions, one that gets the user's input and one that prints the lyrics with the user's name. There are numerous methods to fulfil this prompt and it is up to the participant's discretion which method they choose. The most straightforward execution of this prompt would include having a return statement in the user input function that returns the string that the user typed as input. This value would then be passed to the lyric-printing function through a parameter. Other solutions involve scope or creating global variables, or defining functions within other functions.
Prompt 4	This prompt serves the purpose of gauging whether participants are able to communicate how the final program works. This task will reveal how much of the code works because they deliberately designed it to execute in this way, versus working out of mimicking code from resources. It also reveals whether the participants can accurately describe how information passes through the program.

The exercise is adapted from Dr. Katherine Gibson, a professor at University of Maryland, Baltimore County. (Gibson, 2019)The question demonstrates the coders ability to write a function. The exercise is scaffolded to gradually introduce more levels of complexity.

In order to complete this coding activity, the participant would need to engage with, at minimum, the following topics:

- Function Calls

- Parameter Passing
- Function Syntax
- Returns
- Code Trace

Since there are numerous ways to successfully complete the challenge, participants could also wish to find information on why we use functions, what scope is, and even language specific documentation.

Semi-Structured Post-Activity Interview

After the participant either successfully completed the coding activity or opted to end the activity, they engaged in a post-activity interview. These questions asked them to reflect on their frustrations and to describe their experience. See the full collection of questions in Appendix D. The questions included:

- What did you find most challenging about this exercise?
- When did you feel the urge to reach for help?
- Can you describe the experience of using the support technology/websites?
- What could have made this process easier for you?
- What're you looking for when you Google something? What makes the resource useful or not useful?

Upon completing the experiment, participants received a $15 gift card for their participation in the study.

Data Theming

The following section outlines my methods for data theming for the interviews and coding activity of this study.

Interviews

All of the interviews were recorded and all of the interviews followed the same basic structure of questions. In order to theme the interviews, I listened to the recordings and took summarized notes on the participants' answers. I organized the notes by grouping them by question. I was then able to see how the participants more generally responded to each question. I reviewed the notes to extract themes and determine the prevalence of ideas from the interviews. The themes from the interviews are outlined in the findings.

Coding Activity

I watched the recordings of the experiments and cataloged each action taken by the participant. These actions included what was done they did on the screen through the mouse and keyboard, but also what they said. Through this method, I paired the context to the participants' actions.

Participants completed the final prompt of the exercise, which was a verbal or drawn stack trace. Because of the disparate levels of coding mastery, the information was useful for comparison or measuring the effectiveness of their experience with the resources. For these reasons, the information collected from those stack traces are not featured in the findings of this study.

From the cataloged actions, I reviewed each note and assigned a theme. The themes were generated during the process of reviewing the notes. When a note did not fit into an existing theme, one was created or broadened to accommodate. From the large-scale themes, I was able to breakout subcategories to have more granularity in the findings.

Chapter 4: Novices Using Existing Resources (Phase 1)

Research Question: How do existing just-in-time learning solutions support self-teaching, adult computer science learners who are new to coding?

Findings from Pre-Activity Interview

The pre-activity interview set the basis for how active the participants are in their self-teaching, what sort of resources they use, and what challenges they face. The findings are based on the interviews with the 6 participants from phase 1.

Approaches and Goals

The approaches taken by the participants to learning computer science were connected to their computer science education goals. The participants who took a proactive approach to self-teaching tended to have the goal of learning the concepts for the sake of building that skill, unconnected from a specific project. While the participants who took a more reactive method of self-teaching tended to have the goal of executing a specific project for work or school. Two participants from the phase 1 interviews described their goals for learning coding as wishing to gain a holistic or broad understanding of the foundational computer science topics. These two participants also noted that they prefer to self-teach using tutorials or linear coding demonstrations. This was in contrast to the four participants from phase 1 who are self-teaching code in order to serve a purpose for work or school. These participants consistently noted their preferences for just-in-time resources such as forums, and for code samples.

I categorized proactive self-teaching practices as seeking out guided tutorials, investigating the fundamentals of computer science, and building a broad conceptual understanding. This method also avoids opting out of certain subjects because they temporarily seem irrelevant. I am categorizing reactive self-teaching practices as seeking out answers to emergent coding questions, finding code examples for syntax or that directly apply to their situation, and self-limiting exposure to topics that seem irrelevant.

Many participants were explicit about the connection between their practices and their computer science education goals. One participant noted, "*Unless I have a project, I don't usually try to learn [coding]. I need to be able to apply it right away.*" In this example, the participant is not regularly working on their coding abilities and is only motivated to engage or practice when they have a project to apply it to. Similarly a participant stated, "*I kind of wish there was self-motivation to actually do an hour a day of a Python tutorial to keep my brain thinking about computer science. I only Google coding questions if I'm actively using it elsewhere in my life.*" At the root of needing these skills for application is the need to actually solve a problem. One participant discussed the struggle they feel with spending valuable time on reviewing the basics when there is an urgent and relevant problem to solve. They stated, "*But it's hard to motivate myself to take that time [to read the basics of coding]. Especially when I have a specific problem I'm trying to figure out.*" In this case, the participant is actively not relating learning the basics with their ability to solve their specific problem. On the other hand, a participant that engaged in more proactive self-teaching practices stated, "*I would like to become proficient enough where I stop just Googling things. I would like to know intrinsically how to do it.*" This end-goal has led the participant to take a more structured and proactive approach to learning computer science for the time being.

It is worth noting that there is no inherent benefit from taking a proactive or reactive approach. The approaches are both values methods of reaching different goals.

Self-Teaching Resources

The participants gave numerous examples of resources they used for self-teaching computer science. From the numerous examples of resources described in the interview, I grouped the resources into four categories: forums, tutorials, textbooks, and one-on-one help. Table 5 provides an in-depth description for each type of resource, as well as numerous examples from the interview of each category.

Table 5: Types of Self-Teaching Resources

Category	**Description**	**Examples**
Forums	This category includes websites that have a question/answer format. Some examples are moderated by employees of the platform and some are community moderated. Most examples have a voting system in place to rank the answers.	Stack Overflow, Quora, Yahoo Answers, eLearning Heroes
Tutorials	This category includes websites that have structured, scaffolded instruction on computer science topics. Some of these examples include self-quizzing, code examples, videos, and activities.	Code Academy, Code.org, Khan Academy, W3Schools
Textbooks	This category includes physical textbooks	Learning Python

	and eBooks that have instruction on the topics of computer science. These resources are mostly static and do not provide feedback for the reader.	the Hard Way, JavaScript: The Definitive Guide
One-on-one Help	This category includes any help given to a participant from an individual through a medium other than a forum. This communication happens in-person, via email, or other digital media.	Classmate, teaching assistant, professor, colleague, friend

A final component that is present through each of the resources listed in Table 5 is code samples. This category includes any examples of working code provided online or through files. Code samples are found in each of the methods listed previously, but they can be considered their own learning resource. For many coders, as they gain experience, they are able to learn from and dissect code samples without needing the further context provided by the other resources noted. However, for novice coders, working code samples without context or explanation can be less useful or productive. One participant noted that their early days of learning to code included, "*a lot of getting code from someone else and running it and then changing it for myself and my purpose. I wasn't really sure how it worked though and I usually get stuck.*"

Frustrations and Rewards

The participants expressed a wide variety of frustrations with self-teaching. Conversely, there was a much more unified message about what was rewarding about self-teaching computer science. The frustrations can be categorized as issues with knowledge gaps, difficulties with resources, and problems from external forces. Knowledge gaps refer to any portion of computer science or coding that the participant is not familiar with or does not understand. While difficulties with resources, refers to barriers they encounter with the educational resources they rely on. And external forces describe any frustrations that are triggered from outside of the direct educational experience and they can usually involve other people. Each category is broken into specific frustrations expressed by participants in Table 6.

Table 6: Categories of Frustrations

Category	Frustrations from Participants
Knowledge Gaps	Frustrated by not knowing where to start
	Frustrated by not knowing best practices
	Frustrated by not knowing what terms to use in a web search
Resources	Frustrated by not knowing what resources to trust
	Frustrated when working with software that is under-supported
	Frustrated when writing code in a language that has bad documentation
	Frustrated by setting up coding environment
External Forces	Frustrated when they have a great idea but are not given enough time to build it
	Frustrated knowing there are other people who could do this more quickly

Some of these frustrations were reiterated by the participants when they completed the coding exercise as well. These frustrations are used as inspiration for the design changes described later in the study.

The rewards for participants could all be described as the moment of satisfaction or "eureka" that comes from finally solving a problem and getting something to work. One participant said, the most rewarding part is "*when it works, when it does the thing it was supposed to do.*" However, the reward differed for participants depending on their goals. Participants who were learning coding for the sake of understanding, felt rewarded when they finally understood an underlying concept. They were less gratified by writing code that worked if they could not understand how it worked. A participant in this category even noted that they felt rewarded when they knew their code was "*elegant and most efficiently written*". Worrying about the elegance or efficiency of the code is in contrast to the participants who were learning coding as a reaction to a work or school problem. They noted their relief and reward when they finally got the code to execute the outcome they were planning to reach. Some participants in this category even noted that they specifically did not care how it worked, but just that the outcome was correct and consistent. A participant clarifies, *"I don't care how it works as long as it worked.*"

Summary of Pre-Activity Interview Findings

The interviews yielded great insight into the experience of self-teaching computer science using online resources. The relationship between the participants' goals and their approaches helped determine what sub-populations of novice, self-teaching coders exist. The designs that serve the proactive self-teacher may differ from the reactive self-teacher in the realm of computer science resources. Of those resources, the interviews helped solidify the categories of resources that are most commonly used by this population. The interviews also allowed me to contextualize the just-in-time resources into the greater categories of resources. Additionally, the frustrations and rewards shed light on what the participants identified as their struggles. Some of their frustration or barriers will become clear in the coding activity that they may not note themselves. Finally, the aspects of the self-teaching process that are rewarding could potentially be leveraged in the design of such resources.

Findings from Coding Activity

Nearly 600 actions were catalogued as a part of these six experiments and put into themes. The four main themes emerged from these notes are presented in Table 7.

Table 7: Themes from Phase 1

Theme	Description	Count
Searching for resources	Typing a search into a search engine, evaluating the results, and selecting a resource to open.	96
Browsing selected resources	Reading, evaluating, or interacting with the content of a resource.	94
Adjusting code assignment	Writing code, deleting code, copying/pasting code, or running a program.	91
Working with errors	Reacting to logic and run-time errors	17

Searching for Resources

Across the six iterations of the code activity, participants made forty-two unique searches. Not every search that was made led to a participant opening a resource, and some searches were made where participants opened numerous resources. The full list of searched phrases from phase 1 participants can be found in Appendix F. The following words were searched 3 or more times by participants in the experiment. The most searched words from the experiment can be found in Table 8.

Table 8: Most Searched Words

Word	**Count**
java	19
function	13
code	9
javascript	8
alert	6
class	5
html	4
how	4
define	4
print	4
python	3
variable	3
string	3
return	3
type	3
run	3

The shortest searches were “java program” and “script tags”, which are each two words long. The longest searches were “code snippet for javascript function that prints a string” which is nine words long and “does function have to have return type java” which is eight words long.

Much of the searching and selecting process is out of the scope of the design possibilities for this study. This information is helpful for gleaning the level of understanding the participants had of the content as they worked on the coding assignment.

Browsing Selected Resources and Adjusting Code Assignment

The notes in the Browsing Selected Resources and Adjusting Code Assignment categories were the most relevant to the participants’ experiences with the resources themselves. Searching for resources happens outside of the resource itself and dealing with coding errors happens between the compiler and further resource searching. Browsing Selected Resources and Adjusting Code Assignment are the themes centered around the participants' experiences using resources to improve their code. These two main themes have been broken into relevant subcategories that are described and counted in Table 9.

Table 9: Phase 1 Theme Subcategories

Themes	**Subcategories**	**Description**	**Count**
Browsing selected resources	Interacting with Try-it-Yourself code windows	Participants writing or running code in an interactive code window in a resource. Examples include W3Schools and CodePen.	6

	Reading code samples	Participants reading code samples provided by the resources.	43
	Reading code comments	Participants reading the comments in the code samples provided by the resources.	3
	Reading text	Participants reading the text content in the resources.	34
	Returning to resources	Participants re-opening the resources while working on their code assignment.	7
Adjusting code assignment	Interacting with the compiler	Participants interacting with the compiler in any way beside running the program. This included saving files, stopping program runs, adjusting compiler settings, and making new files.	9
	Copying code	Participants copying and pasting code. Some participants copied code from previous drafts of their code assignment or from the resources.	9
	Programming the assignment	Participants typing directly into their program. These were only instances where participants were not directly duplicating code from a resource that was open at the same time.	34
	Talking the problem aloud	Participants talking through the logic of their coded assignment and of their code draft.	9
	Re-typing code	Participants duplicating code from a resource that is open at the same time by typing it directly into their program.	5
	Running code	Participants running the program and executing the code.	13

These themes and subcategories were the most pervasive observations about the participants from the coding experiment. The majority of the notes describe the participants reading the code or reading the text of a resource. I observed that many participants tried to skip reading text and preferred reading the code samples. There were 43 instances of participants reading code samples and 34 instances of participants reading the text of the resource. I further investigated this pattern and found that 30 times, the participant read the code sample first and then turned to the text as a secondary support. There were only 9 instances of participants reading the page linearly or reading the content in the order the page dictates. In the resources used by the participants, the code samples were often embedded in a paragraph. So while the code samples and paragraph text both seem important to the participants, the priority is to have the code samples first.

Summary of Coding Activity Findings

The coding exercise exposed the ways that the participants solve problems and use resources when self-teaching a computer science topic. The initial four themes provided a framework to use when thinking about their behaviors during the exercise. The insight into the challenges with searching for a resource was enlightening and leads to design ideation later in the study. After narrowing down to two main themes, adjusting code assignment and browsing selected resources, I was able to break out further subcategories of their interactions with the resources and their code authoring. This insight will directly correlate to potential design decisions. Lastly, the prioritization of code samples over explanatory text by the participants was an extremely useful detail for later design.

Findings from Post-Activity Interview

The post-activity interview focused on participants' experiences with the exercise and asked them to reflect on their choices when it came to resources. The interviews were conducted with the 6 participants from phase 1 after completing the coding activity.

Challenges with the Prompts

Many of the participants were not able to complete the prompts, so the level of challenge differed across the participants. An initial challenge for some was that they did not understand the prompt. Some participants had never used or created a function before and did not know what a function was. Understandably, this made the coding prompt difficult to understand, especially with a researcher watching them.

Additionally, some participants did understand the concept of a function, but could not reason why a function would be used for such a simple program. It was a challenge for a few to get past the relative simplicity of the program's design. This frustration was quite expected in retrospect, particularly from participants who have only self-taught computer science with the intentions to solve a business or work problem. They have only attempted to execute code that fulfils a mission they thoroughly understand. The exercise was scaffolded so as to gradually require more facets of functions, which is antithetical to the practices this more reactionary population typically uses. A participant who only regularly codes to parse DNA sequences for their projects noted, "*I would have just found another - hardcoded way to do it. It would have had the same result, and it would have had less to do with programming*." An outcomes focussed mindset made this exercise challenging at its foundation.

General Challenges

The majority of the other challenges fit into the categories of frustration found in the pre-activity interview. One participant noted their frustration from a knowledge gap and described, "*I was frustrated because I can't interpret how to go through a problem when I don't have the words and language needed to talk about Python.*" Similarly, a participant wished for "*more code syntax off of the top of my head*." An external factor that challenged many participants was running out of time and feeling watched. The exercise was not actually timed, but many participants wanted to avoid wasting the researcher's time. Additionally, learning a new concept while being watched caused anxiety for some participants. Some challenges posed by the resources were not having the examples the participants wanted and were looking for. A participant "*wanted examples where it had complete code using multiple functions*" but was struggling to find an example that fit this criteria.

Reaching Out for Help

Participants were asked to reflect on times they felt the urge to reach out for help during the exercise. I was able to answer questions that were unrelated to the exercise, but was careful to not guide their coding or resource choices. The participants' reflections illuminated that their answers had a lot to do with their background in self-teaching. One participant stated, "*I've never really had the resources to ask other people at my job, so I didn't really have that urge*." While a participant who is enrolled in secondary education said they would "*have gone to a professor or my friend who codes*." These vastly different answers reveal that the tendency to ask for help can be related to the options that have been available to you in the past.

Useful Resources vs Unuseful Resources

As each participant had recently interacted with numerous online resources, they were asked to describe what made a resource either helpful to them or what made it unhelpful. A common item that classified a resource as very useful was containing numerous code examples with clear explanations. As one participant explained, "*I can always use more little chunks of code with well annotated explanations or comments*." Similarly, multiple participants noted the benefit of try-it-yourself code portals. One participant noted, "*I don't want to mess up my whole program for a small piece.*" In other words, they like to have a space where they can test how a small piece of code will run before introducing it to their larger program.

The aspects that made a resource unuseful were somewhat surprising. Some noted a lack of trust of informal or crowd-sourced computer science resources. A participant said, "*I just want to know that the code I'm reading is right. It's like I want the teacher to tell me, not another student*." However, in contrast to this concern there was also discomfort with using these resources if they seemed to only be for "real coders". One participant summarized, "*If it is a website that seems geared to coders, I*

might not want to use it because I don't know what they are talking about."
Additionally, some participants who are considered more reactionary self-teachers said they try to avoid anything that is a part of an "online learning experience". The participant stated, "*I just want to see and extract the one thing I'm looking for. I'll know more quickly if I'm in the right place.*" The participant expounded on what makes an "online learning experience". They described it as, "*something that tries to take me on a linear path, from start to finish. They usually have videos too*." Such resources would fall into the category previously listed as tutorials. The post-activity interviews provide more evidence for a relationship between the type of self-teacher (proactive or reactive) and their preferences in materials.

Lastly, while the code samples were vital to making a resource useful, one participant remarked, "*a wall of code is intimidating*." Large program samples that do not have intermittent explanation can have the opposite of the intended effect and actually turn users away from the resource.

Summary of Post-Activity Interview Findings

While many participants struggled with the exercise itself, I was still able to gain significant insight into more general challenges and patterns. The general challenges included knowledge gaps and external forces, with the additional pressures of being watched during a synchronous study. There was also insight into what type of learner may be looking for one-on-one help in a coding challenge like this one. Lastly, I was able to aggregate some key features that make a resource helpful or unhelpful.

Chapter 5: Designs for Self-Teaching Resources

From the insights gained through observing the current state of practice for novice adults who are teaching themselves computer science, I was able to develop potential designs that could leverage the benefits of the online resources used by this population. The final design ideas are rooted in the themes from the code activity experiment. Multiple types of solutions were considered when ideating and sketching for this population. A final design was created in the form of an interactive mockup to be evaluated in the second experiment.

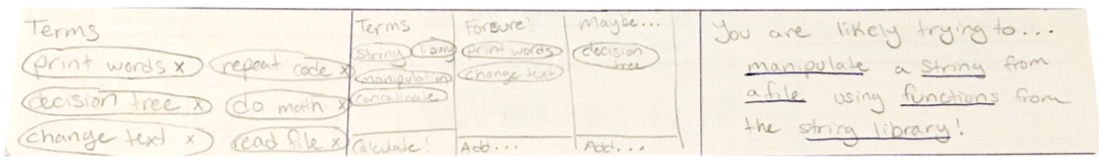
Solution Sketches

With the findings from the first phase of this study, I ideated on some large scale product solutions to the barriers faced by self-teaching, novice computer science learners. The sketching method is a narrative sketch that shows the user's journey and demonstrates the proposed features of the product or solution.

Resource Searching Assistance

This design in Figure 1 is in reaction to the frustrations of not knowing where to start and not knowing what to search in a search engine.

Users struggle to know where to even begin with a coding challenge. They do not know the terminology to search.	An eye-catching, but not invasive pop-up will prompt a new user to engage in the survey.	The survey includes 3 columns in relation to the user's problem. The columns are labeled, Yes, Maybe, No. The fourth column is labeled Terms and is full of computer science concepts.



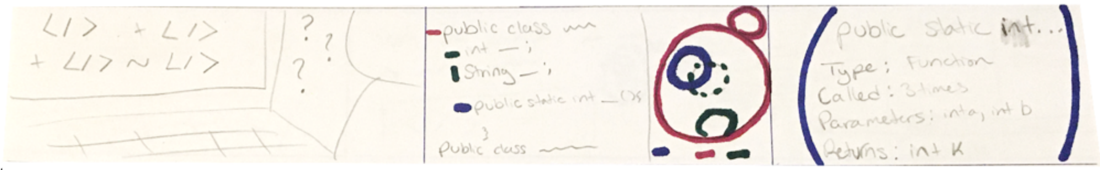
The terms start out in plain English. The user drags the phrases into the relevant columns based on their relation to their problem. The Terms column populates with relevant coding terms based on the selections.	The user moves terms they recognize to the appropriate columns or they add their own. When they press Calculate, the system determines a level of comprehension of the user and a suggested description of the problem the user is working on.	The system will output a problem statement with hyperlinked relevant terms. The links will take them to the definition of the terms. Additionally, the most relevant pages from the site will be listed below.

Figure 1. Resource Searching Assistant Storyboard

This solution is an interesting way to tackle the frustrations the participants feel when searching for useful resources, however it does not particularly assist the population in the way they use these resources. This solution was determined to be out of the scope of this project.

Code Anatomy Visualization

This design in Figure 2 is in reaction to the frustrations with not knowing how to string the code samples together, especially from multiple sources.



The users piece code from many sources together into one solution. However, a lack of understanding of how those pieces fit together leads to broken code.	The code block would get a color-coded overlay for each phrase. These colors would correspond to a 3D visualization of the code architecture.	The user can focus on one component to read its description and further details.

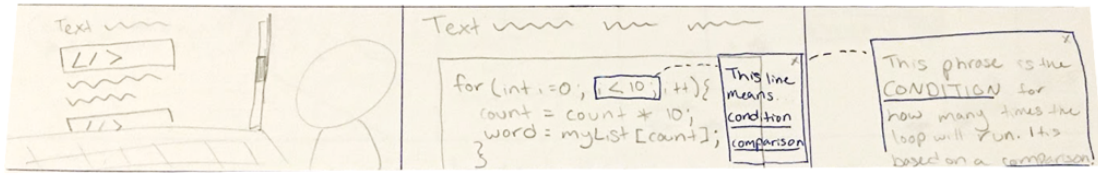
The user zooms in on an overlap to determine the components' relationship. For example, int a is instantiated in the given function.	The user can manipulate the anatomy and see the changes reflect in the code. For example, this would bring the blue variable instantiation into the red class.

Figure 2. Code Anatomy Visualization Storyboard

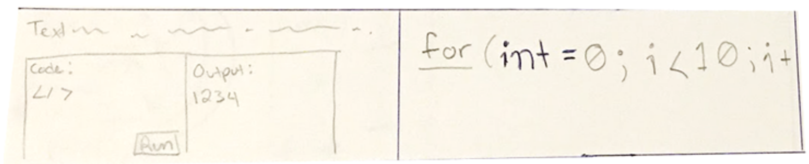
This solution is similar to the numerous methods of code visualization that exist and that are used by teachers and students. It is also a particularly big leap to conclude that a participant would understand how their code links together better from a complicated visualization. Lastly, no participants explicitly expressed a desire for a visualization or tool for seeing the bigger picture.

Code Sample Layout

The design in Figure 3 is in reaction to the observations of how participants interact with the code resources, specifically with code samples. The usefulness of the code samples is dependent on their quality and the placement of the context and explanation of that sample.



Users are reading the featured code blocks first and return to read the paragraphs only after many failed attempts with their own code.	The code block is featured first. Each phrase of code is connected to documentation definitions. When the user hovers over a phrase, they see the definition.	Key terms in the definition are hyperlinked for more detail. This will help users inform their future search vocabulary.



Users can view the code in a split view with the output. The users can change values and see how the output is affected.	Font, coloring, styling will all mirror popular coding editors and be consistent across the code blocks.

Figure 3. Code Sample Layout Storyboard

This design addresses the main barrier that was observed in the previous experiment and is the most rooted in the observations. This design can be executed through an interactive mock-up and most closely addresses the key research questions of this study. For these reasons, the Code Sample Layout design is expanded on for further evaluation.

<u>*Design Ideas from Themes*</u>

Research Question: What are some instructional and interface design changes for just-in-time learning solutions that could improve the learner experience?

After ideating on the possible big-picture solutions, I decided that working on the code sample design would be the most beneficial to answering my research question. I believe that the code sample solution is most directly supported by the observations gained in the first phase of the experiment. With the general plan being to design or mock-up a just-in-time learning resource, I took each theme from my study and translated it to a design possibility. Table 10 summarizes the previously identified themes and subcategories and aligns them with correlating design ideas. Some themes were intentionally left blank because they are not directly related to an online resource, but instead are related to the coding environment, the learner, or their programming ability.

Table 10: Mapping of Design Ideas to Themes and Subcategories

Themes	**Subcategories**	**Design**
Browsing selected resources	Interacting with Try-it-Yourself code windows	Include Try-it-Yourself code windows where possible.
		Have the option to run code samples whenever they are introduced.
		Provide the output of the code sample, even if it is static.
	Reading code samples	Provide code sample early on the page, before large swaths of text where possible.
		Embed important information and context into the code block, so users do not have to cross reference the code samples and

		the paragraphs.
		Provide visual cue that the code sample works or does not work.This is particularly important for forum resources.
		Match the syntax coloring of code editors.
	Reading code comments	Include well-commented code samples.
	Reading text	Put text after the code sample.
		Extract key information from the text and embed it into the code sample.
	Returning to resources	
Adjusting code assignment	Interacting with the compiler	
	Copying code	Provide a button to copy a whole code sample.
		Make sure code is displayed as text and can be highlighted and copied.
		Programs written in languages with spacing requirements (such as tabs in Python) should be spaced correctly when copied and pasted.
	Programming the assignment	
	Talking the problem aloud	
	Re-typing code	Make the resource responsive so that it can be put into a split-screen with a coding window.

	Running code	Include a prompt or reminder to run their code often.

Not every design option listed above is enacted in the mock-up created for this study. The limitations of time and ability made me prioritize certain design options over others. Luckily, this also lessened the number of variables between the first and second phase of experiments.

Final Design

The final design is based on a revised version of the top Stack Overflow page related to functions. Figure 4 shows this page, it can also be accessed here. Figure 5 shows the page redesigned based on the findings from phase 1. The integrity of the question and answer format is retained in this design.

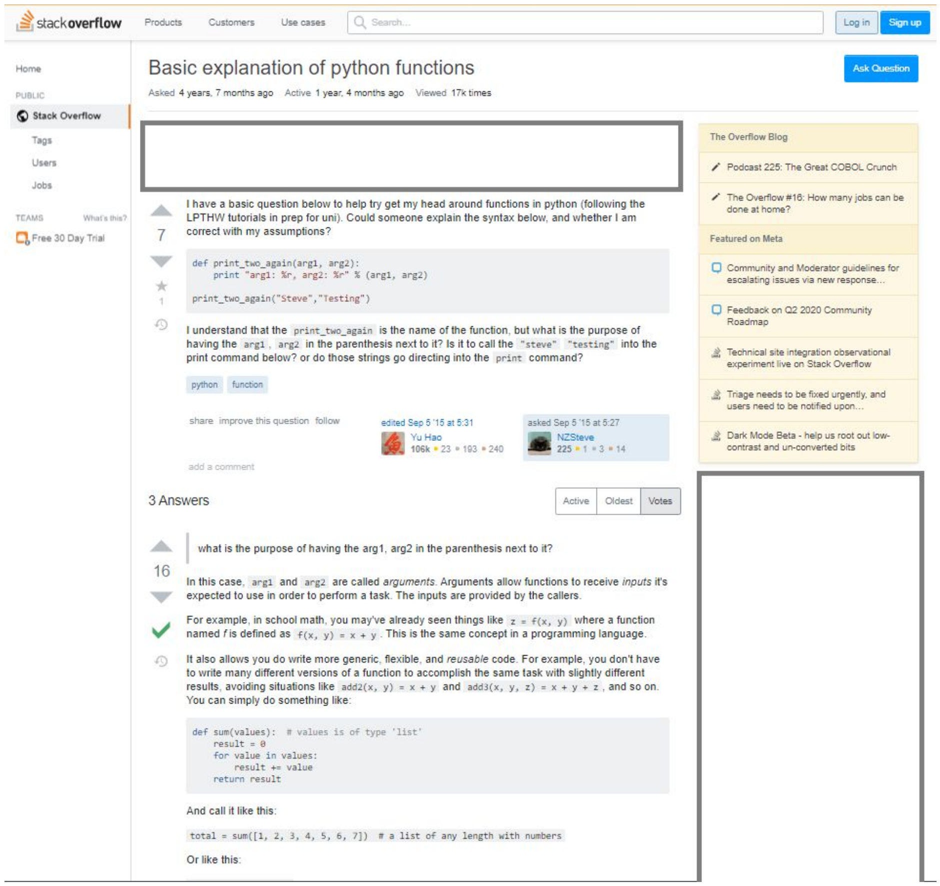


Figure 4. "Basic explanation of python functions" Stack Overflow page.

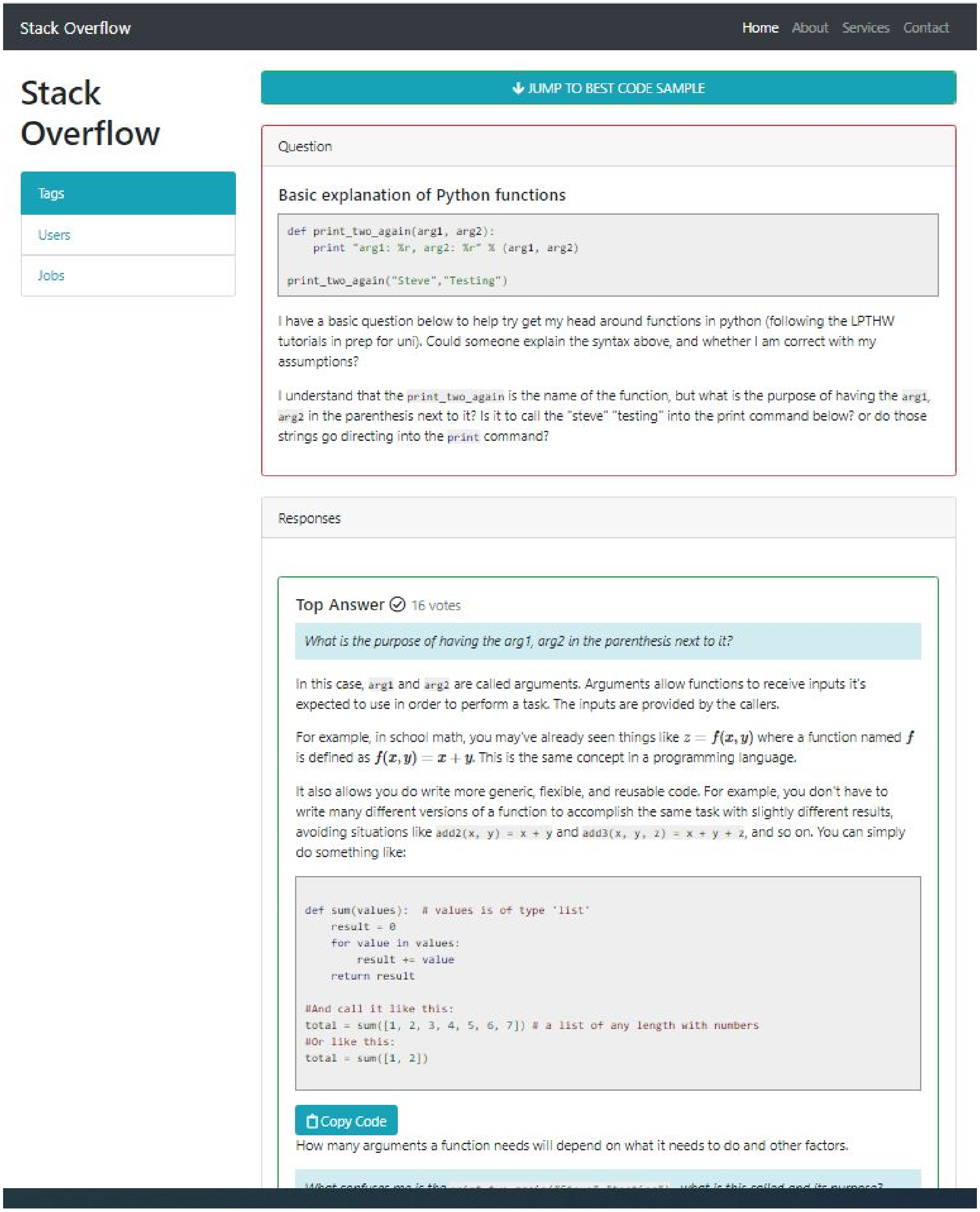


Figure 5. "Basic explanation of python functions" page redesigned for the study.

Design Feature 1: Easy Navigation to Best Code Sample

In order to maintain the question and answer format of Stack Overflow, I did not reorder all of the code samples to be at the beginning of the page. Instead, I included a button that would jump the user to the "best" code sample of the page. I chose the code sample from the highest voted answer as the "best" code sample. See button in Figure 6. Additionally, I moved the code of the question to the top of the question block.

Figure 6. The "Jump to Best Code Sample" button featured at the top of the Stack Overflow mock-up.

Design Feature 2: Contextual Information within Code Samples

I embedded the contextual information from the paragraphs of the page into hover tabs in the code block (Figure 7). The black tabs would appear where a use was hovering over a line of the code. The hover would highlight which areas of the code it was referring to. The language of the tabs was lifted directly from the language in the paragraphs, so that I would not be adding any contextual information that was not present before. The hovers disappeared when a user moved their mouse off of the code.

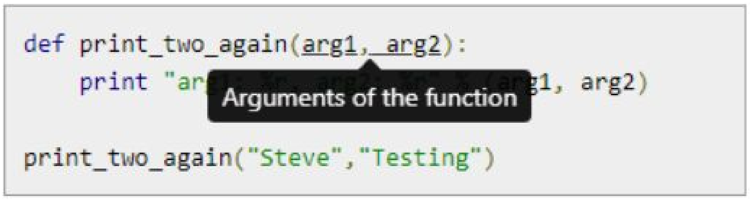
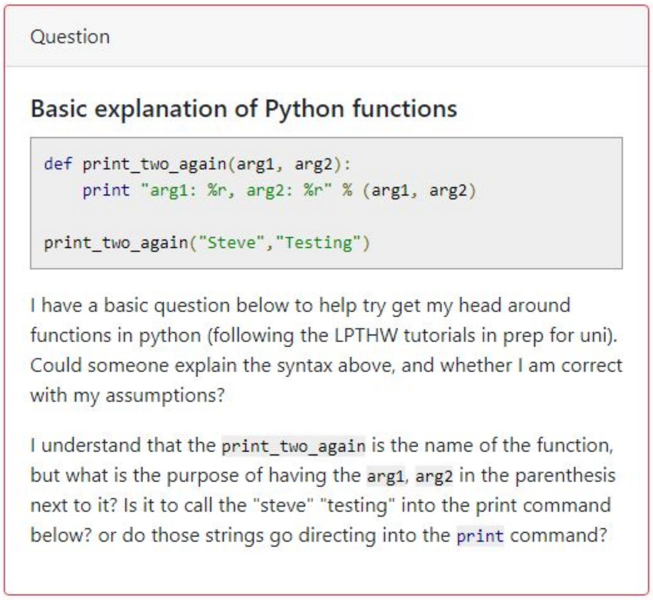


Figure 7. The hover-over tabs of the question code sample. The first tab is labeled "Name of the function" and the second tab is labeled "Arguments of the function".

Design Feature 3: Distinguish Working and Not-working Code Samples

In order to distinguish between code that was flawed and functioning code, I used color. The color of the border around the question (which includes incorrect or incomplete code) is red while the color of the border around the answer was green. See Figure 8.



Question

Basic explanation of Python functions

```
def print_two_again(arg1, arg2):
    print "arg1: %r, arg2: %r" % (arg1, arg2)

print_two_again("Steve","Testing")
```

I have a basic question below to help try get my head around functions in python (following the LPTHW tutorials in prep for uni). Could someone explain the syntax above, and whether I am correct with my assumptions?

I understand that the `print_two_again` is the name of the function, but what is the purpose of having the `arg1`, `arg2` in the parenthesis next to it? Is it to call the "steve" "testing" into the print command below? or do those strings go directing into the `print` command?

Figure 8. The question code block and text from the Stack Overflow mock-up.

Design Feature 4: Support for Copying Exemplary Code

In order to copy the best code samples from the resource, I provided a Copy Code button that would put the entire code block in the clipboard. See Figure 9. The code was formatted so that it would successfully run when pasted into a compiler. The code block itself could also be highlighted and copied without adding spacing or formatting.

Design Feature 5: Syntax Highlighting in Code Samples

The code blocks in the resource were all given a standard coloring system. The code blocks were given a grey background and box to help them stand out. The coloring of the code mimicked many code authoring platforms that the users may be familiar with. For example, program language primitives such as “def”, “print”, or “return” were given a different color. Other features of the program, such as comments or function arguments, were also colored to help distinguish their behavior within the program. This can also help novice coders identify the different components of the code. The resulting program decoration can be seen in Figure 9.

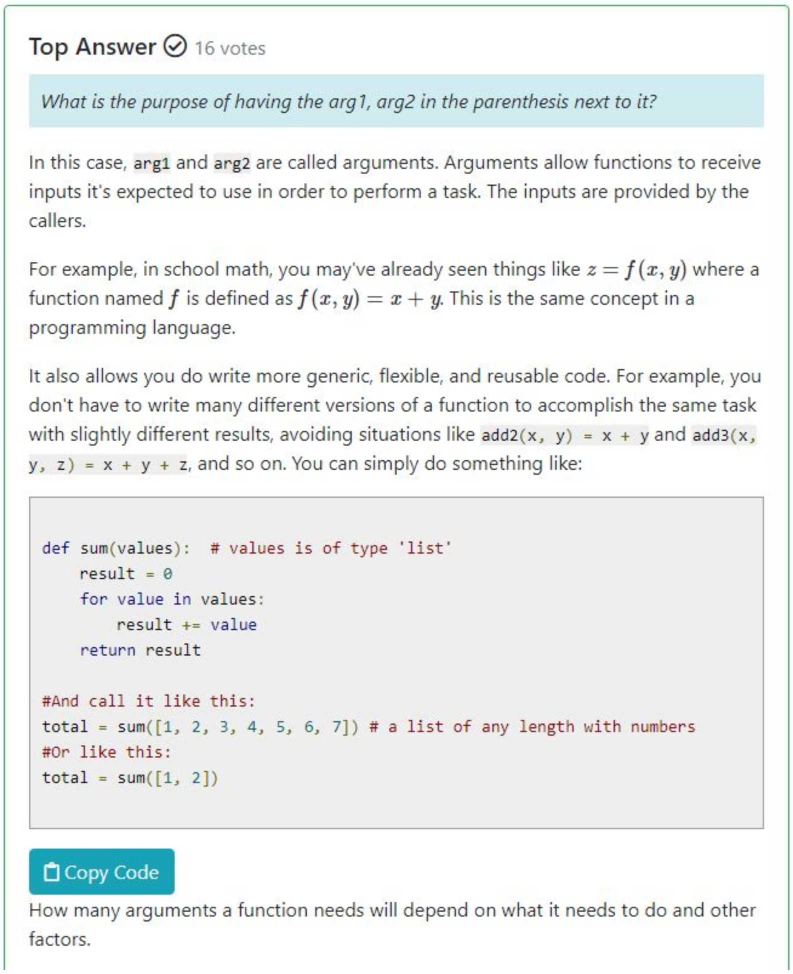


Figure 9. The top answer text and code block from the Stack Overflow mock-up.

Some design elements that were retained from Stack Overflow include the question and answer format and voting for answers. Additionally, I retained that when the answerer quoted the root question, their quotes were specifically styled. This can be seen in Figure 9 in the light-blue box.

Summary of Design Changes

The final design to be tested in the third phase of the study is an interactive mock-up of a Stack Overflow page. The design changes enacted are documented in Table 11 and the corresponding changes are labeled in Figure 10.

Table 11: Design Changes

Themes	**Subcategories**	**Label**	**Design**
Browsing selected resources	Reading code samples	A	"Jump to Best Code Sample" button
		B	Code prioritized over text where possible
		C	Hover-over labels for code blocks
		D	Color-coded questions and answers
		E	Consistent code block styling
Adjusting code assignment	Copying code	F	"Copy to Clipboard" button
			Code is displayed as text and can be highlighted and copied
			Code space correctly when copied and pasted
	Re-typing code		Resource is responsive

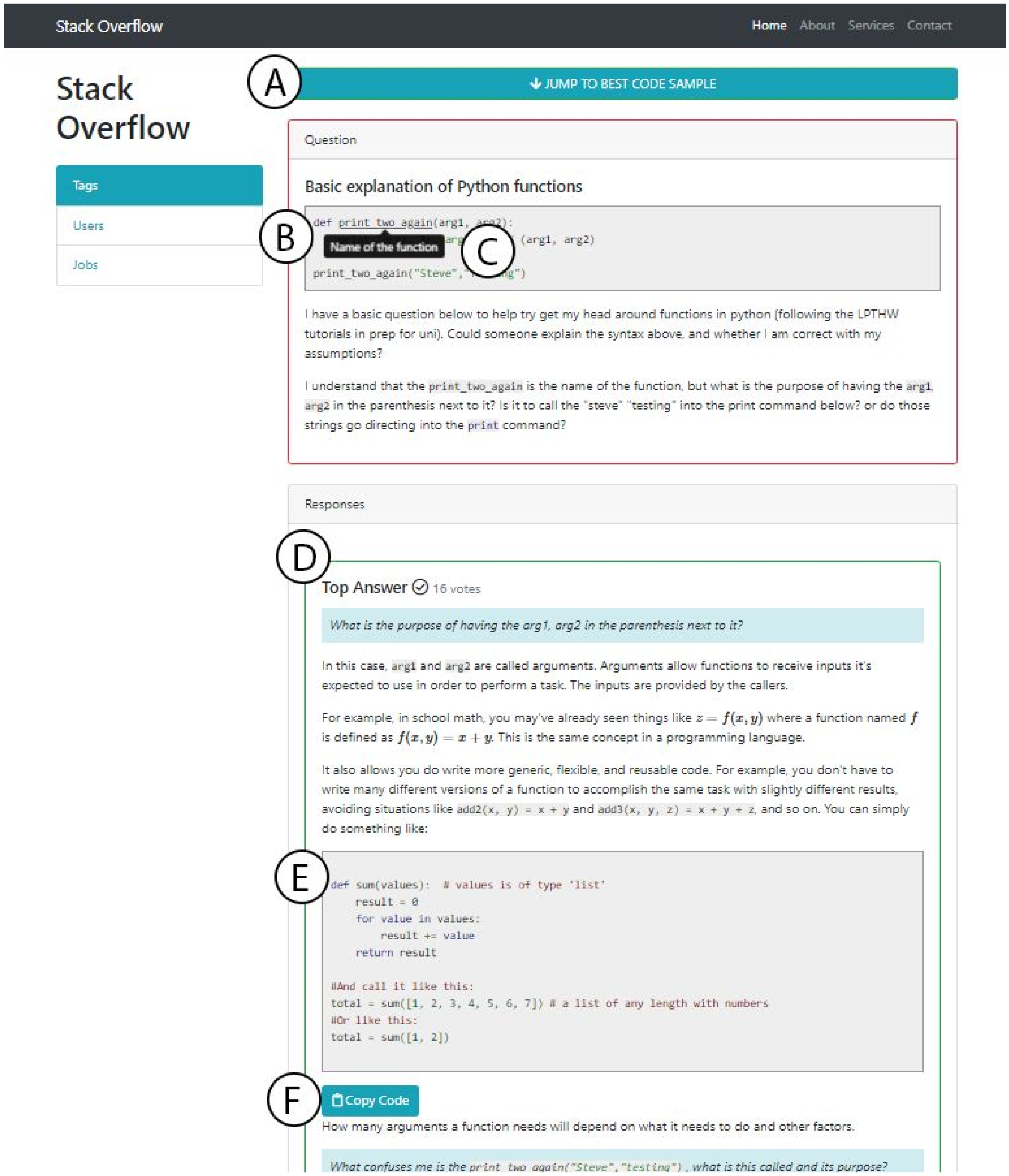


Figure 10. The final design of the resource with corresponding labels to Table 11.

Chapter 6: Evaluation of Resource Design (Phase 3)

The final phase of the study asked a new set of adult programming novices to go through the same protocol as used in the initial study but now using the redesigned set of resources. The goal of this phase was to provide data that could be used to comparatively evaluate the impact of my design. The third phase experiment is also segmented into the pre-activity interview, the coding activity, and the post-activity interview.

Findings from Pre-Activity Interview

I conducted a pre-activity interview for each of the six participants in phase 3. However, the purpose of conducting the interview with this population differs from the first phase. For phase 3, I used the pre-activity interview to get context for where the participants were in their coding career. The interviews were then catalogued and analyzed using the same methods as phase 1. I found that the themes and trends of these interviews were the same as phase 1. The participants still fell into the categories of reactive versus proactive self-teaching styles. Additionally, they expressed the same variety of frustrations with self-teaching computer science. The listed many of the same resources as the first set of participants, all of which fit into the categories of forums, tutorials, textbooks, and one-on-one help.

Findings from Coding Activity

I watched the recordings of the activity and recorded each action. The notes were about the actions taken on the screen, as well as the thoughts expressed by the participants as a part of the think-aloud procedure.

Nearly 600 actions were catalogued as a part of these six experiments and put into themes. The three themes and their descriptions are outlined in Table 12.

Table 12: Phase 3 Themes

Theme	Description
Browsing the resource	Reading, evaluating, or interacting with the content of a resource.
Adjusting code assignment	Writing code, deleting code, copying/pasting code, or running a program.
Working with errors	Reacting to logic and run-time errors

Notably the themes are largely the same as the ones found for the previous phase. The theme "searching for resources" is eliminated because the participants were provided the only resource they were allowed to use.

These general themes were broken into subcategories that are described and counted in Table 13.

Table 13: Phase 3 Theme Subcategories

Theme	Subcategories	Description	Count
Adjusting Code	Working on code	Writing code or deleting code.	62

assignment	Copying code	Participants copying and pasting code. Some participants copied code from previous drafts of their code assignment or from the resource.	13
	Running code	Participants running the program and executing the code.	33
Browsing the resource	Reading question code	Participants reading the code sample that is embedded in the question of the forum.	13
	Reading answer code	Participants reading the code samples that are embedded in the answers of the forum.	15
	Reading question text	Participants reading the text of the question in the forum.	7
	Reading answer text	Participants reading the text of the answers in the forum.	31
	Returning to resource	Participants re-opening the resource while working on their code assignment.	17
	Specific searching	Participants scrolling to find a specific answer in the resource. Some participants used the browser's "find" tool.	24
	Interacting with hovers	Participants read the hover notes on the code samples.	9
Working with errors	Interpreting compiler errors	Participants reading and dissecting the errors from the compiler after running their code.	21
	Talking the problem aloud	Participants talking through the logic of their coded assignment and of their code draft.	47

Reading More Text

There are some key differences between these subcategories and the ones found from the first phase (shown in Table 9). One key difference was that there were significantly more instances of participants reading the text of the resource in comparison to the code blocks. There are two components to my explanation as to why this changed. The first component has to do with the Returning to resource subcategory. There were significantly more instances of participants revisiting the resource while working on their code during this exercise. Since they only had one resource to consult, they were diving deeper into the content and reading more carefully then they naturally would. One participant said, "*I would probably have moved onto a different page at this point, but I'll read it a little bit first.*" The second component is the high volume of specific searching. When participants reached a challenge that would normally garner its own unique Google search, they were forced to only consult with the page given. This resulted in people slowly and linearly scrolling through the resource to find content on the extract they were stuck on. Both of these elements contributed to more instances of participants reading the text.

Running Code

Secondly, there were 20 more instances of participants running their code. I think a possible reason for this change is that participants were willing to experiment more with their code when their pool of resources was limited. Similarly, more participants spent time talking through their code out loud as a means of problem solving, because they could not consult further resources.

Interacting with Hovers

The participants also had helpful interactions with the hover-over tabs that were added to the code blocks. Unfortunately, only half of the participants discovered the hover-over tabs. When I asked why a participant did not read the hovers, they stated "*I'm used to little tabs popping up while I'm browsing things on the internet. It's usually image descriptions or labels, which I'm used to ignoring.*" A second participant noting, "*I didn't assume that the information in the hovers was real.*" This critique about the discoverability and reliability of the feature is supported by the lack of interaction from half of the participants.

Using Question Code

The "Jump to Best Code Sample" button was not utilized by any participants. Additionally, no participants copied code from the "best" code sample. Code was only copied from the code block in the question. This is the opposite of the intention behind the color-coded code blocks. However, since the question code block is featured first, it makes sense that it was so regularly referenced. Additionally, many participants remarked that the "best" code sample did not relate to the prompt enough for them to understand. This is helpful feedback to have from novice programmers dealing with seemingly dissimilar programs. Four participants remarked that they did not read the "best" code sample because it dealt with numbers, while the prompts of the exercise dealt with printing text. However, the entire anatomy of a function (using parameters and returns) was only present in the "best" code sample.

Summary of Coding Activity Findings

In general, the nuances of this particular Stack Overflow page and the limitation of only using one resource, proved extremely challenging for the participants. The majority of changes in the participants' behavior are more readily attributed to the design of the exercise and the content of the Stack Overflow page. However, for participants who did discover the hover-over tabs, their patterns in the reading the code were directly related to the design changes.

Findings from Post-Activity Interview

The six participants of phase 3 engaged in a post-activity interview.

Mismatched Example

A recurring problem for numerous participants was feeling that the code samples did not match their given prompts. All of the content that needed to go into their functions was provided in their starter code, they only needed to create the function around that start code. However, translating the functions in the code samples, to their prompts was commonly too much of a leap. One participant was annoyed that the best answer used code that was different from the code in the question and stated, "*I don't want new code introduced if I'm already trying to figure out the first code.*" Similarly to the issues noted about the "best" code sample, a participant remarked "*I only read the code description if it seems like it does what I'm trying to do.*" This participant reflected on why she did not read the "best" code sample. She stated, "*The*

Happy Birthday program has nothing to do with numbers, so why would I look at code with lists of numbers?"

Having a Different Plan

Participants that had a little more experience with coding actually approached the coding prompts with a plan. In one instance, the participant wanted to use global variables to avoid having to pass values through function parameters. In the second instance, the participant wanted to define a function within another function to also avoid parameters. Both participants spent significant time scanning the resource for any information that would help them execute their plans. Since the resource did not have any of these details, they both had to reroute their plans to include using parameters. When I asked about this dilemma they dealt with, they both replied that they would have used a second resource to continue to execute their original plan. When reflecting on the challenges of this exercise, the participant said, "*honestly, the most challenging part was that I couldn't just Google* "*how do functions operate in python" because I wanted to go back to the global variable method.*"

Looking for Tutorials

Compared to the interviews of phase 1 and the pre-activity interviews of phase 3, the post-activity interviews for phase 3 yielded more requests for step-by-step tutorials. One participant stated, "*The hover-overs were helpful, but it wasn't granular enough. I like the videos where it takes you through the steps*." Specifically, the request for videos was echoed by other participants from this phase. There are two possible reasons for this somewhat sudden wish for tutorial-style materials. One is that the limitation of using one resource made them wish they had a stronger foundation. A participant noted, "*I wish I had just watched a 4-minute video on how functions work in Python before we started this exercise.*" Or the second possible reason is that this phase of participants included more participants who are new to the basics of coding.

Summary of Post-Activity Interview Findings

The underlying challenge for all participants in this study was that they felt limited by only being able to reference one resource. This was reiterated in every post-activity interview and is the underlying cause for the other challenges and behavior changes listed in the findings. By eliminating the "searching for resources" aspect of their behaviors, more concentration was put toward their coding and toward reading the resource in detail. The resource was not specifically tailored to the exercise prompts, but all information necessary for completing the program was available in the Stack Overflow page. However, many participants noted the difficulty and the barrier of having to translate seemingly unrelated examples to their current code.

Chapter 7: Conclusion

The work I completed in this study could be used as a source for numerous new directions of study. Additionally, there are areas of this study that could have been improved or clarified. This study can also be used as reference for any future work that involves the population of adult novices self-teaching computer science.

Discussion

Some ideal next steps for this work would be to apply the designs to more just-in-time resources and see how it affects the coders' experiences. It would also be important to introduce a mechanism for measuring the effectiveness of the material with and without the design changes. The existing study explores how the experiences differ but does not measure whether the participants had any sort of improvement in their abilities to complete the coding activity. Seeing if the designs have any bearing on the users' learning could be useful for any designers of just-in-time learning materials.

There were numerous themes from this study that did not result in further design decisions. These themes could yield new research and findings about self-teaching novice coders and online learning resource design.

A significant area that I did not explore further was the pattern of behaviors that have to do with searching for resources. The searching process really impacted the directions taken by the participants and was a key determinant as to whether participants were going to become distracted with other obstacles. There is potential for a project that attempts to remove barriers from the searching process for novice coders without just curating the content in a site like CodeAcademy or Code.org.

There were also numerous themes from the findings in relation to adjusting code and browsing the resource that were not addressed in this work. The following themes did not result in a direct design decision for this study:

- Returning to resources
- Interacting with the compiler
- Programming the assignment
- Talking the problem aloud

There is potential for intentional design that supports these elements of the novice coder experience.

Additionally, these findings mostly relate to just-in-time learning materials. However, there is potential for these findings to be applied and tested on linear tutorials and coding courses.

Limitations

There were numerous limitations associated with this study. While none are so significant as to undermine the findings, it is important to articulate each limitation so as to understand the bounds of this work. A challenge of this study was adjusting the exercise for different coding languages and for different participant skill levels. In terms of adjusting for different coding languages, there was a particular challenge in translating the coding exercises between JavaScript versus Java and Python. JavaScript is generally used to manipulate the content of a web page or HTML file. Those who had exclusively studied this language were not particularly familiar with programs that print to the console exclusively. This difference in experience created a hurdle for those using JavaScript to conceptualize the purpose of the prompts in the exercise. It also provided a unique challenge for gathering user input. This language has numerous ways to get user input, and if the version I chose for the code samples were not what the participants are familiar with, there was an extra challenge for them in the exercise.

Secondly, I had to be prepared to simplify the coding prompt depending on participant ability. This adjustment could take the form of allowing the participants to stop when they were stuck or to open and view the code provided in future prompts (Appendix E) before successfully completing the code challenges for the first prompts. I also asked additional probing questions to get more levels of detail in what the participants did or did not understand about the code they were manipulating.

A significant limitation of my study that should be addressed with any future work was the design of the phase 3 experiments. The behaviors and practices of the participants in this study were inauthentic in that they only had access to a single resource. This change was problematic for two reasons. The first reason was that cutting out the searching step of the process for the participants actually eliminated some of their self-education and problem solving strategies. The participants used online searches to inform and improve further topic discovery. By having a singular resource, they were not able to broaden their understanding of options. The lack of options was evident from the instances described from the study where participants were looking for specific solutions. Participants took significant time searching for a specific, alternative solution in the provided resource, instead of adopting the solution that was evident from the code provided. The second reason this change was an issue was that participants altered their natural behaviors. One participant noted, "*I used this [resource] like they were the only notes I took into a test with me, which is never the case when I'm trying to code for work*". The observations of participants reading the resource linearly and revisiting the resource while working on their code many times in a row are unique to the study design and would not be particularly replicated if participants had more resource options.

If this study were to be replicated, I would advise applying the same interface designs to multiple Stack Overflow pages that show a wider variety of examples of code.

An important takeaway from this study was that novice coders rely on code samples being recognizably similar to their own projects. When a variety of resources are available, participants will not spend time trying to take leaps to relate to programs that deal with significantly different data. So in any new versions of the experiment, I would suggest finding Stack Overflow pages that use similar data to the prompted assignment.

A limitation in the resource design was the discoverability of the hover-over tabs. There were repeated times where participants would ask a question out loud that could be answered by the information in the hovers, but the participants did not notice the pop-up. When this occurred some participants would later find the answer in the text of the page, others would forge ahead with their questions going unanswered. The usability of the tabs was successful for those who found them. I think an improved design would be hyperlinking all of the text in the code blocks with the mouse icon changing to the clicking hand. By clicking a line of code, a panel to the right of the code will open and populate with the most thorough definition that can be aggregated from the content available. This design more closely mimics the designs in the sketch found in Figure 3.

The accessibility of the design choices were not explored in this study. Two aspects of the design solely relied on color. The visual cue that code was or was not likely to execute correctly was a red or green colored border. This messaging relies on cultural understanding of those colors and having acute color recognition. Similarly, the visual cues to help dissect the components of the code samples also relied on having acute color recognition or distinction. This aspect of the code sample design was more subtle and less vital than recognizing if code would execute or not. Overall, the accessibility of the design changes, and design of just-in-time resources in general could be analyzed with the sole purpose of improving accessibility.

Implications

One of the most important findings from this study is the relationship between a novice coder's motivations for learning and their self-teaching technique. This finding could inform work for designers of both just-in-time resources and linear tutorial experiences. Designers could tailor resources to better support proactive versus reactionary self-teachers. Designers, teachers, and anyone who interacts with self-teaching adult novices should be aware of this distinction so that they can provide materials that support the learners' goals.

Additionally, in this study I was able to aggregate some user opinions about what makes a useful and unuseful resource. The distinctions between useful and unuseful could be used by future resource designers as they are creating materials or by teachers when they are suggesting resources to novices.

Supervisors or teachers of adult novice coders should also be aware of the amount of time and effort spent searching and evaluating self-teaching resources when facing a coding challenge. In a scenario where a supervisor is asking a novice coder to work on something that requires a new skill, they should be aware of the added time that will be used to aggregate useful resources on the topic. This process may affect their expected timelines and potentially prompt the supervisors to provide more resources from the outset.

Participants in phase 3 who are more reactive in their self-teaching methods started expressing interest in having a linear tutorial experience after becoming frustrated with the lack of quick, just-in-time options. Instructional or interface designers of linear tutorials may adjust their products to be prepared for scenarios where reactive self-teachers turn to their resources.

The findings from phase 1 could result in numerous new iterations and designs that tackle one of the many barriers found in the self-teaching process. Two of the sketches created in phase 2 were out of scope for this thesis project but could be used as a future project.

A Final Thought

As the technology field continues to grow, research should continue to focus on the population of adult novice learners that are self-teaching computer science. Through gaining further understanding of how novice coders use just-in-time resources, we can discover design changes that improve their learner experience. This process can, in turn, prepare learners with a stronger foundation for future computer science education.

Appendices

Appendix A: Participant Recruitment Text

Teaching Yourself to Code?
Consider being a part of a 1-2 hour study where we observe and test applications of people teaching themselves to code.

We are especially interested if you are teaching yourself but are considering formal computer science education in the future.

If you participate you will receive a $15 gift card.
If interested email: clindema@umd.edu

Appendix B: Participant Consent Form

CONSENT TO PARTICIPATE

Project Title	Self-Teaching Computer Science
Purpose of the Study	*This research is being conducted by **Carrie Lindeman** at the University of Maryland, College Park. We are inviting you to participate in this research project because you are an adult who is teaching yourself computer science. The purpose of this research project is to improve computer science learning resources.*
Procedures	*The procedures involve answering questions about your computer science education and background for 15-20 minutes. Completing computer science practice problems while being recorded on video for 20-30 minutes. We will use the attached laptop camera to record visual and audio of you during the exercise OR will record a video conference with screen recording and camera of participants device. Answering follow-up interview questions for 20 minutes. The total amount of time spent participating in the study will not exceed 1 hour and 30 minutes.*
Potential Risks and Discomforts	*There is no more than minimal risk associated with participating in this study.* *There is a possibility of breach of confidentiality, but the steps taken to minimize this include only having researchers review recordings and notes from the participant sessions. Recording and saving video recordings locally on a password protected device and on a password protected and encrypted cloud storage service. Deleting any footage where participants reveal personally identifiable information.*
Potential Benefits	*There are no direct benefits from participating in this research. However, possible benefits include improved experience with just-in-time computer science learning solutions.*
Confidentiality	*All information collected in this research is confidential, and participants will not be identified by their name or other identifiable indicators. All data will be accessible only by the researchers.* *Any potential loss of confidentiality will be minimized by storing data in password protected cloud storage.* *If we write a report or article about this research project, your identity will be protected to the maximum extent possible. Your information may be shared with representatives of the University of Maryland, College Park or governmental authorities if you or someone else is in danger or if we are required to do so by law.*

Compensation	*You will receive $15 gift card. You will be responsible for any taxes assessed on the compensation.* *If you will earn $100 or more as a research participant in this study, you must provide your name, address and SSN to receive compensation.* *If you do not earn over $100 only your name and address will be collected to receive compensation.*
Right to Withdraw and Questions	*Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify.* *If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the investigator:* ***Carrie Lindeman*** ***3520G Van Munching Hall*** ***7699 Mowatt Ln, College Park, MD 20742*** ***clindema@umd.edu*** ***717-253-7886***
Participant Rights	*If you have questions about your rights as a research participant or wish to report a research-related injury, please contact:* University of Maryland College Park Institutional Review Board Office 1204 Marie Mount Hall College Park, Maryland, 20742 E-mail: irb@umd.edu Telephone: 301-405-0678 *For more information regarding participant rights, please visit:* https://research.umd.edu/irb-research-participants *This research has been reviewed according to the University of Maryland, College Park IRB procedures for research involving human subjects.*
Statement of Consent	*Your signature indicates that you are at least 18 years of age; you have read this consent form or have had it read to you; your questions have been answered to your satisfaction and you*

	voluntarily agree to participate in this research study. You will receive a copy of this signed consent form. *If you agree to participate and agree to being video recorded, please sign your name below.*	
Signature and Date	**NAME OF PARTICIPANT** **[Please Print]**	
	SIGNATURE OF PARTICIPANT	
	DATE	

Appendix C: Demographic Survey

Survey

Full name:

Age:

Gender:

Race:

- ❏ White
- ❏ Hispanic or Latinx
- ❏ Black or African American
- ❏ Native American or American Indian
- ❏ Asian / Pacific Islander
- ❏ Other

Highest level of education reached:
Institution:
Major or area of focus:

Current Occupation:

What have you used to teach yourself computer science?

- ❏ Online classes
- ❏ Textbook
- ❏ Forums
- ❏ Online Guides
- ❏ Tutorial videos
- ❏ Googling specific questions
- ❏ Free classes from universities

What are your motivations for learning computer science?

How do you plan to use your computer science knowledge?

Appendix D: Semi-Structured Interview Questions

Semi-Structured Interview Questions

General Questions

- How have you approached learning computer science?
- What resources do you use?
- What websites do you use?
- How often do you work on learning computer science?
- What are your goals for your computer science education?
- Do you have plans for formal computer science education?
- Do you use computer science in your job?
- How long have you been trying to learn?
- What’s the most frustrating about teaching yourself?
- What’s the most rewarding part of learning computer science?

Exercise Based Questions

- What did you find most challenging about this exercise?
- When did you feel the urge to reach for help?
- Can you describe the experience of using the support technology/websites?
- What could have made this process easier for you?
- How will you move forward to practice these skills?
- What’re you looking for when you Google something? What makes the resource useful or not useful?
- What resources do you like and why?
- When you feel like you need help, where do you go? Any sites?

Appendix E: Code Activity Prompts

The following exercises are adapted from Dr. Katherine Gibson from University of Maryland, Baltimore County. (Gibson, 2019)

Python

Program:

```
print("Happy birthday to you!")
print("Happy birthday to you!")
print("Happy birthday, dear Maya...")
print("Happy birthday to you!")
```

Prompt 0:

- What does this program do?
- What do you think about this program's design?
- How would you change or improve this program? Why?
- Have you written programs like this one before?

Prompt 1:
Rewrite this program so that it uses a function to print the "Happy birthday to you!" lyric.

Program:

```
birthdayName = input("Whose birthday? ")
print("Happy birthday to you!")
print("Happy birthday to you!")
print("Happy birthday, dear "+birthdayName+"...")
print("Happy birthday to you!")
```

Prompt 2:
Rewrite this program so that it uses a function to get the user input.

Program:

```
birthdayName = input("Whose birthday? ")
print("Happy birthday to you!")
print("Happy birthday to you!")
print("Happy birthday, dear "+birthdayName+"...")
print("Happy birthday to you!")
```

Prompt 3:
Rewrite this program so that it uses functions to get the input and print the song lyrics. The suggested functions are “getBirthdayName” and “printLyrics”.

Prompt 4:
Explain a code trace of your final program, (can include drawing or doing a stack trace). Explain why you would break this program up into functions.

Anticipated final program:

```
def getBirthdayName():
   birthdayName = input("Whose birthday? ")
   return birthdayName

def printLyrics(name):
   print("Happy birthday to you!")
   print("Happy birthday to you!")
   print("Happy birthday, dear "+name+"...")
   print("Happy birthday to you!")

printLyrics(getBirthdayName())
```

JavaScript

Program:

```
<script>
console.log("Happy birthday to you!");
console.log("Happy birthday to you!");
console.log("Happy birthday, dear Maya..");
console.log("Happy birthday to you!");
```

```
</script>
```

Prompt 0:

- What does this program do?
- What do you think about this program's design?
- How would you change or improve this program? Why?
- Have you written programs like this one before?

Prompt 1:

Rewrite this program so that it uses a function to print the “Happy birthday to you!” lyric.

Program:

```
<script>

    function go(){
        var name = document.getElementById("firstname").value;
        console.log("Happy birthday to you!");
        console.log("Happy birthday to you!");
        console.log("Happy birthday, dear "+name+"...");
        console.log("Happy birthday to you!");
    }

</script>

<html>
First name:<br>
<input type="text" name="firstname" id="firstname"><br>
<button onclick="go()">Submit</button>
</html>
```

Prompt 2:

Rewrite this program so that it uses a function to get the user input.

Program:

```
<script>
```

```
    function go(){
        var name = document.getElementById("firstname").value;
        console.log("Happy birthday to you!");
        console.log("Happy birthday to you!");
        console.log("Happy birthday, dear "+name+"...");
        console.log("Happy birthday to you!");
    }

</script>

<html>
First name:<br>
<input type="text" name="firstname" id="firstname"><br>
<button onclick="go()">Submit</button>
</html>
```

Prompt 3:
Rewrite this program so that it uses functions to get the input and print the song lyrics. The suggested functions are “getBirthdayName” and “printLyrics”.

Prompt 4:
Explain a code trace of your final program, (can include drawing or doing a stack trace). Explain why you would break this program up into functions.

Anticipated final program:

```
<script>

    function getBirthdayName(){
        return document.getElementById("firstname").value;
    }

    function printLyrics(){
        var name = getBirthdayName();
        console.log("Happy birthday to you!");
```

```
        console.log("Happy birthday to you!");
        console.log("Happy birthday, dear "+name+"...");
        console.log("Happy birthday to you!");
    }

</script>

<html>
First name:<br>
<input type="text" name="firstname" id="firstname"><br>
<button onclick="printLyrics()">Submit</button>
</html>
```

Java

Program:

```
public class HBD
{
   public static void main(String[] args)
  {
    System.out.println("Happy birthday to you!");
    System.out.println("Happy birthday to you!");
    System.out.println("Happy birthday dear Maya...");
    System.out.println("Happy birthday to you!");
  }
}
```

Prompt 0:

- What does this program do?
- What do you think about this program's design?
- How would you change or improve this program? Why?
- Have you written programs like this one before?

Prompt 1:

Rewrite this program so that it uses a function to print the "Happy birthday to you!" lyric.

Program:

```
import java.util.Scanner;

public class HBD {
    public static void main(String[] args) {

      Scanner input = new Scanner(System.in);
      System.out.print("Whose birthday? ");
      String birthdayName = input.next();

      System.out.println("Happy birthday to you!");
      System.out.println("Happy birthday to you!");
      System.out.println("Happy birthday dear " + birthdayName +
"...");
      System.out.println("Happy birthday to you!");
    }
}
```

Prompt 2:
Rewrite this program so that it uses a function to get the user input.

Program:

```
import java.util.Scanner;

public class HBD {
    public static void main(String[] args) {

      Scanner input = new Scanner(System.in);
      System.out.print("Whose birthday? ");
      String birthdayName = input.next();

      System.out.println("Happy birthday to you!");
      System.out.println("Happy birthday to you!");
      System.out.println("Happy birthday dear " + birthdayName +
"...");
      System.out.println("Happy birthday to you!");
    }
}
```

Prompt 3:

Rewrite this program so that it uses functions to get the input and print the song lyrics. The suggested functions are “getBirthdayName” and “printLyrics”.

Prompt 4:
Explain a code trace of your final program, (can include drawing or doing a stack trace). Explain why you would break this program up into functions.

Anticipated final program:

```
import java.util.Scanner;

public class HBD {
    public static String getBirthdayName(){
        Scanner input = new Scanner(System.in);
      System.out.print("Whose birthday? ");
      String birthdayName = input.next();
      return birthdayName;
    }
    public static void printLyrics(String name){
        System.out.println("Happy birthday to you!");
        System.out.println("Happy birthday to you!");
        System.out.println("Happy birthday dear " + name + "...");
        System.out.println("Happy birthday to you!");
    }
    public static void main(String[] args) {
      printLyrics(getBirthdayName());
    }
}
```

Appendix F: Searched Phrases

- Alert trigger html
- Alert function javascript
- Alert Trigger HTML
- alert function javascript html
- Javascript function html for alert
- multiple java alert boxes in a row
- How to code a function java
- how to call a function in java
- for loop java
- defining variables in python
- defining variable passed into function python
- Define function java
- define function java stackoverflow
- Java insert variable into list
- Java insert variable into string
- java call function within a class
- java program
- Iterate over range integers python
- java iterate over range integers
- define class attributes java
- pass parameters in java class
- does function have to have return type java
- How to pass return type java method
- set void return type
- error, class, interface, or enum expected
- instantiateclass java
- java cannot find symbol
- define string in java
- how to load main class in java
- function print javascript
- vs code run command
- vs code run code
- vscode hotkey cheat sheet
- visual studio code run code mac
- print in vs code
- script tags
- script tags
- print hello world in javascript
- code snippet for javascript function
- code snippet for javascript function that prints a string
- javascript print using a function

References

2020 Online Education Trends Report (p. 26). (2020). BestColleges. https://res.cloudinary.com/highereducation/image/upload/v1584979511/BestColleges.com/edutrends/2020-Online-Trends-in-Education-Report-BestColleges.pdf

Antonis, K., Daradoumis, T., Papadakis, S., & Simos, C. (2011). Evaluation of the effectiveness of a web-based learning design for adult computer science courses. *IEEE Transactions on Education, 54*(3), 374–380. https://doi.org/10.1109/TE.2010.2060263

Benigno, V. & Trentin G. (2000). The evaluation of online courses. *Journal of Computer Assisted Learning, (16)*3, 259-270.

Bureau of Labor Statistics, U.S. Department of Labor. (2019). Computer and Information Technology. Occupational outlook handbook, Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Chatterjee, P., Kong, M., & Pollock, L. (2020). Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software, (159)*, 1-13.

Darby, R. (2018, February 20). Knowledge on demand: What does 'just-in-time learning' mean for your corporate training? *EDGE Learning Media*. https://edgelearningmedia.com/articles/knowledge-demand-just-time-learning-mean-corporate-training/

Gibson, K. (2019). *CMSC201 Computer Science I for Majors Lecture 10 – Functions* [PowerPoint Presentation]. https://www.csee.umbc.edu/courses/201/spring19/docs/slides/CMSC%20201%20-%20Lec10%20-%20Functions.pdf

Glassman, E., & Russell, D. (2016). DocMatrix: Self‐teaching from multiple sources. *Proceedings of the Association for Information Science and Technology, 53*(1), 1–10. https://doi.org/10.1002/pra2.2016.14505301064

Guo, P. J. (2017). Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 7070–7083. https://doi.org/10.1145/3025453.3025945

Larson, Q. (2017, May 4). *The 2018 New Coder Survey: 31,000 people told us how they're learning to code and getting jobs as…*. FreeCodeCamp.Org. https://www.freecodecamp.org/news/we-asked-20-000-people-who-they-are-and-how-theyre-learning-to-code-fff5d668969/