# DataSnap: Enabling Domain Experts and Introductory Programmers to Process Big Data in a Block-Based Programming Language

Jonathon D. Hellmann

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Eli Tilevich, Chair
Dennis G. Kafura
Clifford A. Shaffer

June 5, 2015
Blacksburg, Virginia

Keywords: block-based languages, end-user programming, big data, Snap!

# DataSnap: Enabling Domain Experts and Introductory Programmers to Process Big Data in a Block-Based Programming Language

Jonathon D. Hellmann

Abstract

Block-based programming languages were originally designed for educational purposes. Due to their low requirements for a user's programming capability, such languages have great potential to serve both introductory programmers in educational settings as well as domain experts as a data processing tool. However, the current design of block-based languages fails to address critical factors for these two audiences: 1) domain experts do not have the ability to perform crucial steps: import data sources, perform efficient data processing, and visualize results; 2) the focus of online assignments towards introductory programmers on entertainment (e.g. games, animation) fails to convince students that computer science is important, relevant, and related to their day-to-day experiences.

In this thesis, we present the design and implementation of DataSnap, which is a block-based programming language extended from Snap!. Our work focuses on enhancing the state of the art in block-based programming languages for our two target audiences: domain experts and introductory programmers. Specifically, in this thesis we: 1) provide easy-to-use interfaces for big data import, processing, and visualization methods for domain experts; 2) integrate relevant social media, geographic, and business-related data sets into online educational platforms for introductory programmers and enable teachers to develop their own real-time and big-data access blocks; and 3) present DataSnap in the Open edX online courseware platform along with customized problem definition and a dynamic analysis grading system. Stemming from our research contributions, our work encourages the further development and utilization of block-based languages towards a broader audience range.

This project has been supported in part by the NSF Award #1444094.

# Dedication

To my little one, Andrew, who has given me the greatest sense of joy and delight than I have ever known.

# Acknowledgments

This Master's thesis has been quite the whirlwind of an experience. One moment it seems that I am starting out on this grand and adventurous quest, and the next moment I am standing among friends after my thesis defense reminiscing about all of the late nights without any sleep, the numerous projects shared with friends in the past, and many lunch discussions which were just as likely to be inspiring as they were to be outrageous. This isn't to say that it was a short experience or period of time, just that the time flew by as it seems to do when one is thoroughly engaged in a task (as well as engaged in quite a few other life changing events like the birth of our first child). However, through all of my up and downs, my triumphs and failures, my loving wife has always been there for me to encourage me and to push me to be my best. Thank you, Juliana, for your understanding, your gentle heart, and for everything that you have done for me. You are truly the best gift that God has given to me.

I also now want to give the biggest thanks to my parents, as I am who I am today only because of their immense sacrifice for my well-being. Through the last few weeks of raising our now two-week old son, I have gained the first-hand experience and understanding of the magnitude of the sacrifice that you have given for me. Thank you so much for everything and I will be sure to pass this same love and efforts spent forward to the next generation.

To my dear friend, Harshal Hayatnagarkar, whose guidance and wisdom has been like a lantern in a dark tunnel at times when I felt like I had forgotten the way forward. I will never forget our Friday seminar shenanigans, how you taught me algorithmic strategies for how to get the most bang for your buck during numerous Chipotle to-go orders, and our good times laughing together at a local restaurant or at the lab. Thank you for being there for me many times, just a phone call away, when I needed your technical expertise and wisdom throughout this project.

To my dear friend, Cory Bart, who essentially showed me the launching pad for my thesis idea and project. Your previous work provided me with a strong foundation from which to start, and your approachable persona and willingness to go over and teach me concepts greatly helped me throughout my work. Thank you for all of your generous help.

To Zheng Jason Song, whose incredible confidence in me inspired me to take on challenges that I didn't think I would be able to achieve. Thank you for your time spent in discussion,

on walks together, and for your camaraderie during this entire process.

I especially want to thank my advisor, Dr. Eli Tilevich, who has helped me significantly throughout my graduate school studies. You have kept me focused on the goal at times when my ambition would have me dream up plans that would most likely take years more than my allotted study time to accomplish. To my and our research group's benefit, you also have quite the way with words, as I knew that a proposal or paper draft revision with your help can experience quite the positive impact, perhaps even disguising that it's original author was even a graduate student who is quite wordy and verbose at times. Thank you for your support, your sacrifice of time, and most importantly, your ability to somehow impart both humorous but true life lessons as well as practical lessons all in one conversation, time and time again.

I'd also like to thank the members of my committee, Dr. Dennis Kafura and Dr. Cliff Shaffer. Dr. Kafura had many meetings with me and was influential in making sure that I could clearly articulate my claims and purpose for my research. Dr. Shaffer, who also taught my online education systems graduate course, has been a great help in encouraging me to take my developed platform further and further towards its application in the educational realm. He has also been a great help in the editing of this thesis paper. Thank you to you both for your encouragement and time offered for the benefit of this thesis project.

I must also thank many other individuals within the Virginia Tech Computer Science department, especially Dr. Todd Stevens and Ms. Margaret Ellis. Our great times together teaching undergraduate courses I will never forget. You both have become my friends as well as my colleagues. Vivek Akupatni and Peeratham (Karn) Techapalokul, your help on parts of this thesis project towards its use on the Open edX platform has been very much appreciated. To my software engineering group at Virginia Tech, you have provided me with much inspiration and motivation in this research work. Lastly, I would like to thank all of my unmentioned family, friends, and colleagues who have helped me in ways large and small throughout these few years during my study at Virginia Tech.

# Contents

# List of Figures

# Chapter 1

# Introduction

Block-based programming languages were originally designed for educational purposes. Their user-friendly interface encourages and motivates non-computer professional users to learn programming concepts. Snap! [37] is a recent example of a block-based programming language that was developed by Jens Mönig at MioSoft Corporation and Brian Harvey at the University of California, Berkeley. It features a web-based interface for users to drag and drop blocks to form programs.

Users program by dragging and dropping blocks from a block palette onto a scripts panel to form a program. Figure 1.1 shows a simple block-based programming language problem as a demonstration. This particular problem notifies the user whether they should take their umbrella that day based on the chance of precipitation.


Figure 1.1: An example block-based programming language program

The program shown is built by snapping together different blocks. A block, for the purpose of block-based languages, is a method which can execute a series of commands, typically represented by a shape in the block-based interface. Thus the blocks in the example problem shown are: "when I am clicked", "if condition, then perform a set of commands", "is value 1 less than value 2?", etc.

When a user runs their program, each block is run in order from top to bottom, but blocks that are nested inside of other blocks are considered to be dependencies of the outer block and are executed first. When a block is invoked, the method written for that particular block is run, and a result may be rendered to the screen, a variable's value might be updated, or some other action specified by the block will take place.

Block-based language platforms are typically composed of a few different components. The main components of the Snap! web interface are a blocks palette, block categories panel, scripts panel, and a stage (Figure 1.2). The scripts panel is an area onto which users can drag and drop blocks to form their program. The block palette holds the blocks which the user can access. Above the block palette is the block categories panel, which is used to divide the blocks into groups. The stage displays any variables or sprites that have been added. Users drag blocks from the blocks palette over to the scripts panel, run their scripts, and results are typically updated on the stage on the right side.

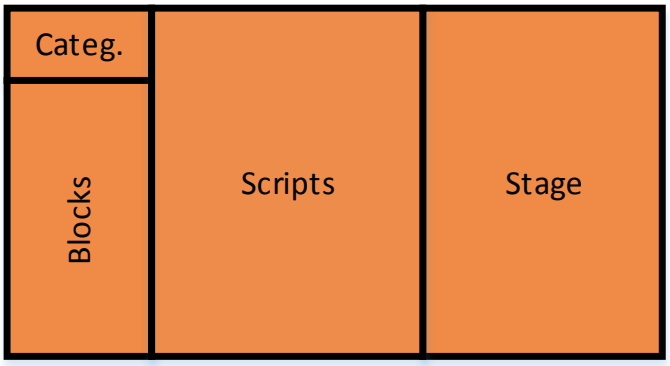



Figure 1.2: The main components of the Snap! web-based interface.

Block-based languages so far have been used mainly for educational purposes. A few online block-based programming platforms exist to aid this cause, such as Snap!, Scratch, Blockly, and Tynker. These platforms let students develop, run, and share their code with each other within the browser. These platforms are often used in conjunction with an educational course. A few example courses which pair block-based programming with teaching computer science are the "Introduction to Programming with Scratch in Education course from Google's Computer Science for High School (CS4HS) program and "Creative Computing" course for educators from the ScratchEd research team at Harvard both use Scratch [54], while code.org's "Hour of Code" and "20 Hour Curriculum" use Blockly [32].

## 1.1 Overview

*Block-Based Languages For a New Audience: Domain Experts and Professionals*

Can block-based languages be used by domain experts and professionals? Can these programming platforms be used in situations where professionals can develop meaningful programs? When was the last time you saw a block-based language that was not solely focused on teaching educational concepts? These questions, and other questions like these, delve into the existing problem that block-based languages as they currently exist are mainly focused on educational instruction and are underutilized in industry and in common society.

Block-based languages (e.g., Blockly, Snap!, Scratch, etc.) were originally designed for teaching purposes. Take Snap! as an example. Users of Snap! use intuitive drag and drop actions in order to snap blocks together to build programs. Snap! and other block-based languages are so successful in educational contexts because the syntactical errors inherent in a typical text based programming language are no longer present in a block-based language, and the user is presented with an easy to use interface in which to program. The affordances of block-based languages aid introductory students to more quickly learn computer science concepts in comparison to a traditional programming language like C, Java, Python, etc.

However, we believe the affordances of block-based languages are not reserved only for students, but can be used to aid professionals and domain experts also. In this paper, we describe how we extended Snap! from being an educational programming language into a powerful tool for domain experts and professionals, with little knowledge or experience in computing.

*Block-Based Languages For Introductory Programmers*

Furthermore, we also extend the state of the art in block-based languages for the previously mentioned audience of introductory students to address an existing problem: K-12 and online educational platforms (e.g. Massive Open Online Courses (MOOCs)) are both plagued by low retention rates and low interest in the computer science discipline. Educational theory posits that improper academic motivation models can be responsible for large percentages of students either becoming disinterested in computer science or never completing their courses.

Specifically, the current focus of online programming assignments on entertainment (e.g., games, animation, etc.) is particularly misdirected; it fails to convince students that the covered technical content is important, relevant, and related to their day-to-day computing experiences. After presenting our work towards DataSnap for the domain expert audience, this paper presents our work towards DataSnap for the introductory programmer audience. In this part of the paper, we incorporate real-time and big data in the context of big data

problem solving into the DataSnap platform. We present our openly-accessible data sets and our elementary blocks which can be used by teachers to create their own data sets within DataSnap. Then, we present sample problems to show the utility of DataSnap offered to introductory programmers, and discuss our architecture and implementation.

*Summary*

We believe that block-based languages can benefit a variety of audience types. Block-based languages have originally been used to aid in teaching computer science concepts. We believe the K-12 and introductory college communities which use block-based languages will benefit from the introduction of real-time and big data sources. We also believe block-based languages have not yet been presented in a favorable way to the professional community in industry and society in order to be effectively used by domain experts. Our work targets these two specific research areas. Thus the purpose of this thesis is summarized as follows:

> To study the challenges of block-based languages offered to two specific audiences: 1) domain experts and 2) introductory programmers.

## 1.2 Target Audience

We will now more clearly define our target audience of domain experts and professionals. First we will start with an overview of the personnel involved in domain expert work. In many corporations, domain experts rarely work alone, but work along with policy officials who need to make informed policy decisions as well as with technical teams that include software developers. The software developers are often tasked with developing platforms which can aid the domain expert in solving questions in their specific domain. Thus there is a chain of people involved in the efforts towards performing domain expert tasks.

Consider the following scenario as one such example which demonstrates block-based languages benefiting the entire chain of people involved in making a health policy decision. First we will introduce the un-optimized scenario, and then we will introduce the optimized scenario. The scenario goes as follows: A Centers for Disease Control and Prevention (CDC) policy official reviews the latest influenza-like-illness simulation results and makes note that a few parameters of a flu-epidemic simulation need to be changed in order to make an informed decision. An epidemiologist, on behalf of the the policy official, contacts the software development department and notifies the development team that specific parameters need to be changed. The epidemiologist in this scenario has little access to editing the parameters or methods of the simulation themselves. The software developers make the required changes, and the results slowly but surely percolate back to the domain experts and then back to the policy official who can review the simulation and make an informed health policy decision.

Now consider the following scenario, which represents the optimized version: A Centers for Disease Control and Prevention (CDC) policy official reviews the latest influenza-like-illness simulation results and makes note that a few parameters of the simulation need to be changed in order to make an informed decision. An epidemiologist working at the CDC opens up the simulation, which is presented in an online block-based interface. The block-based interface is specifically designed to make it easy for domain experts to tune simulation parameters. The epidemiologist makes the required parameter changes, and sends the simulation back over to the policy official, all within the same day. The policy official reviews the simulation and makes an informed health policy decision. Meanwhile, software developers continue to work behind the scenes to develop customized blocks which are tailored specifically for the epidemiologists' needs for their simulations.

Contrasting these two scenarios, we can see that in the second scenario the epidemiologist, who we consider the domain expert in this scenario, was given control over tweaking simulation parameters and methods. The epidemiologist was able to tweak the program as asked by the policy official without ever waiting on the software development team to develop a solution, as in the first scenario. The epidemiologist could also tweak the simulation as many times as they wanted without having to contact the software development team. In the second scenario, the time spent towards making the health policy decision was reduced and the communication overhead was reduced. Furthermore, the epidemiologist / domain-expert was left in charge of handling the domain that they knew best (epidemiology) while the software developer was left in charge of handling the tasks that they knew best (software development).

In both of these scenarios there were many people involved in making the health policy decision, and each one had a different level of both domain expertise and technical expertise. It is often the case that this differing level of expertise causes many communication issues: the domain experts have a hard time expressing the things they need, and the software developers have a hard time interpreting what it is that the domain experts need to have developed.

| | Domain Expertise | Technical Expertise |
|---|---|---|
| **Policy Official** | Low-Medium | Low |
| **Domain Expert (Non-Technical)** | High | Low |
| **Domain Expert (Power-User)** | High | Medium |
| **Developer** | Low | High |
| **Experienced Developer** | Low | High |

Figure 1.3: The spectrum of users for DataSnap and their domain and technical expertise.

However, computational thinking skills have become ever more important in industry as computers have touched and enhanced nearly every domain. This change has affected both the college curriculum and the workplace. Colleges are encouraging students to take computational thinking courses, and companies would benefit from hiring those that know basic programming or data analysis / manipulation skills. Because of these changes, the workplace has seen an increase in employees with computational thinking skills. Thus there are many employees who are an expert in a particular domain who also have an ability to perform basic programming functions. And there are also those that are more technically proficient yet have some basic domain knowledge. There is a spectrum of users with differing levels of domain-based and technical expertise (Figure 1.3).

This spectrum of users with varying technical and domain expertise is right where our work is situated: Our work aims to help bridge the gap between the spectrum of domain experts and software developers through a common medium that each user can play a role in. The online block-based interface will serve as a common ground [63, 58], providing both the domain experts and the software developers with a common base to communicate and exchange ideas through.

| | Responsibilities | Capabilities |
|---|---|---|
| **Policy Official** | Manages multiple domains. | Views results and makes informed decisions. |
| **Domain Expert (Non-Technical)** | Manages one domain. | Can edit block parameters. |
| **Domain Expert (Power-User)** | Manages one domain. | Can make their own custom blocks in the GUI if needed. |
| **Developer** | Develops block-based language platform. | Can edit the program source code to develop customized blcoks. |
| **Experienced Developer** | Develops and deploys block-based language platform. | Full capability to edit source code / deploy instances of platform. |

Figure 1.4: The spectrum of users for DataSnap and their responsibilities and capabilities.

Figure 1.4 further details the spectrum of users that we target for our online block-based interface called DataSnap and their work responsibilities and the capabilities they could be expected to perform in DataSnap. At the top are policy officials who manage many domains, but may have a low domain expertise in each specific domain. These officials generally do not have much time to learn the domain in detail because they have to manage many different systems. However, they are in charge of receiving advice from domain experts whom them manage, viewing results, and making informed decisions.

The next group is the non-technical domain experts. The non-technical domain experts typically manage one domain and have a high domain expertise but a low technical proficiency. Thus in our block based interface, most of the non-technical domain experts will not be tasked with performing any tasks that are deeply technical in nature. The non-technical domain experts can instead be expected to be capable of changing parameters in the block-based interface in order to suit their needs.

The following group is the power-user domain experts. This group has the same domain knowledge as the non-technical domain experts, but has a higher technical expertise. This technical expertise might come in the form of understanding computational thinking or computer science concepts, for example. This group will not shy away from performing basic programming or analytical tasks. This group will be expected to be able to make their own customized blocks within the DataSnap graphical user interface (GUI). These

customized blocks will serve as methods which can perform a function for the domain expert group.

Developers are the next group involved. Developers typically have a low domain expertise and a high technical expertise. They will not be able to solve domain specific questions but can provide the tools that the domain experts need in order to solve these questions. The developers can be expected to edit the program source code in order to develop customized blocks that cannot be created within the platform GUI. Developers are also responsible for adding to or changing the program GUI.

Next, the experienced developers have a low domain expertise and a high technical expertise as well. These developers have full capability to edit the program source code and to deploy instances of the tool on the web and various mobile devices. These developers will be responsible for the project architecture and deployment services of the block-based interface platform.

### 1.2.1 A comparison of DataSnap to Excel-based data processing

This section discusses the situations in which it would be most beneficial to use DataSnap, or most beneficial to use Excel or a similar data modeling tool such as JMP or LabVIEW.

First, the periodicity of the data must be taken into account. Let us take as an example a company that has some Excel-based spreadsheets that must be updated with new data. The Excel spreadsheets include formulas or macros that perform calculations in order to process the data. When working with data that is updated perhaps once a year or once every month, it is manageable to work in Excel. The data can be copied in, and the formulas and macros changed. If the data needs to be updated once every week or twice a week, the workload of copying, pasting, and updating all functions is still manageable, but there is significantly more work. Moving towards data that is updated once a day or several times a day, the workload quickly starts to become totally unmanageable as the time for data management surpasses the time involved for data analysis / user processing.

In comparison, a program built in an online block based language like DataSnap can take advantage of the simple and convenient methods of accessing and importing real-time data: through programming blocks. These data-access blocks can be utilized to acquire and update data hundreds or thousands of times per day. Furthermore, the imported data does not have to be constrained to the tabular format as it does in Excel. Users also do not have to be familiar with the names of macros or the required syntax in order to get the desired data. Customized blocks which utilize HTTP methods can even be made for data sources which currently are not in the block-based environment.

Excel-like programs for processing data will typically work best when the periodicity of the data is low, based on the overhead work of managing the data. Excel has very little support for receiving real-time and big data. Furthermore, for the real-time data that is available,

users do not know how to access this data because they have to type out commands in order to access it. Users might fail to develop these commands because they simply do not know what to type in order to specify the parameters and syntax that is required to get the information that they need.

In conclusion, an Excel-based approach may work best when the periodicity of the data is very low and the user is comfortable with having all of their data represented in a tabular format. An approach which uses DataSnap or a block-based programming platform will work best when the periodicity of the data is high, when users are working with data that cannot easily be represented in a tabular format, or when users prefer to have scaffolding for data import.

## 1.3 Research Contributions

In the following subsections we will describe the research contributions that this thesis presents. Our research contributions are divided into two parts: 1) DataSnap for Domain Experts; and 2) DataSnap for Introductory Programmers.

### 1.3.1 Research Contributions: DataSnap for Domain Experts

***Contribution 1***

*DataSnap abstracts a subset of the once very complex, unreachable big data problem solving environment, and provides this environment for use by domain experts, and those with little to no coding experience.*

Users of DataSnap are empowered with the ability to carry out a subset of big data processing operations. Hence, a wider community of users, currently lacking access to big data processing, or frustrated with typical data programming environments, should be able to wield the "big data" tool for their day-to-day tasks, previously only accessible to those experts in advanced computing technologies.

The previously mentioned big data problem solving subset which is abstracted, includes the following functionality:

1. Big data import into the problem solving environment
    - Compare managing and reading in files in Excel, R, Python, Java, Hadoop HDFS,

etc. to DataSnap's easy and convenient import of JSON and CSV data.

2. Big data processing methods
    - Compare select methods on tabular data, maximum, minimum, average, median, sum, and product methods in Excel, R, Python, Java, Hadoop, SQL to convenient processing methods in DataSnap
3. Results visualization
    - Compare writing code to develop visualizations in Python, Java, etc. to DataSnap's programmatic use of visualization blocks.

***Contribution 2***

*We are the first to use a block-based language (DataSnap) to process big data for domain experts in not just one specific domain, but present a generic solution which domain experts of multiple fields can use.*

Other researchers have used block-based languages to provide solutions in one specific domain, but these solutions cannot be transferred over to other domains. These solutions are built specifically for the researcher's particular domain. We present a generic solution which is not constrained to one particular domain.

***Contribution 3***

*We designed the interfaces in the Snap! runtime for big data import and results visualization; the interfaces are intuitive and easy-to-use.*

***Contribution 4***

*We designed a system architecture for DataSnap that allows users to develop and execute their own server-based "cloud" methods.*

Users are able to drag and drop blocks to form their own sequence of commands, which we call

"cloud" methods, that will execute on our compute-intensive processing server. Furthermore, the code input and code execution procedures are separated into different environments with different computational power, to speed up the data processing procedures.

Other researchers have used block-based languages to execute server-side scripts [81, 69, 80], but the user did not have the ability to use blocks to develop their own server-side scripts. DataSnap allows the user to define, save, and share their own customized cloud based algorithms.

***Contribution 5***

*We implemented the designed system and through case study we show its relevance and utility towards domain expert use.*

### 1.3.2 Research Contributions: DataSnap for Introductory Programmers

***Contribution 6***

*DataSnap integrates relevant social media, geographic, and business-related data sets into online educational platforms as a means of convincing introductory online learners that computing is a powerful tool for solving challenging and relevant problems across a variety of fields. DataSnap shifts the focus from over-implemented game and animation development towards real-world, big data problem solving.*

***Contribution 7***

*In addition to providing access to data from six real-time and big-data sources, DataSnap provides the necessary tools for others to develop their own real-time and big-data data access blocks. DataSnap is the first platform to offer both a sample of convenient data access methods and the tools to develop further data access methods for use by teachers in courses.*

## 1.4 Paper Roadmap

Previously in this thesis, we have introduced our target audience and our research contributions. The rest of this thesis is structured as follows. Chapter 2 reviews related work in this field. Chapters 3 and 4 discuss DataSnap as it is presented to the two different target audiences: domain experts and introductory programmers. Then, Chapter 5 presents our work towards hosting and grading DataSnap problems in the Open edX platform. Chapter 6 discusses opportunities for future work. Lastly, Chapter 7 presents our conclusions for this project.

# Chapter 2

# Related Work

## 2.1 Big Data Problem Solving

There are a few implementations of web-based, block-based programming environments that have an educational focus. Scratch [54], and Snap! [37] are both aimed primarily at the K-12 audience. Scratch supports project "remixing", which is similar to version control "forking" of projects, and has a strong social media presence which lets students comment and share projects. Snap!, the extended re-implementation of Scratch, allows users to build their own blocks, and is differentiated from Scratch through its capability for users to use first class lists, procedures, and continuations.

Another block-based language is Blockly [32]. Blockly is structurally similar to Scratch and Snap!, but has a few additional features such as the ability to convert blocks to JavaScript, Python, Dart, or XML. Unlike Scratch and Snap!, Blockly has less support for sprite animation. All three of these languages use drag and drop methods to snap together programming blocks, use color coding on different block types, and give users the ability to create procedures or functions. However, Scratch, Snap!, and Blockly have so far been geared towards game and animation development, while the block-based language that we present, DataSnap, is geared towards real-world, big data problem solving.

Other researchers have introduced approaches of using block-based languages as tools to teach a specific concept to students. Bags [33], a web based application for the teaching of relational operations and data analysis, allows users to snap blocks together in order to learn relational algebra. Bags shows a table of the data in what was previously the stage region of Snap!. DBSnap [71, 72], a program similar to Bags, also focuses on teaching relational algebra concepts however it's blocks are built in a tree-like structure. Blocks are snapped into the receiving nodes of each tree-like element. In DBSnap, blocks consist of either data set blocks or operator blocks, and programming is performed following the relational model of data. These two tools are specific in their focus of teaching the educational concept of

relational algebra, and thus are only focused on the data filtering process instead of the broader range of functions in which a block-based language is capable.

As previously mentioned in claim 1, DataSnap abstracts the big data problem solving environment for use by domain experts and those with little to no coding experience. The current big data problem solving environment includes working in R, Python, Java, Hadoop, or SQL environments [43, 51, 18, 57].

Some attempts at abstracting the big data problem solving environment include using higher-level languages. Take DryadLINQ or Matlab's matrix based operations for example [30]. DrayLINQ's goal is to make distributed computing on large compute clusters simple enough for every programmer. However, this environment is complex enough to only be suitable for those with a background in computing.

Labrinidis et. al. discuss the challenge data scientists have when data must be exported from an SQL database, processed, and then imported back into the SQL database [49]. In these cases, complex hand-coded SQL often must be generated in order to perform data analytics [67]. This tediously written code is error-prone and forces the analyst to focus more on managing their data than on analyzing their findings.

Dittrich et. al [24] discusses the "clear need of many organizations, companies, and researchers to deal with big data volumes efficiently." This author and many other authors proceed to teach techniques, such as job optimization, physical data organization, program self-tuning, and many others which will boost performance of Hadoop MapReduce jobs [24, 39, 28, 25]. Although performance benefits can be achieved by smarter programming methods, such performance benefits will only be acquired by those that are capable of setting up a system like Apache Hadoop. A wide audience of users will still not be able to use this tool because it is reserved for experts in technical fields.

In the preface of *Hadoop: The Definitive Guide* [83], the author mentions Hadoop's common theme: "Stripped to its core, the tools that Hadoop provides for building distributed systems - for data storage, data analysis, and coordination - are simple. If there's a common theme, it is about raising the level of abstraction - to create building blocks for programmers who just happen to have lots of data to store, or lots of data to analyze, or lots of machine to coordinate, and who don't have the time, the skill, or the inclination to become distributed systems experts to build the infrastructure to handle it."

Indeed, platforms like Hadoop have raised the level of abstraction for big data storage and processing, to the point where computer professionals need not be distributed systems experts themselves. However, Hadoop users must at least hold a high level of technical proficiency to setup the complex environment. Although our system only offers a subset of methods that a Hadoop environment offers, we hope to similarly raise the level of abstraction in the big data problem solving environment, and offer this environment for an even broader range of users, particularly, domain experts and non-computer-professionals.

Resulting visualizations exist in many different traditional and web-based programming plat-

forms. Matplotlib, developed by John D. Hunter, is a library for making 2D plots of arrays in Python [42], for example. Java Treeview allows the visualization of microarray data [68] for visualization problems. Plenty of other Java GUI-based visualizations have been developed. Web-based JavaScript visualization libraries include Highcharts [41], FusionCharts [64] and D3JS [13].

As mentioned in claim 2, we are the first to use a block-based language to process big data for domain experts in not just one specific domain, but present a generic solution which domain experts of multiple fields can use. In recent years, researchers have found the benefits of block-based languages in other application scenarios or domains for their user-friendly interfaces: 1) Marron et. al. and Bensalem et. al. have studied the potential usage of programming interactive applications using Blockly [56, 11]; 2) Silva et. al. [72] have studied using Blockly as a database query tool, while Harel et. al. [36] build a web service following the design pattern of Blockly; 3) Trower et. al. [78] studied using Blockly to control a bluetooth connection, which shows the great potential usage of Blockly in sensory data processing and the Internet of Things (IoT); 4) Varga et. al. have published a series of papers [81, 69, 80] concerning the specific field of materials design. Their approach also involves the separation of coding and code execution, but their platform is designed for only one domain-specific usage. 5) Agrawal et. al. [1] studied how to use Blockly to visualize fabrication. 6) Similarly, Leo et. al studied how to build bioinformatics applications on the Hadoop platform: Biodoop [50]. 7) Kim et. al studied programming robots with LEGO Mindstorms NXT hardware with a visual environment called mindstorms visual programming language [46].

Our project also falls in the realm of end-user software engineering (EUSE). Ko et. al. and Burnett et. al, respectively, discuss the current and future-oriented outlook on end-user software engineering and end-user programming [47, 14]. Lizcano et. al discusses a web-centered approach to EUSE and how to develop for end users without programming skills [52]. Alternatively, approaches towards improving data processing through the use of spreadsheets include [82, 16]. These solutions are promising for working with data that is in tabular format.

All the above mentioned approaches show that such block-based languages are no longer used solely as educational tools; researchers have seen their great advantage in user-friendly interfaces and are trying to apply such block-based languages for sensor control, visualization, interactive applications, and other domain-specific purposes. In this paper, we discuss how we extended Snap!, one of the block-based languages, adding to it the ability to serve domain experts in numerous fields with general data processing and visualization for domain experts. In order to realize our research goals, we designed DataSnap to provide the generic interfaces for importing, processing, and visualizing data.

## 2.2 Block-Based Languages for Education

From the educational perspective, our group has studied how to use real world big data as a data source to motivate beginners to learn computer programming. Bart et. al used real-time web data in introductory computer science projects [9], and discussed big data in the context of pedagogy and technology in computational thinking courses [8]. Similar studies involved creating introductory computer science projects that were stimulating, relevant, manageable, and which utilized large data sets [76, 77].

Many authors have discussed the high attrition rates in the computer science discipline [10, 12, 70, 65, 7]. Beaubouef and Mason discuss the high number of declared majors in computer science, yet the low number of graduates in the field. They proceed to investigate the possible causes for high attrition rates for computer science students [10]. Biggers et. al performs a study comparing graduating CS seniors with students that leave computer science.

The study included results such as: "Students who left the CS major have an overwhelming perception that CS is an asocial, coding-only field with little connection to the outside world", and "For the leavers, exposure to computer science was often limited to the intensely programming-focused first year courses. Students described these courses as full of assignments in which they could not see real world relevance" [12]. These results further indicate a need for more relevant problems and data for students for work with. Repenning et. al. focused on game design and computational thinking in public schools [66]. Some projects studied the effect of pair programming, citing increased retention rates and student confidence [59, 15], however, we believe these studies could benefit from addressing the issue of providing real-world, relevant problems to students in courses.

Other researchers have studied how to engage with students, how to enable and encourage success, and how to introduce real world problems in introductory courses [3, 10, 5, 35, 48, 38]. Anderson et. al studied the introduction of data programming in a CS1 course [2]. Dorling et. al. studied the transition from graphical based languages to text-based languages like Python [26] and Ball et. al. studied the use of the tile-based language named TouchDevelop [6]. The study involved the use of graphical languages in conjunction with, instead of in place of, text-based programming languages in order to draw shapes on a canvas.

Some researchers focused primarily on teaching educational concepts using block-based languages [62, 84, 55, 53, 34]. Other researchers studied women and minority groups in particular. Aspray et. al. studied the retention of underrepresented minority graduate students in computer science [4]. Cohoon studied strategies for how to improve female retention in the computer science major [19], and Fisher et. al. studied the influences and processes which cause women to either attach themselves or detach themselves to the field of computer science [29].

# Chapter 3

# DataSnap for Domain Experts

## 3.1 Introduction

Block-based programming languages were originally designed for educational purposes: their user-friendly interface encourages and motivates non-computer professional users to learn programming concepts. Due to their low requirements for a user's programming capability, such languages have great potential to serve domain experts with little programming knowledge as a data processing tool. However, the current design of block-based languages fails to provide domain experts with the ability to perform crucial steps: import data sources, perform efficient data processing, and visualize results.

We present the design and implementation of DataSnap, which is a block-based programming language extended from Snap!. The advantage of DataSnap lies in: 1) enabling domain experts to import data either from public data sets like Twitter, Google Geocoding, National Weather service, etc, or from the user's private data sets stored in the cloud; 2) leveraging the computational resources of cloud servers to speed up the processing of big data; 3) combining third-party visualization libraries written in JavaScript to provide domain experts with easy-to-use interfaces to visualize their results. Through a case study, we demonstrate how DataSnap can enable domain experts with little programming skills to analyze data and visualize results within a web browser in a "drag and drop" style.

To see the benefits of DataSnap, consider the following example. An epidemiologist has data on the flu from the Centers for Disease Control and Prevention's Influenza-like Illness Surveillance Program (ILINET). This data includes the total number of patient visits at a reporting location, the number of patients that had a positive indication for the flu, and the percent weighted influenza-like illness (ILI). The data source includes data reported each week since 1997, broken down by the 10 health and human services (HHS) regions of the United States. Suppose, for example, the epidemiologist would like to determine which week was the peak flu week in each of the past 15 years, and visualize the distribution of the flu

by HHS region in an interactive world map.

There have been some spreadsheet (e.g., Excel)-based approaches for this purpose. However, these approaches are not as intuitive as using the "drag and drop" method in Snap!. Other approaches involve pre-configured social media analysis [22, 21, 20], but DataSnap's intention is to let the end-user define the algorithms involved in the analysis.

We want to enable such domain-specific users to: 1) easily import their data, and use the "drag and drop" method to express how they want to process their data; 2) be able to use external compute resources to speed up the calculation procedure; and 3) easily visualize their results.

Although currently Snap! as a programming language can easily enable users to write programs using "drag and drop" methods which require no programming background for domain experts, it fails to satisfy the above mentioned requirements: 1) it is designed without an interface to import big data; 2) the runtime of the language only uses the resource of the browser, or say, the desktop of users, which is not enough to process big data; 3) it currently only supports the visualization of simple strings, numbers, and list items, with no support for maps / ordinary figures like bar charts, line graphs, and pie charts.

In this paper, we mainly focus on how to enable Snap! as a powerful tool for domain experts to process big data. In specific, to deal with the above mentioned research issues, we 1) designed the big data import interface / data visualization interfaces for the Snap! language runtime; 2) modified the runtime architecture of Snap!: we separated the browser-based programming interface of inputting code and the actual code execution to different environments, the interactive coding procedure still uses Snap! while some of the code execution procedures are offloaded to the cloud to take advantage of superior computational resources. We implement the proposed interfaces and architecture, and through a case study we prove that DataSnap can successfully provide an easy-to-use data processing platform for domain experts.

The contribution of this paper is four-fold:

- We are the first to use Snap! to process big data for domain experts in not just one specific domain, but to present a generic solution which domain experts of multiple fields can use.
- We designed the interfaces in the Snap! runtime for big data import and results visualization; the interfaces are intuitive and easy-to-use.
- We designed a system architecture for DataSnap, so that the code input and code execution procedures can be separated into different environments with different computational power, to speed up the data processing procedures.
- We implemented the designed system and through case study we show its relevance and utility.

Section 2 presents an overview of DataSnap, which includes a basic architecture and description of the elements added to block-based languages. Sections 3 and 4 present the interface design for data import and visualization, respectively. Section 5 proceeds to discuss the system design for leveraging cloud resources in big data processing. Section 6 presents a relevant case study, and Section 7 concludes the chapter on DataSnap for Domain Experts.

## 3.2 DataSnap Overview

First we will briefly introduce the architecture of DataSnap. DataSnap is a block-based language based on Snap!. Therefore, just like Snap!, it is composed of JavaScript files, which include Snap!'s JavaScript sources as its core. However, to achieve the aforementioned functionality, we add three JavaScript modules as plugins to Snap: the data input module, the visualization module, and the cloud processing module, as shown in Figure 3.1.

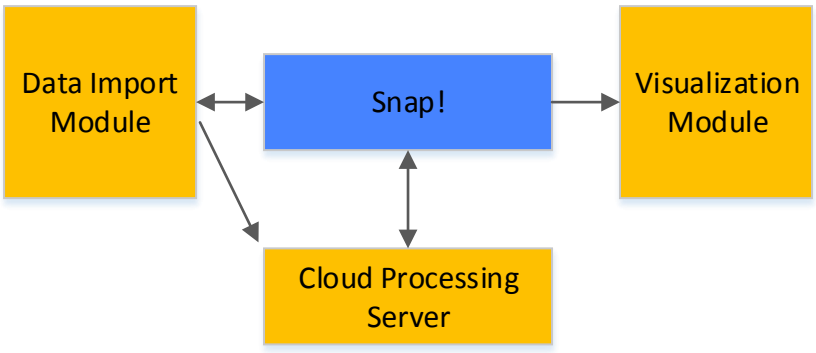



Figure 3.1: The DataSnap architecture

When a user runs DataSnap in a browser, the DataSnap user interface is almost the same as that of Snap!, except for five new categories in the category palette, along with their corresponding blocks in the block palette. As shown in Figure 3.2, three new categories of blocks are added that are mainly designed for server-side functionality: Cloud Processing, Cloud Variables, and Cloud Data Import; while two new categories of blocks are added that are mainly designed for client-side functionality: Data Import and Visualization.

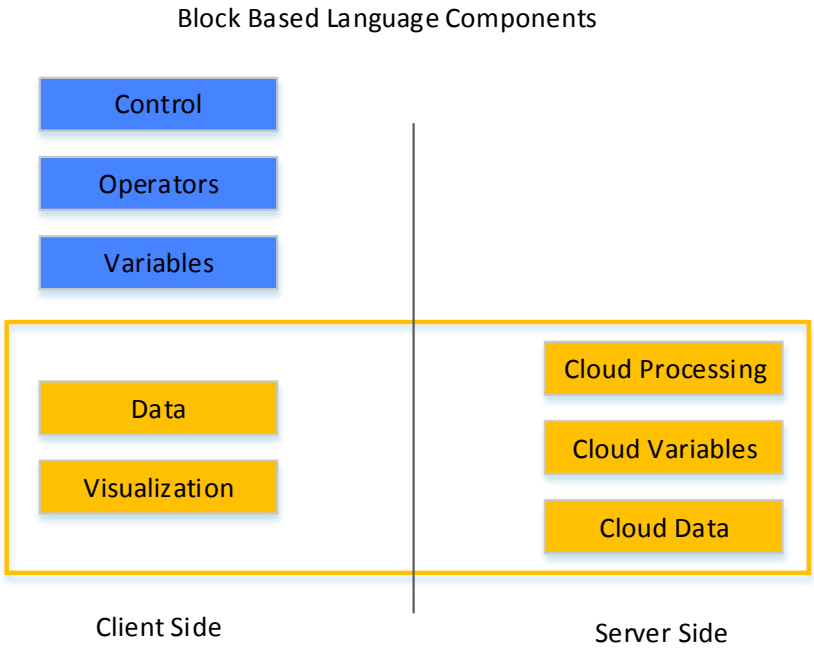



Figure 3.2: The contribution of server-side general purpose domain expert cloud processing blocks and visualization tools for the first time in block-based languages. The new elements to block-based languages are circled.

When a user runs DataSnap, the above mentioned blocks will enable him/her with the ability of performing the following functionality. We will introduce in detail how these funtionalities are achieved in the coming sections.

- Import user data directly to our server
- Leverage cloud resources in order to perform big data processing calculations and data filtering
- Send the resultant data from the server to the web-browser
- Generate a visualization of the data.

## 3.3 Data Import

Because domain experts and professionals commonly collect data specific to their fields, they need to be provided with a convenient way of importing this data into DataSnap, so that it can be processed and visualized. However, this initial programming step presents an

interesting challenge, as data sources from various domains are often very large and would overload the web browser if fully imported.

In typical desktop-based programming environments, the size of data sets was not always a concern, with the user being limited only by the size of their computer's storage devices. In a web-based interface, however, the user is greatly limited to how much data their browser can handle. Modern web browsers are often limited to using only megabytes of data, which for some domain experts, would be nowhere near being able to utilize the full set of data that they have collected. Thus, our first contribution related to data importing in block-based languages is an interface in the DataSnap runtime for importing and handling big data in a block-based environment. Specifically, we introduce the concept of cloud variables, explained in detail later, as well as basic operational elements for server-based data import.

Some code executions which are less consuming are still needed on the client-end (e.g., result displaying and visualizing). Our second contribution in data importing is a set of blocks that define the basic operational elements of client-end data import.

Our third contribution is that we also provide some openly-accessible, fundamental datasets as web services to DataSnap users. By presenting simple block definitions, we allow users to select a data source dynamically from all provided data services, as well as select all available data fields for further usage.

### 3.3.1 Server-end Data Import

Domain experts and professionals typically have sets of data that they have collected that are stored either on their personal computer or on a server. This data is often stored in tabular form in documents such as in Excel files, CSV files, Google Sheets documents, or other document types. In our implementation, we chose not to support all of these document types. However, we have decided to support the importation of CSV files, due to this file's format not being tied to any particular operating system or platform.


Figure 3.3: The cloud data import blocks in DataSnap.

Figure 3.3 shows the big data import blocks. Four sample data source blocks are provided

in DataSnap. These blocks return a URL to the specified data source. In the fifth block, the user can input the URL of their own CSV file as a parameter. When this block runs, the corresponding JavaScript function connects to a web service provided by the server, which initializes the download procedure of that CSV file.

### Cloud Variables

Cloud variable storage allows users to store strings, numbers, and the results of data processing operations on our server. The reason cloud variables are necessary is because the results of intermediate processing methods may still be too large to transfer to the client web browser. Additionally, storing these results on the server would avoid the unnecessary transferring of the data from the server to the client and back to the server for the next processing operation, resulting in improved performance.

There are two kinds of cloud variables on our server end: server_cloud_variables and user_cloud_variables. They are both key-value pair dictionaries. Both of them are user-specific, which means that the server maintains different server_cloud_variables and user_cloud_variables for each user, distinct by the unique user id assigned to each user when he/she initializes a DataSnap instance using a browser. Note that the keys of server_cloud_variables are auto-incrementing integers and that the values are results of a data processing method. The keys of the user_cloud_variables are strings, assigned by the user as the name of variables, and the value is a reference to a result previously stored in the server_cloud_variables dictionary.

Each time a data processing procedure (such as importing remote data files, a select method, max method, etc.) is finished, the results of the data process are automatically stored as server_cloud_variables, and the key of the variable (the auto-incremented id) is passed back to the DataSnap client. If the user wants to reference the result, he/she assigns a variable name (a string) to associate with the key of the server_cloud_variables, and the server stores the variable name and the corresponding id in the user_cloud_variables dictionary.

Figure 3.4 shows the result from the user executing a block that sets a cloud variable equal to the result of a “get max” method. The “get max” method performs an HTTP request and the processing operation is performed on the server. A reference to the resulting pandas.DataFrame object is stored in the server_cloud_variables dictionary and is associated with a key value of 14. This key value is passed back to the client as a reference_number for use in the “set” method. The set method then makes an HTTP request that sends the variable name, sel_rows, and the reference_number, 14, to the server. The server then associates the key sel_rows in the user_cloud_variables dictionary with the appropriate value that was stored in the server_cloud_variables dictionary.

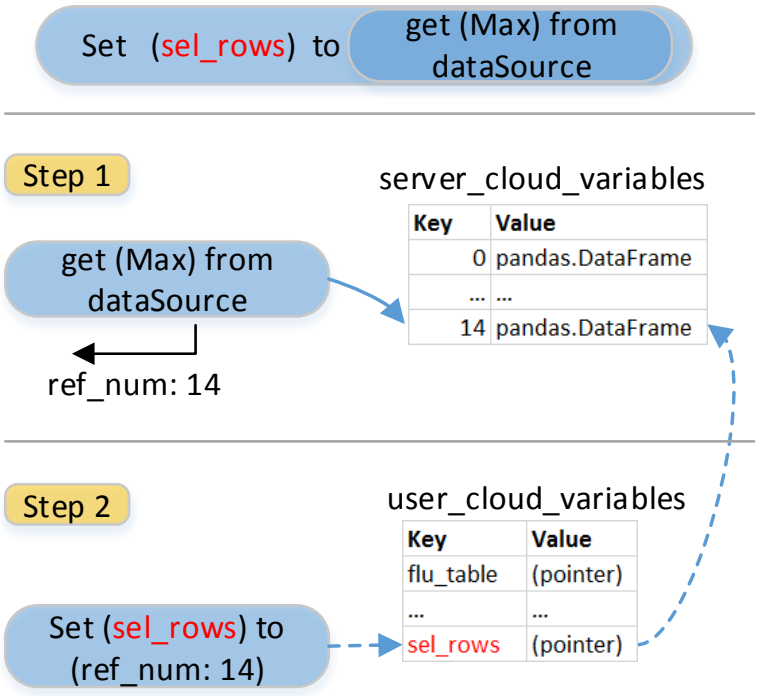



Figure 3.4: Demonstration of cloud variable storage, through the use of the server_cloud_variables and the user_cloud_variables dictionaries.

### Retrieving Cloud Variables


Figure 3.5: The cloud variable blocks added to DataSnap.

When users need to retrieve the results back from the server for display, they may use the blocks provided in Figure 3.5 to get the results. First the variable name (a string) is set to the block that executes the last data process procedure, and then the `retrieve data from cloud variable (variable name)` block is used to retrieve the results back.

Next we explain the critical role played by the server_cloud_variables and user_cloud_variables. DataSnap executes in a way that the inner blocks run first, followed by the outer blocks. As is shown in Figure 3.4, the client-end receives the returned key of server_cloud_variables that points to the results of the last execution. If the user needs the results, then the user may use user_cloud_variables to save the results. Otherwise,

the results saved in server_cloud_variables are not referenced, and they will be cleaned periodically.

### 3.3.2 Forming Server-Based "Cloud" Methods

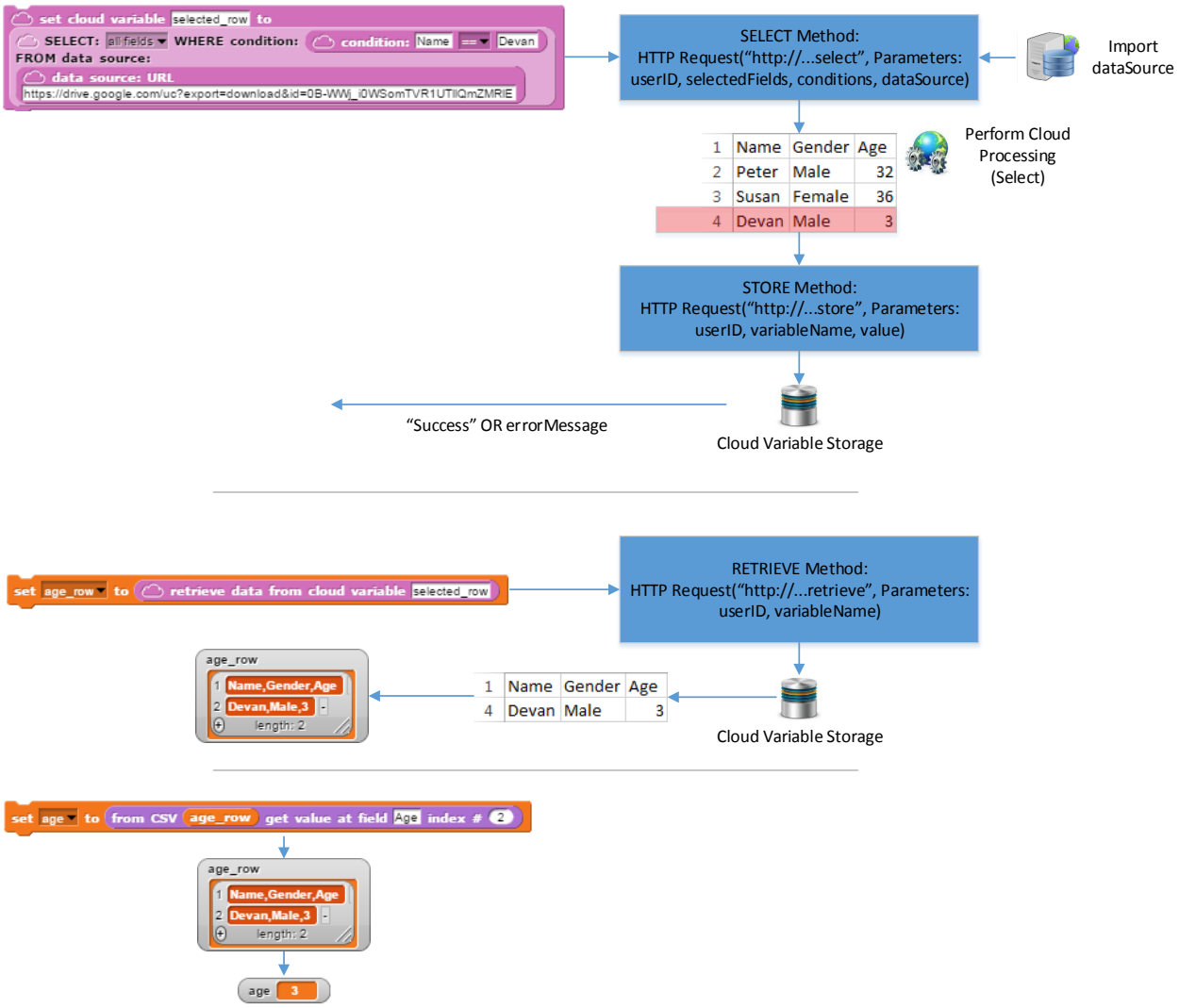



Figure 3.6: A demonstration of how to form server-based "cloud" methods by using the cloud data import, cloud variables, and elementary blocks. This simple example retrieves the age of the person named Devan.

In Figure 3.6, we demonstrate how one can use blocks to define server-based "cloud" methods. First, the server-end data import function is used to download a CSV file. Next, a cloud processing method is performed on the CSV file. In this scenario, a "select" method is performed where the selected row(s) must meet the given condition: Name equals Devan.

Then, the filtered result from the server is stored in cloud variable storage and a success message or an error message is sent back to the client.

Next, the value from the cloud variable selected_row is retrieved from the server and sent to the client. The value is assigned to the variable age_row. The next block then parses the CSV data to acquire the specific cell value of the CSV. In this scenario, the parser finds the value at the field / column named "Age" at the second row. This value (3) is assigned to the variable named age.

The above mentioned flows can be saved as a server-based cloud method, namely, a block: its parameters are the URL of a CSV file, and the name of the person we want to search, and the output is the person's age. DataSnap supports the import and export of such user-defined methods as xml files. These xml files can be created, stored, shared, and imported from/into any instance of DataSnap as a user convenience. Figure 3.7 shows the end result: the custom block and it's block definition.

Block:


Block Definition:

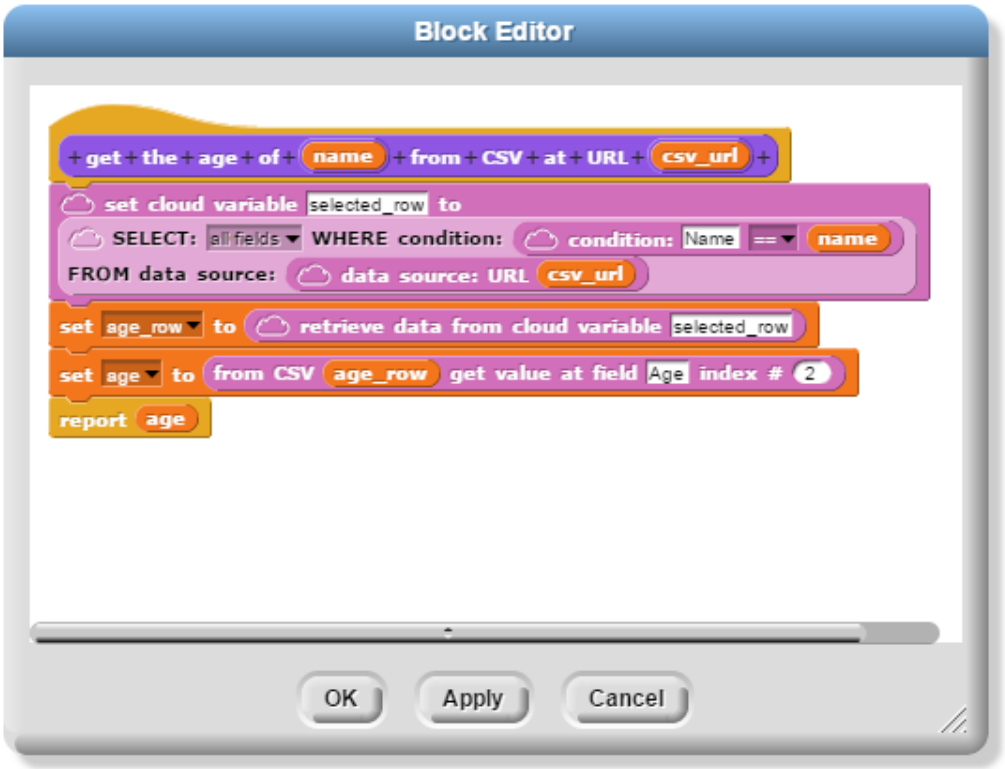



Figure 3.7: A server-based "cloud" method. The block and it's block definition are shown.

## 3.4 Visualization

DataSnap provides users with the opportunity to visualize their data on a map. With this addition, users gain the ability to perform data processing operations on their data and then display the results on a map programmatically. Through the addition of this visualization, users will be able to better relate to and better understand their data, and this increased understanding will help them contribute newly discovered information to their respective fields.


Figure 3.8: The six mapping blocks which coincide with the DataSnap mapping visualization.

The user can add the following three elements to the map: markers, circles, and points. Markers are shown as the Google red pinpoint symbol, which is used in Google Maps. Circles are drawn as having a fixed radius in miles, and points are drawn as having a fixed size in screen pixels. Corresponding with these three elements are block implementations to both add the element to the map, or remove the elements from the map. The mapping blocks are held within the Visualization tab in DataSnap, and the blocks are shown in Figure 3.8. As shown, the user can programmatically specify the latitude, longitude, and size of the element by typing in specific values. However, instead of typing these static latitude and longitude values, users can combine and nest blocks from the openly-accessible data sets; geocoding blocks which retrieve latitude and longitude values for a string the user enters can be nested inside the mapping blocks.

Mapping is the first visualization that has been fully designed and implemented in DataSnap. Because the DataSnap front-end client is built on web-based technologies, there is no limit to what other visualizations can be added. Indeed, other such JavaScript libraries for displaying line graphs, bar graphs, pie charts, and much more have already been developed and can easily be integrated for display on DataSnap; examples of such libraries are D3.js, Highcharts, and include many others.

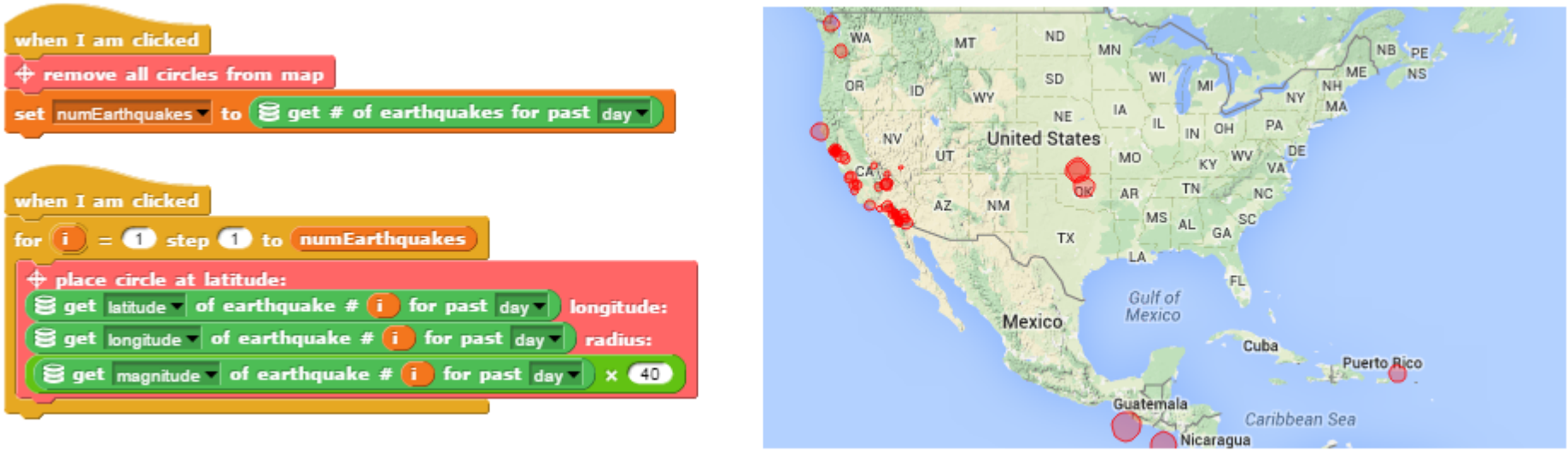



Figure 3.9: Data visualization example. https://www.google.com/maps, 2015

Figure 3.9 shows an example of data visualization using a Google Map. One can map out all of the earthquakes that have occurred during the past day by creating two methods. The first method removes all the circles from the map and retrieves the number of earthquakes that have occurred in the past day. The second method iterates over the number of earthquakes and places a circle at the latitude and longitude that the earthquake occurred. Additionally, the size of the circle corresponds to the magnitude of the earthquake.

## 3.5 System Design for Leveraging Cloud Resources in Big Data Processing

As we have mentioned earlier, in Snap! the user places blocks within their browser, so that the browser can then execute the code expressed by these blocks. For local processing, Snap! manages to perform efficiently simply due to the high computational power of modern computers. However, considering that DataSnap needs to be able to process potentially large amounts of data, the complexity of data operations, including "select", "max()", and "min()," may increase non-linearly with the increase of the data sizes. Therefore, to achieve high efficiency, DataSnap needs to separate code input and code execution: the user may input code using their own browser, but the code execution, especially the operations that are extremely time consuming should be executed in another environment with superior computational resources.

In this section, we first introduce the general runtime architecture of DataSnap, and then give the detailed design of both the DataSnap client side, in which the users form block scripts, as well as the cloud server side, in which the scripts are executed.

### 3.5.1 Runtime System Design Overview

Figure 3.10 shows the architecture overview of the designed runtime system for DataSnap. It consists of two parts: the front-end for user programming and the server-end for code execution.

The DataSnap scripts are written by domain experts using the DataSnap front-end running in their browsers. As mentioned earlier, the DataSnap front-end primarily consists of a few basic HTML elements, CSS markup, and JavaScript files.

The server-end provides web services using an Apache Web Server. We chose to utilize Python Flask to efficiently get a web application up and running quickly, and because of its lightweight but extensible core. The server-end consists of cloud variable storage, data process controllers, internal data process modules, and external data process controllers.

When the data processing requests are received from users, the requests are forwarded to one

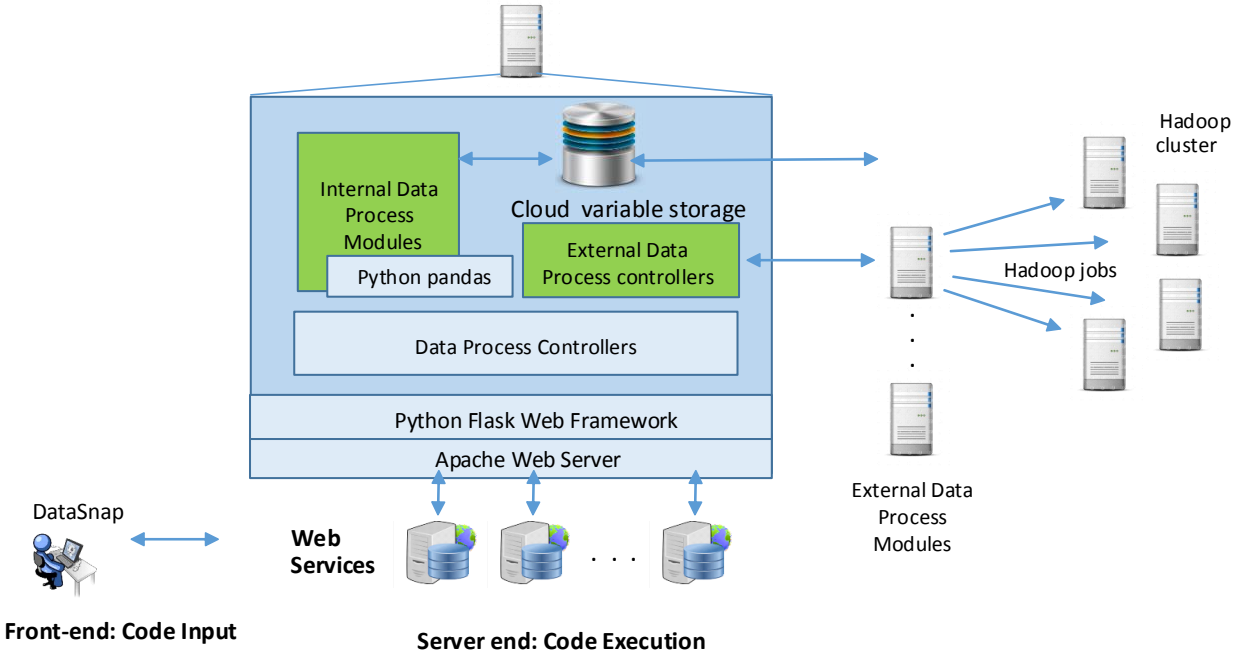



Figure 3.10: Runtime System of DataSnap

of the data process controllers. The data process controller then decides whether to process the request using the local computational resources on this server, or computational resources from external servers. If the data process controller decides to process the request locally, the request is forwarded to the internal data process module. Alternatively, if the data process controller determines that the request needs to be handled by an external server, the request is forward to the external data process controller. The external data process controller then sends the request to an external data process module, which is responsible for communication with an external data process module.

Each data process module, whether internal or external, defines a specific compute server as its host and a set of data processing methods which this host is able to perform. The internal data process module, for example, can perform the following methods: 1) select; 2) get maximum; 3) get minimum; 4) get average; 5) get median; 6) get sum; 7) get product. The internal data process module is described in further detail in the corresponding subsection below.

### 3.5.2 Interaction between Client-end and Server-end

When compute-intensive operations are included into a script in the client-end, the logic behind these operations is not executed by the client-end. Instead, the client-end's runtime makes remote calls to the corresponding web services of the server-end, passing along all the

required data source URL references and parameters (Figure 3.12).

When the server receives these remote requests, it first decides whether to forward the request to either an internal or an external data process module. The data process module receives the data source url and parameters, and the required data sources files are then downloaded. The data process module then performs the data processing operation. Results of the operation are stored in cloud variable storage and a reference to the object in cloud variable storage is sent back to the client.

### 3.5.3 Data Process Modules

#### Internal Data Process Modules

The internal data process module on our server is based on the Python pandas high performance data analysis library [61, 60]. The data processing methods which are a part of the internal data process module are shown in Figure 3.11. The first block resembles a SQL style SELECT block, in which a user can select specific rows and fields of a data source based on the given conditions. The parameters that a SELECT block takes are the selected fields, the conditions, and the data source. For the processing operations provided, the data source is expected to be in tabular form and saved in the CSV format.

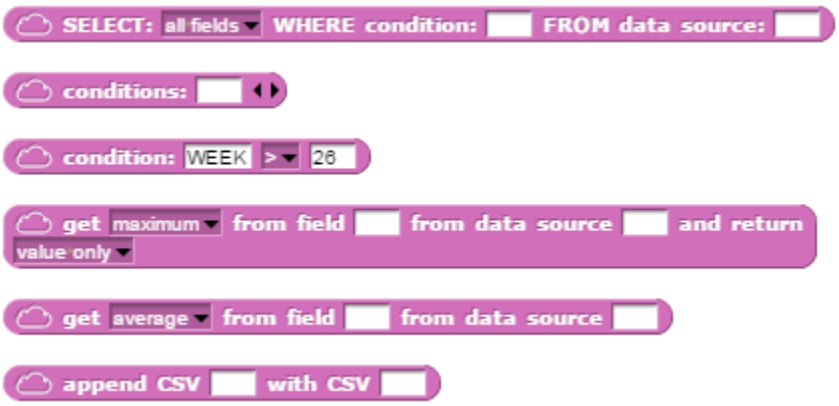
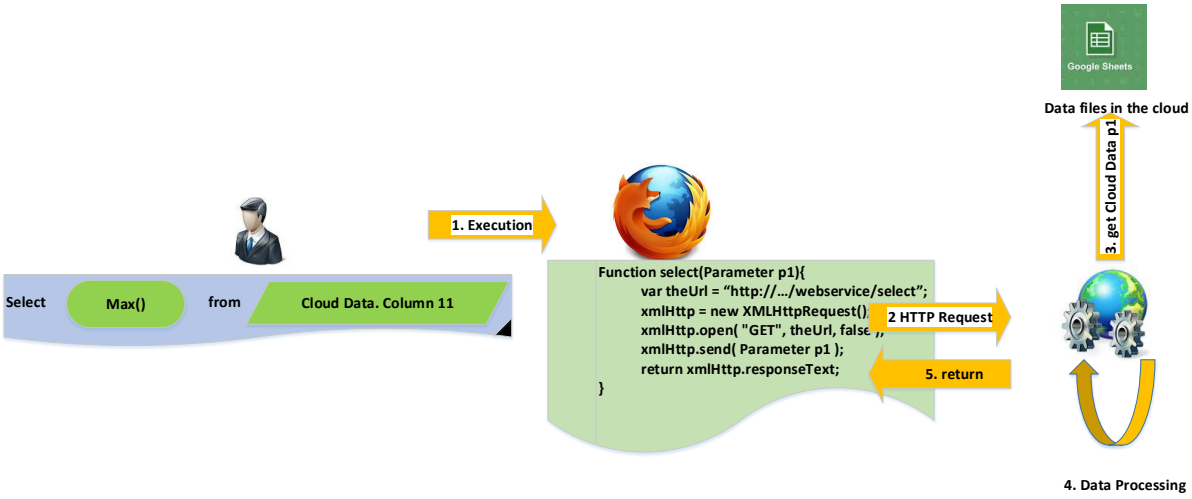



Figure 3.11: The big data processing blocks within the internal data process module in DataSnap

For the first parameter, the selected fields parameter, the default value is “all fields”, which will select all fields or “columns” of the data source. Each value in the first row of the data source is used as the field name for it’s particular column. If the user would like to select only certain fields, they can type in the fields they would like to select in the fields block. The fields block can then be dragged into the fields slot, replacing the “all fields” selection.

For the second parameter, the condition parameter, conditions are defined as having a con-




via

Figure 3.12: Communication Between Client-end and Server-end

dition field, a condition operator, and a condition value. The condition field shown in the figure, for example, helps to select all rows which have a value greater than 26 in the column labeled "WEEK". The condition operators available are: 1) == 2) != 3) >4) >= 5) <and 6) <=. Strings typed into the condition field text box must coincide with a valid field in the data source or an error will be thrown. The "equal to" and "not equal to" condition operators can be used with either strings or numeric values, however the other condition operators must be used with numeric values. The "condition" or "conditions" block must be placed in the condition slot of the SELECT block. In the case that the "conditions" block is used, multiple "condition" blocks are placed in the input slots of the "conditions" block. Lastly, the third parameter of the SELECT block is the data source, as was mentioned earlier in the paper. The parameters are sent as part of the request from the client to the server.

The next two blocks "get maximum" and "get average" provide the following big data processing operations: 1) get maximum; 2) get minimum; 3) get average; 4) get sum; and 5) get product. The operations act upon one particular field of the data source and can return either the 1) the value only, or 2) the entire row. Returning "value only" for a maximum block, for example, will only return the maximum value found in that column, while returning "entire row" will return the entire row that the maximum value was found in.

Lastly, the append CSVs block appends two CSVs into one CSV. This method adds the rows of the second CSV to the end of the first CSV. This method is particularly useful when results from other processing methods need to be combined before the next processing method is performed. Please note that both objects must have the same fields or else an error will be generated.

### External Data Process Modules

As introduced earlier, an external data process module is a compute server host that is capable of performing data processing methods. An example external data process module is the proposed Hadoop cluster shown at the right in Figure 3.12. This particular compute server would be capable of carrying out high throughput MapReduce operations for example. The integration of a Hadoop cluster as an external data process module in DataSnap would be especially effective in speeding up MapReduce operations on even larger data sets which the Python pandas library would likely not process as quickly.

An important functionality to mention is that methods from the internal and external data process modules are completely swappable. If the same exact DataSnap programming block can map to either a method on an internal or an external module, the data process controller will be responsible for routing the request to the appropriate module. Take the "get maximum" block for example. This method is a MapReduce operation which can be performed by either the Python pandas internal data process module or the proposed Hadoop external data process module. The data process controller will determine to which module the request is routed. However, the end-user is abstracted away from these implementation details and is presented only with the simple "get maximum" block in the DataSnap user interface.

The routing decision of the data process controller mentioned in the previous paragraph would be dependent on factors such as the size of the data source and the complexity of the processing operation. Thus the routing would ensure that the user's data processing requests are performed as quickly as possible by utilizing the most effective data process module. Because DataSnap has an extensible design, we encourage others to develop and integrate their own compute-servers and external data processing modules, to meet the compute-specific needs of their particular domain.

## 3.6 Case study: Epidemiology Example

Figure 3.14 will be used to demonstrate our case study. In this problem, three blocks have been developed by a domain expert in epidemiology who needs to analyze flu data. The data is based on the Centers for Disease Control and Prevention's (CDC) flu data, collected for its Influenza-Like Illness Surveillance Program (ILINET).

Figure 3.13 shows a subset of the data from this data source. REGION refers to either the entire nation or one of the 10 HHS regions of the United States (major city representing each region: 1. Boston 2. NYC 3. DC 4. Atlanta 5. Chicago 6. Dallas 7. Kansas City 8. Denver 9. San Francisco 10. Seattle). ILITOTAL refers to the total number of influenza-like illnesses reported. % Weighted ILI is equal to the percentage of people who were reported to have an influenza-like illness out of the total number of patients seen, weighted by region.

| | REGION | YEAR | WEEK | ILITOTAL | % WEIGHTED ILI |
|---|---|---|---|---|---|
| 2 | Region 1 | 1997 | 40 | 44 | 0.49853479 |
| 3 | Region 2 | 1997 | 40 | 3 | 0.374963154 |
| 4 | Region 3 | 1997 | 40 | 32 | 1.35427537 |
| ⋮ | | | | | |
| 9121 | Region 10 | 2015 | 11 | 135 | 1.021887152 |

Figure 3.13: CDC ILINET flu data table.

In this scenario, the epidemiologist has created an interface in DataSnap, consisting of three methods that can be executed by the epidemiologist herself or her colleagues. The import/export feature of DataSnap makes it easy to share the created interfaces with other domain experts.

The figure shows the result after clicking on the "Map out the peak flu week for the (2014) flu season" block, with the generated data displayed in the top right section called the data region, and a visualization of the strength of the flu in the 10 HHS regions of the United States in the bottom right section, which is called the visualization region. The size of the circle for each region is based on the % Weighted ILI value for each region, with larger circles representing a larger presence of the flu. Based on the visualization, one can conclude that in the peak flu week for the 2014-2015 flu season, the flu had a larger presence in the eastern half of the United States than in the western half. From viewing our data, one can see that Region 6 had the highest weighted ILI, and that Region 1 had the lowest weighted ILI. Results can then be compared over multiple years.

To gather more information about the peak flu week, one can click on the next block, "Get the peak week of the flu season for year (2014)." The data region will then populate with more information about the peak flu week. The next block "Get the peak week of each flu season from year (1997) to (2014)," will retrieve the peak flu weeks of each flu season for the past 15 years. The user can scroll through the displayed metadata to learn additional information about each peak flu week.

In this example, if the epidemiologist would like to view the algorithm that the programmer used to create the data on display, all they would have to do is right click on the block and click edit. The block definition will then display, letting them analyze the program's data selection steps.

Let us contrast this example with a typical data analysis scenario. Typically, on a website that shows an analysis of ILI data, the viewer is able to view the end-result, which is the filtered data or the generated visualization. However, they are left to blindly trust the implementation on how the data or the visualization was generated. DataSnap provides both the ability to view the generated data and visualizations as wells as the ability to transparently view the program algorithms that were used to generate the data and visualizations. Fur-

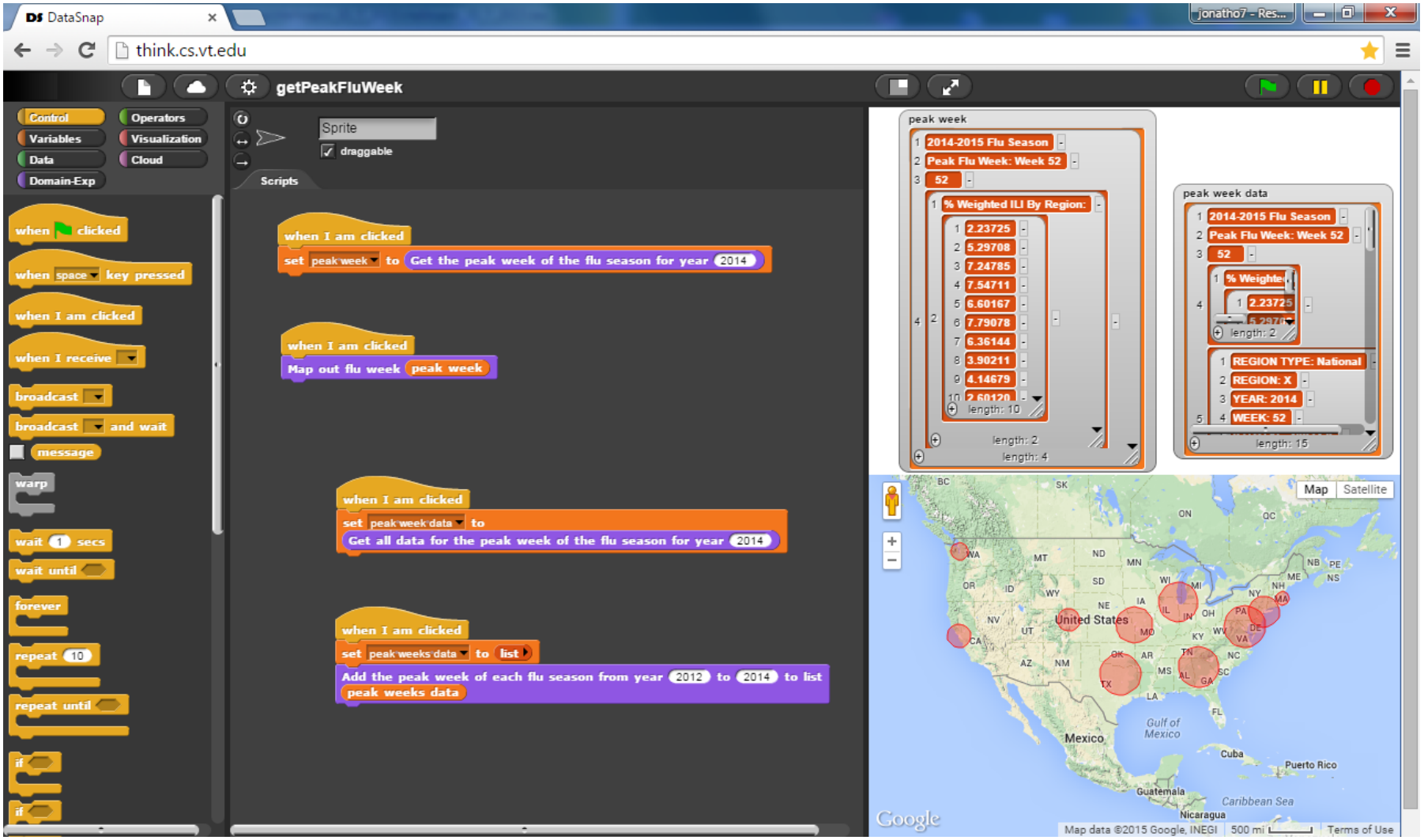



Figure 3.14: A flu data analysis program designed for an epidemiologist, shown in the DataSnap platform. https://www.google.com/maps, 2015

thermore, results can easily be shared with other peers in their field. The block definitions of the first two epidemiology blocks are shown in Figures 3.15 through 3.21. The rest of the block definitions are shown in Appendix A.

The key element of this example is that it shows how DataSnap abstracts the once very complex, unreachable big data problem solving environment, and provides this environment for use by domain experts, and those with little to no coding experience. DataSnap essentially lowers the barriers to entry for using a big data problem solving environment, making it accessible to a wider user audience.

To illustrate this point, let us take the Apache Hadoop MapReduce environment for example. In order for a user to compute using a MapReduce problem, first the user must be familiar with a professional programming language such as Java, or a similar language for Hadoop's variant programs, in order to program a MapReduce program. They must also be adept at using a terminal and following the non-trivial steps in order to first install and configure the single node cluster which can run their Apache Hadoop MapReduce programs, and all of this system's dependencies. The user must also learn how to import files using the Hadoop Distributed File System (HDFS). These tasks are not trivial tasks, typically adopted by advanced programmers. Although setting up a similar big data processing environment in Python or Java would be somewhat easier, users with little programming experience would still have a hard time programming an algorithm for big data problem solving.

By contrast the DataSnap computing platform provides a web-based application that excludes the necessity for any complicated installation procedures, and whose program composition process eliminates the possibility of introducing syntactical errors. Users are empowered with the ability to carry out big data processing and visualization operations. Hence, a wider community of users, currently lacking access to big data processing, or frustrated with typical data programming environments, should be able to wield the "big data" tool for their day-to-day tasks, previously only accessible to those experts in advanced computing technologies.

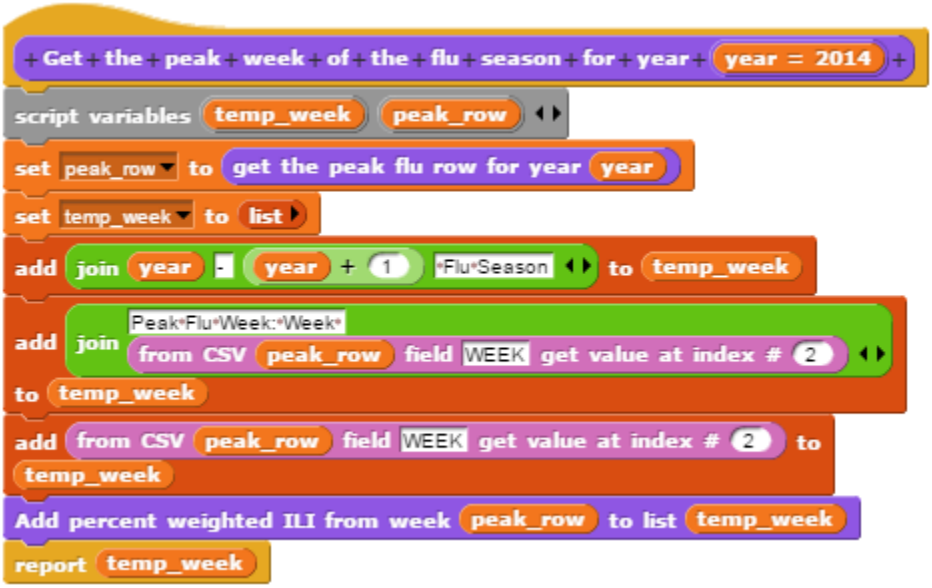



Figure 3.15: The "Get the peak week of the flu season for year (year)" block definition.

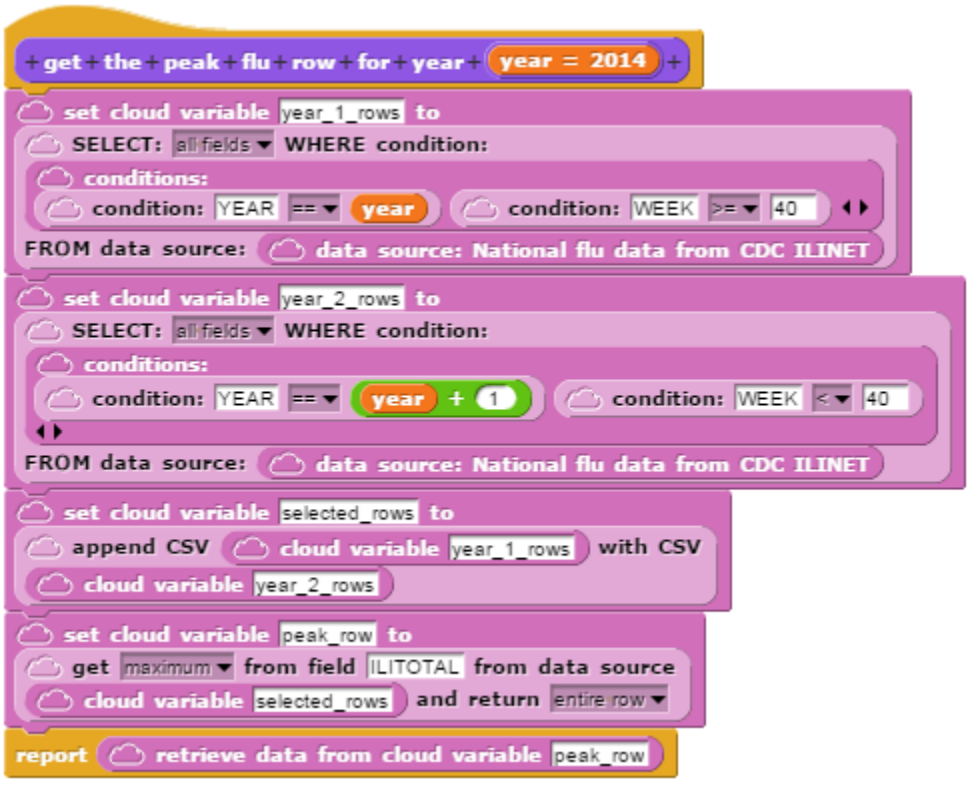



Figure 3.16: The "Get the peak flu row for year (year)" block definition.

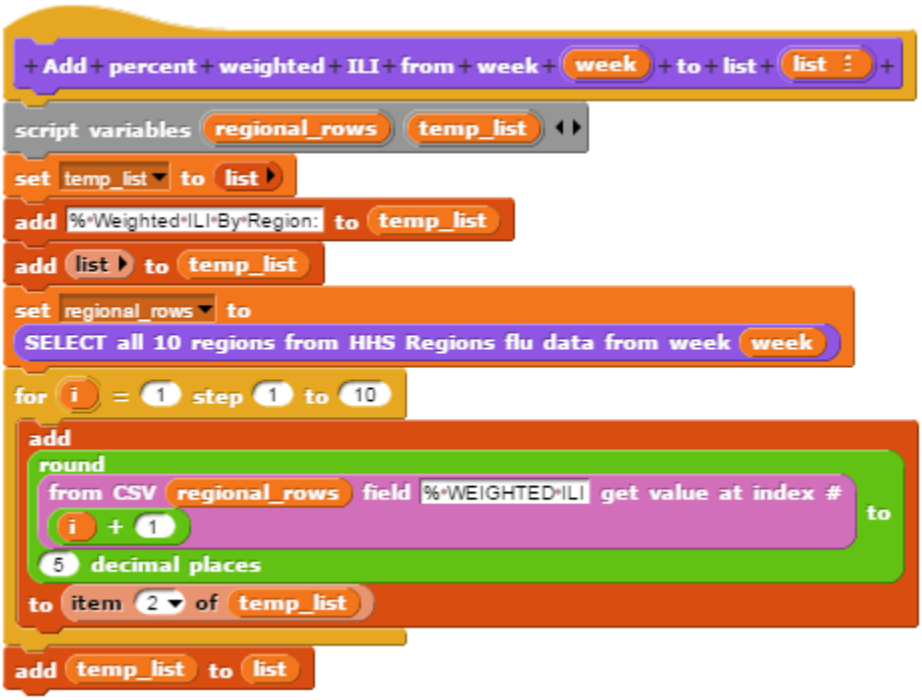



Figure 3.17: The "Add percent weighted ILI from week (week) to list (list)" block definition.

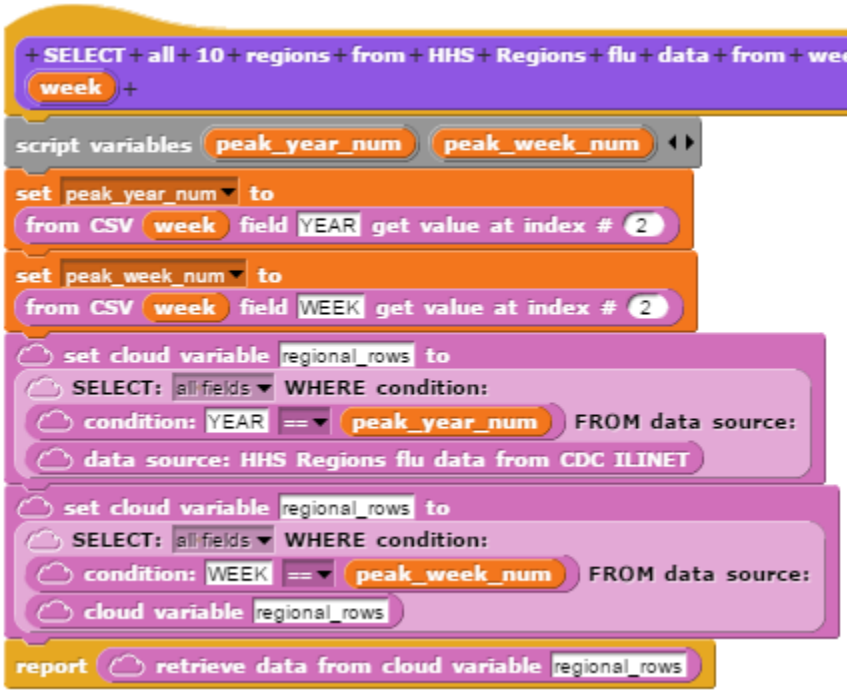



Figure 3.18: The "Select all 10 regions from HHS Regions flu data from week (week)" block definition.


Figure 3.19: The "Map out flu week (week)" block definition.

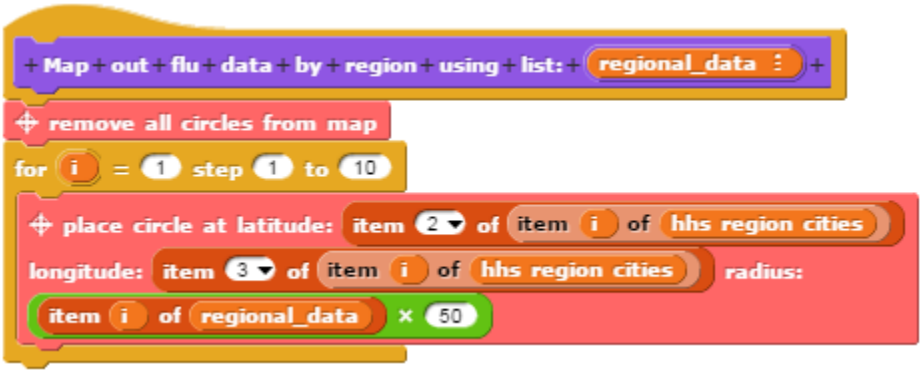



Figure 3.20: The "Map out flu data by region using list (regional_data)" block definition.

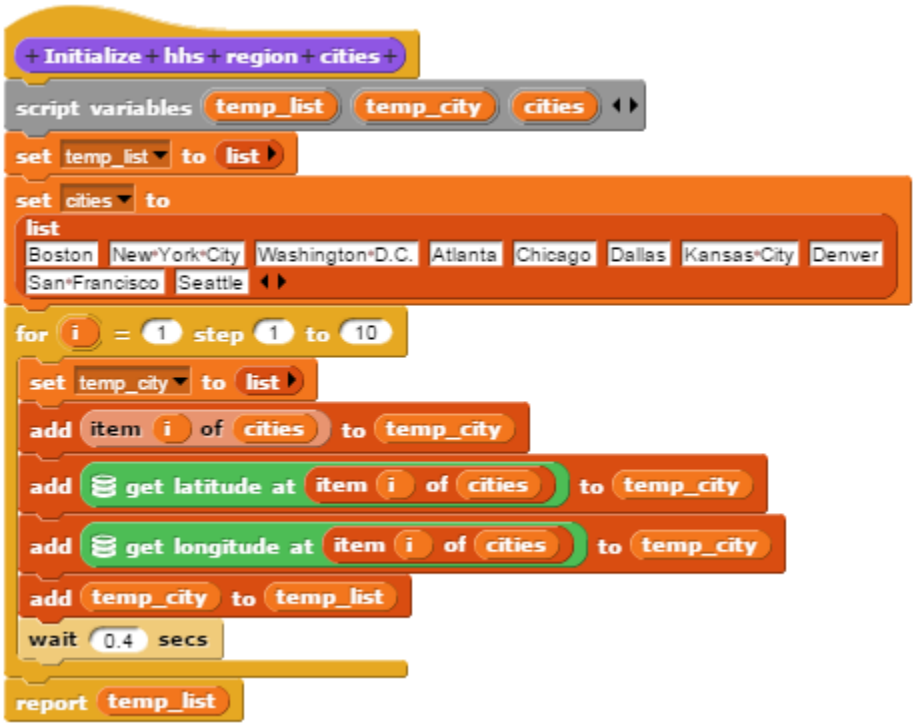



Figure 3.21: The "Initialize hhs regions cities" block definition.

## 3.7 Reference Implementation

### 3.7.1 Design Motivations

DataSnap is the starting point of a greater and broader system to come, where multiple front-end clients (Snap, Blockly, Scratch, etc.) utilize the big data processing server that is already in place. Each front-end client may have a different interface and a different audience of users, but each one can benefit from the big data processing capabilities provided by DataSnap. Thus compatibility with multiple clients was a major design motivation for this project and our work towards this goal will be described in the sections below.

DataSnap is also designed with extensibility at its core. We envision a system where even more big data processing operations are defined, more powerful compute servers or more powerful computation mechanisms are used (Hadoop scripts, etc.), and more data formats are supported (csv, json, xml, doc, xls, etc.).

### 3.7.2 Cloud Variables API

The Cloud Variables API allows users to store strings, numbers, and the results of data processing operations on our server. The reason the Cloud Variables API is necessary is because the results of intermediate processing methods, in the form of CSV dataframes, may still be too large to transfer to the client web-browser. Additionally, storage of these results on the server would avoid the unnecessary transferring of the data from the server to the client and back to the server for the next processing operation, resulting in improved performance.

Therefore, we chose to have each of the big data processing methods store their results server-side in a Python dictionary called server_cloud_variables. The server_cloud_variables dictionary holds a key for each client called user_id, and each users variables are stored under this key. Each front-end client user generates one user_id via a Globally Unique Identifier (GUID) generator method. For our purposes, the user_id is a 16 character string.

Figure 3.22 outlines the Cloud Variables API. The doSetCloudVariable request corresponds to the "set cloud variable (name) to (value)" block and the doRetrieveDataFromCloud-Variable request corresponds to the "retrieve data from cloud variable (name)" block. The parameters for both of these requests are shown at the right of the figure.

The isValueAReferenceIndex is a boolean value with a value of true meaning that a reference index to a variable that is within server_cloud_variables will be used in the assignment. A value of false means that the cloud variable will be assigned directly to whatever is typed into the "value" field of the block. A string or an integer can be typed into this field. Variable_name and variable_value, as their name suggests, correspond to the "name" and "value" fields of the blocks. If a cloud variable name is reused, the old value is overwritten.

| Cloud Variables API<br>API URL Base: http://think.cs.vt.edu/snap/**api/cloudvariables** | |
|---|---|
| **URL** | **Parameters** |
| /doSetCloudVariable | user_id, isValueAReferenceIndex, variable_name, variable_value |
| /doRetrieveDataFrom CloudVariable | user_id, variable_name |

Figure 3.22: The Cloud Variables API

### 3.7.3 Internal Data Processing API

Figure 3.23 shows the Internal Data Processing API. As it's name suggests, the select request corresponds to the "select" big data processing block. The number of conditions, condition field, condition operator, and condition value are sent as request parameters. If multiple conditions are used, additional parameters are used: condition_field0, condition_field1, etc. DataSourceType is a string which specifies the type of the data source: either "cloud_variable" or "url". If the value "cloud_variable" is used, this specifies that the dataSourceValue is a reference index to a cloud variable. If the value "url" is used, this specifies that the datSourceValue is a url string. The file at this url will be downloaded for use in the cloud methods.

MethodSet1 is the request which corresponds to the "get maximum" block. The big data processing operations available in this block are 1) get maximum; and 2) get minimum. The name of the specific operation is provided as the operation_type parameter. Field corresponds to the CSV column which should be traversed. DataSourceType and dataSourceValue are the same as in the select request. Lastly, the returnType parameter is a string that is equal to "value only" if the user only wants the output value without the entire row, or "entire row" if the user would like to retrieve the row where the output value was found.

MethodSet2 is the request which corresponds to the "get average" block. The big data processing operations available in this block are 1) get average; 2) get median; 3) get sum; and 4) get product. The same parameters are used as in methodSet1 except the returnType parameter is not included. In the cases of these processing operations, specifying a return type would not make sense because the output value is not specific to any specific row(s) of the data set.

Lastly, the append request corresponds to the "append CSV" block. The parameters dataSourceType and dataSourceValue are identical to the parameters in the previous blocks except that the dataSourceType must be equal to "cloud_variable". A url which corresponds to a CSV file cannot be used in this implementation. The two CSV's must already be stored

as cloud variables on the server.

The Internal Data Processing API is utilized by the DataSnap front-end. The API is provided to other developers so that other block-based programming environments may be able to benefit from the use of the big data processing commands in their platforms. Our code for this project is open source, and more big data processing methods are welcome to be added to our code repository.

| **Internal Data Processing API**<br>**API URL Base:** http://think.cs.vt.edu/snap**/api/internaldataprocessing** | |
|---|---|
| **URL** | **Parameters** |
| /select | user_id, numberOfConditions condition_field, condition_operator, condition_value, dataSourceType dataSourceValue |
| /methodSet1 | user_id, operation_type, field, dataSourceType, dataSourceValue, returnType |
| /methodSet2 | user_id, operation_type, field, dataSourceType, dataSourceValue |
| /append | dataSourceType0, dataSourceValue0, dataSourceType1, dataSourceValue1 |

Figure 3.23: The Internal Data Processing API

## 3.8 Conclusion

The design of block-based programming languages can yield benefits that extend beyond user-friendly interfaces that encourage and motivates non-computer professional users to learn programming. Due to their low expectations of programming capability, block-based languages have great potential to benefit domain experts, who commonly possess little programming expertise. However, the current design of mainstream block-based languages fails to provide domain experts with the ability to import data sources, perform efficient data processing, and visualize results.

In this paper, we present the design and implementation of DataSnap, a block-based programming language we derived and extended from Snap!. As compared to Snap!, DataSnap offers the following advantages. 1) It enables domain experts to import data either from public data sets or from their private data sets stored in the cloud; 2) it leverages the computational resources of cloud servers to speed up the processing of big data; 3) it combines third party visualization libraries written in JavaScript, and provides domain experts with

easy-to-use interfaces to visualize their data results. Through a case study, we demonstrate how DataSnap enables domain experts with little programming skills to analyze real data sources and visualize results within a web browser in a “drag and drop” style.

# Chapter 4

# DataSnap for Introductory Programmers

## 4.1 Introduction

Online educational platforms (e.g., Massive Open Online Courses (MOOCs)) have become increasingly popular means of delivering computing curricula. These platforms, unfortunately, are plagued by extremely low retention rates, particularly in introductory courses. Educational theory posits that improper academic motivation models can be responsible for large percentages of online students never completing their courses. Specifically, the current focus of online programming assignments on entertainment (e.g., games, animation, etc.) is particularly misdirected. It fails to convince students that the covered technical content is important, relevant, and related to their day-to-day computing experiences. This paper presents the design and implementation of DataSnap, a programming platform which incorporates real-time and big data in the context of big data problem solving in order to address this problem.

DataSnap combines both 1) relevant social media, geographic, and business-related, data sets and 2) big data operations, as a means of convincing introductory online learners that computing is a powerful tool for solving challenging and relevant problems across a variety of fields. DataSnap was developed as an extensible platform which is not tied to a particular front end client, enabling other online platforms to improve their motivation and retention rates by leveraging our technology to enrich their curricula. Several big data problem solving examples will demonstrate its usefulness.

Despite their popularity, these online educational programming platforms have so far failed to realize their promise as a means of motivating and retaining a diverse population of students. Namely, current online students are often turned away by the curriculum's prevalent focus on entertainment, such as games and animation. Not developing the necessary motivation to

study computer science, many students are dropping out having failed to see computing as a tool for solving real world problems in their lives [10]. Additionally, women and minorities remain woefully underrepresented in computing, with the typical over-emphasis on games and animation contributing to the under-representation [19].

The widely recognized and well-validated MUSIC model of academic motivation [45] explains this lack of motivation [44]. It suggests that students become internally motivated when they perceive that (1) they are eMpowered; (2) the content is Useful; (3) they can be Successful; (4) they are Interested; and (5) they feel Cared for by the instructor and other students. The MUSIC model shows that existing educational programming platforms fail to provide useful and interesting content to a diverse audience; not everyone is interested in games and animations, and it is hard to see this kind of development as generally useful [35]. A new, promising approach is to teach introductory computation in the context of big data science to solve real-world problems[3]. Unfortunately, these platforms lack support for accessing real-time and big data sources, which can provide a unique context for teaching real-world problem solving. Indeed, addressing this technological shortcoming has the potential to empower students to explore real-world problems that are truly important to them.

We propose that educational programming platforms must address the needs of a diverse set of students by integrating real-time and big data sources into assignments. Thus we present DataSnap, a programming platform which incorporates real-time and big data in the context of big data problem solving, realized for two different levels of audiences: introductory programmers and non-expert programmers. For introductory programmers, DataSnap provides convenient ways of accessing the following real-time and big data sources through browser based content provider APIs: 1) National Weather Service data, 2) Stock data (Google Finance), 3) GeoCoding (Google GeoCoding Service) data, 4) United States Geological Survey (USGS) Earthquakes data 5) Twitter data, and 6) Reddit data. Data access is provided to learners client-side through programming "blocks" (See Figure 4.1), and server-side via our web API.

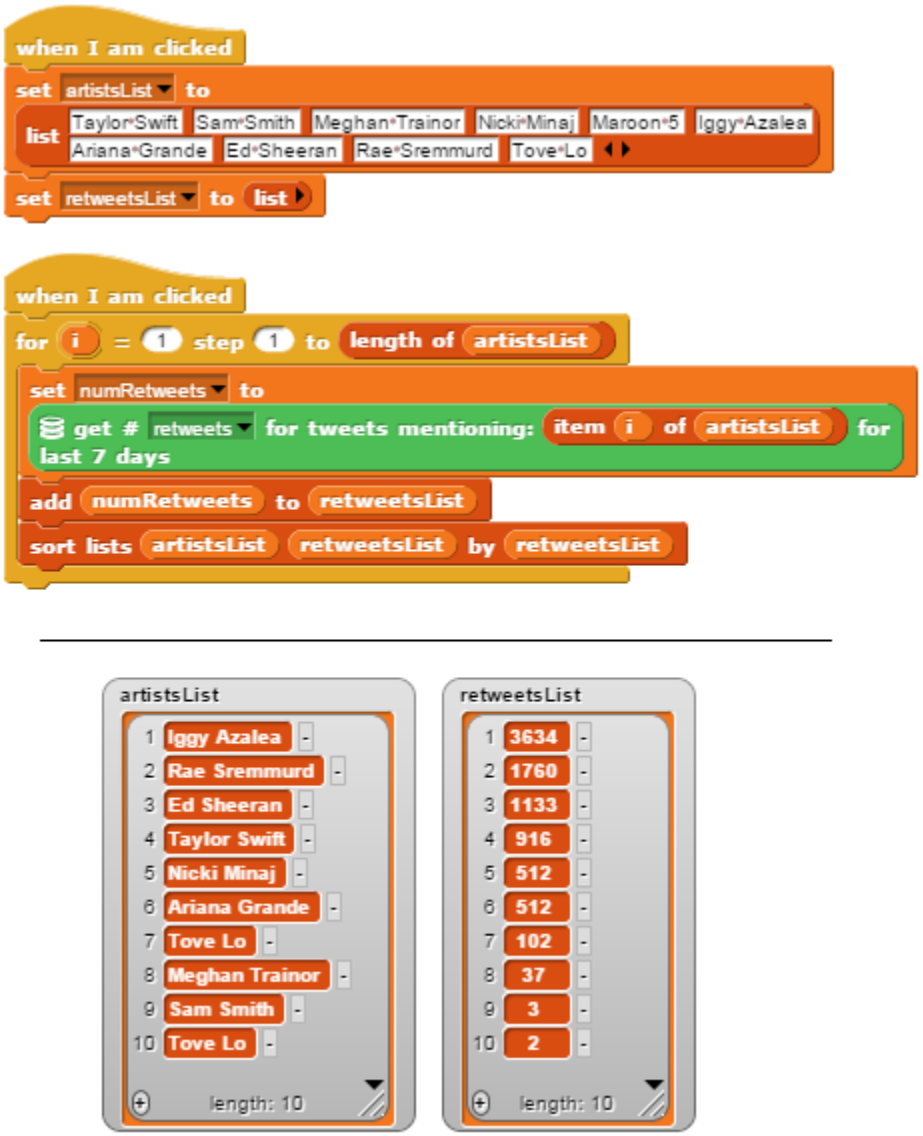



Figure 4.1: Top: An example DataSnap program which lets users determine the most popular music artist based on Twitter retweets. The first method initializes the artistsList and retweetsList variables. The second method retrieves the Twitter data and places the results in the list called retweetsList. Bottom: The results of running the DataSnap program.

## 4.2 Educational Context

### 4.2.1 Educational Context Overview

Access to data should be simple and convenient, yet expressive for introductory programmers. Because the introductory programmer audience may range from K-12 to introductory college students, differing levels of difficulty for data access are implemented through different levels of scaffolding. The Reddit library, for example, expects users to know how to manipulate lists, and is appropriate for a more advanced audience. The weather library, however, does not provide data to users in the form of lists, but gives data in simpler formats such as strings and integers. Thus the weather library may be more appropriate for a less experienced audience.

Figure 4.2 outlines the differences between the different data sets. The data sets have been divided into three suggested educational levels, with three data sets designed for beginners, two for intermediate users, and one for advanced users. The educational levels are determined based on the data types that the blocks return, the 3V Model [23], and the context that the data is traditionally used in. The data sets that only return strings and numbers have a beginner educational level. One exception to this distinction is the Twitter data set which is in the intermediate educational level because it deals with high velocity data with a social media context. The Earthquakes data set, which is in the intermediate educational level, affords the use of data which is more relevant if mapped out. Lastly, the Reddit data set is in the advanced educational level because it requires users to utilize lists, and lists of lists, in addition to strings and numbers for return types. In the figure, data sources are in parentheses in the library column. Although a particular educational level is given, each audience level is able to use all of the libraries in some way. Suggested audience levels are Beginner: K-12, college; Intermediate: 6-12, college; Advanced: 9-12, college.

Attention to the big data properties from the 3V model must also be considered. The 3V model properties are: Velocity, Volume, and Variety, and are used to define collections of data sets which could be considered to be "big" in a particular way. Velocity is defined as "the rate at which new information is added to the system." The data sets with high velocity include: Weather, Stocks, Earthquakes, Twitter, and Reddit. Next, volume refers to "the total quantity of information, usually measured in bytes or number of records." The data sets with high volume include the Earthquakes and the Reddit data sets. Lastly, variety is defined as "the format or formats of the data." The data sets with high variety include the Earthquakes, Twitter, and Reddit data sets.

One such example problem which can be given to introductory programmers is shown in Figure 4.1. In this problem, the student is instructed to determine which music artist is the most popular based on social media data (e.g. the Twitter data set). Because popular phrases, people, or topics could be likely to be tweeted and retweeted a larger number of times, the number of retweets could signify how popular a particular phrase is.

| Library | Educational Level | Velocity, Volume, Variety | Data Types |
|---|---|---|---|
| Weather (NWS) | Beginner | High, Low, Low | strings, numbers |
| Stocks (Google) | Beginner | High, Low, Low | strings, numbers |
| GeoCoding (Google) | Beginner | Low, Low, Low | numbers |
| Earthquakes (USGS) | Intermediate | High, Med, Med | strings, numbers, JSON data |
| Twitter | Intermediate | High, Low, Med | numbers |
| Reddit | Advanced | High, High, Med | strings, numbers, lists, lists of lists |

Figure 4.2: A table of the six datasets for introductory programmers.

The student is first instructed to create a list of artists that they would like to analyze for popularity, and also to declare a variable which will store the results of the analysis (retweetsList). Next, they are instructed to retrieve the number of retweets about a particular artist, and then use iteration to add the results for each artist to a list. Through the development of this program, the student will learn how to assign values to variables, retrieve data from a data set, add items to a list, and iterate through a for loop. Further extensions of this problem could be to sort the list of artists and retweets in descending order, using various sorting algorithms.

### 4.2.2 Importing Openly-Accessible Data into DataSnap

In addition, access to common sources of social media, geographic, and finance-related data has become an important tool in both the common user's and the domain expert's tool set. Data from these sources are made public by their respective organizations and should be easily accessible to all, thus we call these data sources "openly-accessible". Data from such sources can be used for making meaningful decisions. For example, social media data sets are rich with information that can help companies develop or redirect business strategies, researchers predict outbreaks or epidemics, and government leaders make key decisions [17]. Our second contribution related to data importing in block-based languages is the addition of six openly-accessible data sources to block-based programming languages, as well as the generic resource that allows users to import data from any data source accessible via a web API call.

The openly-accessible blocks are designed to provide easy and convenient ways for users to import real-time data from common sources into DataSnap. The open-access blocks are shown in Figure 4.3. Data sources include:

- National Weather Service data
- Google Finance stocks data
- Google Geocoding Service data
- United Sates Geological Survey earthquakes data
- Twitter data
- Reddit data

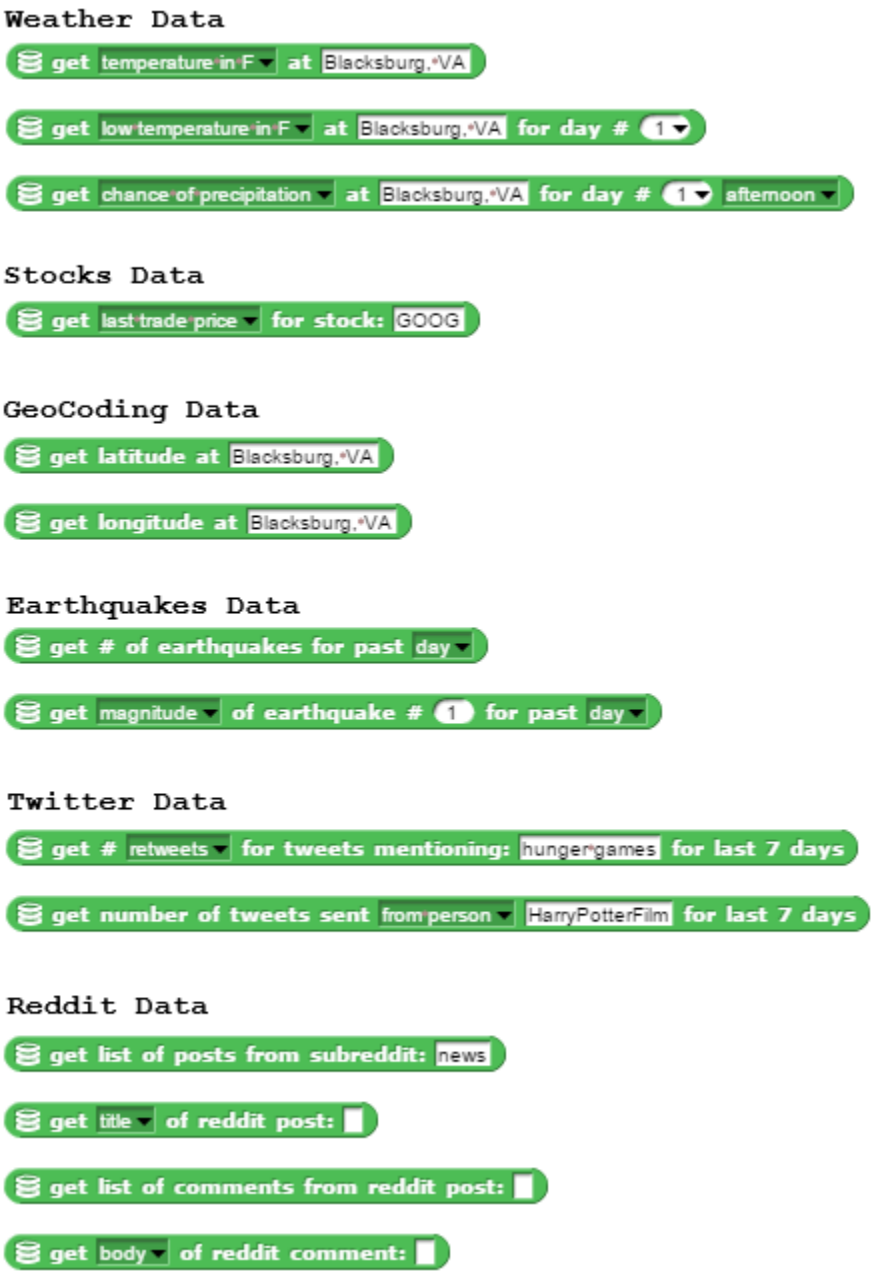



Figure 4.3: Each of the six data set's programming blocks, which provide simple and convenient access to data.

### 4.2.3 Elementary Blocks

The blocks from the six data sets previously mentioned provide a sample set of data for students and teachers to work with, however, we foresee situations where a teacher may

want to use data from other web sources. We define six basic elementary blocks for client-end data import and management specifically for this purpose. Through the combination of these elementary blocks, a teacher or student can develop their own customized data access blocks. The block specifications follow:

- `from (url) get json`
- `from (json) get keys`
- `from (json) (key OR array index) get value`
- `from (url) get csv`
- `from (csv) get fields`
- `from (csv) (field) (index) get value`

When the first block runs, it initializes an HTTP Request Client, to fetch the json file from the given URL. When the second block runs, it uses the json parser to get all the keys from the given json. When the third block runs, it uses our json parser to traverse through the given json to the desired key value pair. Blocks four through six are conceptually the same, except using a csv parser instead of a json parser.

By combining these six basic elementary blocks and other blocks, we are able to provide complicated data import and management methods for users to import and work with both CSV and JSON data formats. Using these blocks, teachers and students are able to form their own data sets for use in their classes and personal projects. As a proof of concept, we have used these elementary blocks to define the previously mentioned data sets shown in Figure 4.3. The corresponding block definitions are shown in Appendix B.

### 4.2.4 Caching of Data

The data retrieval process for each use of a block usually takes between 0 and 5 seconds, from the time the user clicks on the block to the time the retrieved value is printed out on screen, stored in a variable, or utilized in some other way. Because each click of a data access block causes a query to its corresponding web API, there is a potential to over-request data from the API server and throttle the API rate limit, and also a potential to slow down the student as they must wait for each data access request to finish.

Repeated uses of the earthquake blocks, for example, such as in a for loop structure, could potentially query the USGS Earthquakes API once for each use, overloading their server, and also slowing down the student. Therefore, precautions have been taken to avoid the misuse of the data sets through the use of client-side caching. Although in our implementation caching

is only utilized for the weather and earthquakes data set, it is capable of being utilized for any data set that is developed. Next, we will explain how this caching strategy works.

After the request from the client front-end to the back-end is made, the report (weather report, earthquake report, etc.) is stored in a JavaScript object called CacheController. CacheController is an object which contains three methods: addReport(), hasCachedReport(), and getCachedReport() and contains a dictionary object for each data set. Each dictionary stores reports retrieved from the back-end. The addReport() method for the earthquake data set takes as argument both a period (hour, day, week, or month) and a dateTime object. Based on an expiration time and the dateTime object for the last report that was stored, the hasCachedReport() will return true if the report is not expired, and false if the CacheController either does not have a report or has an expired report. The getCachedReport() method will either return a cached report or will retrieve a new report if the old report has expired.

The caching system in place effectively eliminates the need to retrieve data from the data source web API each time a student requests data through the use of a block. This enhances the data sets that utilize this caching strategy, and minimizes the harmful effects of naive beginner programming mistakes.

## 4.3 Sample Lessons and Problems

### 4.3.1 Sample Problem - Decision

Logic and decision making is a very important concept for K-12 students to learn. Our first sample problem aims to let students practice logic and decision programming concepts through the use of the weather data set. The sample problem goes as follows.

You are getting ready for school today and you have a limited amount of space left in your backpack for any extra things. However, you need to decide whether you should take your umbrella with you to school. Design a program that will give you a helpful message based on the chance of precipitation for this afternoon. If the chance of precipitation is:

- 20 or less, set the message to "You can leave your umbrella at home today."
- between 21 and 40, set the message to "You might want to take your umbrella today."
- between 41 and 60, set the message to "You should strongly consider bringing your umbrella."
- between 61 and 100, set the message to "You definitely should bring your umbrella today."

This educational problem will teach students how to declare and assign values to variables, retrieve data, use mathematical operators, and use decision control structures. The student will be able to use the current chance of precipitation in their city while they are working on the problem. The student will then be encouraged to change the chance of precipitation to values ranging from 0 to 100 in order to test their program. A teacher's testing framework, (proceed to Chapter 5 for more information), can then change the value of the "chanceOfPrecip" variable to test the edge cases of the student's solution, and then give the student a grade based on these tests. Figure 4.4 shows the sample student solution.

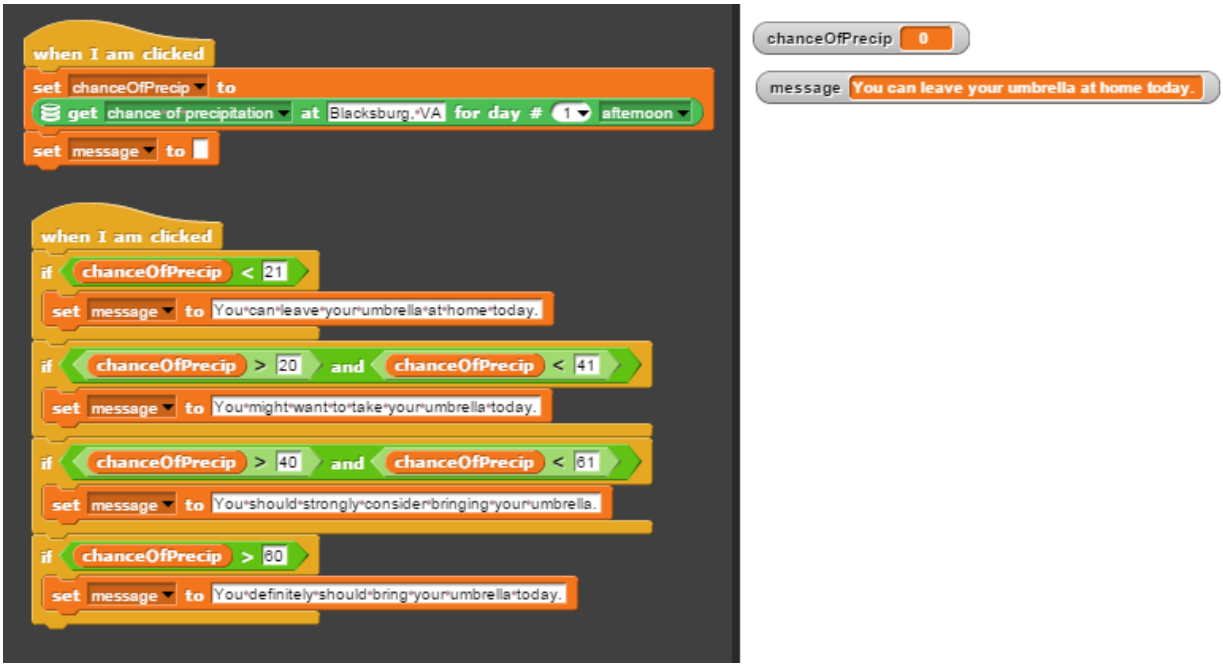



Figure 4.4: Decision Problem - Weather Data

### 4.3.2 Sample Lesson - Iteration

Iteration is a very common and valuable concept taught in computer science courses. However, iteration problems are usually designed and taught without the student's context in mind. Students are immersed in a environment where live data is ubiquitous: Twitter and Reddit feeds, facebook posts, weather updates, etc. We believe that the student will feel more motivated if they are encouraged to work with data that exists in their current context, which should include the student's environment as well as the live data that is in it.

In the following example, we present a lesson which uses earthquake data to teach iteration. The sample lesson text for this example will be provided, but a teacher is encourage to modify this text to suit their educational needs.

### Iteration

Iteration is a very valuable technique which can let a user repeat a specific set of commands in order to achieve their goal. Iteration can be used to repeat the same set of commands once, twice, hundreds, or even thousands of times so that the user does not have to keep entering in the same code over and over again. A few examples of using iteration are:

- Repeatedly adding together a students grades from a gradebook to receive the total and then dividing the total by the number of assignments to calculate the average grade.
- Repeatedly printing out the magnitude of each of the worlds earthquakes that have occurred in the past day or week.

Iteration is defined as: "the act of repeating a process with the aim of approaching a desired goal, target or result." Furthermore, 'Each repetition of the process is also called an iteration, and the results of one iteration are used as the starting point for the next iteration." (Wikipedia: Iteration)

In part 1 of this example, first a program without iteration will be shown. Then in part 2 and 3, programs with iteration will be shown which will demonstrate the usefulness of using iteration. Specifically, iteration will be used to repeat a command hundreds of times.

### Part 1

This program retrieves from the U.S. Geological Survey the number of earthquakes that have occurred in the past day and then prints out this number. Then, it prints out the magnitude of each of the first five earthquakes, waiting 3 seconds between each print. Since iteration is not used in this example, you will notice that the same command is repeated five times.

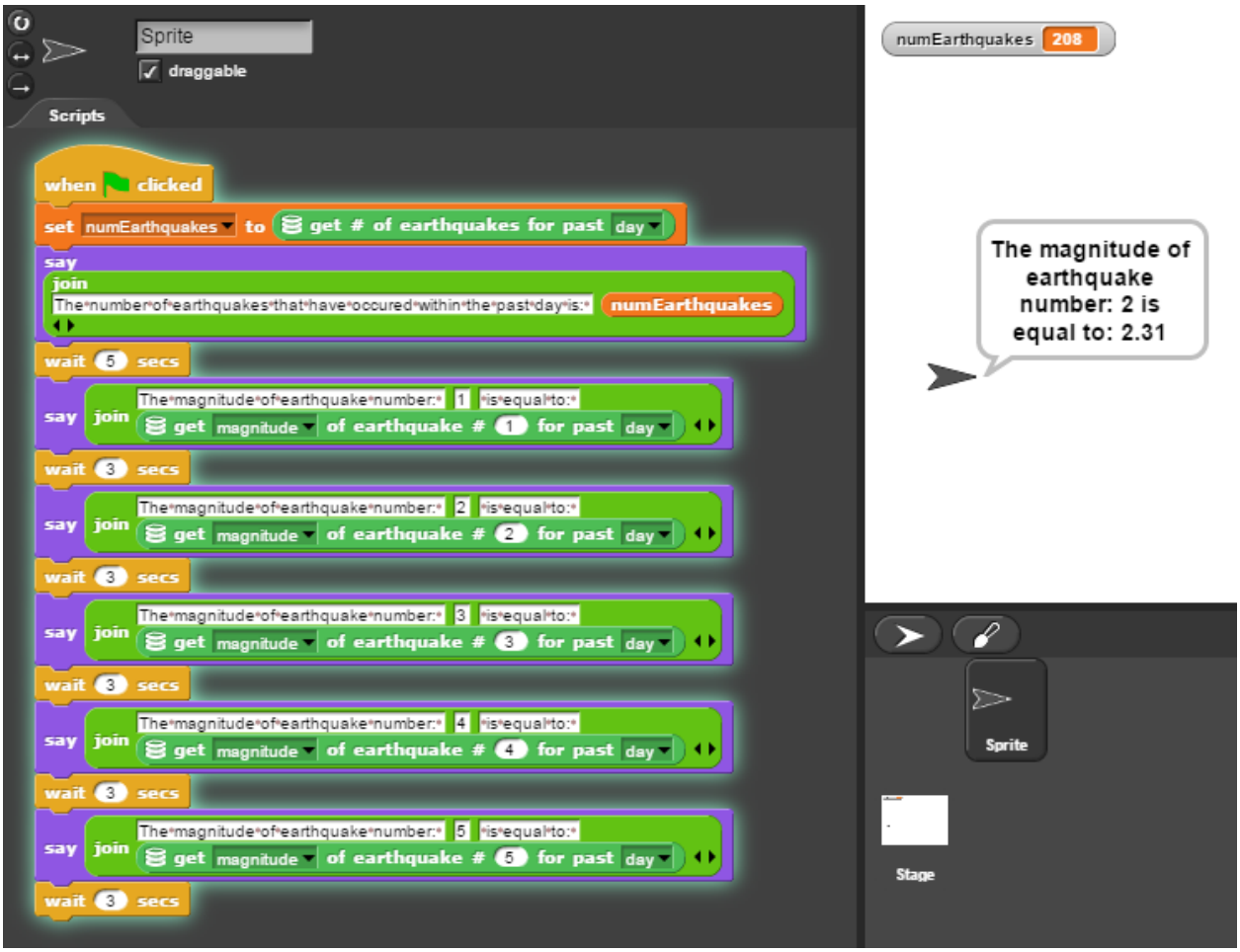



Figure 4.5: Iteration example - Part 1

### Part 2

This program produces the exact same results as the program in Part 1. However, notice instead of having 5 separate commands which print out the magnitude, there is only one command shown which prints out the magnitude. This command is repeated 5 times through the use of the for loop control block. Using a for loop is especially useful when you need to repeat a command many times, as you will see in Part 3.

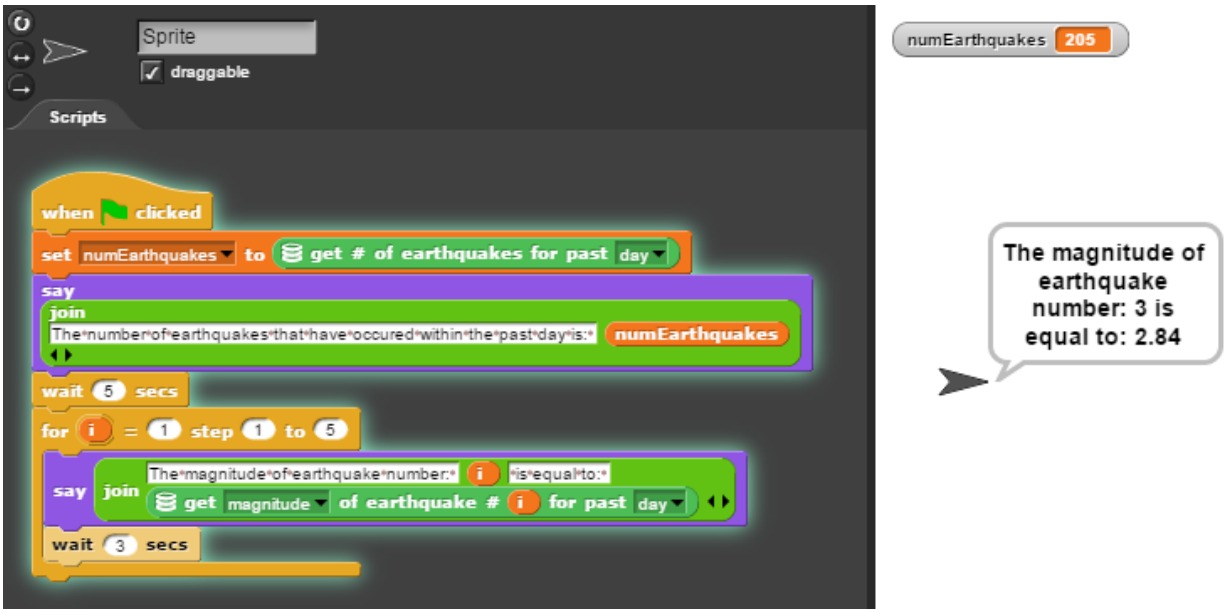



Figure 4.6: Iteration example - Part 2

### Part 3

Instead of only printing out the magnitudes of the first earthquakes, this program prints out the magnitude for every single earthquake that occured in the past day. This number can usually be over 80+ earthquakes. This example especially shows the power of iteration. Instead of manually adding the same command 80+ times, this command could be written once, and repeated easily with the use of iteration.


Figure 4.7: Iteration example - Part 3

### 4.3.3 Sample Problem - Iteration

In this example, an example taken from earlier in this report has been presented along with its educational context and extension problem. We encourage teachers to modify this program to suit their educational needs, whether that means offering no blocks to start off with, or offering more blocks to help scaffold the learning experience. The sample student solution is shown in Figure 4.8.

#### Visualizing Data - Earthquake Data

Design a program to map out the earthquakes that have occurred within the past day. First, you will need to retrieve the number of earthquakes that have occurred within the past day by using the earthquakes data access blocks. You will then need to use iteration in order to place a circle on the map with the latitude, longitude, and magnitude of the earthquake. The size of the circle represents the strength of each earthquake. For this example, multiply the magnitude of the earthquake by 20 so that it is large enough to be seen on your map.

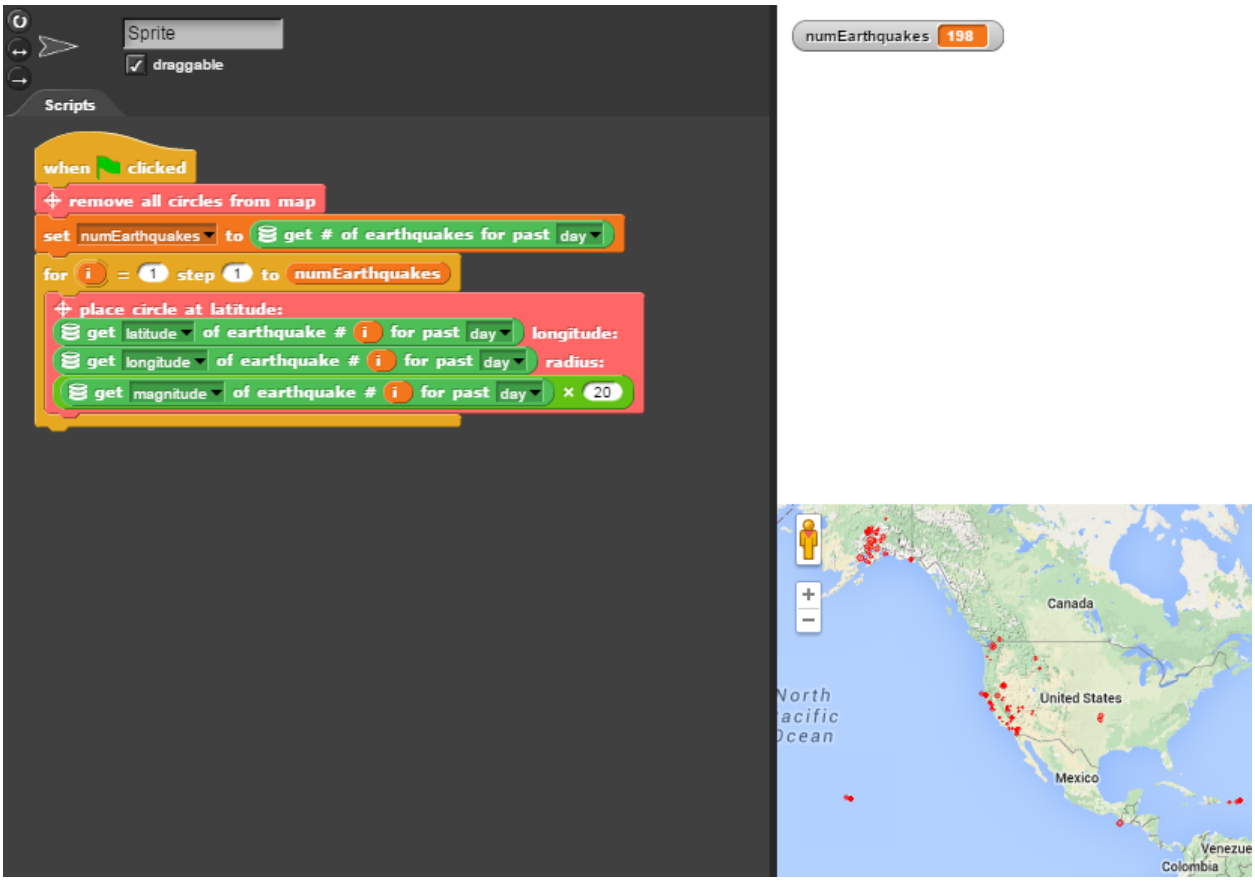



Figure 4.8: Iteration Problem - Visualizing Earthquake Data. https://www.google.com/maps, 2015

Extension Problem: Find the earthquake with the highest magnitude. Print out this magnitude and its location description. The sample student solution is shown in Figure 4.9.

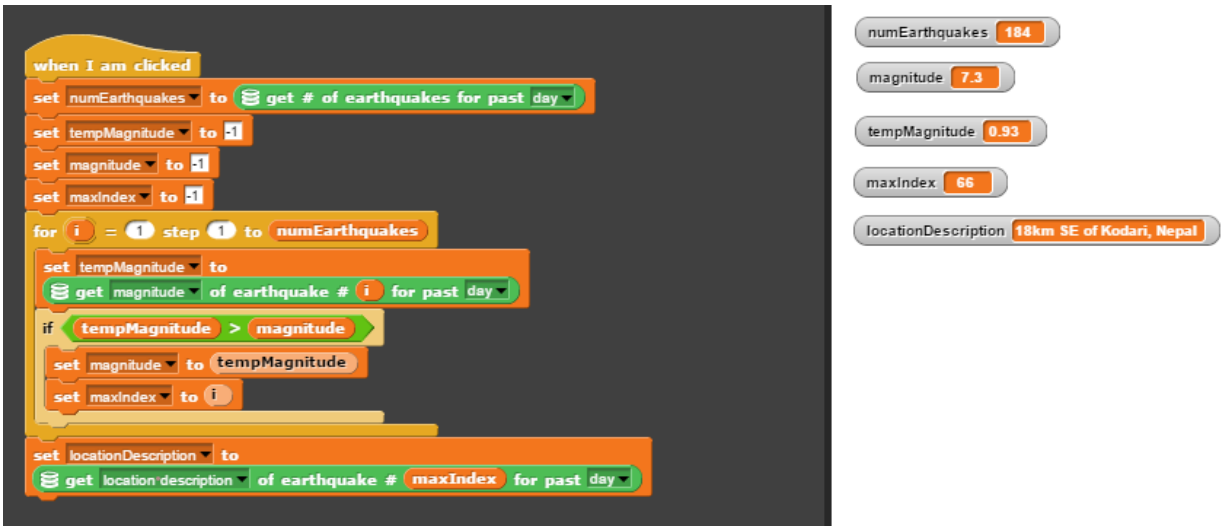



Figure 4.9: Iteration Extension Problem - Highest Earthquake Magnitude

## 4.4 Reference Implementation

### 4.4.1 Data Sets Implementation

This section will further detail the data set blocks discussed earlier in the paper. Figure 4.3 depicts these blocks. Each block may include drop-down menus which let the user pick which data they would like to retrieve. For the first weather data block, users can also retrieve: (1) temperature (2) windchill, (3) humidity, (4) windspeed in mph, (5) wind direction in degrees, (6) a weather description, (7) the visibility in miles, (8) the dewpoint temperature, and (9) the pressure in inches. Users can also access data broken down by the day, and also broken down by afternoon/night periods. Users can type in any location into the textbox that they would like to retrieve weather information from. A few example problems that can be given to students are: Convert temperature in Fahrenheit to Celsius; Suggest what clothes to wear based on the weather; Show the forecast for this week for the city that you live in.

For the stocks data set, users can also retrieve: (1) last trade price, (2) last trade date and time, (3) change percentage, (4) change amount, and (5) exchange name. Example problems using this data set are: For ten stocks, can you put the stocks that are up for the day in one list, and the stocks that are down for the day in another list? Can you sort these lists? If a person owned 50 shares of AAPL, how much money did they earn or lose today?

For the geocoding data set, the user can type in any location for which they would like to retrieve the latitude and longitude. The location entered must be recognizable by Google's GeoCoding service to return appropriate results. Example problems include: Map out the

top five most populous cities, with circles which represent the population size. Place on a map the current weather from 10 cities throughout the world.

For the earthquakes data set, the user can retrieve the number of earthquakes that have occurred in the following time periods: (1) hour, (2) day, (3) week, and (4) month. The following information can be retrieved for each individual earthquake: (1) magnitude, (2) latitude, (3) longitude, (4) location description, (5) url, and (6) a JSON which holds more additional info. A few problems include: What is the largest magnitude earthquake that occurred this week? This month?

For the Twitter data set, users can retrieve the number of retweets and the number of favorites for tweets that mention a particular phrase in the last 7 days. Users can also get the number of tweets with the following options: (1) from a particular person, (2) to a particular person, and (3) referencing a particular person, for the last 7 days. A few example problems are: Which upcoming box-office movie will earn the most money, based on retweets?

For the Reddit data set, users can type in a subreddit category that they would like to receive posts from. From a post, users can retrieve the following data: (1) title, (2) content, (3) subreddit, (4) author, (5) date, (6) ups, (7) downs, (8) id, and (9) whether the content is a url link. For each post, users can then get a list of comments for which particular comment information can be obtained. Because Reddit uses a nested forum style on its site, users will have to learn how to deal with lists that are nested within lists (i.e. comments of a post, comments of comments). An example problem to give to students is: Which subreddit category is the most popular?

### 4.4.2 Elementary Blocks Implementation

DataSnap adds the additional functionality that lets students or professors access data from any web API that returns data in JSON format. The blocks which support this functionality are shown in Figure 4.10. The first block, “from URL get JSON”, sends a request from the client to our server, which makes the request to the web API. Our server then returns the JSON data in the form of a string with proper JSON indentation.

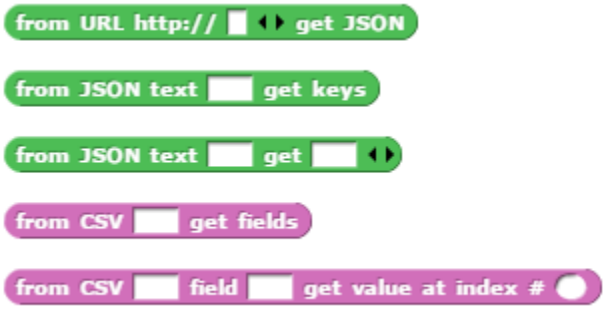



Figure 4.10: Elementary Blocks

For the user's convenience, the user is able to break up the URL into sections that are later concatenated before the request is sent. This is achieved by clicking on the arrows next to the URL field. In the example blocks shown in Figure 4.11, the top block shows the URL as one string, and the bottom block shows the URL separated by parameter. When these blocks are run, a request is made to the National Weather Service to get the weather report for Blacksburg, VA.

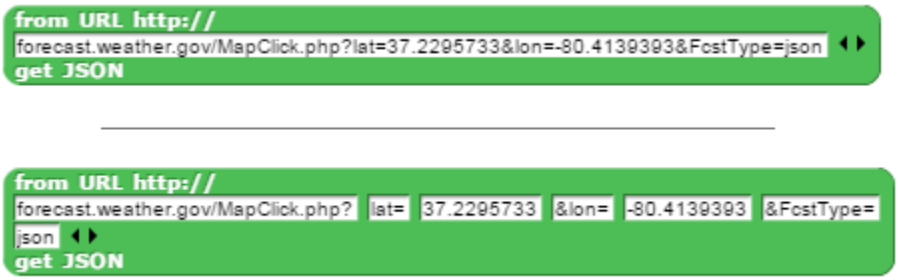



Figure 4.11: From URL get JSON block

Because of DataSnap's block-based nature, blocks can be inserted into each URL text field. If we wanted to get the weather report for a different city, for example, we could type in a different latitude and longitude value, but we could also use the blocks from the GeoCoding data set to get the latitude and longitude for us. In Figure 4.12, we have replaced the static values for the latitude and longitude values with values retrieved from another API request to the Google GeoCoding service.

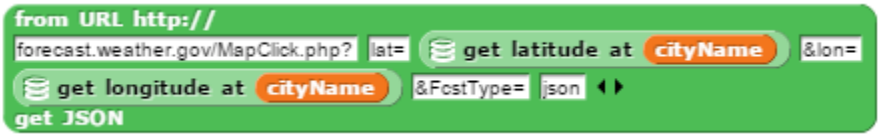



Figure 4.12: From URL get JSON block - with customized URL

The next block "from JSON text (text) get keys" accepts a JSON string as its first argument. This block returns a list of the keys that are contained in this JSON.

The next block "from JSON text (text) get (key or array index)" accepts a JSON string as its first argument, and a series of keys to traverse through as its following arguments. The block will first determine whether the string is in valid JSON format, form a JSON object from the string that is entered, and then traverse to the desired key/value pair based on the series of keys that are provided as arguments. The user can click the left and right arrows to add or remove key or array index fields.

The next two blocks are CSV parsing blocks. The first block expects as an argument a CSV in DataSnap. A CSV in DataSnap was implemented by using a List variable and using commas to separate the values. No spaces should be entered before or after the commas. The first item of the list should contain the field names. The subsequent items of the list should contain the rows of the CSV. An example CSV is shown in Figure 4.13. Notice that each row starts out with an integer for the index, and that the fields row should start out with a comma instead of an integer (as the field names row should not have an integer index).

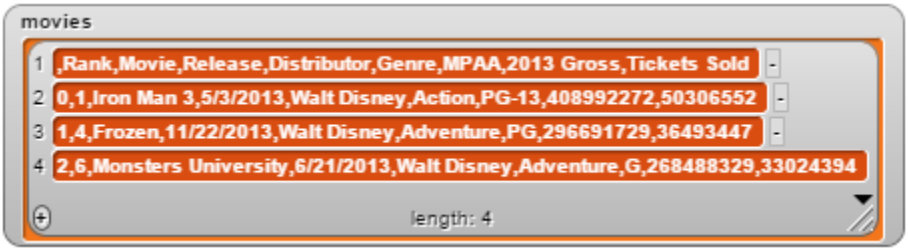



Figure 4.13: Example CSV in DataSnap

The last block is the "from CSV (CSV) field (field) get value at index num (num)" block. This block expects a CSV as its first argument and a valid field name as its second argument. The third argument, the index number, is used to specify which row of the CSV to look in. Since the column (field name) and row (index number) are both specified, this cell value can be extracted and returned.

### 4.4.3 Data Service API

The Data Service API is a RESTful API which specifies the interface between the front-end client and the server back-end for the openly-accessible data blocks. On the server, a call to the Data Service API involves utilizing the Python CORGIS libraries in order to retrieve the requested data. To illustrate this process, we will describe the scenario of what happens when a "get temperature" block is invoked in DataSnap.

The first step in the process is that the user's selection and entered text for the "get temperature" block are captured as parameters to be used in an HTTP GET request. A GET request which includes these parameters is then sent to the DataSnap back-end server through the Data Service API. The location string is sent to the CORGIS Python weather library, which returns a weather report including temperature, windchill, etc. for that day and the next 5 days.

More specifically, the CORGIS Python weather library sends the location string to the Google GeoCoding web API which returns the appropriate latitude and longitude associated with this address. The latitude and longitude values are then sent as parameters in a request to the National Weather Service web API, which returns a weather report in JSON format. From this JSON, the CORGIS library extracts out only the requried fields in order to form a new JSON object. The DataSnap back-end server then sends the weather report to the client front-end, which caches the weather report and selects the appropriate fields from the JSON to return in the method call.

DataSnap exposes the following Data Service API for use by front-end clients. API URL Base: `http://think.cs.vt.edu/snap/api/dataservice`

| **Data Service API**<br>**API URL Base:** http://think.cs.vt.edu/snap**/api/dataservice** | |
|---|---|
| **URL** | **Parameters** |
| /weather | location |
| /stocks | stockQuery |
| /location | location |
| /earthquakes | earthquakePeriod |
| /twitter | twitterFactor, twitterQuery |
| /redditPosts | subreddit |
| /redditComments | postID |
| /urlRequestForClient | urlString |

Figure 4.14: The Data Service API

Each request returns a report object in JSON format which the client is responsible for

traversing in order to acquire the desired data. The first 7 URLs correspond to the 6 data sets for introductory programmers. The last URL corresponds to the "from URL get JSON" block. This Data Service API is utilized by our DataSnap front-end, and we encourage other block-based programming platforms to utilize our Data Service API in order to include data access in their platform.

## 4.5 Conclusion

Current online educational platforms have fallen short in realizing their promise as a means of motivating and retaining students. Given the chance to work with real-time and big data for solving problems that are relevant to their lives, students will be more actively engaged in their courses and more likely to continue to study computer science. In this chapter of the thesis, we have introduced DataSnap, a block-based visual programming language, to set the stage for real-time and big data problem solving in a block-based programming environment. We have provided real-time and big datasets as well as elementary blocks for teachers to define their own data sets. We presented our sample problems and our reference implementation, in order to demonstrate our platform's utility and further explain our implementation details.

# Chapter 5

# Architecture and Affordances of DataSnap Hosted on Open edX

## 5.1 Introduction

DataSnap is an online block-based programming language in which users can drag and drop blocks onto a canvas, snapping together blocks in order to form programs. DataSnap was originally designed for teaching computer science concepts to K-12 students. However, DataSnap currently exists as an online standalone tool and does not provide any integration with online courseware platforms. Thus teachers do not have a scalable way to assign DataSnap programming problems to students because of the following limitations: 1) there is no support for identifying a student; 2) students cannot submit their developed code solution for a grade; 3) student programs cannot be automatically graded.

In this thesis, we have both addressed these limitations and also integrated other features which provide additional benefits. Specifically, we have 1) hosted DataSnap problems which can be administered to hundreds / thousands of online users through the utilization of the Open edX courseware platform; 2) provided a way for teachers to define their own customized DataSnap problems using the DataSnap interface itself; 3) developed a dynamic analysis grading system which is capable of handling real-time and big data access blocks and 4) provided a robust data collection system for collecting interaction data from students.

Our work in this thesis is built upon the block-based programming environments that already exist. Other online block-based programming systems such as Google's Hour of Code or Blockly's sample of hosted games offer students the chance to receive feedback from their code. Our project is distinguishable mainly by its capability to offer teachers the ability to define their own customized block-based programming problems to give to students, and by the inclusion of real-time and big data access blocks for use by students. The contribution of this paper includes three main components:

- We are the first to present instructor-based development of customized block-based language problems which can be developed in the block-based language itself. We have implemented this model in the DataSnap programming environment. This model can be implemented in any block-based language.
- We present a model for performing dynamic analysis grading for programs which use real-time and big data access blocks in block-based languages.
- We present our system for user-related data collection for block-based languages. Our system captures user data for further analysis by data analysts and teachers of online courses.

In this paper, we will introduce our design requirements in section 2, an overview of our DataSnap xblock in section 3, and DataSnap problem definition in section 4. Section 5 and 6 will discuss the dynamic analysis process and interaction tracking data, respectively. Lastly, section 7 will discuss the educational affordances of hosting DataSnap in an online course platform, and section 8 will conclude.

## 5.2 Design Requirements

As mentioned in the introduction, our design requirements for this project included issuing customized DataSnap problems to students, keeping track of which students accessed these problems, giving grades, and collecting student information. One option was to develop the infrastructure for supporting student authentication, grading systems, displays, and notifications ourselves, however, there are plenty of learning management systems (LMS) that are available that can be utilized. Thus our first design requirement was that we needed to utilize a learning management system to provide basic classroom management services. We chose to utilize the open-source open courseware development platform called the Open edX platform because it provided an infrastructure to support all of these services. In comparison to other LMS systems, such as Scholar and Sakai, Open edX is most suitable for systems that are offered to large online audiences, such as in Massive Open Online Courses (MOOC) courses, or for hosting projects that are complementary to a high school / college course's material.

Our second design requirement was that our DataSnap problems would have to include access to real-time and big data. As part of our previous work, we developed real-time data access blocks which acquired National Weather Service data, Google Finance stocks data, Google geocoding data, United States Geological Survey earthquakes data, Twitter data, and Reddit data. The main motive of this requirement was to parallel the motive behind our work on Snap for introductory programmers: to allow students and teachers to work with data that is relevant to their lives, in the hopes of convincing students that computer science is important and related to their day-to-day experiences. It is our aim that this work will

encourage more students to study computer science and retain the population of students that is currently studying in this field. Mentioned earlier in this paper, but presented once again for the reader's convenience, the real-time and big data access blocks are shown in Figure 5.1.

Our third design requirement was that defining a customized DataSnap problem to give to students must be so simple that it would require no previous programming knowledge on the teacher's part. We didn't want the teacher to have to open up another programming environment and develop code in Python or JavaScript, for example, in order to define their customized problem. Thus we decided that the process of defining a customized DataSnap problem must be able to be completed directly in the block-based DataSnap interface itself. This requirement meant that teachers could define their customized DataSnap problem in the abstracted, higher-level DataSnap interface in which they and their students are already familiar. We believe that this design requirement will make defining a DataSnap problem accessible to an audience ranging from experienced programmers to those that have no experience in traditional programming languages.

Adherence to this requirement also provides users the opportunity to be able to call web API's to access data through the use of our data-access blocks rather than through writing complex script-based web API calls in Python or JavaScript. Thus this design requirement also supports our work of allowing students to work with data that is relevant to their everyday lives.

## 5.3 DataSnap Xblock Overview

### 5.3.1 Introduction to the Open edX Courseware Platform

First, we aim to familiarize the reader with the Open edX framework. One definition of Open edX is: "the open-source release of the edX platform developed by the non-profit organization founded by Harvard and MIT" [79]. EdX is designed as a platform which can help teachers offer interactive online classes and MOOCs. The Open edX system contains three different views: a user view, an author view, and a research view. The user view is typically a student's view, which allows the student to take courses, view their progress, participate in forums, wiki, online chat, email, view a class calendar, and many more other functions. The author view, which is also known as the "studio", is where teachers can create and customize courses. Lastly, the researcher view is a view designed for teachers or researchers who would like to perform data analytics. Our work touches each of these three views, which will be discussed shortly.

However, first we will discuss the internals of what makes up an EdX course. The basic building block of an EdX course is known as an xblock. Xblocks are "Python classes that implement a small web application. Like full applications, they have state and methods,

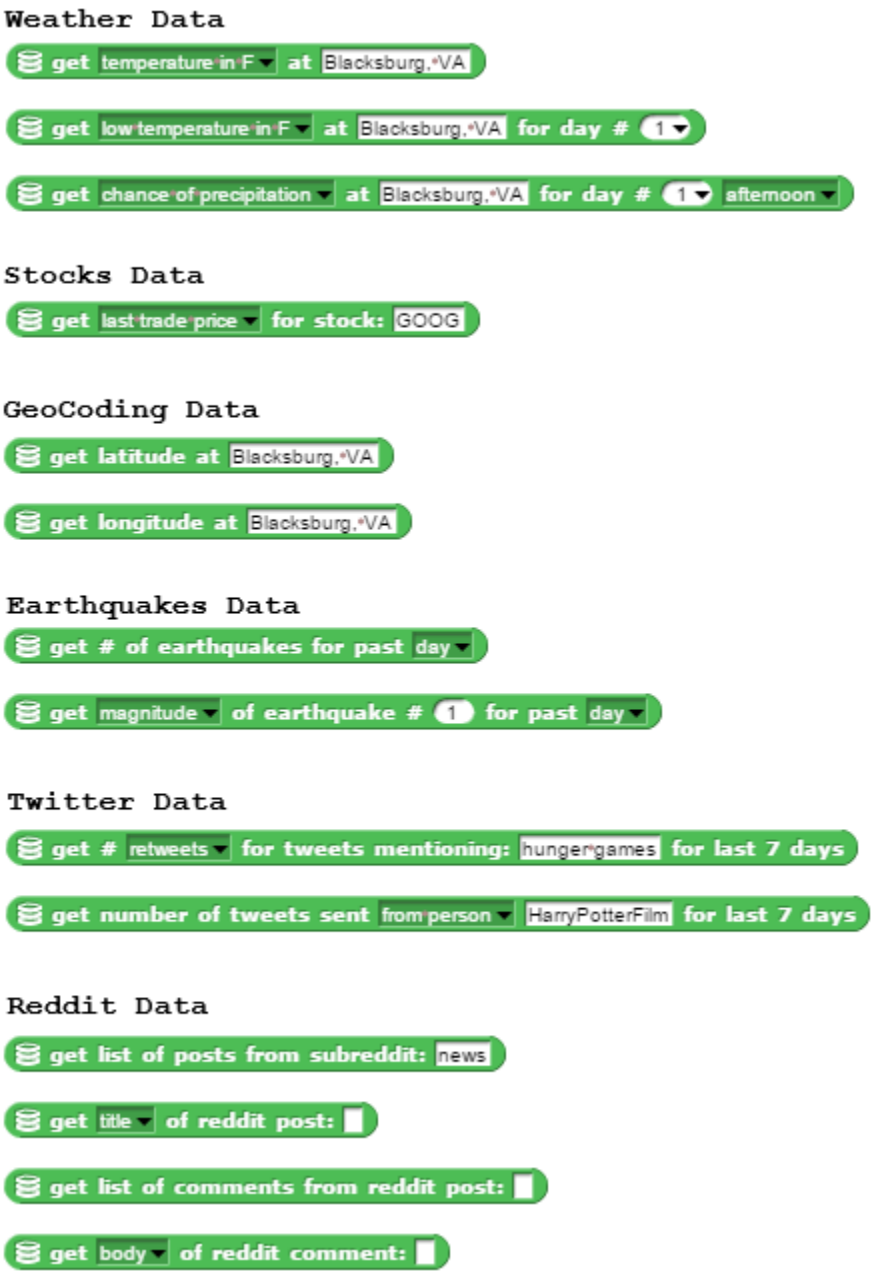



Figure 5.1: Each of the six data set's programming blocks, which provide simple and convenient access to data.

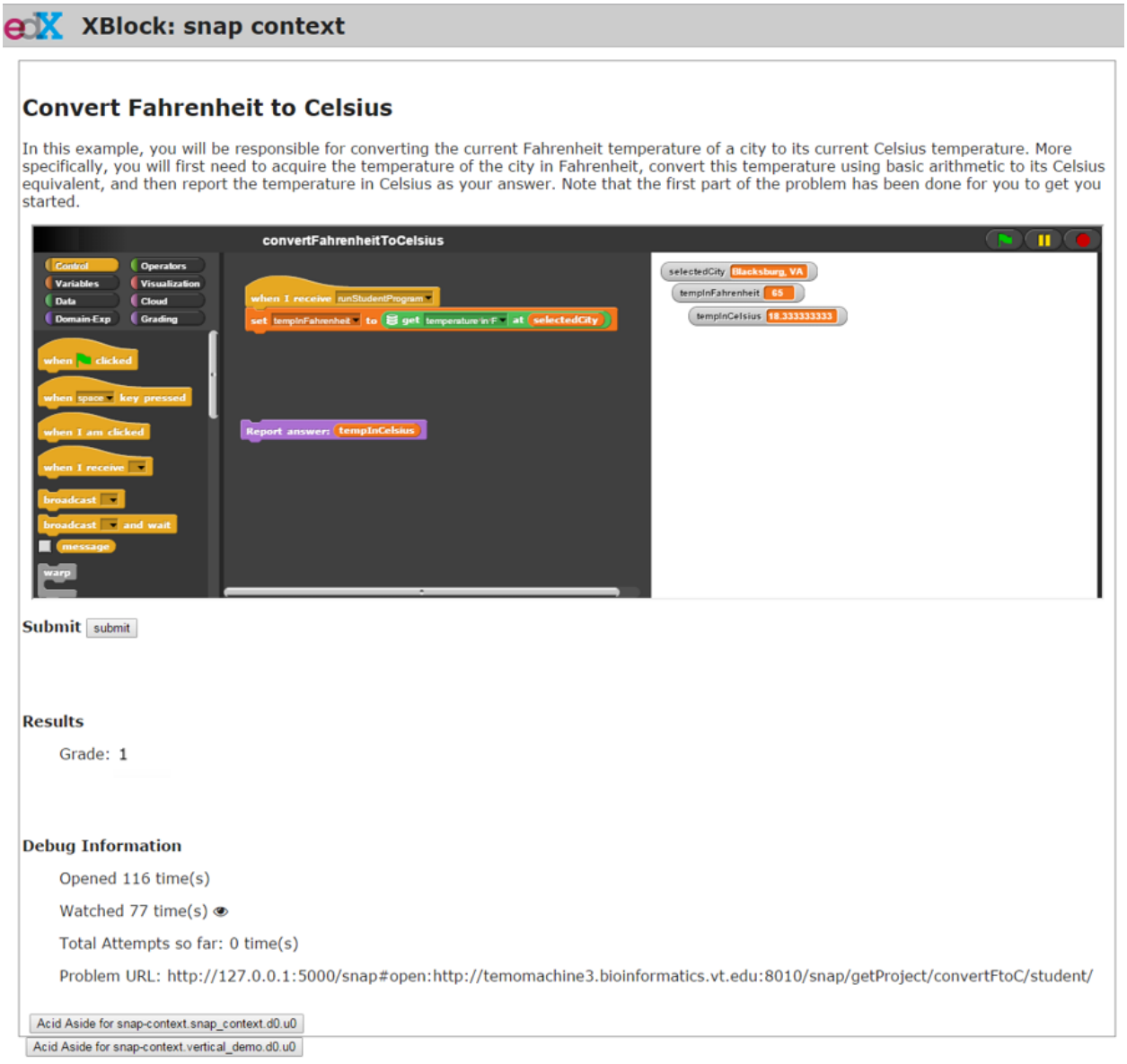



Figure 5.2: The DataSnap XBlock.

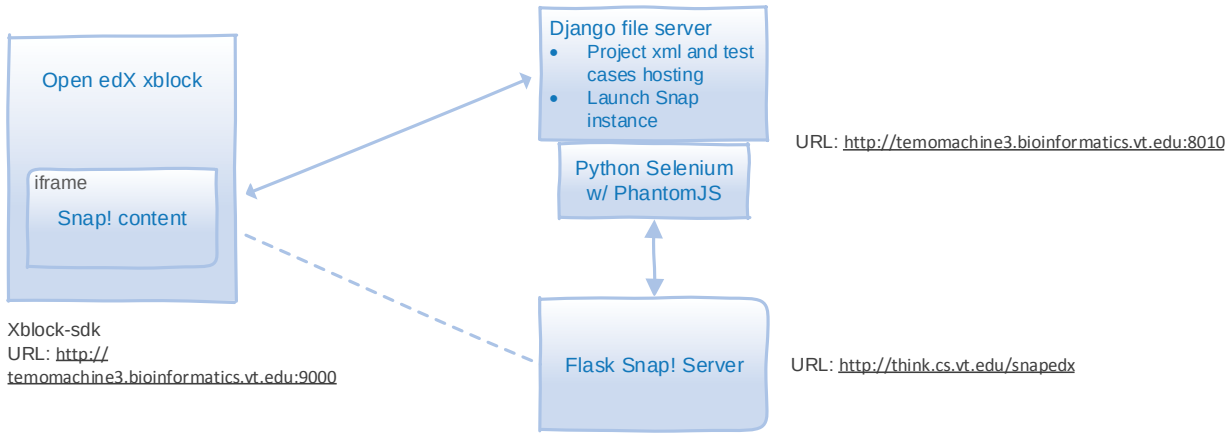



Figure 5.3: The project architecture

and operate on both the server and the client" [27]. Xblocks do not run by themselves but run within an xblock runtime. The xblock runtime provides services such as storage, URL mapping, analytics, user authentication, progress tracking, grading, and forums. Example xblock runtimes include edX Studio, edX LMS, and the Xblock Runtime for Google App Engine. Each of these runtimes can be used by teachers to manage their online courses.

### 5.3.2 DataSnap Xblock

In our project, based on the Open edX infrastructure already in place, the proper way for us to host our DataSnap content would be to host it in an xblock which could be run by an xblock runtime. Figure 5.2 shows our developed xblock loaded with a temperature conversion problem as a sample programming exercise. In this problem, the teacher has used the Open edX studio view to customize their own DataSnap problem xblock. Each xblock consists of a problem title, problem text, the DataSnap programming environment hosted in an iframe, a submission button, and a grade. Some additional debugging information is provided at the bottom.

When the student accesses the xblock, they are presented with the Open edX user view of the xblock. It is the student's responsibility to read the problem text and then drag and drop blocks from the block palette on the left of the DataSnap environment to the scripting pane in the middle, in order to form a program to solve the problem. The student's variables are displayed on the right side of the DataSnap environment. Once the student believes they have formed a valid program, they can click on the submit button, which will run a dynamic analysis grading test. Their grade will then be displayed in the results section.

Although providing teachers with a way to host their own customized DataSnap problems is of primary importance, collecting data about student interaction is also very important for teachers and data analysts. The data that our xblock collects includes: 1) the number

of times the problem was opened; 2) the number of times the problem was watched, with 10 seconds or more of having the window open as the criteria for what counts as one watch count; 3) the total number of submission attempts; 4) The student program xml which corresponds to their highest graded program and also their last submitted program; 5) The student test outputs of the dynamic analysis; 6) The teacher test outputs of the dynamic analysis; 7) Whether the problem was fully solved (Student received a 100 percent score); 8) interaction tracking data. The DataSnap interaction tracking data will be discussed more fully in one of the following sections. This data is collected per student and is accessible to teachers via the Open edX researcher view.

Our architecture, as shown in Figure 5.3, involves a deployed xblock-sdk with the DataSnap content hosted in an iframe. The xblock-sdk is the Open edX the testing platform for developers to test their xblocks. On the right side of the figure depicts our Django server. The Django server is used to 1) store the teacher program xml, student skeleton xml, and test case files 2) generate the URL required to launch a snap instance with the student skeleton xml 3) run the dynamic analysis grading test for the teacher program using Python Selenium with PhantomJS. Our customized DataSnap content is hosted at http://think.cs.vt.edu/snapedx and includes the real-time data access and grading blocks. The DataSnap server is built on the lightweight Python Flask microframework. Because three servers are used to host content, a communication method is needed in order to exchange information. Communication between the xblock and the DataSnap iframe occurs by the window.postMessage() JavaScript method. Communication between the xblock and the Django server occurs by HTTP Ajax requests.

## 5.4 DataSnap Problem Definition

Now we will discuss how a teacher can define their own customized DataSnap problem. First let us start with the general definition for a function: f is a function which requires inputs $x_1$, $x_2$ ... $x_n$, and which generates outputs $y_1$, $y_2$, ... $y_n$ : ($y_1$, $y_2$, ... $y_n$) = f($x_1$, $x_2$, ... $x_n$). Breaking down this problem we have:

1. x: program inputs
2. f: program algorithm
3. y: program outputs.

Therefore, for the definition of a problem, a teacher must be required to define the variables which are associated with the program inputs, the variables which are associated with the program outputs, and also provide a program (algorithm) which can use the inputs to produce the outputs. This program will be used to run dynamic analysis to grade the student

program. Let us use a temperature conversion problem as a simple example: A student will be responsible for writing a program to convert the current Fahrenheit temperature of a city to its current Celsius temperature.

In this example, the program input, defined by the teacher, can either be a Fahrenheit temperature or the name of a city of which the current Fahrenheit temperature can be acquired. We will choose the latter option, the city name, as this will let the student work with the temperature of the city that they are in at the time that they are developing the solution. The program output can be defined as the temperature in Celsius.

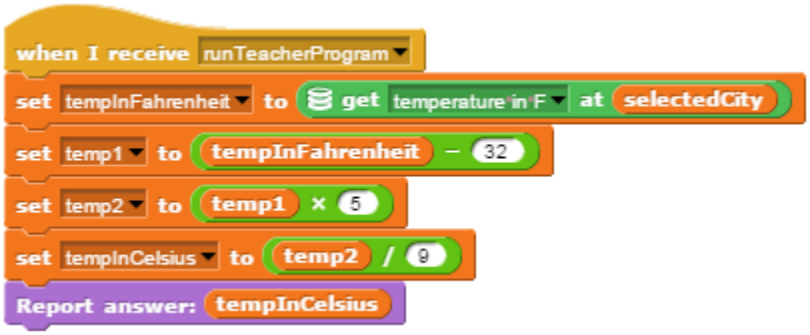



Figure 5.4: The teacher program for the convert Fahrenheit to Celsius problem.

The teacher can then use the block-based DataSnap programming language to form a program which converts the temperature in Fahrenheit to the temperature in Celsius. A sample teacher program is shown in Figure 5.4. This teacher program is exported as an xml which will be used in the dynamic analysis grading process discussed later.

The next required item is the student skeleton program, which is a DataSnap program that will load when the student opens the xblock. The student skeleton program will define what DataSnap blocks and what variables the student will start out with when they load their xblock. This program is also exported as an xml, and its definition is optional. If this program is not defined, the student will start out with the standard DataSnap interface with no blocks or variables added.

The last required item from a teacher is a set of test inputs to be used for the dynamic analysis test cases, written in JSON format. The test cases for this example are shown at the top of Figure 5.5. The first JSON object contains the name of the first input variable name as the JSON object key and an array of values which serve as the test case inputs as the JSON object value.

## 5.5 Dynamic Analysis

Our second and third design requirements, namely, utilizing the real-time data access blocks and letting teachers define a DataSnap problem in the DataSnap language itself, lead to some required changed in our dynamic analysis design. For our dynamic analysis, although initially we planned to run the teacher tests ahead of time and store the results, we now had to run both the student tests and the teacher tests at the same time to ensure both tests ran using the same up-to-date data. The resulting dynamic analysis control flow is shown in Figure 5.5.

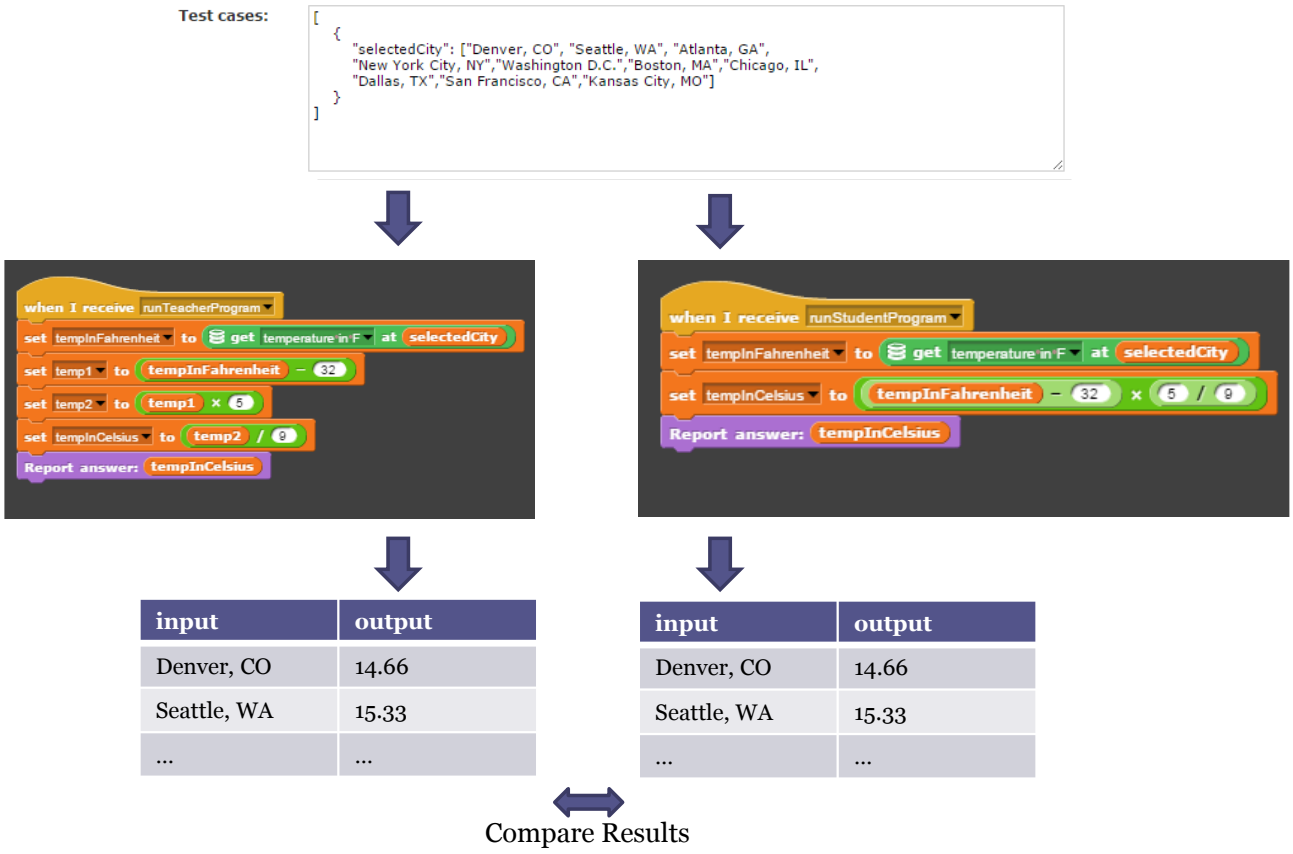



Figure 5.5: The dynamic analysis grading control flow.

In the dynamic analysis process, the student outputs are created by running the test cases directly in the student's browser instance of DataSnap and sending the student outputs from DataSnap over to our xblock. In our example, first the input variable "selectedCity" is assigned to the value of the first test case input, which is "Denver, CO" in this case. The student program is then run and the temperature in Celsius is reported using the "Report answer" block. This process is repeated for each of the test case inputs, with the reported values stored in an array as the student test outputs. Note that for each invocation of

the student program, the "get temperature in Fahrenheit at (selectedCity)" block will run, causing a web api call to the National Weather Service to be invoked. When the tests are finished, the following data will be passed back to our xblock:

- finished: Boolean - Indicates whether the dynamic analysis finished.
- errorMessage: String - An error message which describes why the dynamic analysis failed.
- studentOutputs: array - An array of values as the dynamic analysis outputs.
- programXML: String - The program represented in xml format, stored as a string.

Figure 5.6 shows the message communication between the xblock and DataSnap, and also introduces the grading loop which is the main component of the dynamic analysis. Before any messages are passed, event handlers are registered to associate a message type with a JavaScript function. For example, when the MESSAGES_TYPE.RESULT message is received by the xblock, the handle_results_from_snap JavaScript method will be invoked. Event handlers must be used for message communication between the xblock and the DataSnap content in the iframe because JavaScript does not allow blocking calls, and furthermore, web browser interaction should not be halted while the tests are being performed.

When the submit button is clicked, a message is sent from the xblock to DataSnap. This message is handled and the student tests are started. Because each individual student test may use web API calls to retrieve real-time or big data from external sites, we cannot be sure how long each individual test may take. Therefore, the gradingLoop function is called every 100 milliseconds to manage the dynamic analysis control flow.

Each time the gradingLoop function is run, it checks to see if it can start the next test by checking the individualTestsStatus array. Each index of individualTestsStatus has a value that is either "complete" if the test has already run on this input number, "running" if the test is still running on this input number, or "incomplete" if the test has yet to be run. The gradingLoop function will start the next test according to the values of this status array. Once all of the individual tests are finished running, a message is sent from the DataSnap content in the iframe to its parent, which is the xblock. The xblock then receives the message and calls the appropriate functions to store the student outputs, student program xml, compare the student outputs with the teacher outputs, and display a grade to the student. If an error occurs during the grading process, an error message is passed back to the xblock. If the student tests do not return a message within 30 seconds, the xblock assumes an error has occurred and handles this situation appropriately.

The dynamic analysis process for the teacher program occurs somewhat differently however. For our grading process, our third design requirement also meant that instead of running a server side Python script in order to grade a student's program, we would need to initialize an

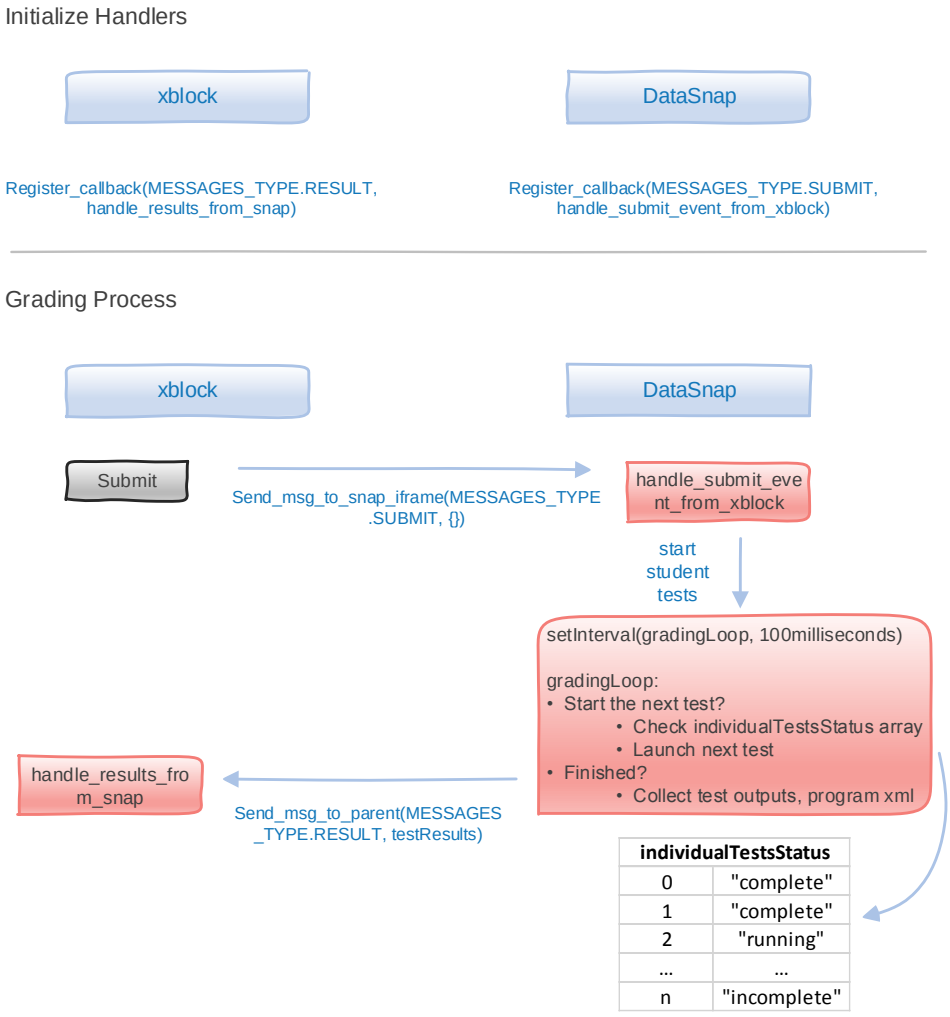



Figure 5.6: Message communication between the xblock and the DataSnap iframe and the grading loop used to perform the dynamic analysis.

instance of DataSnap in order to run the teacher program. We decided we would initialize an instance of DataSnap on our Django server, import the teacher program, run the test cases, and then retrieve the teacher outputs. We utilized Python Selenium, which is a package "used to automate web browser interaction from Python" [31] and PhantomJS, which is a "headless WebKit scriptable with a JavaScript API" [40] in order to initialize the headless DataSnap instance on our Django server. We passed the name of our dynamic analysis grading function, "startTeacherTests" to the DataSnap instance in order for the teacher tests to start. When the tests finish, the teacher program data will be returned to our Django server. Upon the return of this data to the Django server, a handler method will be invoked, and will forward this data to the DataSnap xblock to be compared to the student output data.

## 5.6 Interaction Tracking Data

This paper presents a system for user-related data collection for block-based languages. Specifically, our system records information for the following interaction events:

- drop a block
- delete a block
- button press
- run script
- rename a variable

When a block in DataSnap is dragged from the block palette, it must be dropped somewhere on the scripts pane. However, there are a few ways in which it can be dropped. The block can be dropped with a position described as "UNDER" another block, "ABOVE" another block, or simply "ON" the scripting area. The position field is used to describe where the block is dropped. Next, the target field describes the associated position target, which may be another block or the scripting pane. The text below is an example of a DROP interaction event, in which the "doAnswer" block is dropped under the "doSetVar" block:

> Object type: "DROP", block: "<block s="doAnswer">¡block var="tempInCelsius"/></block>", position: "UNDER", target: "<block s="doSetVar"><l>tempInFahrenheit</l><block ...></l><block var="selectedCity"/></block></block>"

Delete events simply record the message type "DELETE" and the target block which is deleted. A button press records the message type "BUTTON_PRESS" and the name of the button that was pressed. Run script events record the message type "RUN_SCRIPT" and the block associated with the script which is executed. Renaming variable events record the message "RENAME" and the new and old variable names.

Many more events can be captured from the DataSnap programming environment, however we chose to focus on these five fundamental events. These events are critical in analyzing the student program structure as it is being built by the student. These events also record how many times the student runs their own program, which can indicate how many times they have tested their program. The events also indicate where variable renaming was required, which may indicate a possible restructuring or refactoring of their program. We believe a set of powerful static analysis methods can be built using both these interaction events and the student program xml.

## 5.7 Educational Affordances

We believe that DataSnap problems hosted in Open edX offer multiple educational affordances to students. The following affordances are offered by DataSnap with or without hosting on Open edX: data collection and authentic context-awareness; and the following affordances are offered by DataSnap hosted in an Open edX context: methodical analysis, real-time evaluation, and traditional/MOOC class hosting capabilities.

DataSnap users are exposed to an environment that is capable of retrieving data from external sources through the use of data access blocks. Data retrieval was one of the main additions which sets DataSnap apart from the original Snap! platform. In Snap!, users would have to manually type in their data into text fields or lists. DataSnap offers a dynamic way of retrieving data: through web API calls. Users are able to collect this data, store it in various data structures, and perform data operations to meet their goals. Thus data collection is the educational affordance offered by DataSnap which helps users to collect relevant data, relate to their data, and use data to solve real-world problems in their everyday lives.

The real-time and big data sources also contribute to a sense of context-awareness for the user. The context of the user includes the environment in which they currently reside, which includes the current weather, Twitter tweets, Reddit posts, stock market information, and any other live information that is part of the user's environment. Furthermore, with the number of internet connected devices reaching a number around 10 billion [73], it is evident that we live in a world that is rich in data. Students in today's society would do well to be aware of how to take advantage of the many real-time data sources that are available, so that they could make meaningful decisions in their future jobs and careers. We believe DataSnap gives users a chance to practice computer science and computational thinking

concepts while working in an environment that is rich in data. A typical K-12 or college course often places the user in an environment with a "stale" context, such as one with pre-loaded or old data. We, however, believe the user should be placed in an environment which more closely matches their current context which is rich in data, and we believe DataSnap is a step in the right direction towards achieving this goal.

DataSnap also encourages users to use methodical analysis, or the "analysis means to break material into its constituent parts and to determine how the parts relate to one another and to an overall structure or purpose" [74]. To solve a programming problem hosted on DataSnap in Open edX, users must break down the question into its parts and solve each part. The user is required to form together blocks which are interrelated in order to achieve an overall purpose. Users can define their own methods and variables, and form their own program structure. Furthermore, each student's program structure may be slightly different. We believe DataSnap fosters the development of program planning, critical thinking, and problem solving skills.

DataSnap offered in Open edX also offers the real-time evaluation affordance. Through the dynamic analysis grading process, students are able to receive timely feedback on their developed program. Students can then modify their program if they did not achieve the grade they would like to have, or simply if they would like to restructure or reorganize their program, and resubmit their program. The real-time evaluation leads to a shorter feedback cycle. It is plausible that students limit their number of submissions based on the time and effort required to receive feedback on their submissions. The dynamic analysis grading process leads to improvements in this area, in comparison to the traditional model of letting students submit once and having the teacher grade the submissions manually after the deadline. We believe DataSnap's use of real-time evaluation will lead to shorter feedback cycles, more program interaction and therefore an opportunity for students to better understand the concepts taught by the teacher.

Lastly, DataSnap with a dynamic analysis grading infrastructure also opens up the possibility for hosting problems in a traditional or MOOC class. Our DataSnap xblock can be installed in an Open edX platform and used by teachers for programming assignments or projects at scale. Previously, without the existence of a DataSnap xblock with dynamic analysis, it would be unrealistic to expect teachers to be able to manually grade a large number of student programs. With our work, it is now possible for teachers to provide these programming problems to hundreds of students and also to let the students receive timely feedback during the submission phase of the problem. Grades received on a DataSnap programming problem can be integrated with a gradebook assignment on Open edX, removing the need for manual grade entry. Thus teaching computational thinking and computer science concepts using a Snap! based environment can be performed with a larger audience scale than was previously possible.

## 5.8 Sample Problems

This section will describe a few of the DataSnap problems hosted on Open edX. The first example is the convert Fahrenheit to Celsius problem that was brought up earlier in the paper.

The second example involves using the earthquakes data set to find the earthquake that has occurred within the last time period that has the highest magnitude. Using some form of iteration is encouraged, as there may be a large number of earthquakes that have occurred. There are often 100 - 500 small earthquakes that occur each day. The student is provided with the skeleton program shown in Figure 5.7 to start the program with.

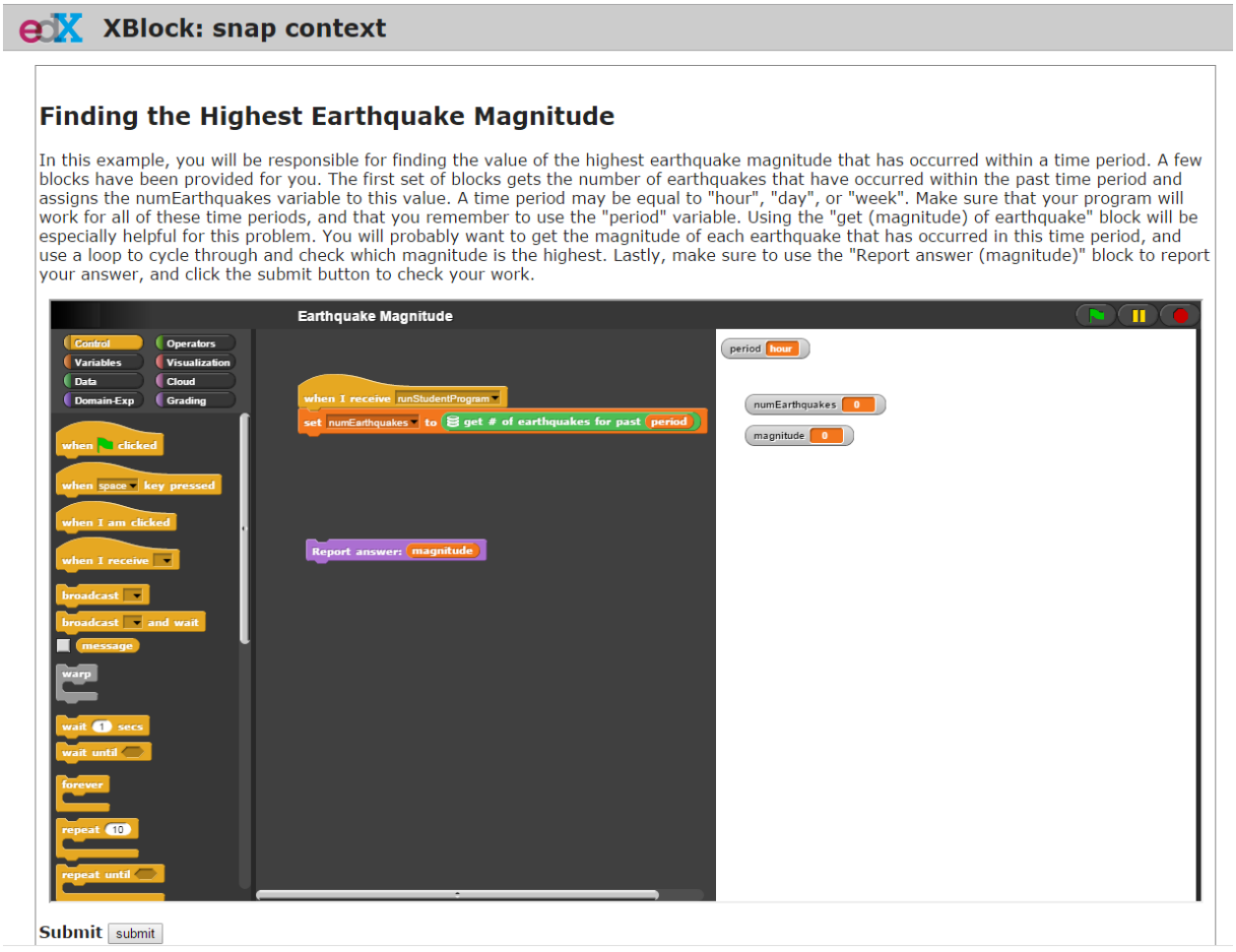



Figure 5.7: Earthquake Magnitude Problem - Problem Skeleton

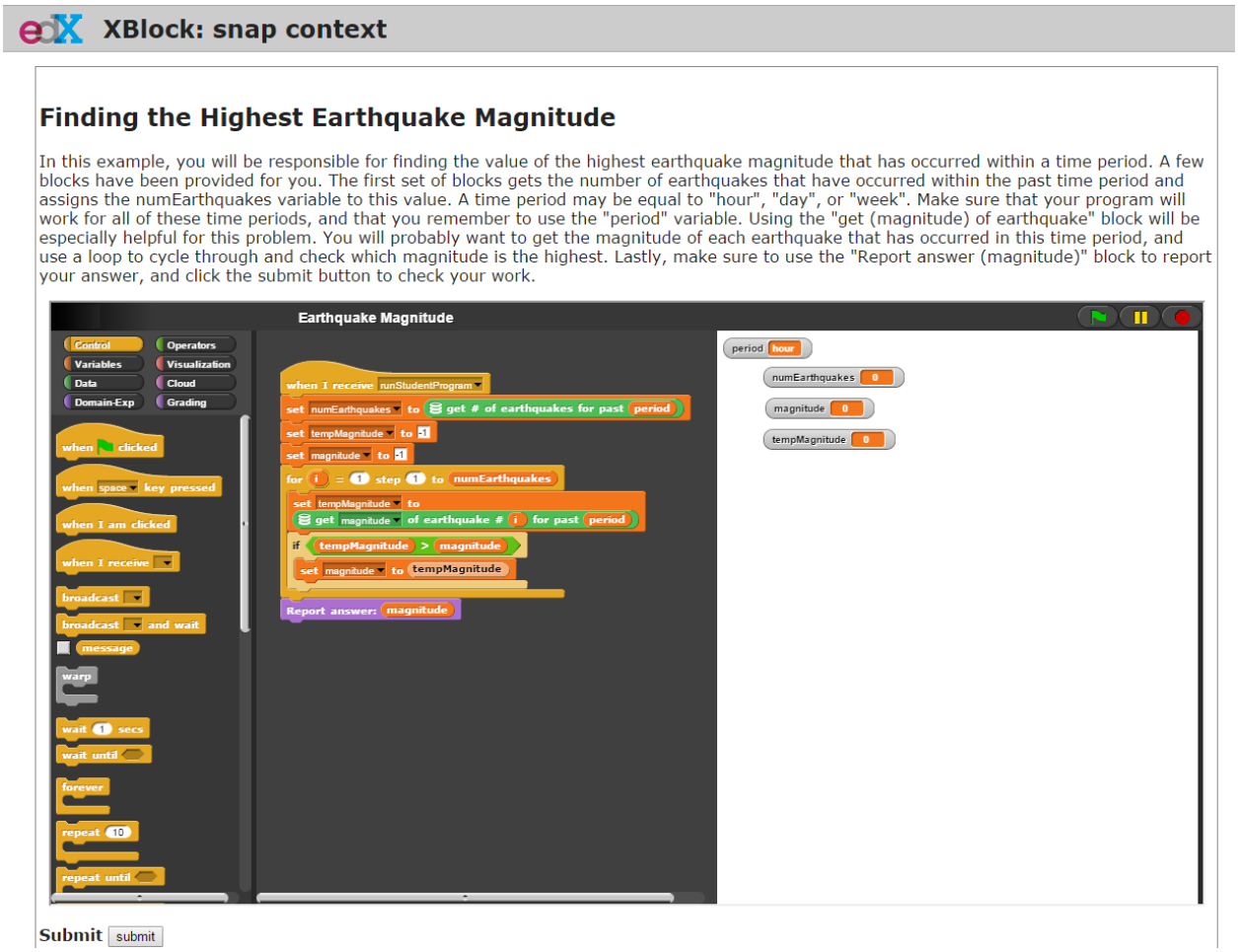
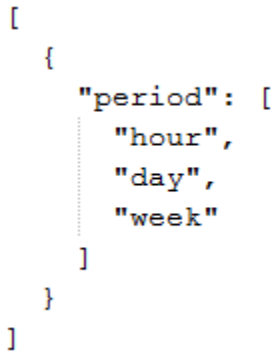



Figure 5.8: Earthquake Magnitude Problem - Student Solution

A sample student solution is shown in Figure 5.8, and the dynamic analysis test inputs are shown in Figure 5.9. The student solution shown uses a “for loop” structure in order to solve the problem. Notice that whenever the data set blocks are used, the “period” variable is used, so that the period can be changed and the program will find the highest earthquake magnitude for this period. When the dynamic analysis tests run, the student and teacher program is run with period equal to “hour”, “day”, and “week” sequentially. The student outputs are compared to the teacher outputs in order to determine the student grade.



```
[
  {
    "period": [
      "hour",
      "day",
      "week"
    ]
  }
]
```


Figure 5.9: Earthquake Magnitude Problem - Test Inputs

The third example is an extension of the second example. It involves having the student store the index of the earthquake with the highest magnitude, and then retrieving the location description of this earthquake. The student program skeleton is shown in Figure 5.10. The test inputs for this program are the same as in the previous problem. A sample student solution is shown in Figure 5.11. This sample problem helps to further enrich the student's skills in iteration through practice.

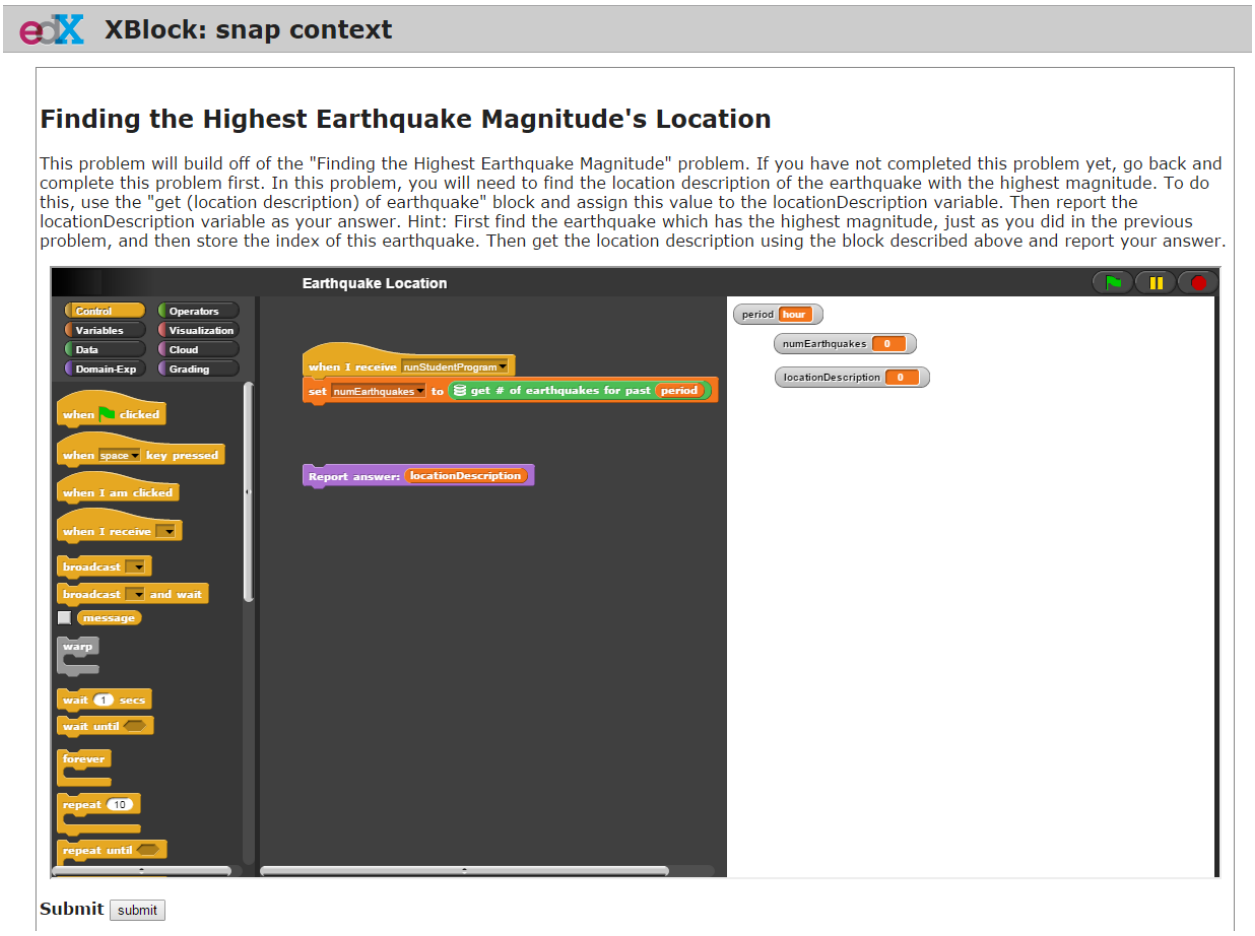



Figure 5.10: Earthquake Location Problem - Problem Skeleton

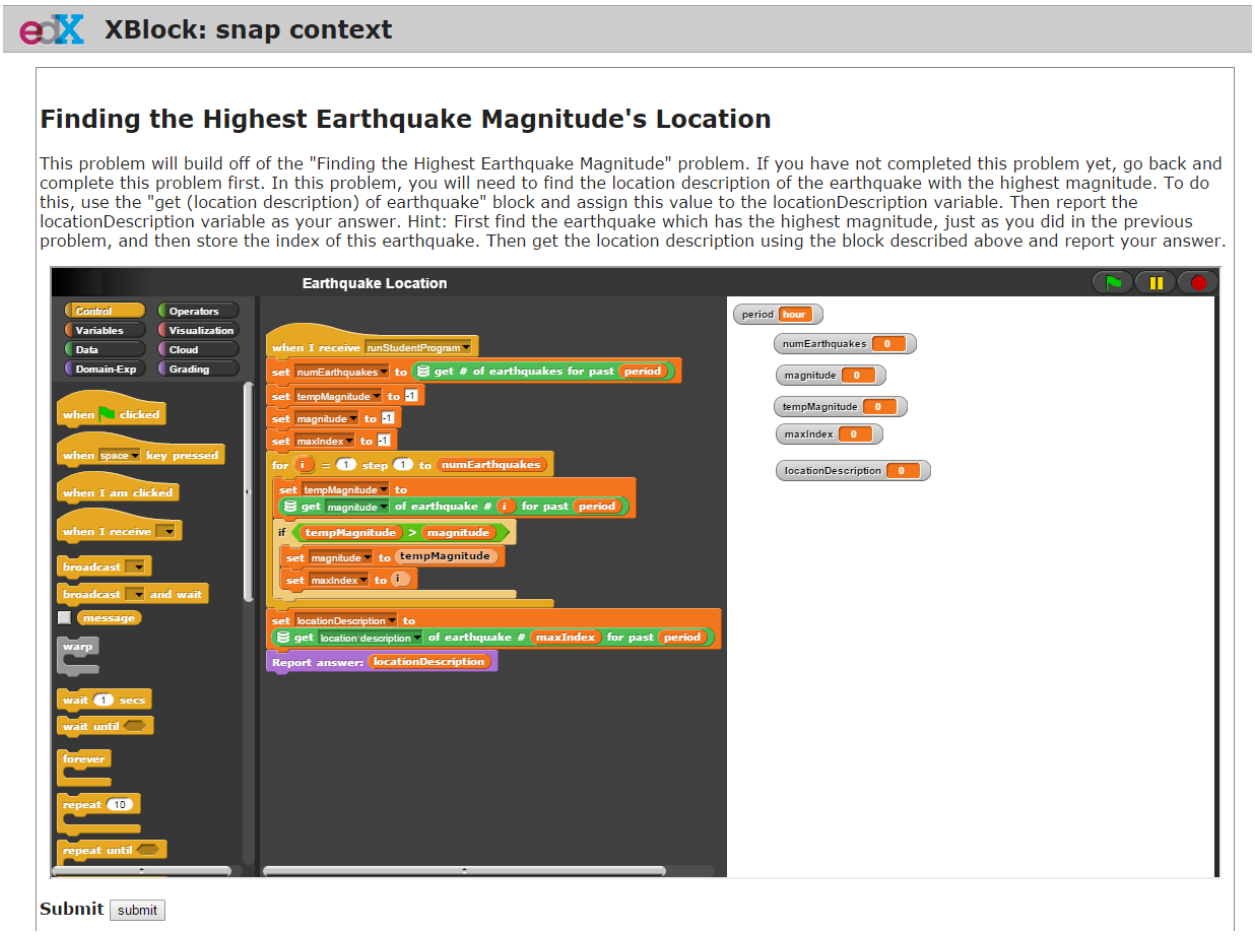



Figure 5.11: Earthquake Location Problem - Student Solution

Our fourth example is a problem that uses the stocks data set and simple math calculations. The student skeleton and student program are shown in Figures 5.12 and 5.13. The test inputs are shown in Figure 5.14. In this problem, students will practice retrieving data, performing division, and rounding numbers. This program is simpler than the earthquakes programs and can be used to help students get more familiar with how to declare and use variables.

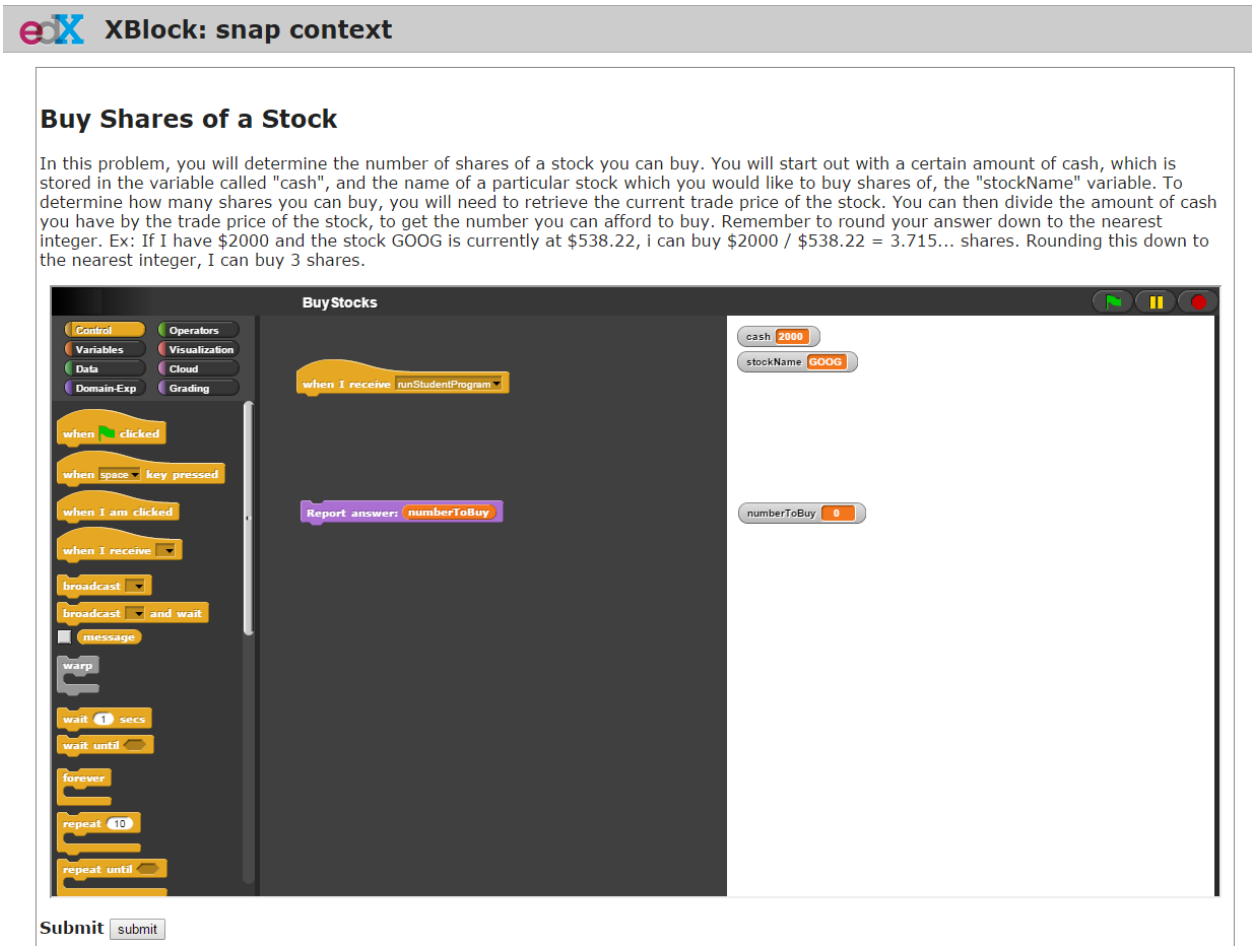



Figure 5.12: Buy Shares of a Stock Problem - Problem Skeleton

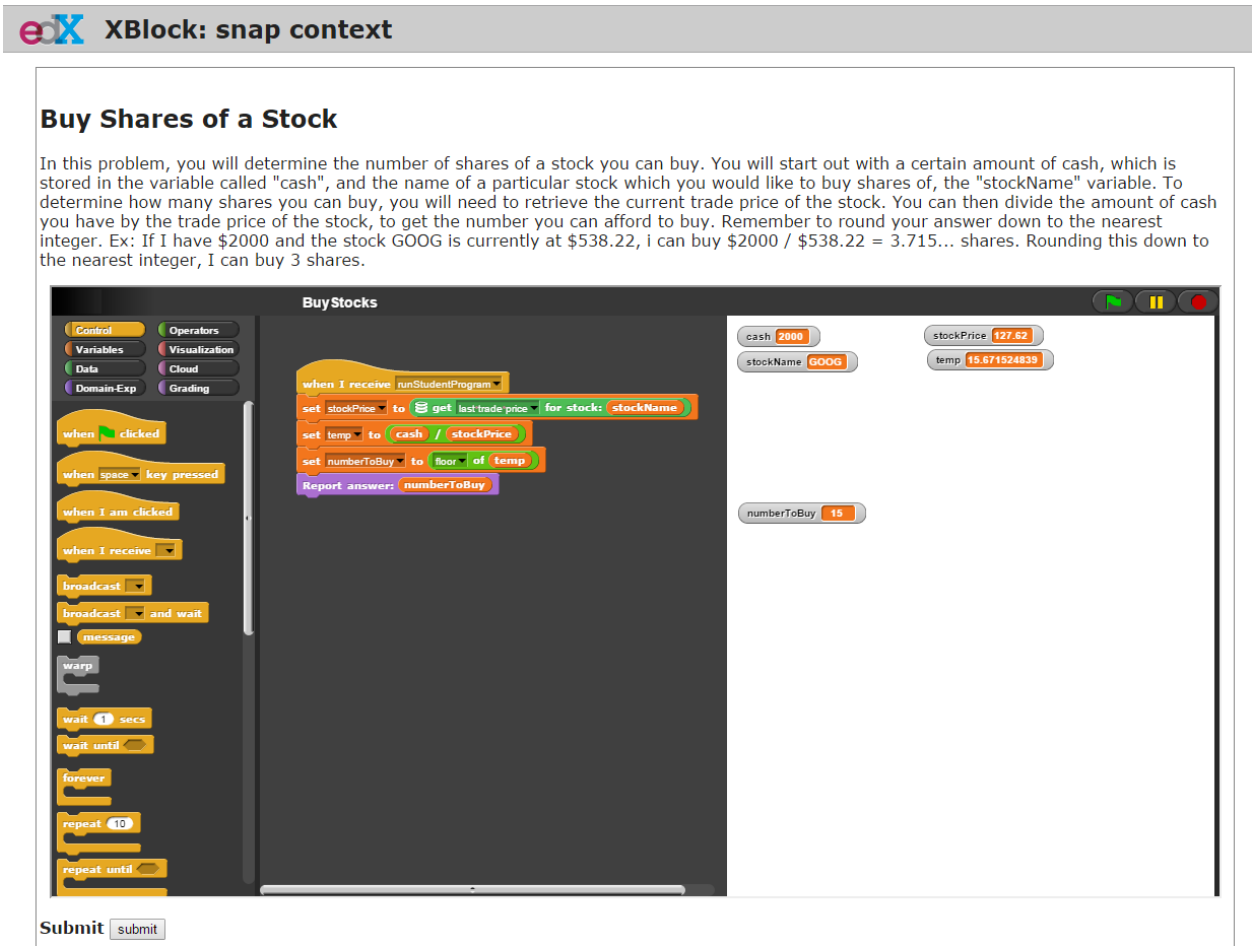



Figure 5.13: Buy Shares of a Stock Problem - Student Solution

```
[
    {
      "cash": [2000, 2000, 2000, 2000, 1, 1, 1, 1, 0, 0, 0,
      0, 100000, 100000, 100000, 100000, 962.99, 962.99,
      962.99, 962.99]
    },
    {
      "stockName": ["GOOG", "AAPL", "MSFT", "AMZ", "GOOG",
      "AAPL", "MSFT", "AMZ", "GOOG", "AAPL", "MSFT", "AMZ",
      "GOOG", "AAPL", "MSFT", "AMZ", "GOOG", "AAPL", "MSFT",
      "AMZ"]
    }
]
```

Figure 5.14: Buy Shares of a Stock Problem - Test Inputs

## 5.9 Conclusion

In conclusion, this paper presents the contribution of our model of instructor-based development of customized block-based language problems which can be developed in the block-based language itself. Our implementation is realized through the DataSnap programming language hosted in the Open edX courseware development platform. The paper presents a model for performing dynamic analysis grading on programs which use real-time and big data access blocks in block-based languages. Lastly, our paper presents our system for user-related data collection for block-based languages. The collected data is stored in the Open edX platform for further analysis by teachers and researchers.

# Chapter 6

# Future Work

Our work has opened up a few more avenues of research in the realm of block-based programming languages. The following sections describe our planned future work in a number of different areas: expanding visualization support in Section 6.1, configuring Virginia Tech NDSSL simulations in Section 6.2, expanding support for domain experts in Section 6.3, introducing DataSnap in educational settings in Section 6.4, and DataSnap in Open edX: test cases, programming patterns, and course hosting in Section 6.5.

## 6.1 Expanding Visualization Support

As mentioned in Section 3.4, DataSnap includes a Google Map visualization along with its block-based programming blocks in order to add markers, circles, and points. However, adding more visualization displays and blocks to DataSnap would be very beneficial towards both domain experts and introductory programmers. Along this trajectory, research towards what visualization blocks should be paired with what visualization displays can be performed.

A few example visualization displays include line graphs, bar graphs, and pie charts. These displays could serve a useful purpose in aiding both domain experts and introductory programmers. Domain experts often need to chart their data, and teachers and students in courses could make use of these additional data visualizations. Visualizations can be integrated through utilizing a JavaScript library such as C3, D3, Highcharts, or a library not mentioned.

Along with the added visualization displays (line graph, bar graph, pie chart, etc.), blocks would need to be designed and developed which can programmatically configure the visualization displays. For a line graph, for example, the blocks "Set y-axis scale to (number)", "Set y-axis data to (list)", and many other blocks would need to be developed. This research trajectory begs many questions, a few of which are listed as follows: "What visualization

blocks are required to configure the visualization displays?", "Can one visualization block be paired with multiple visualization displays?", and "Can this type of programming be introduced to students to teach computer science or computational thinking concepts?".

## 6.2 Configuring Virginia Tech NDSSL Simulations

On Monday, April 27, 2015 DataSnap was presented to the Network Dynamics Simulation Science Laboratory (NDSSL) at Virginia Tech. As defined on their site, NDSSL "creates informatics tools for societal problems by developing synthetic information systems and associated analytical methods appropriate for very large complex systems" [75]. A few of NDSSL's applications include:

- SIBEL, which "supports running numerous experiment situations that generate distributions of outcomes to gain an appreciation of the time-varying state (the dynamics) of an epidemiological event."
- FluCaster, which is a "disease surveillance and situation assessment tool that uses social media crowd sourcing and complex mathematical models to track and predict the spread of communicable diseases."
- Synthetic Information Viewer (SIV), which is a web-based tool used to visualize a synthetic population at a desired level of disaggregation.

From our meeting we discussed utilizing DataSnap in three ways:

1. To specify simulation parameters.
2. To specify the data that needs to be transferred from one application to another application.
3. To customize and configure user view options.

Regarding the first item, "To specify simulation parameters", in applications such as SIBEL, FluCaster, and SIV users must specify their simulation parameters. Let us take a malaria epidemic simulation in the SIBEL application as an example. In this application, a user can run a simulation with the parameters: social distance intervention with a compliance level from 0 - 100% by 20% intervals; a closing of primary education schools intervention with a compliance level from 30 - 100% by 10% intervals; an antiviral prophylaxis intervention on a subpopulation of healthcare workers when the disease reaches a percent infected rate of 0.5% to 3% by 0.5% intervals. Based on the combination of the user's stated parameters, the SIBEL application will run 6 x 8 x 6 = 288 simulations.

However, in the state of a malaria outbreak a policy official must act quickly and each simulation can take quite some time to perform. Instead of having the application run a simulation for each combination of all parameters, a block-based language such as DataSnap can be used to let the user specify the combinations of parameters for which they would like to run simulations. The block-based language would give the user more power to specify the relationships between each parameter also. For example, a user can specify that the social distance intervention variable X should vary based on a bell curve centered at 40%, that the school closure variable Y should vary linearly, and that the percent infected rate Z should be determined by an exponential growth rate.

With the user's enhanced control of the specification parameters, two valuable benefits can be achieved: 1) greater control over the specification of parameter values; 2) an opportunity to reduce the number of total simulations that need to be run, saving simulation processing time, and thus saving human lives during an outbreak situation.

Regarding item 2, "To specify the data that needs to be transferred from one application to another application," DataSnap can be used to let the user specify which data should be transferred from the SIBEL application to the FluCaster or SIV application, for example. If the set of data can be specified, only some of the data would have to be transferred instead of all of the data between applications.

Regarding item 3, "To customize and configure user view options," DataSnap can be used to let the user customize the view of the online application. For example, one user may be concerned about transportation and would like only transportation related tabs and data to be shown on the online application. Another user may be concerned with urban planning and would like to specify that only population density and land-use data be shown in the online application. The important point here is that the user instead of the software developer is able to customize and configure the user view options. Instead of making separate applications for each end-user group, a user would be able to completely customize their configuration using blocks in a block-based language like DataSnap

## 6.3 Expand Support for Domain Experts

As mentioned in subsection 3.7.1 "we envision a system where even more big data processing operations are defined, more powerful compute servers or more powerful computation mechanisms are used (Hadoop scripts, etc. ), and more data formats are supported (csv, json, xml, doc, xls, etc.)". DataSnap could be made even more supportive of domain experts if more big data processing methods could be developed. Text based processing could be especially helpful for those that work on text analysis or social media data analysis.

Direct integration with Google Sheets for importing and editing data may also be helpful for some users. In contrast, the current system supports the import of CSV files that happen to be hosted on Google Drive, but editing CSV files on Google Drive is not as convenient as

editing the native Drive format. In addition, although domain experts in a specific field will develop their own block palettes, block palettes for specific domains other than epidemiology could be designed in order to get this process started. Lastly, support for all HTTP methods (GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH) can be developed for DataSnap to open up a wide range of possibilities. The current implementation of DataSnap only utilizes the HTTP GET request.

## 6.4 Introduce DataSnap in Educational Settings

Next, as our infrastructure for real-time and big data problem solving in a block based language is now in place, we can now introduce it in educational settings in order to study questions such as "What type of data sets (social media, business-related, geographic, etc.) do students indicate are most relevant to their lives?", "What visualization techniques are most wanted / used by students and domain experts?", and "What additional big data processing methods are most wanted by students and domain experts?". These questions can be answered by collecting data from courses offered in K-12 and college level settings.

## 6.5 DataSnap in Open edX: Test Cases, Programming Patterns, and Course Hosting

Future work in the research direction of hosting DataSnap in Open edX includes the possibility of providing teachers with the ability to write their own test cases used to grade the sample DataSnap (or other block-based programming language) problems; Teachers can be given the ability to write their own unit tests inside of a block-based language, similar in format to the Java JUNIT testing framework. Block-based unit testing may present a new set of difficulties which can be compared to traditional unit testing frameworks.

Another useful research area would be to analyze and categorize student programming patterns. The collected student program xml and interaction tracking data will be very helpful in aiding this scenario. The program xml shows the entire student program structure which will aid in performing a static program analysis, and the interaction data will indicate what buttons students click on, how many times they run their program, which blocks they use, how many interactions typically happen between each time the program is run, etc.

Furthermore, more work needs to be done in order to support hosting a full course with DataSnap hosted on Open edX. Our implementation of the DataSnap xblock is hosted on the xblock-sdk, but the xblock would need to be hosted on an Open edX fullstack. The xblock would need to be tested further to ensure quality when a large number of students access the program, and a course modules would need to be defined. Our implementation has presented sample programming assignments to aid research / development in this direction.

# Chapter 7

# Conclusions

This thesis presented DataSnap, a block-based programming language extended from Snap!. In this thesis, we have presented our work in enhancing the state of the art in block-based programming languages for our two target audience: domain experts and introductory programmers. We have researched how to improve three specific areas for domain experts: importing data sources, performing big data processing operations, and visualizing results. Through a case study, we have demonstrated how DataSnap can enable domain experts with little programming skills to analyze data and visualize results within a web browser in a "drag and drop" style.

For the introductory programmer audience, we have researched how to provide convenient and easy-to-use data big data access blocks for teaching big data problem solving. We have also provided the necessary tools for teachers to develop their own real-time and big-data access blocks for use in their courses.

Through our presentation of DataSnap in Open edX, we have provided a framework to host DataSnap problems in an Open edX xblock for later use in Open edX courses, provided a way for teachers to define their own customized problems, and introduced a dynamic analysis grading system for problems which feature real-time data. Our work in this area also includes collecting interaction tracking data and program structure data, which can be used in further studies on student programming patterns.

# Bibliography


[1] H. Agrawal, R. Jain, P. Kumar, and P. Yammiyavar. Fabcode: visual programming environment for digital fabrication. In *Proceedings of the 2014 conference on Interaction design and children*, pages 353–356. ACM, 2014.

[2] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and B. Tribelhorn. A data programming cs1 course. 2015.

[3] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and S. A. Wolfman. Introductory programming meets the real world: using real problems and data in cs1. In *SIGCSE*, pages 465–466, 2014.

[4] W. Aspray and A. Bernat. Recruitment and retention of underrepresented minority graduate students in computer science. In *Report on a Workshop by the Coalition to Diversity Computing*, 2000.

[5] M. Ball, L. Mock, J. McKinsey, Z. Machardy, D. Garcia, N. Titterton, and B. Harvey. Oh, snap! enabling and encouraging success in cs1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 691–691. ACM, 2015.

[6] T. Ball, S. Burckhardt, J. de Halleux, M. Moskal, and N. Tillmann. Beyond open source: The touchdevelop cloud-based integrated development environment.

[7] L. J. Barker, C. McDowell, and K. Kalahar. Exploring factors that influence computer science introductory course students to persist in the major. In *ACM SIGCSE Bulletin*, volume 41, pages 153–157. ACM, 2009.

[8] A. C. Bart. Situating computational thinking with big data: Pedagogy and technology. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 719–719. ACM, 2015.

[9] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Transforming introductory computer science projects via real-time web data. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 289–294. ACM, 2014.

[10] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[11] S. Bensalem, M. Bozga, J. Quilbeuf, and J. Sifakis. Optimized distributed implementation of multiparty interactions with restriction. *Science of Computer Programming*, 98:293–316, 2015.

[12] M. Biggers, A. Brauer, and T. Yilmaz. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. In *ACM SIGCSE Bulletin*, volume 40, pages 402–406. ACM, 2008.

[13] M. Bostock. D3. js. *Data Driven Documents*, 2012.

[14] M. M. Burnett and B. A. Myers. Future of end-user software engineering: beyond the silos. In *Proceedings of the on Future of Software Engineering*, pages 201–211. ACM, 2014.

[15] J. C. Carver, L. Henderson, L. He, J. Hodges, and D. Reese. Increased retention of early computer science and software engineering students using pair programming. In *Software Engineering Education & Training, 2007. CSEET'07. 20th Conference on*, pages 115–122. IEEE, 2007.

[16] K. S.-P. Chang and B. A. Myers. A spreadsheet model for handling streaming data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2015.

[17] H. Chen, R. H. Chiang, and V. C. Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188, 2012.

[18] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492, 2009.

[19] J. M. Cohoon. Toward improving female retention in the computer science major. *Communications of the ACM*, 44(5):108–114, 2001.

[20] S. Cook, C. Conrad, A. L. Fowlkes, and M. H. Mohebbi. Assessing google flu trends performance in the united states during the 2009 influenza virus a (h1n1) pandemic. *PloS one*, 6(8):e23610, 2011.

[21] C. D. Corley, D. J. Cook, A. R. Mikler, and K. P. Singh. Using web and social media for influenza surveillance. In *Advances in Computational Biology*, pages 559–564. Springer, 2010.

[22] A. Culotta. Towards detecting influenza epidemics by analyzing twitter messages. In *Proceedings of the first workshop on social media analytics*, pages 115–122. ACM, 2010.

[23] P. Dave. Big data - what is big data - 3 vs of big data - volume, velocity, and variety. `http://blog.sqlauthority.com`.

[24] J. Dittrich and J.-A. Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proceedings of the VLDB Endowment*, 5(12):2014–2015, 2012.

[25] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li. A novel approach to improving the efficiency of storing and accessing small files on hadoop: a case study by powerpoint files. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 65–72. IEEE, 2010.

[26] M. Dorling and D. White. Scratch: A way to logo and python.

[27] edX.org. Xblocks. `http://xblock.readthedocs.org/en/latest/guide/xblock.html`.

[28] M. J. Fischer, X. Su, and Y. Yin. Assigning tasks for efficiency in hadoop. In *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, pages 30–39. ACM, 2010.

[29] A. Fisher, J. Margolis, and F. Miller. Undergraduate women in computer science: experience, motivation and culture. In *ACM SIGCSE Bulletin*, volume 29, pages 106–110. ACM, 1997.

[30] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics. *interactions*, 19(3):50–59, 2012.

[31] P. S. Foundation. Selenium 2.45.0. `https://pypi.python.org/pypi/selenium`.

[32] N. Fraser et al. Blockly: A visual programming editor, 2013.

[33] J. Gorman, S. Gsell, and C. Mayfield. Learning relational algebra by snapping blocks. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 73–78. ACM, 2014.

[34] J. Gray, H. Abelson, D. Wolber, and M. Friend. Teaching cs principles with app inventor. In *Proceedings of the 50th Annual Southeast Regional Conference*, pages 405–406. ACM, 2012.

[35] M. Guzdial and A. E. Tew. Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *Proceedings of the second international workshop on Computing education research*, pages 51–58. ACM, 2006.

[36] D. Harel and G. Katz. Scaling-up behavioral programming: Steps from basic principles to application architectures. In *Proceedings of the 4th International Workshop on Programming based on Actors Agents & Decentralized Control*, pages 95–108. ACM, 2014.

[37] B. Harvey, D. Garcia, J. Paley, and L. Segars. Snap!:(build your own blocks). In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 662–662. ACM, 2012.

[38] B. Harvey and J. Mönig. Bringing no ceiling to scratch: can one language serve kids and computer scientists. *Proc. Constructionism*, 2010.

[39] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272, 2011.

[40] A. Hidayat. Phantomjs. *Computer software. PhantomJS. Vers*, 1(7), 2013.

[41] J. Highcharts. Interactive javascript charts library.

[42] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[43] R. Ihaka and R. Gentleman. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314, 1996.

[44] B. D. Jones. Motivating students to engage in learning: The music model of academic motivation. *International Journal of Teaching and Learning in Higher Education*, 21(2):272–285, 2009.

[45] B. D. Jones and G. Skaggs. Validation of the music model of academic motivation inventory: A measure of students motivation in college courses. In *International Conference on Motivation, Frankfurt, Germany*, 2012.

[46] S. H. Kim and J. W. Jeon. Programming lego mindstorms nxt with visual programming. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pages 2468–2472. IEEE, 2007.

[47] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, et al. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, 43(3):21, 2011.

[48] D. Kumar. Digital playgrounds for early computing education. *ACM Inroads*, 5(1):20–21, 2014.

[49] A. Labrinidis and H. Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[50] S. Leo, F. Santoni, and G. Zanetti. Biodoop: bioinformatics on hadoop. In *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*, pages 415–422. IEEE, 2009.

[51] S. Leo and G. Zanetti. Pydoop: a python mapreduce and hdfs api for hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 819–825. ACM, 2010.

[52] D. Lizcano, F. Alonso, J. Soriano, and G. Lopez. A web-centred approach to end-user software engineering. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(4):36, 2013.

[53] D. J. Malan and H. H. Leitner. Scratch for budding computer scientists. In *ACM SIGCSE Bulletin*, volume 39, pages 223–227. ACM, 2007.

[54] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[55] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1):367–371, 2008.

[56] A. Marron, G. Weiss, and G. Wiener. A decentralized approach for programming interactive applications with javascript and blockly. In *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*, pages 59–70. ACM, 2012.

[57] V. Mayer-Schönberger and K. Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[58] J. C. McCarthy, V. C. Miles, and A. F. Monk. An experimental study of common ground in text-based communication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 209–215. ACM, 1991.

[59] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8):90–95, 2006.

[60] W. McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[61] W. McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.

[62] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari. Learning computer science concepts with scratch. *Computer Science Education*, 23(3):239–264, 2013.

[63] A. Monk. Common ground in electronically mediated communication: Clarks theory of language use. *HCI models, theories, and frameworks: Toward a multidisciplinary science*, pages 265–289, 2003.

[64] S. Nadhani and P. Nadhani. *FusionCharts Beginner's Guide: The Official Guide for FusionCharts Suite*. Packt Publishing Ltd, 2012.

[65] J. Peckham, P. Stephenson, J.-Y. Hervé, R. Hutt, and M. Encarnação. Increasing student retention in computer science through research programs for undergraduates. *ACM SIGCSE Bulletin*, 39(1):124–128, 2007.

[66] A. Repenning, D. Webb, and A. Ioannidou. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 265–269. ACM, 2010.

[67] P. Russom et al. Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, 2011.

[68] A. J. Saldanha. Java treeviewextensible visualization of microarray data. *Bioinformatics*, 20(17):3246–3248, 2004.

[69] J. Sallai, G. Varga, S. Toth, C. Iacovella, C. Klein, C. McCabe, A. Ledeczi, and P. T. Cummings. Web-and cloud-based software infrastructure for materials design. *Procedia Computer Science*, 29:2034–2044, 2014.

[70] E. Seymour, N. M. Hewitt, and C. M. Friend. *Talking about leaving: Why undergraduates leave the sciences*, volume 12. Westview Press Boulder, CO, 1997.

[71] Y. N. Silva and J. Chon. Dbsnap: Learning database queries by snapping blocks.

[72] Y. N. Silva and J. Chon. Querying databases by snapping blocks.

[73] R. Soderbery. How many things are currently connected to the "internet of things" (iot)?

[74] T.-H. Tan, M.-S. Lin, Y.-L. Chu, and T.-Y. Liu. Educational affordances of a ubiquitous learning environment in a natural science course. *Educational Technology & Society*, 15(2):206–219, 2012.

[75] V. Tech. Network dynamics and simulation science laboratory. `http://www.vbi.vt.edu/ndssl`.

[76] E. Tilevich, C. A. Shaffer, and A. C. Bart. Creating stimulating, relevant, and manageable introductory computer science projects that utilize real-time web-based data. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 743–743. ACM, 2014.

[77] E. Tilevich, C. A. Shaffer, and A. C. Bart. Creating stimulating, relevant, and manageable introductory computer science projects that utilize real-time, large, web-based datasets. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 711–711. ACM, 2015.

[78] J. Trower and J. Gray. Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 5–5. ACM, 2015.

[79] S. University. About openedx. `http://online.stanford.edu/openedx`.

[80] G. Varga, C. R. Iacovella, J. Sallai, C. McCabe, A. Ledeczi, and P. T. Cummings. Enabling cross-domain collaboration in molecular dynamics workflows. In *COLLA 2014, The Fourth International Conference on Advanced Collaborative Networks, Systems and Applications*, pages 41–47, 2014.

[81] G. Varga, S. Toth, C. R. Iacovella, J. Sallai, P. Völgyesi, A. Ledeczi, G. Karsai, and P. T. Cummings. Web-based metaprogrammable frontend for molecular dynamics simulations. In *SIMULTECH*, pages 171–178, 2013.

[82] M. Vaziri, O. Tardieu, R. Rabbah, P. Suter, and M. Hirzel. Stream processing with a spreadsheet. In *ECOOP 2014–Object-Oriented Programming*, pages 360–384. Springer, 2014.

[83] T. White. *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[84] U. Wolz, H. H. Leitner, D. J. Malan, and J. Maloney. Starting with scratch in cs 1. In *ACM SIGCSE Bulletin*, volume 41, pages 2–3. ACM, 2009.

# Appendix A

# Epidemiology Block Definitions


Figure A.1: The "Get all data for the peak week of the flu season for year (year)" block definition.

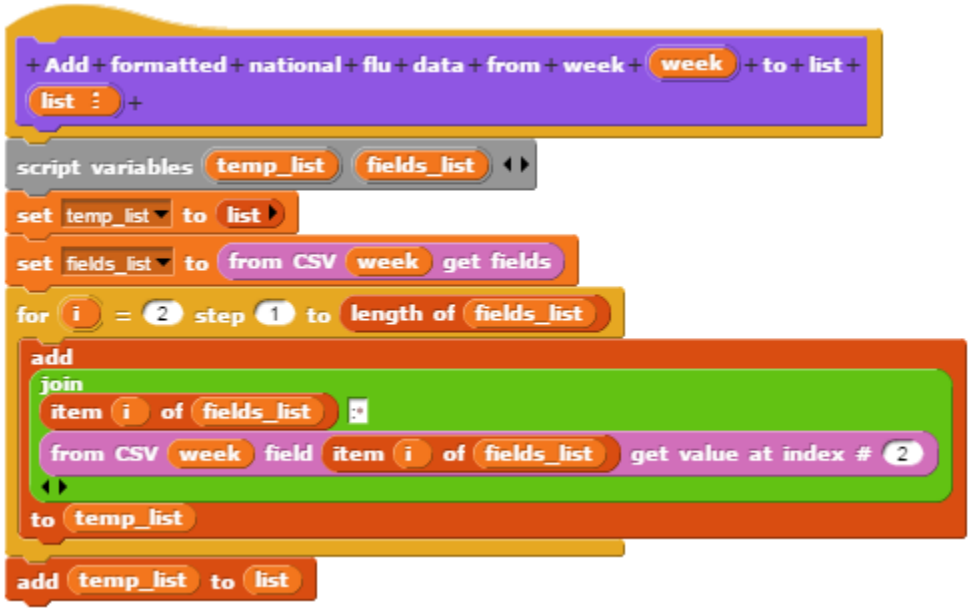



Figure A.2: The "Add formatted national flu data from week (week) to list (list)" block definition.


Figure A.3: The "Add formatted regional flu data from week (week) to list (list)" block definition.


Figure A.4: The "Add the peak week of each flu season from year (year_start) to (year_end) to list (list)" block definition.

# Appendix B

# Data Set Block Definitions

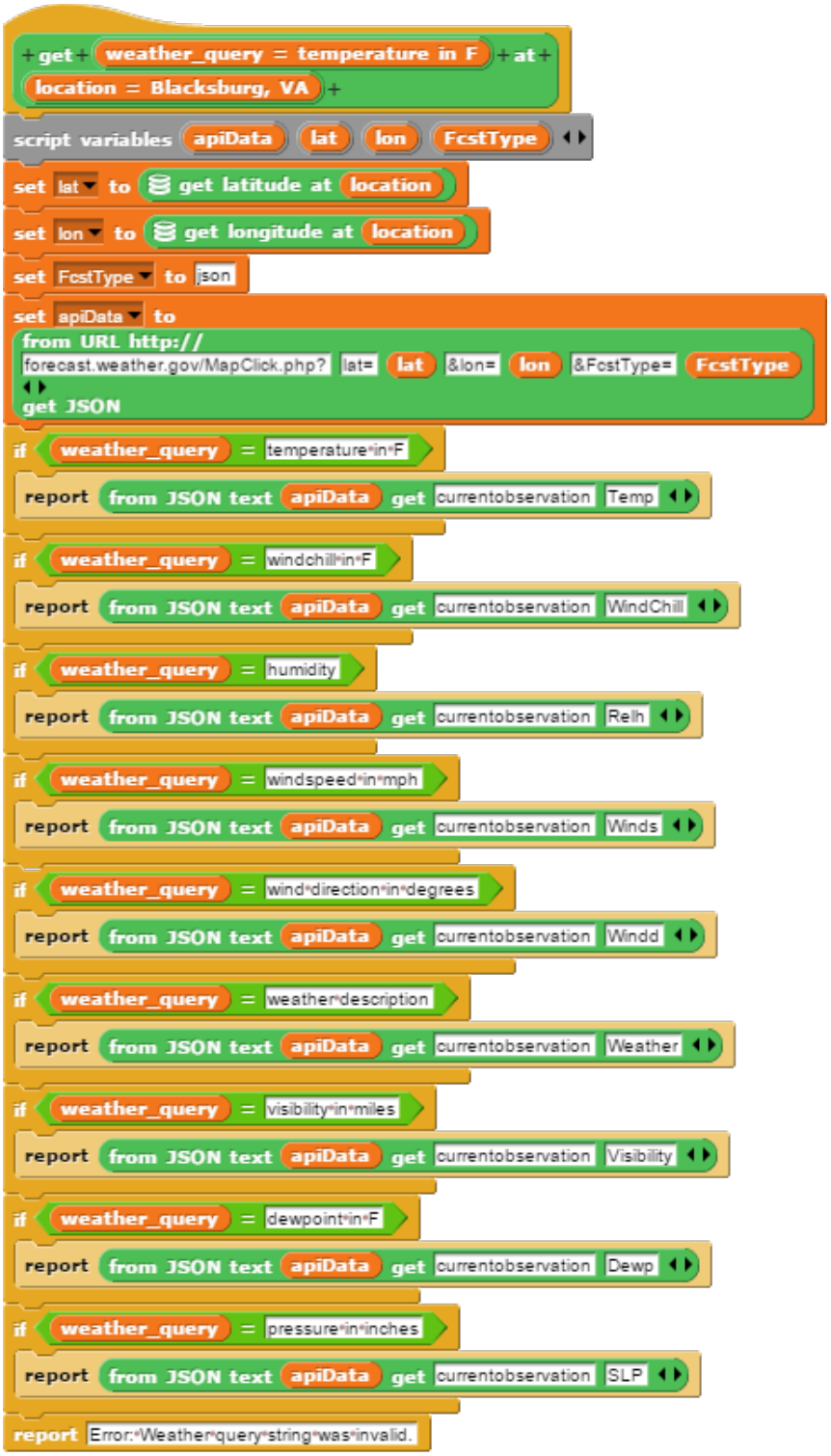



Figure B.1: The "get (temperature in Fahrenheit) at (location)" block definition.

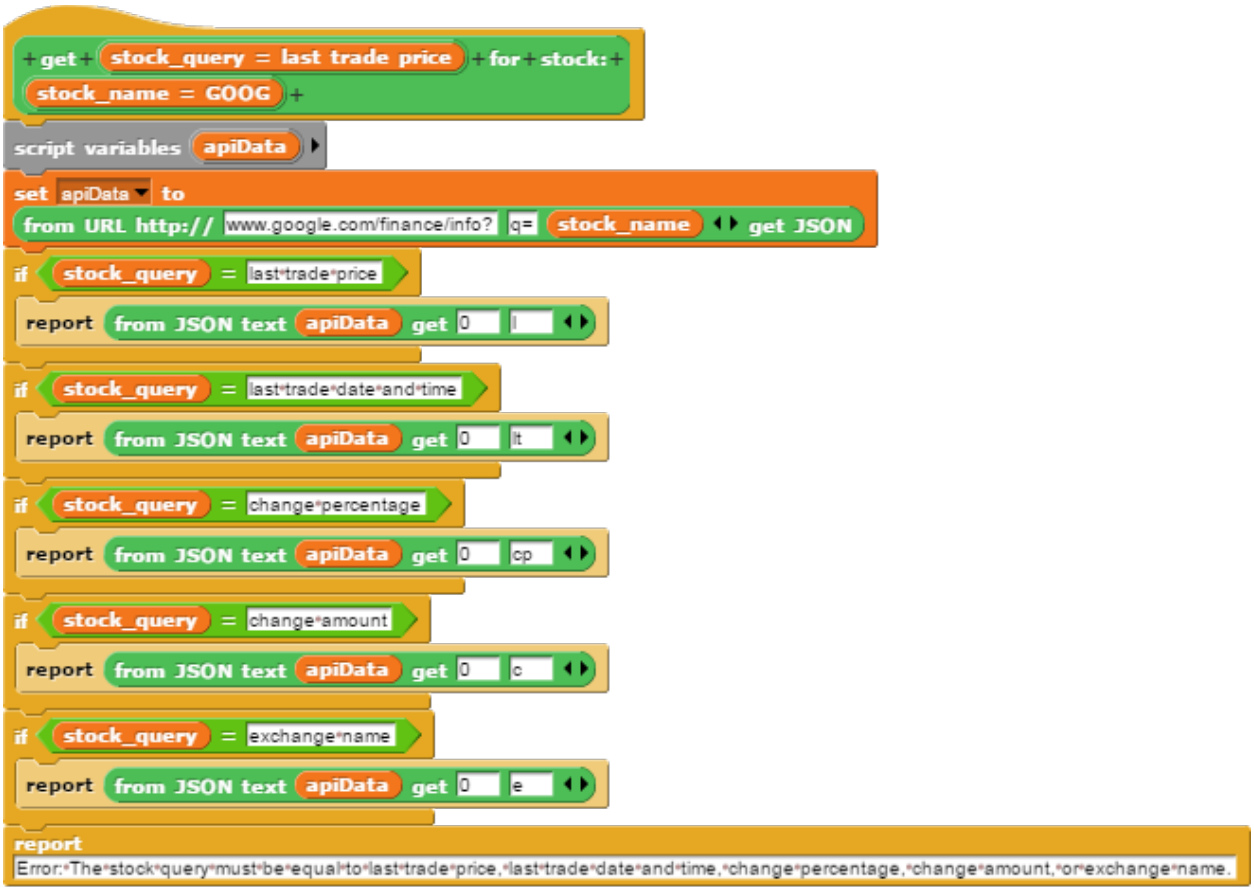



Figure B.2: The "get (last trade price) for stock: (stock)" block definition.

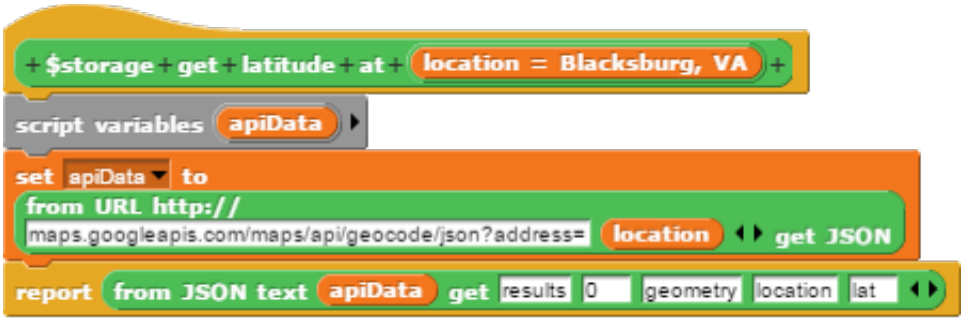



Figure B.3: The "get latitude at (location)" block definition.

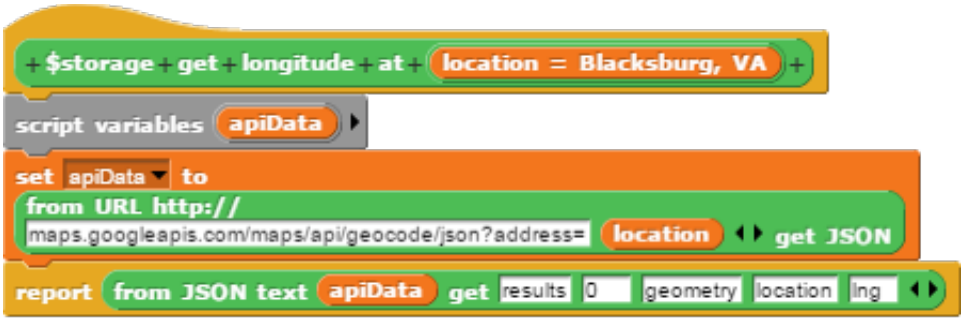



Figure B.4: The "get longitude at (location)" block definition.

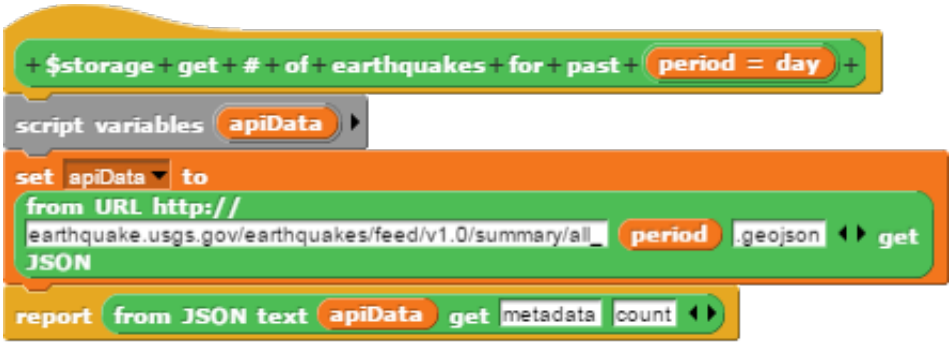



Figure B.5: The "get number of earthquakes for past (period)" block definition.

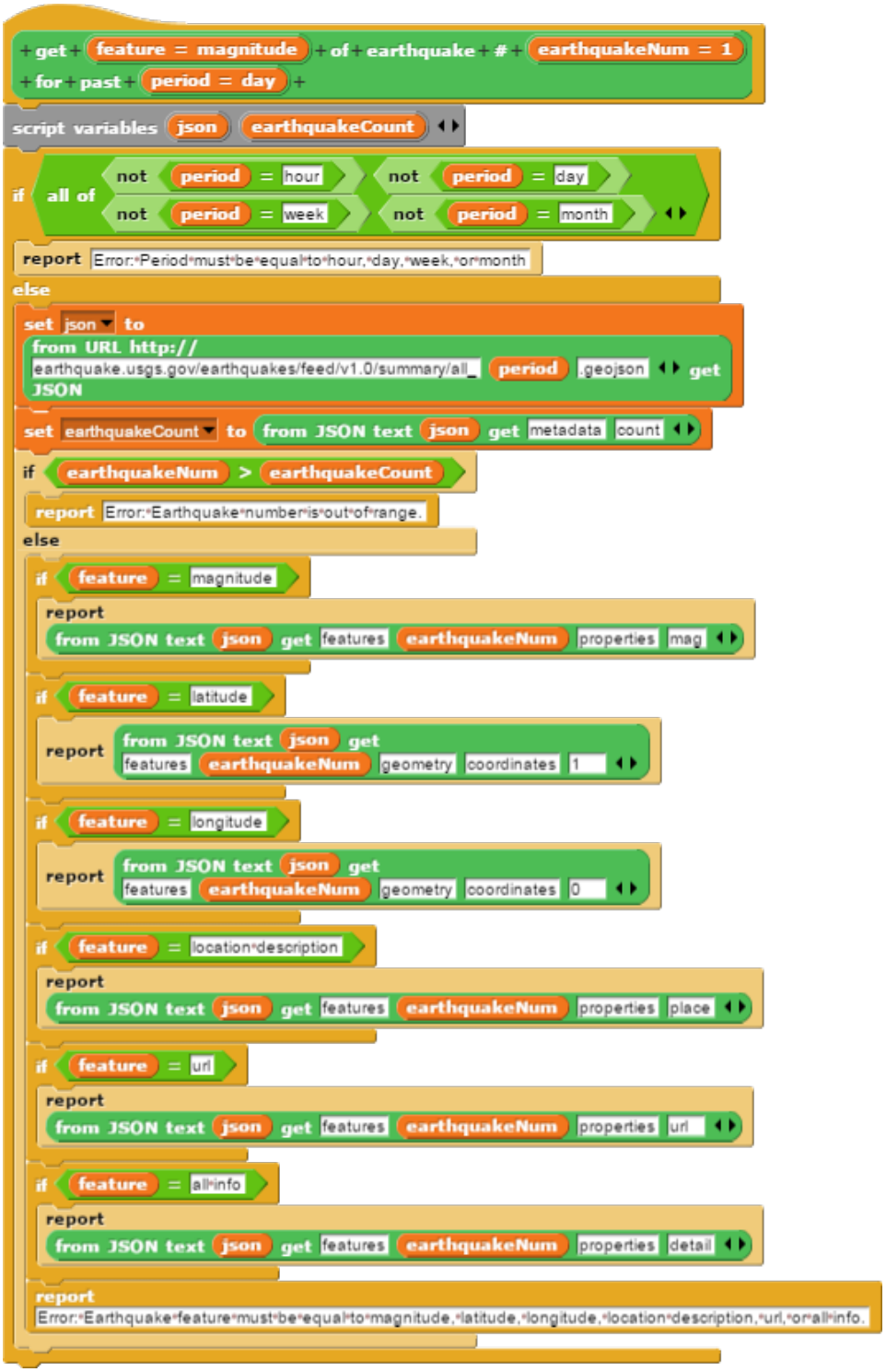



Figure B.6: The "get (location description) of earthquake number (number) for past (period)" block definition.