TRABAJO FIN DE MÁSTER

CURRICULUM DESIGN: COMPUTATIONAL ALGEBRA I

Masters in Bilingual and Multicultural Education

Gabriel Antonio Terrasa Jr

Advisor: Marcos Garasa Mayayo

Academic Year 2022-2023

Table of Contents

Abstract

Computational Algebra I teaches Algebra I while providing students with a foundation in computer programming. As students learn about the typical Algebra I mathematical concepts, they will apply their knowledge by solving real-world problems with programming. The curriculum is designed to align with the state of Maryland Algebra I standards and closely follows the Illustrative Mathematics curriculum. As such, the course is targeted towards mixed-grade or single-grade classes with students years 7th and 9th grade for an entire academic term of 180 school days. The curriculum design uniquely uses programming as one of the primary methods for solving problems related to math concepts, which has become an essential and ubiquitous tool in modern science technology engineering and math (STEM) fields, including biology, astronomy, and geology. The course uses the Python language as it is the most common programming language used in scientific programming. By using programming in the Algebra I course, students will develop essential skills in applied math, programming, critical thinking, and problem solving that they can use throughout their high school math and professional careers.

Keywords

Scientific Computing, Scientific Programming, Python, Project-Based Learning, Applied Math, Problem Solving

1 Introduction

1.1 Context and justification

To begin, the curriculum examines the importance of computer programming as an application of mathematics in student careers, and proposes a way to include programming into the mathematics curriculum, specifically in Algebra I.

What is the goal of secondary math education? One perspective, from the Common Core standards (Maryland State Department of Education, 2012) is to "specify the mathematics that all students should study in order to be college and career ready," implying career readiness as the final goal. In Baltimore County Public Schools (Secondary Mathematics, n.d.), the stated goals are to solidify "students' readiness for post secondary mathematics courses and opportunities that rely on a strong understanding and application of mathematical concepts and skills," which again focuses on the idea of "readiness." Yet not much explanation is given for what students being ready for their careers means. The author focuses on the word "application" mentioned in the Baltimore County Public Schools definition. It is the author's belief that students having the ability and confidence to apply their knowledge at the next step in their careers is the highest level of "readiness" possible, and it is therefore the responsibility of mathematics educators to teach materials and use methods that support the capacity for application in their students.

One way mathematics is applied in careers is through programming (edX team, 2021). In fact, the use of programming in STEM and other fields is growing to a point of ubiquity ("7 Reasons Why Kids Should Learn to Code," 2020), and it is increasingly being seen as an essential digital literacy skill (21K School, 2022). The New York Times writer Steve Lohr gave the example of how even some lawyers have used programming to improve their craft, writing programs to form word clusters in Supreme Court rulings. Lohr goes on to state "it's the same in every field, from marketing to manufacturing to medicine. Code, it seems, is the lingua franca of the modern economy" (Lohr, 2017). Furthermore, in 2016, 20% of all jobs in the United States which paid at least a living wage required coding ("Beyond Point and Click," 2016), and the US Bureau of Statistics estimates a growth of 25% in jobs in software development, quality assurance, and testing by 2031 (U.S. Bureau Of Labor Statistics, 2023).

While computer programs might not all require that a student be able to factor a polynomial or calculate the area of a circle, they typically require an understanding of the underlying arithmetic and statistics (Brooks, 2020; Sweigart, 2012), and they all require critical thinking and abstraction skills (Pappano, 2017) which go hand in hand with mathematics education (Sukma & Priatna, 2021). The author argues that the ability

to apply mathematics through programming is one of the most important mathematics skills in the 21st century, and therefore an integral part of students being "college and career ready."

However, mathematics and computer science being taught together is rare. Most examples come from specific teachers running their own blogs or websites (Hobenshield Tepylo & Floyd, 2016; PLOS Blogs, 2016), and there is little being done on a school or school-system level to incorporate programming into existing mathematics courses. This curriculum design will fill this gap, being the first of its kind to propose the integration of computer programming as a central part of the way mathematics is taught on the scale of a full academic year.

1.2 Objectives

The principal objective of this curriculum design, called *Computational Algebra I*, is to provide a model for how secondary mathematics can better prepare students for their careers by incorporating computer programming, a necessary digital literacy skill, into coursework while continuing to comply with stipulated state curriculums, specifically in the state of Maryland. It does so by providing an implementation of the Maryland Algebra I curriculum in which programming is taught alongside the mathematical content as well as used to reinforce it. It is the first of its kind to propose the integration of programming as a central part of the way mathematics is taught.

Computational Algebra I fills the student requirement in the state of Maryland for Algebra I while building a foundation in applying the math that students learn through programming. Students will also deepen their understanding of mathematics through their programming assignments. Students entering the course need no background in programming. The language to be used is Python, the most common language used in data science (Canales Luna, 2023), and is introduced assuming no familiarity with programming or the language. Programming will be done in Python notebooks and it is recommended they be used through the Google CoLab environment. The curriculum design includes materials for an entire year course.

The design aligns with the state of Maryland Algebra I standards (Maryland Department of Education, 2020). Specifically, it is based on the units layed out in the Illustrative Mathematics curriculum (Illustrative Mathematics, 2019) which is currently used in the Baltimore County, Howard County, and Montgomery County school districts (*Algebra I Year at a Glance Baltimore County*, 2022; Howard County Public Schools, n.d.; Montgomery County Public Schools, n.d.). That said, Maryland is a Common Core state, and this curriculum could also be applied in other states using the Common Core math standards. The curriculum is designed for use in classrooms in the United States for mixed-grade or single-grade classes with students years 7th and 9th grade for an entire academic term of 180 school days (Maryland State Department of Education, 2023b).

2 Rationale

2.1 General aspects

Several programs and standards for mathematics state their goals as being student "readiness" for their careers and their future math coursework (*Common Core State Standards for Mathematics*, n.d.; *Secondary Mathematics*, n.d.). When thinking of "readiness," the inevitable question arises "what does it mean to be ready?" Often, readiness in mathematics is talked about as a student's ability to apply the math they've learned to a career in STEM (Hewer, 2022; Shumow, 2023). But what specifically does it look like to apply the math they are learning in a STEM career? The answer is often through programming, specifically scientific computing.

Programming has become ubiquitous and essential in the world of STEM. For example, top universities like MIT and Carnegie Mellon have started degree programs specifically for computational biology (*Computational Biology*, 2019; Murphy, 2023). This field is considered a "paradigm shift" in modern biology (Murphy, 2023) of growing importance in which programming and data science are applied to biology for research in fields like DNA and RNA sequencing and analysis and evolutionary models (Fan et al., 2010; Noble, 2002). But biology is not the only STEM field in which programming and mathematics are increasingly used together. Programming for analyzing data is considered essential in astronomy (Zingale et al., 2016), and similar claims have been made for all sorts of sciences from geology to zoology (Faller et al., 2021). The general use of programming in science is called "scientific programming" and as noted by the National Institute of Standards and Technology, "nearly every scientific study today involves computation of some sort, " specific uses including automating data collection tasks, building and executing statistical models, and analyzing data (Faller et al., 2021; National Institute of Standards and Technology, 2016).

While there has been considerable research into project-based and problem-based approaches to math learning (Craig & Marshall, 2019; Lee, 2022; Savery, 2006), there are no published curricula which use computer programming, a necessary tool in modern STEM fields, as a primary method for solving problems or carrying out projects related to the math concepts. This is the gap to be filled by this curriculum design. If math courses are to prepare students for their future careers, they must make central to their curricula the use which has been synonymous with the application of math in STEM: programming.

Algebra I was chosen for this curriculum because it is the first of the required mathematics tests administered to high school students in Maryland (Maryland State Department of Education, 2023a) and is the first course in the high school mathematics pathways for the counties which use the Illustrative Mathematics Algebra I curriculum (*Algebra I Year at a Glance Baltimore County*, 2022; Howard County Public Schools, n.d.; Illustrative Mathematics, 2019). The programming skills that students pick up could then be used for the rest of their high school math careers and beyond. Further, in Algebra I students are introduced to several topics that

are compatible with learning programming. For example, as students are learning about analyzing data and statistics in Unit 1, they can learn how to do arithmetic and call functions to aggregate the data sets. Then in Unit 4, as students are learning about mathematical functions, they can at the same time learn about creating functions in programming and see how the two concepts relate to and are different from each other.

2.2 Analysis of Existing Materials

The main contribution of this course is the addition of programming into the mathematics curriculum at the scale of a whole year. Being the first of its kind, there are no models of attempts to do the same thing for an entire year. However, the author looks towards existing materials used in different contexts for inspiration. Materials reviewed will include existing materials for teaching programming and materials at the intersection of programming and mathematics.

2.2.1 Existing Programming Materials

Educational materials for programming are tremendously broad and easily accessible. Online materials include game-based learning like CodeCombat to more course-based platforms such as Code Academy and freeCodeCamp. These are excellent resources for setting the content and order of a course on Python, but are not designed to be used as classroom courses.

One of the most common classroom implementations of a programming course is AP Computer Science whose goals are to create proficient programmers by the end of the course (College Board, n.d.). The units in the AP Computer Science course follow a logical progression for building up the complexity of the programs students can work with. The curriculum has ten units, starting with primitive types, moving to objects, boolean expressions, iteration, classes, arrays, and finally covering recursion. The focus on data types and objects at the beginning is fundamental for students to understand how pieces of their code interact with others. However, much of these technical details, specifically around topics like inheritance and recursion are not necessary for writing the types of programs that would be useful in math class, and notably lack key skills like making plots and graphs. The class is also based on Java, often used for building applications and not as common in STEM as Python (Canales Luna, 2023).

To teach the specific programming needs for science, a plethora of online courses exist for Python. Each of these courses will have their own order and scope of materials. For example, the freeCodeCamp course “Scientific Computing with Python”(freeCodeCamp, n.d.), starts with variables and expressions, followed by logic, the functions, loops, networking and web services, and visualizations. Notably missing from this course is how to plot data, using programing like matplotlib, which would be essential in the Algebra I classroom where students are learning to visualize functions.

2.2.2 Existing Materials at the Intersection of Programming and Mathematics

In approaching teaching programming alongside mathematics, one of the essential questions is how to balance the programming skills with the math skills. Reina Ishibashi at Carroll College describes several ways to approach this balance. Among these are taking measures to ensure active engagement from all students through turn taking turns and coding quizzes, emphasizing the connection between the coding assignments and math, and finally the use of teacher-written code (Carroll College Library & Learning Commons, 2021). Ishibashi notes that students need to have enough programming knowledge to understand any teacher-provided code snippets and to complete the projects, but they do not necessarily need to have the amount of expertise to be able to write all that code on their own. The approach of providing code snippets is heavily utilized in the *Computational Algebra I* course so that students can perform more complex tasks that enhance their understanding of the mathematics material even if they have not covered in depth the required structures to be able to write the entire program themselves.

Another approach to the balance between math and programming is presented in the course "Coding in Math" from CodeHS (*Coding in Math Syllabus*, n.d.). The curriculum describes units with both math and computer science learning outcomes. For example, their first unit focuses on the math concepts, factors, and greatest, common factor, and they take advantage to cover the programming concepts of printing, modulus, variables, if statements, and for loops. This way, the progression of the students' programming and math skills grow concurrently and symbiotically. That is why in this curriculum design, each unit has both math and programming learning outcomes as opposed to teaching the math and programming as separate units. The principal fallback of the CodeHS course is that it is not meant for an entire year in-class course, and is rather supplementary material for inside or outside the classroom.

There are other approaches in which mathematics has been incorporated into the computer science curriculum inside the classroom. For example, Diane Hobenshield Tepylo & Lisa Floyd have written an excellent example of the way they used modulus division in programming to reinforce high school student understanding of remainders (Hobenshield Tepylo & Floyd, 2016). Tepylo and Floyd start with a simple problem for students: remainders, and through a series of programming steps, develop its meaning. This curriculum draws inspiration from Tepylo and Floyd, starting almost every programming lesson with an exercise which demonstrates the need for the programming skills to be practiced. For example, Unit 1 Lesson 7 starts with a small data set for which students must calculate the mean by hand and later poses the question of what would happen with a data set of far more numbers. Furthermore, students are almost always asked to interpret the results of the calculations they perform, meant to prompt students to reflect on the meaning of a calculation like a mean.

There is also a set of work in programming activities that can be done within traditional math class, often with Scratch (Larson & Goetz, n.d.; Vaidyanathan, n.d.). Scratch, a block-based programming language, is

an excellent tool for teaching students to code; however, it does not have the same flexibility or capability for scientific computing as Python does.

Finally of particular interest would be the book *Introduction to Scientific Programming with Python* by Joakim Sundnes which introduces the types of skills students would need to apply programming to their math classes, but without covering the math (Sundnes, 2020). It is from this book that the ordering of the programming was taken. The chapters of the book follow: Computing with Formulas (Chapter 2), Loops and Lists (Chapter 3), Functions and Branching (Chapter 4), arrays and plotting (Chapter 6), with some small changes. For example Chapter 5 about user input and error handling has been deprioritized as it will be included in less detail throughout the labs. This ordering is both intuitive, as evidenced by Sundnes's approach, and allows the students to gain the programming knowledge that they need to approach the math problems presented to them. For example, students reach programming functions and mathematical functions at the same time so that they are able to program and use the mathematical functions as they learn to manipulate them.

3 Background

3.1 Definition of Important Concepts

3.1.1 General Concepts

Coding and Programming: In popular culture, the difference between programming and coding has become clouded in recent years. Even technology teachers' understandings are inconsistent (Corradini et al., 2018). However, within computer science, the two are considered different terms. In a conference paper from 2016, professor Bell concisely describes the difference:

> *Coding in popular culture has come to mean what is more accurately called programming, and, more generally, software development. The term 'coding' has traditionally referred to only a small part of the whole process of software development. Creating new software involves several aspects, including:*
>
> - *analysis: identifying the needs for which a solution will be developed*
> - *design: sketching how the solution will work*
> - *coding: converting the proposed solution to a computer language*
> - *testing: checking that the solution works as intended, including being reliable and usable*
> - *debugging: tracking down why parts of it don't work as intended.*
>
> (Bell, 2016)

The objectives of the proposed course are to go beyond coding and teach students to analyze, design, test, and debug as they work through the coursework. For this reason, the term *programming* is adopted almost exclusively throughout this curriculum design unless specifically referring to the creation of code.

Scientific Computing and Scientific Programming: The National Institute for Standards and Technology says "nearly every scientific study today involves computation of some sort; this broad class of computer applications is known as scientific computing"(National Institute of Standards and Technology, 2016). Another related term is scientific programming which Faller et. al. defined as "any time a computer program is used for science research"(Faller et al., 2021). The author will differentiate the two definitions, using scientific computing to refer to the general field of the use of computer programming in science and using scientific programming to refer to specifically the act of developing and writing programs for use in science. While scientific computing is not specifically the focus of the course, by teaching mathematics and programming together, many of the problems used naturally fall into scientific programming. For example in Lab 1 when students analyze temperature data, that would be considered scientific programming. However, in Unit 2 Lesson 11, when students analyze sports statistics, that might not necessarily be considered scientific programming. However, students are using many overlapping skills such as statistical analysis, iteration, and logical operators.

Programming Language: Britannica Encyclopedia defines a programming language as a language for expressing a set of detailed instructions for a digital computer (Hemmendinger, 2022). Britanica, along with several other online programming educational resources also distinguish high-level and low-level programming languages. Low-level languages require a deep understanding of the internal mechanisms of the computer and its memory while high-level languages typically shield the programmers from much of that thinking, sometimes at the cost of speed (Fowler, 2020; Hemmendinger, 2022; *High and Low Level Languages*, n.d.). Python is intended to be a high-level language, providing features for fast script development (Python Software Foundation, 2023a). The fact that it is a high-level language and its corresponding ease of use are a part of the reason it has been chosen for this course.

Data Literacy and Data Visualization Literacy: Wolff et. al. define data literacy as "the ability to understand and use data effectively to inform decisions" (Wolff et al., 2016). Wolff et. al. also importantly discuss in a later paper the gap between traditional data explored in the classroom and the data students are now more likely to encounter in the world. Classroom data sets might be in small quantities or personally collected, and the typical data used in application is typically much larger and more complex (Wolff et al., 2019). It would seem that modern data literacy crucially must include modern data. Such quantities of data can only most effectively be analyzed through programming. Further, in a literature review, Börner et. al. review definitions of a subset of data literacy, data visualization literacy. These definitions can be condensed to the ability to understand, draw meaning from, and generator data visualizations (Börner et al., 2019). Engebretsen and Kennedy discuss the increasing importance of creating and understanding data visualizations as the collection and communication of data become more common (Engebretsen & Kennedy, 2020). Both data visualization literacy and data literacy more broadly are considered desired learning outcomes for students in this curriculum.

Abstraction: Ryan Michael Kay at freeCodeCamp talks about abstraction as simply "a less detailed representation of an object or concept in nature" (Kay, 2022). Daniel Garcia from the University of California, Berkeley, spoke about abstraction in a New York Times interview as "concealing layers of information," saying that "concealing layers of information makes it possible to get at the intersections of things, improving aspects of a complicated system without understanding and grappling with each part. Abstraction allows advances without redesigning from scratch" (Pappano, 2017). The skill of abstraction is essential in both programming and mathematics. For example, analyzing a real-life situation and picking out the important details to write an equation is an act of abstraction because not all the details of the situation are necessary in order to model it and predict what will happen in the future.

3.1.2 Programming Vocabulary

The following programming vocabulary can be generally defined across programming languages. However, as Python is the language of choice for this course, the focus will be on defining these terms within the Python programming language. It should also be noted that in order to teach this course, proficiency in Python programming language is recommended. Seven high-level concepts and packages are explored below, but it would be impractical to include all of the necessary knowledge to appropriately prepare an instructor of this course.

Python Notebook and Google CoLab: A Python notebook is a tool for running code in a modular way. Google CoLab is a common online platform for editing and running Python as it does not require Python or any packages to be installed on the local computer. Further, it has support for Markdown text which allows the instructor to make *base files* which include explanations and questions in addition to places for students to write their code.

Variables: Variables are often one of the first things addressed when learning a programming language. Various online resources refer to variables as a way to reference or store a value for later use in the program (Tutorials Point, n.d.; W3 Schools, 2023b). Programming variables do not differ from mathematical variables aside from specifics of their implementations in different programming languages. However one of the difficult concepts for some students may be that variables in programming, unlike what they likely will have seen until this course, need not represent just one value, but rather can store objects that hold many values in them, like a list of values or even a function.

Iteration**:** Iteration is a process in which a piece of code is repeated, often changing a value on each iteration (KS3, n.d.). It's a topic that while it has parallels to mathematical concepts like discrete sums likely will not have been touched in any courses prior and will need to be taught to the students from scratch.

Function (Programming): W3 schools, a common online resource for learning programming, defines a Python function with three main statements: (1) "A function is a block of code which only runs when it is called." (2) "You can pass data, known as parameters, into a function." (3) "A function can return data as a result" (W3 Schools, 2023a). Other online resources, though more verbose, also hit these main points as part of the definition of a function (Gallagher, 2020; Programiz, n.d.; W3 Schools, 2023a). Notable in statements (2) and (3) is the use of the word *can*, meaning it is not necessary to have explicit inputs and outputs of your function. These being optional parts of the function is what makes them fundamentally different from a mathematical function. Gallagher also notes that functions are common in many programming languages and are not unique to the language of choice, Python (Gallagher, 2020). For the purpose of this course, "function" will refer to any "function" or "method" defined under the "callable types" part of the Python Data Model, the

main difference between a function and a method being whether or not they are part of a programming object (Python Software Foundation, 2023b), a topic that will only be covered briefly in this course .

Package (Programming): Code written by another author that is imported and used within a program will be referred to as a "package" in this curriculum design.

Matplotlib: A commonly used Python package used for making graphs.

Numpy: A commonly used Python package used for vector math or math done on a series of numbers.

3.1.3 Course Vocabulary

Programming Labs: The term to be used for the projects that students do throughout the year combining mathematical and programming understandings.

Mini-Quizzes: The term to be used for the formal assessments done in each of the units to reinforce the learning of math and programming concepts. These mini-quizzes are intended to take no more than 25 minutes, not occupying an entire class period.

Real-World Problem: In this course, a real-world problem refers to any math problem specifically designed for students to use their knowledge from the course to analyze and better understand a situation related to physical or social phenomena.

3.2 Teaching Methods

Experiential Learning Theory: Experiential learning is defined as "a powerful and proven approach to teaching and learning that is based on one incontrovertible reality: people learn best through experience" (Kolb, 2014). It has been used in the math and engineering classrooms before to much success in increasing student achievement and improving student attitudes toward the material (Félix-Herrán et al., 2019; O'Brien et al., 2021; Uyen et al., 2022). Experiential learning will be largely implemented through the use of programming labs in this course.

Project-Based Learning (PBL): Utah Valley University Office of teaching and learning defines PBL as learning in which "students have to produce an artifact to demonstrate their mastery of content" (Utah Valley University Office of Teaching and Learning, n.d.). In a study about high school math and PBL, Craig and Marshall trace the history of the PBL and develop a defitionion rooted in a desire for contextualizing education and "engaging [students] in authentic, relevant challenges" (Craig & Marshall, 2019). Lee, in comparing several elementary project-based math lessons, noted that they all focus on "a real-life context with a challenging problem, constituting a fundamental difference between a [Project-Based Mathematics] lesson and a conventional mathematics lesson"(Lee, 2022). The author implements project-based learning by making real-world problems the central context of the exercises students explore and by frequently prompting students to make and justify decisions that go beyond presenting a number that falls out of a calculation.

Conventional Learning Methods: Craig and Marshall contrast PBL with conventional learning methods which are defined as including "teacher-centered" methodology and "emphasizing drill and mastery" (Craig & Marshall, 2019). This is in other words, what is typically thought as school instruction, with the teacher lecturing while the students repeat and drill specific skills. The course will focus its methods on PBL, and some conventional learning methods will be implemented in the instruction of concepts prior to students working on their projects.

Open-Ended Prompts: In contrast to some traditional methods, many of the questions used in this curriculum will be open-ended, that is to say without one specific answer. Students will be asked to justify their beliefs with the math that they have learned. Such opinions are without right or wrong answers, and the emphasis in grading will be on their justifications.

3.3 Standards

This curriculum is implementable within the current Algebra I mathematics classrooms in the state of Maryland. So, the basic standards of this course must be identical to those stipulated by the state of Maryland, which are based off of the Common Core (Maryland State Department of Education, 2012). The addition of programming being the main contribution of this curriculum design, below are the Algebra I standards thought to be particularly enhanced by the addition of programming. For the fundamentals of programming, standards have also been taken from the AP Computer Science A curriculum. Finally, additional learning standards have been identified by the author.

Standards from the State of Maryland (Maryland State Department of Education, 2012)

A1.M.4 Interpret the solution to a real-world problem in terms of context.

A1.M.6 Solve multi-step contextual word problems with degree of difficulty appropriate to the course, requiring application of course-level knowledge and skills articulated in the standards.

A1.R.1 Given an equation, reason about the number and nature of the solutions.

A1.R.2 Given a system of equations, reason about the number of solutions.

A1.R.3 Reasoning based on the principle that the graph of an equation in two variables is the set of all its solutions plotted in the coordinate plane.

A1.R.4 Identify an option that would refute a conjecture/claim.

A1.R.5 Identify a correct method and justification given two or more chains of reasoning.

A1.R.9 Construct, autonomously, chains of reasoning that will justify or refute propositions or conjectures about functions.

A1.R.10 Express reasoning about transformations of functions.

A.APR.A.1 Add, subtract, and multiply polynomials.

A.APR.B.3 Identify zeros of polynomials when suitable factorizations are available and/or use the zeros to construct a rough graph of the function defined by the polynomial

A.REI.B.3 Solve linear equations and inequalities in one variable, including equations with coefficients represented by letters.

A.REI.B.4 Solve quadratic equations in one variable.

A.REI.C.6 Solve systems of linear equations exactly and approximately (e.g., with graphs), focusing on pairs of linear equations in two variables

A.REI.D.11 Approximate the solutions to an equation of the form $f(x) = g(x)$ using the point(s) of intersection of the graphs of the equations $y = f(x)$ and $y = g(x)$ intersect.

A.CED.A.1 Create equations and inequalities in one variable and use them to solve problems. Include equations arising from linear and quadratic functions, and simple rational and exponential functions.

A.CED.A.2 Create equations and inequalities in two or more variables to represent relationships between quantities; graph equations on coordinate axes with labels and scales.

A.CED.A.3 Represent constraints by equations or inequalities, and by systems of equations and/or inequalities, and interpret solutions as viable or nonviable options in a modeling context.

A.CED.A.3 Represent constraints by equations or inequalities, and by systems of equations and/or inequalities, and interpret solutions as viable or nonviable options in a modeling context.

A.CED.A.4 Rearrange formulas to highlight a quantity of interest, using the same reasoning as in solving equations.

A.SSE.A.1 Interpret expressions that represent a quantity in terms of its context.

A.SSE.A.2 Rewrite linear, quadratic and exponential expressions.

A.SSE.B.3 Choose and produce an equivalent form of an expression to reveal and explain properties of the quantity represented by the expression.

F.BF.A.1 Write a function that describes a relationship between two quantities.

F.IF.A.1 Understand that a function from one set (called the domain) to another set (called the range) assigns to each element of the domain exactly one element of the range. If f is a function and x is an element of its domain, then $f(x)$ denotes the output of f corresponding to the input x. The graph of f is the graph of the equation $y = f(x)$.

F.IF.A.2 Use function notation, evaluate functions for inputs in their domains, and interpret statements that use function notation in terms of a real-world context.

F.IF.A.3 Recognize that sequences are functions, sometimes defined recursively, whose domain is a subset of the integers.

F.IF.B.4 For a function that models a relationship between two quantities, interpret key features of graphs and tables in terms of the quantities, and sketch graphs showing key features given a verbal description of the relationship.

F.IF.B.5 Relate the domain of a function to a graph and, where applicable, to the quantitative relationship it describes.

F.IF.B.6 Calculate and interpret the average rate of change of a function (presented symbolically or as a table) over a specified interval. Estimate the rate of change from a graph.

F.IF.C.7 Graph functions expressed symbolically and show key features of the graph, by hand in simple cases and using technology for more complicated cases.

F.IF.C.8 Write a function defined by an expression in different but equivalent forms to reveal and explain different properties of the function.

F.IF.C.9 Compare properties of two functions each represented in a different way (algebraically, graphically, numerically in tables, or by verbal descriptions).

F.LE.A.1 Write a function defined by an expression in different but equivalent forms to reveal and explain different properties of the function.

F.LE.A.2 Solve multi-step contextual problems with degree of difficulty appropriate to the course by constructing linear, or exponential function models, where exponentials are limited to integer exponents.

S.ID.B.6 Represent data on two quantitative variables on a scatter plot and describe how the variables are related.

N.RN.B.3 Apply properties of rational and irrational numbers to identify rational and irrational numbers.

N.Q.A.1 Determine an appropriate scale for a graph. Use dimensional analysis to convert units.

N.Q.A.2 Select an appropriate quantity for a real-world context.

S.ID.C.7 Interpret the slope (rate of change) and the intercept (constant term) of a linear model in the real-world context of the data.

S.ID.C.8 Compute (using technology) and interpret the correlation coefficient of a linear fit.

S.ID.C.9 Distinguish between correlation and causation.

Standards from the AP Computer Science A Curriculum

VAR-1.A Create string literals

VAR-1.C Declare variables of the correct types to represent primitive data.

CON-1.A Evaluate arithmetic expressions in a program code.

CON-1.E Evaluate Boolean expressions that use relational operators in program code.

CON-2.A Represent branching logical processes by using conditional statements.

CON-2.C Represent iterative processes using a while loop.

CON-2.E Represent iterative processes using a for loop.

CON-2.G Represent nested iterative processes.

MOD-2.C Describe the functionality and use of program code through comments.

Original Standards by the Author

AS-1 Scientific Computing

AS-1.1 Import packages and functions from the standard library of Python

AS-1.2 Write original Python functions with zero, one, or more arguments

AS-1.3 Describe the difference between a numpy array and a Python list

AS-1.4 Create line graphs, scatter plots, and bar graphs using matplotlib

AS-1.5 Customize axes, axis labels, and titles of matplotlib figures

AS-2 Applied Computing

AS-2.1 Independently decide what calculations should be used as guides for real-world decision making

AS-2.2 Use computed values to justify real-world decisions

4 Course Plan and Description

4.1 Course Description

This curriculum design, called Computational Algebra I, is designed to fill the student requirement in the state of Maryland for Algebra. Specifically, it is based on the units layed out in the Illustrative Mathematics curriculum (Illustrative Mathematics, 2019) which is currently used in the Baltimore County, Howard County, and Montgomery County school districts (*Algebra I Year at a Glance Baltimore County*, 2022; Howard County Public Schools, n.d.; Montgomery County Public Schools, n.d.). That said, Maryland is a Common Core state, and this curriculum could also be applied to other states that work towards the Common Core Math Standards. At the same time, the course aims to build a foundation in applying the math that they learn through programming, with students deepening their understanding of mathematics through their programming assignments.

Students entering the class should have the requisite skills from their math classes up through the eighth grade as described in the state of Maryland standards. This includes understanding of the number system, equations and some exposure to functions and geometry (*Grade 8 Mathematics: Performance Level Descriptors*, 2022).

Students entering the course need no background in programming. The language to be used is Python, the most common language used in data science (Canales Luna, 2023), and will be taught assuming no familiarity.

In this way, the course should prepare them for the following courses required of them, Geometry and Algebra II (Maryland State Department of Education, 2023a) in either a traditional setting or in a course with a computational component.

4.2 Course Breakdown

4.2.1 Overall Course

Unit	Major Topics	Standards	Assessment
(1) One-Variable Statistics (4 weeks)	**Math:** summarize, represent, and interpret on a single count or measurement variable **Programming:** variables, strings, arithmetic, calling functions, length of lists and strings	**Mathematics Standards** N.Q.A.2, A1.R.5, A1.M.4, A1.M.6 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A **Author-Defined Standards** AS-2.1, AS-2.2	***Math Mini Quiz:*** Students calculate the mean, median, mode, and range of a small set of numbers by head, interpret these measures to comment on the dataset. ***Programming Mini Quiz:*** Students declare variables, print a string to the screen, and perform basic addition, subtraction, multiplication, division, and modulo. ***Programming and Math Lab:*** Given data with daily temperatures in different cities students decide which city they would like to live in and why
(2) Linear Equations, Inequalities, and Systems (6 weeks)	**Math:** Writing and modeling equations, manipulating equations and understanding structure, systems of equations and inequalities **Programming:** iteration, logic,	**Mathematics Standards** N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.LE.A.1, F.LE.B.5, S.ID.C.7, A.SSE.A.2, A.SSE.B.3, A.CED.A.1, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.LE.A.2, A.REI.B.3, A.REI.B.4, A.REI.C.6, A.CED.A.3, A.CED.A.4, S.ID.B.6	***Math Mini Quiz:*** Given a word problem, students write an equation to represent a value in the situation; students interpret a graph and interpret it for a real life situation. ***Programming Mini Quiz:*** Students use a loop to implement the fibonacci

	user input	**Programming Standards** VAR-1.A, VAR-1.C, CON-1.A, CON-2.A, CON-2.C, CON-2.E **Author-Defined Standards** AS-2.1, AS-2.2	sequence. ***Programming and Math Lab:*** Students write a program that takes user input and calculates a value from an equation the students develop to match a specific situation (for example, a converter from farenheit to celcius, a program which calculates the area of a polygon given the dimensions, a program that calculates the tip and the total cost of your meal at a restaurant).
(3) Two-Variable Statistics (3 weeks)	**Math:** two-way tables, scatter plots, correlation coefficients, estimating lengths **Programming:** importing functions from packages, using matplotlib to graph data	**Mathematics Standards** N.Q.A.2, S.ID.C.7, F.LE.A.2, S.ID.B.6, S.ID.C.8, S.ID.C.9 **Programming Standards** MOD-2.C, F.IF.B.4 **Author-Defined Standards** AS-1.1, AS-1.4, AS-2.1, AS-2.2	***Math Mini Quiz:*** Given two small sets of data, students must make a two-way table and a scatter plot, determining which situation is best for which, and answer some interpretive questions. ***Programming Mini Quiz:*** Given two lists of data, students perform some basic operations and interpret the answers; students make a scatter plot using matplotlib, including the necessary import. ***Programming and Math Lab:*** Students revisit the data from their projects from unit one, adding graphs to their

			justifications and seeing if their decision about where they want to stay changes.
(4) Functions (4 weeks)	**Math:** what is a function, representations, function notation, inputs and outputs, inverse functions **Programming:** writing functions, arguments, return values	**Mathematics Standards** N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.B.6, S.ID.C.7, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.IF.A.2, A.REI.D.11, F.IF.A.3, F.IF.C.7, A1.R.3, A1.R.4, A1.R.5 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A, MOD-2.C **Author-Defined Standards** AS-1.4, AS-1.2, AS-1.1, AS-2.1, AS-2.2	***Math Mini Quiz:*** Students determine if a graph represents a function; students graph and answer interpretive questions about two functions. ***Programming Mini Quiz:*** Students write and call two functions with one input. ***Programming and Math Lab:*** Students expand their project from Unit 2 to include new cases or conditions. The calculations for each condition should be done in a separate function.
(5) Introduction to Exponential Functions (5 weeks)	**Math:** Growth, exponential functions, percent growth and decay, comparing linear and exponential functions **Programming:** importing functions from packages, creating effective graphs for presentation with matplotlib	**Mathematics Standards** N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.B.6, S.ID.C.7, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.IF.A.2, A.REI.D.11, F.IF.A.3, F.IF.C.7, A1.R.3, A1.R.4, A1.R.5 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A, MOD-2.C	***Math Mini Quiz:*** *G*iven several graphs, students categorize the types of growth demonstrated including: exponential growth, exponential decay, and linear functions; students write an exponential equation for a described real-life scenario and answer questions about it. ***Programming Mini Quiz:*** Given lists of data, students make line graphs, including the necessary import, and print statistics calculated by imported functions.

		Author-Defined Standards AS-1.4, AS-1.2, AS-1.1, AS-2.1, AS-2.2	***Programming and Math Lab:*** Students choose a situation (given options) that can be modeled with <u>exponential growth or decay</u> and graphically investigate the effect of changing different parameters on the results. For example in y = a(1 + r)^x , what happens as a increases? What happens as r changes? What does this mean for the result in the specific situation chosen?
(6) Introduction to Quadratic Functions and Their Forms (5 weeks)	**Math:** Quadratic Functions, Quadratic Expressions, Features of graphs of quadratic functions **Programming:** numpy arrays	**Mathematics Standards** N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.A.3, F.IF.B.4, F.IF.B.5, F.IF.C.8, F.LE.A.1, A.SSE.A.2, A.SSE.B.3, A.CED.A.4, A.APR.A.1, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.LE.A.2, A.REI.B.4, F.IF.C.9, A1.R.1, A1.R.2, A1.R.3, A1.R.5, A1.R.10 **Programming Standards** VAR-1.A, VAR-1.C, MOD-2.C **Author-Defined Standards** AS-1.1, AS-1.2, AS-1.3, AS-1.4, AS-1.5, AS-2.1, AS-2.2	***Math Mini Quiz:*** Given several graphs, categorize or identify a, h, k for quadratic equations of the form y = a(x-h)² + k; students write the quadratic equation from a graph. ***Programming Mini Quiz:*** Students use basic numpy array operations to answer questions about a real-world scenario. ***Programming and Math Lab:*** Students choose a situation (given options) that can be modeled with <u>quadratic functions</u> and graphically investigate the effect of changing different parameters on the results. For example in y = a(x – h)^2 + k , what happens as a increases? What happens as h and k change? What does

			this mean for the result in the specific situation chosen?
(7) Quadratic Equations (4 weeks)	**Math:** Finding Solving quadratic equation, completing the square, quadratic formula **Programming:** objects and properties	**Mathematics Standards** N.RN.B.3, N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.C.8, F.LE.A.1, A.SSE.A.2, A.SSE.B.3, A.APR.A.1, A.APR.B.3, A.CED.A.2, A.CED.A.3, A.CED.A.4, F.BF.A.1, F.LE.A.2, A.REI.B.4, F.IF.C.9, A1.R.1, A1.R.2, A1.R.3, A1.R.5, A1.R.9, A1.R.10 **Programming Standards** VAR-1.A, VAR-1.C, MOD-2.C **Author-Defined Standards** AS-1.2, AS-1.4, AS-1.5, AS-2.1, AS-2.2	***Math Mini Quiz:*** Students convert quadratic equations between standard, factored, and vertex form; students write a quadratic equation from a described situation and answer questions about it. ***Programming Mini Quiz:*** Students write a small class for a described situation. ***Programming and Math Lab:*** Students write a quadratic class with functions for visualizing, printing different forms, and finding the roots; students use this class to analyze a situation of choice by the students (examples: ball position, calculating areas for gardening).

4.2.2 Unit Breakdowns

Unit 1: One-Variable Statistics (4 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2"></td><td colspan="2">Content</td></tr>
<tr>
<td>Goals:

Students will be better able to:

● Draw conclusions and information from a variety of plot types
● Calculate and interpret measures of center (median and mean) and spread (range, interquartile range, and standard deviation)
● Use the Google CoLab platform to write and run code by themselves
● Drawing their own conclusions from and answering open ended questions with data by themselves</td>
<td>Mathematical Understandings:
● The mean and median are both measures of center each with their own strengths and weaknesses.
● Range, interquartile range, and standard deviation are measures of spread each with their own strengths and weaknesses.
● Dot plots, histograms, box plots, and line graphs can all be used to visualize single variable datasets each with their own strengths and weaknesses.

Programming Understandings:

● Python executes top to bottom and left to right
● In Python Notebooks (like in Google CoLab), a cell of code is not executed until you run it
● The console is where a program outputs and is where the program can interact with the user
● Variables are used to store and access values without code
● Variables names in Python have restrictions
● Values have types which can include integers, floats, and strings</td>
<td>Essential Questions:

Why is programming used to process data?

How can data analysis be targeted to answer real-world questions?</td>
</tr>
</table>

	● Modulo is the same as taking the remainder of a division operation	
Mathematics Standards N.Q.A.2, A1.R.5, A1.M.4, A1.M.6 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A **Author-Defined Standards** AS-2.1, AS-2.2	*Skill Acquisition*	
	Mathematical Skills: ● Calculate the mean, median, and range of a one-variable data set by hand ● Identify and explain the significance of quartiles dot plots, histograms, and box plots	**Programming Skills:** ● Open a Python notebook using Google CoLaboratory ● Run individual cells of code in Google CoLaboratory ● Print to console ● Declare a variable ● Perform arithmetic operations between numbers and variables ● Given a list, use Python builtin functions and indexing to: ○ Find the number of values contained ○ Find the sum of the values contained ○ Put a numerical list in order ○ Find a value at a ¡n index in a List, for the purpose of finding the median of a dataset

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students will take a quiz in which they calculate the mean, median, mode, and range of a small set of numbers by head, and interpret these measures to comment on the dataset. Situations used include free-diving times, javelin throwing distances, and 100m race times.
Programming	***Programming Mini Quiz:*** Students take a quiz in which they declare variables, print a string to the screen, and perform

	basic addition, subtraction, multiplication, division, and modulo.
Math and Programming Lab	***Programming and Math Lab:*** Students complete a lab in which given data with daily temperatures in different cities, they decide which city they would like to live in and why.

Stage 3 - Learning Plan

Week(s) 1-2:

Lesson 1 (1 days): Student introductions, the difference between statistical and non-statistical questions

Lesson 2 (2 days): Getting to know the students in the class through single count variables, review of the measures they should know entering the class (mean, median, range)

Lesson 3 (1-2 days): How to read a dot plot and histogram

Lesson 4 (1-2 days): *Math Mini Quiz 1*, standard deviation, quartiles

Week(s) 2-3:

Lesson 5 (2 days): What is programming? Getting started with Google CoLab, “Hello World”

Lesson 6 (1 day): Declaring variables, arithmetic, modulo with Python

Lesson 7 (1 day): Practice problems solving for the mean and median using Python

Lesson 8 (1-2 days): *Programming Mini Quiz 1* and review the answers

Lesson R.1 (1-2 days, optional): Review of the mathematical content from weeks 1-2

Week 4:

Lesson 9 (5 days): *Programming and Math Lab 1*, (optional) students present their findings

Unit 2: Linear Equations, Inequalities, and Systems (6 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:

● Create and use mathematical expressions to express and analyze real-world situations
● Validly manipulate mathematical equations to create equivalent equations
● Solicit and use user input to perform arithmetic operations in Python
● Use loops to execute iterative processes
● Use boolean logic to create branched programs</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:

● “Constraints” express limitations on real-world situations.
● “Equivalent equations” define the same relationships and have the same solution.
● Two graphs intersecting represent the solution to a system of linear equations
● Linear equations with the form y = mx + b have their y-intercept at b and a slope of m.

Programming Understandings:

● Strings cannot be added in the same way integers and floats can.
● For loops and while loops are two different ways of executing iterative tasks each of which with their own weaknesses and strengths.
● If and else statements can be used to make branches of a program, where each branch is only executed when a logical condition is met.
● Python error messages show the type of error, the location of the error, as well as attempt to provide detailed information about what specifically</td><td>Essential Questions:

What is a “constraint” and how can it be represented mathematically?

How does the type a Python value affect the operations that can be performed on it?</td></tr>
</table>

<table>
<tr><td></td><td>went wrong in the execution of the program.</td><td></td></tr>
<tr><td rowspan="2">Mathematics Standards

N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.LE.A.1, F.LE.B.5, S.ID.C.7, A.SSE.A.2, A.SSE.B.3, A.CED.A.1, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.LE.A.2, A.REI.B.3, A.REI.B.4, A.REI.C.6, A.CED.A.3, A.CED.A.4, S.ID.B.6

Programming Standards

VAR-1.A, VAR-1.C, CON-1.A, CON-2.A, CON-2.C, CON-2.E

Author-Defined Standards

AS-2.1, AS-2.2</td><td colspan="2">Skill Acquisition</td></tr>
<tr><td>Mathematical Skills:
Express equations from the constraints imposed by real-world problems
Determine which quantities change and which remain constant in a real-world situation
Manipulate an equation to create an equivalent equation to isolate a desired quantity
Solve systems of 2-variable linear equations</td><td>Programming Skills:
Use a for and while loop to perform repeated and iterative computational operations
Use the input function to solicit and process user input through the console
Use if and else statements to correctly control the logical flow of a program
Read and understand a Python error message to determine the nature and location of a bug in code</td></tr>
</table>

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students take a quiz in which given a word problem, they write an equation to represent a value in the situation. Students also interpret a graph and interpret it for a real life situation. Situations used include jet ski fuel usage and emptying one water cooler into another.
Programming	***Programming Mini Quiz:*** Students take a quiz in which they use a loop to implement the fibonacci sequence.
Math and	***Programming and Math Lab:*** Students complete a lab in which they write a program that takes user input and calculates a

Programming Lab	value from an equation the students develop to match a specific situation (for example, a converter from farenheit to celcius, a program which calculates the area of a given the dimensions, a program that calculates the tip and the total cost of your meal at a restaurant).

Stage 3 - Learning Plan

Week(s) 1-2:

Lesson 1 (1 day): Review of symbols <, >, ≤, ≥ and how to write single-variable inequalities from situations, including identifying which quantities can change and which ones can't

Lesson 2 (1-2 days): How to write single-variable and and two-variable equations from situations, including identifying which quantities can change and which ones can't

Lesson 3 (2 days): Graphing inequalities and sets of equations, describing the meaning of their intersections and overlaps

Lesson 4 (1 days): More practice problems making inequalities and equations from situations, graphing them, and reasoning about their results

Lesson R.2 (1-2 days, optional): Review the programming concepts from unit 1

Week(s) 3-4:

Lesson 5 (1 day): "Equivalent equations" and practice determining if equations are equivalent

Lesson 6 (1-2 days): Manipulating equations by making equivalent equations to solve equations for specific variables

Lesson 7 (1 day): Practice practice problems making questions from situations and solving them

Lesson 8 (1 day): *Math Mini Quiz 2*, the meaning of m and b within $y = mx + b$

Week 5*:

Lesson 9 (1-2 day): *If/Else* logic, and *and/or* combinational logic

Lesson 10 (1 day): *For* loops

Lesson 11 (1-2 day): *While* loops, and practice problems

Lesson 12 (1 day): *Programming Mini Quiz 2* and review the answers

Week 6:

Lesson 12 (4-5 days): *Programming and Math Lab 2*, (optional) students present their findings

**While these lessons have been put in week 5, they can also be dispersed throughout weeks 1-4 as they do now require knowledge specific knowledge of the content*

Unit 3: Two-Variable Statistics (3 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:

● Solicit and use user input to perform arithmetic operations in Python
● Use loops to execute iterative processes
● Use boolean logic to create branched programs</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:

● Two-way tables and scatter plots are both ways of examining two variable data sets, each with their own weaknesses and strengths in specific situations.
● Pearson correlation coefficients go from -1 to 1, with 1 being a strong positive correlation and -1 being a strong negative correlation.
● Relative frequencies represent the percentage of the time an outcome occurred and can at times be taken as a probability for future events.

Programming Understandings:

● A package is a code written by outside authors with the intended purpose of being used by others.
● Packages can be used via the import and as statements in Python.
● When using Matplotlib, graphics will not be displayed until the show function is called.</td><td>Essential Questions:

What is the difference between a two-way table and a scatter plot?

Why are certain types of plots better suited for certain types of data?

What are packages in Python and are they used in scientific computing?</td></tr>
<tr><td>Mathematics Standards</td><td colspan="2">Skill Acquisition</td></tr>
</table>

N.Q.A.2, S.ID.C.7, F.LE.A.2, S.ID.B.6, S.ID.C.8, S.ID.C.9 **Programming Standards** MOD-2.C, F.IF.B.4 **Author-Defined Standards** AS-1.1, AS-1.4, AS-2.1, AS-2.2	**Mathematical Skills:** ● Calculate relative frequencies given a two-way table ● Interpret the meaning of relative frequencies in a variety of real-world situations ● Determine the sign and strength of a correlation between two variables ● Interpret the meaning of correlation coefficients in a variety of real-world situations	**Programming Skills:** ● Import packages, including matplotlib ● Use functions imported from external packages ● Make and show a scatter plot using matplotlib

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students take a quiz in which given two small sets of data, they must make a two-way table and a scatter plot, determine which situation is best for which, and answer some interpretive questions. Situations used include demographic analysis.
Programming	***Programming Mini Quiz:*** Students take a quiz in which given two lists of data, they perform some basic operations and interpret the answers. They also make a scatter plot using matplotlib, including the necessary import. Situations used include comparing GDP and life expectancy data.
Math and Programming Lab	***Programming and Math Lab:*** Students complete a lab in which they revisit the data from their projects from unit one, and add graphs to their justifications to see if their decision about where they want to stay changes.

Stage 3 - Learning Plan
Week 1: **Lesson 1 (1-2 day):** What are two-way tables, calculating relative frequencies **Lesson 2 (1 day):** Scatter plots **Lesson 3 (1-2 day):** Correlation coefficients, visually determining what is the line of best fit **Lesson 4 (1-2 day):** Practice problems with real life situations analyzing graphs for to extract information, *Math Mini Quiz 3* **Week 2:** **Lesson 5 (1 day)**: What are packages? How to import packages **Lesson 6 (1 day):** Matplotlib, making a figure and showing it, students graph various sets of given data **Lesson 7 (1-2 days):** *Programming Mini Quiz 3*, Real-world data set problems. Students make scatter plots and describe the correlation coefficient **Week 3:** **Lesson 8 (5 days):** *Programming and Math Lab 3*, (optional) students present their findings

Unit 4: Functions (4 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:

● Describe mathematical and programming functions
● Use functions and function notation to solve real world-problems
● Combine mathematical and programming function knowledge to write code which solves real-world problems</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:

● Functions are relationships between variables in which each input has exactly one output.
● An independent variable represents the input of a function and the dependent variable represents the output of a function.
● The average rate of change can be defined can be used to characterize the way functions change over an interval
● Linear functions have a constant rate of change.
● Piecewise functions are functions that can be described as sub-functions, each of which serving over a specific part of the domain.

Programming Understandings:

● A programming function is a function that can be reused later.
● The def keyword is used to define functions in Python.
● Functions have arguments which are given when calling the function and are available only within that function.</td><td>Essential Questions:

What is a function in mathematics?

What is a function in programming?

How are programming and mathematical functions similar and different?

What are the independent and dependent variables of a situation?

What is a piecewise function and how does it differ from other functions?</td></tr>
<tr><td></td><td colspan="2">Skill Acquisition</td></tr>
</table>

Mathematics Standards	**Mathematical Skills:**	**Programming Skills:**
N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.B.6, S.ID.C.7, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.IF.A.2, A.REI.D.11, F.IF.A.3, F.IF.C.7, A1.R.3, A1.R.4, A1.R.5 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A, MOD-2.C **Author-Defined Standards** AS-1.4, AS-1.2, AS-1.1, AS-2.1, AS-2.2	● Determine if a graph depicts a function ● Write equations in function notation ● Use function notation to write equations and interpret the relationship between variables for real-world situations ● Interpret the meaning of intercepts, maximums, and minimums of graphed functions ● Analytically find the intersection of two functions by setting them as equal to each other ● Determine a reasonable domain and range for a function given a situation ● Analytically find the inverse of a function ● Write the equation from and graph piecewise functions, accurately interpreting the values at the end of each interval	● Declare a function with one input and one return valuable ● Call custom functions with one input and one return valuable ● Declare and call functions with more than one variable ● Decide which part of a script can be put into a function to make the code easier to read and reusable ● Accurately determine which variables in a program are available within a function

Stage 2 - Evidence	
Mathematics	Students take a quiz in which they determine if a graph represents a function. Students also graph and answer interpretive questions about two functions. Situations used include train travel and body temperature regulation.
Programming	Students take a quiz in which they write and call two functions with one input each. Situations used include finding the area and circumference of a circle.
Math and Programming Lab	Students complete a lab in which they expand their project from Unit 2 to include new cases or conditions. The calculations for each condition should be done in a separate function.

Stage 3 - Learning Plan
Week(s) 1-2: **Lesson 1 (1 day):** Definition of a function as each input leading to one output **Lesson 2 (1-2 days):** Writing equations in function notation **Lesson 3 (2 days):** Visually analyzing minimums, maximums, and intercepts of graphs **Lesson 4 (1-2 days):** Average rate of change **Lesson 5 (2 days):** Piecewise functions **Lesson 6 (1 day):** *Math Mini Quiz 4*, review the answers **Week(s) 2-3:** **Lesson 7 (1 day):** What is a programming function? How are they similar or different from programming functions? **Lesson 8 (1-2 day):** Writing programming functions, when to make functions **Lesson 9 (1-2 day):** Real-world problem practice **Lesson 10 (1 day):** *Programming Mini Quiz 4*, review the answers **Week 4:** **Lesson 11 (4-5 days):** *Programming and Math Lab 4*, (optional) students present their findings **Lesson 12 (1 day)**: Inverse functions

Unit 5: Introduction to Exponential Functions (5 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:
● Describe and perform basic calculations on exponential growth functions
● Effectively communicate graphs using matplotlib</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:
● Exponential functions take the form $y = a(b)^x$.
● Exponential percent growth and decay take the form $y = a(1+r)^x$.
● Pure exponential decay functions never reach zero.
● Continuous functions can be calculated at an infinite number of input points, which discrete have specific valid inputs.
● The value of exponential functions will eventually surpass the value of linear functions.

Programming Understandings:
● Packages have functions within them which can be imported without importing the entire package.
● Graphics are intended for communication, and as such proper labeling is an essential part of making a graphic.</td><td>Essential Questions:

How are exponential functions different from linear functions?

What happens as a, b, and r are manipulated in an exponential function?

Why do exponential decay functions approach but never reach 0?

What is the purpose of a graphic? How can we use Python to serve that purpose?</td></tr>
<tr><td>Mathematics Standards</td><td colspan="2">Skill Acquisition</td></tr>
</table>

N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.B.6, S.ID.C.7, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.IF.A.2, A.REI.D.11, F.IF.A.3, F.IF.C.7, A1.R.3, A1.R.4, A1.R.5 **Programming Standards** VAR-1.A, VAR-1.C, CON-1.A, MOD-2.C **Author-Defined Standards** AS-1.4, AS-1.2, AS-1.1, AS-2.1, AS-2.2	**Mathematical Skills:** ● Explain the difference in behavior between linear and exponential functions ● Explain the meaning of a, b, and x in an exponential function ● Distinguish exponential decay and exponential growth from a graph and from an equation ● Determine based on the description of a situation if it will be modeled by exponential growth or decay ● Calculate a change rate from a graph or a table ● Convert between exponential function and percent growth or decay functions	**Programming Skills:** ● Import specific functions from packages use the *from* keyword ● Using Matplotlib, add important labels to a chart such as a title, axis labels, and legend

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students take a quiz in which given several graphs, they categorize the types of growth demonstrated including: exponential growth, exponential decay, and linear functions. They also write an exponential equation for a described real-life scenario and answer questions about it. Situations used include the spread of mold.
Programming	***Programming Mini Quiz:*** Students take a quiz in which given lists of data, they make line graphs, including the necessary import, and print statistics calculated by imported functions. Situations used include calculating the area and circumference of a circle.
Math and Programming Lab	***Programming and Math Lab:*** Students complete a lab in which they choose a situation (given options) that can be modeled with exponential growth or decay and graphically investigate the effect of changing different parameters on the results. For example in y = a(1 + r)^x , what happens as a increases? What happens as r changes? What does this mean for the result in the specific situation chosen?

Stage 3 - Learning Plan
Week(s) 1-2: **Lesson 1 (2 days):** Introduction of exponential growth through a discrete example of the giving a kid a dollar every day versus one cent and doubling it each day, derivation of the formula **Lesson 2 (1 day):** Comparing the average rate of change of the exponential and linear function explored in Lesson 1 **Lesson 3 (1-2 days):** Finding growth factor of an exponential **Lesson 4 (1-2 day)**: Expansion to the continuous case **Lesson 5 (2 days)**: Introduction of exponential decay as having a growth factor less than 1 **Week(s) 2-3:** **Lesson 6 (2-5 days)**: Real-world problem solving with exponential growth and decay **Lesson 7 (1 day):** *Math Mini Quiz 5*, review the answers **Week(s) 3-4:** **Lesson 7 (1 day):** Why make graphics? Exploring useful and unuseful graphics **Lesson 8 (1 day):** How to make graphics useful and readable with Matplotlib, title, axis labels, colors, legends **Lesson 9 (2 days):** Practicing makings graphics to communicate contextual information using Matplotlib; tick marks and text **Lesson 10 (1-2 days):** Importing specific functions from packages using the *from* keyword, using matplotlib to plot exponential growth and decay **Lesson 11 (1 day):** *Programming Mini Quiz 5*, review the answers **Week 5:** **Lesson 12 (5 days):** *Programming and Math Lab 5*, (optional) students present their findings

Unit 6: Introduction to Quadratic Functions and Their Forms (5 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:

● Describe standard, vertex, and factored forms of quadratic equations and what the constants in each form represent.
● Write a quadratic equation to describe an appropriate real-world situation
● Use external code in their own Python programs</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:

● Quadratic functions can describe quantities that go up and then down.
● Each term in the quadratic function representing a free falling object has a meaning.
● Quadratics can be written in “standard form,” “vertex form,” and “factored form” each making particular pieces of information easily readable from the equation.
● Equations representing real-world phenomena often have restricted domains in which they are valid.

Programming Understandings:

● Numpy is a common Python package that is used for computing with list-like data (arrays).
● Numpy arrays are often better for numerical calculations on large groups of numbers than Python lists.
● Numpy arrays allow for easier manipulation of entire arrays, for example multiplication of an entire array by a constant or pointwise multiplication of equal-dimensioned</td><td>Essential Questions:

What are standard, vertex, and factored forms of quadratic equations? Why analyze equations using different forms?

When is it best to use Numpy and when is it best to use a Python list?

What are the advantages and disadvantages of using external code?</td></tr>
</table>

<table>
<tr><td></td><td>arrays.</td><td></td></tr>
<tr><td rowspan="2">Mathematics Standards

N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.A.3, F.IF.B.4, F.IF.B.5, F.IF.C.8, F.LE.A.1, A.SSE.A.2, A.SSE.B.3, A.CED.A.4, A.APR.A.1, A.CED.A.2, A.CED.A.3, F.BF.A.1, F.LE.A.2, A.REI.B.4, F.IF.C.9, A1.R.1, A1.R.2, A1.R.3, A1.R.5, A1.R.10

Programming Standards

VAR-1.A, VAR-1.C, MOD-2.C

Author-Defined Standards

AS-1.1, AS-1.2, AS-1.3, AS-1.4, AS-1.5, AS-2.1, AS-2.2</td><td colspan="2">Skill Acquisition</td></tr>
<tr><td>Mathematical Skills:
Manipulate equations to convert quadratic equations between standard, vertex, and factored form
Describe the ways the constants in the standard, vertex, and factored forms relate to properties of the graphs and the real-world situations they represent
Given a real-world situation modeled by a quadratic function, write the quadratic function.
Determine a valid domain and range of a quadratic function that describes a real-world phenomenon</td><td>Programming Skills:
Import Numpy into a Python notebook
Differentiate between numpy arrays and Python lists in terms of form and usage
Perform arithmetic on entire Numpy arrays
Learn new numpy functions through examples</td></tr>
</table>

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students take a quiz in which given several graphs, they categorize or identify a, h, k for quadratic equations of the form $y = a(x-h)^2 + k$. They also write the quadratic equation from a graph. Situations used include the shape of a contact lens.
Programming	***Programming Mini Quiz:*** Students take a quiz in which they use basic numpy array operations to answer questions about a real-world scenario. Situations used include comparing GDP and life expectancy.

Math and Programming Lab	***Programming and Math Lab:*** Students complete a lab in which they choose a situation (given options) that can be modeled with quadratic functions and graphically investigate the effect of changing different parameters on the results. For example in y = a(x – h)^2 + k , what happens as a increases? What happens as h and k change? What does this mean for the result in the specific situation chosen?

Stage 3 - Learning Plan

Week 1:

Lesson 1 (1 day): What is numpy? What is a numpy array? Why is it useful?

Lesson 2 (2 days): Numpy array arithmetic and indexing

Lesson 3 (1 day): Practice problems making graphics of functions using numpy using *arange* and numpy arithmetic

Lesson 4 (1 day): *Programming Mini Quiz 6*, review the answers

Weeks 2-3:

Lesson 5 (2 days): Introducing quadratic functions through free-falling objects

Lesson 6 (3 days): Interpreting intercepts and vertices of graphs in the context of free-falling objects and other real-world applications

Lesson 7 (5-7 days): *Programming and Math Lab 6*, (optional) students present their findings

Weeks 4-5:

Lesson 8 (2-3 days): Factoring quadratic equations, introduction to the idea that $(x+m)(x+n)=x+(m+n)x+mn$ and the special cases in which m and n have the same absolute value

Lesson 9 (2-3 days): Converting from standard form to vertex form

Lesson 10 (4 days): Real-world problems writing quadratic equations, converting between the forms, and interpreting the constants of the different forms

Lesson 11 (1 day): *Math Mini Quiz 6*, review the answers

Unit 7: Quadratic Equations (4 weeks)

<table>
<tr><th colspan="3">Stage 1 - Desired Results</th></tr>
<tr><td rowspan="2">Goals:

Students will be better able to:
● Solve quadratic equations using a variety of methods, choosing the most appropriate based on the formula and the represented situation
● Determine when it is beneficial to write a custom class to solve a programming problem
● Write a custom class to help solve a programming problem</td><td colspan="2">Content</td></tr>
<tr><td>Mathematical Understandings:
● Quadratic formulas cannot be solved using the same methods as linear equations.
● The quadratic formula is a generalizable way to find the zeros (real or imaginary) of a quadratic equation.
● The number of times a quadratic equation crosses the x-axis is equivalent to the number of real zeros that equation has.
● If the product of two numbers is zero, then at least one of these numbers must be zero.
● The plus minus symbol can be used to express two solutions to a quadratic equation with one expression.
● Knowing the rational and irrational composition of two terms in a sum or two factors in a product, the rationality of the result can be determined.

Programming Understandings:
● A class in Python allows users to make their own custom data types.
● All class methods have self as their first</td><td>Essential Questions:

Why can’t many quadratic equations be solved with the same methods used for linear equations?

What is the quadratic formula and why is it useful?

What is an object?

What is a class?

Why write a custom class?</td></tr>
</table>

	argument which refers to the object itself. ● Instance variables are properties held within the instance of an object that can be accessed later.	
Mathematics Standards N.RN.B.3, N.Q.A.1, N.Q.A.2, A.SSE.A.1, F.IF.B.4, F.IF.B.5, F.IF.C.8, F.LE.A.1, A.SSE.A.2, A.SSE.B.3, A.APR.A.1, A.APR.B.3, A.CED.A.2, A.CED.A.3, A.CED.A.4, F.BF.A.1, F.LE.A.2, A.REI.B.4, F.IF.C.9, A1.R.1, A1.R.2, A1.R.3, A1.R.5, A1.R.9, A1.R.10 **Programming Standards** VAR-1.A, VAR-1.C, MOD-2.C **Author-Defined Standards** AS-1.2, AS-1.4, AS-1.5, AS-2.1, AS-2.2	*Skill Acquisition* **Mathematical Skills:** ● Solve a quadratic equation using factoring ● Solve a quadratic equation using "complete the square" ● Use the quadratic formula to find the factors and the solutions to quadratic equations ● Describe the relationship between the number of factors and the number of x-intercepts ● Describe why knowing the rational and irrational composition of two terms in a sum or two factors in a product, the rationality of the result can be determined	**Programming Skills:** ● Declare a class in Python ● Create a constructor using the *__init__* method which saves instance variables ● Create methods for a custom class, correctly using the *self* keyword as the first argument ● Properly access instance variables within a method ● Design a class to best fit the use case of a given situation

Stage 2 - Evidence	
Mathematics	***Math Mini Quiz:*** Students take a quiz in which they convert quadratic equations between standard, factored, and vertex form. They also write a quadratic equation from a described situation and answer questions about it. Situations used include the trajectory of a basketball player dunking.

Programming	***Programming Mini Quiz:*** Students take a quiz in which they write a small class for a described situation. Situations used include a social media user.
Math and Programming Lab	***Programming and Math Lab:*** Students complete a lab in which they write a quadratic class with functions for visualizing, printing different forms, and finding the roots. They also use this class to analyze a situation of choice by the students (examples: ball position, calculating areas for gardening).

Stage 3 - Learning Plan

Weeks 1-2:

Lesson 1 (1 day): Motivate the need for special methods for solving quadratic equations by attempting to isolate x in a quadratic equation, solving quadratic equations from the factored form

Lesson 2 (1-2 day): Solving quadratic equations is equivalent to finding the x-intercepts of a quadratic, determining the number of real solutions from the graph

Lesson 3 (1-2 day): Quadratic formula derivation and usage

Lesson 4 (1-2 day): Rational and irrational sums and products

Lesson 5 (1 day): *Math Mini Quiz 7*, review the answers

Weeks 2-3:

Lesson 6 (1 day): What are objects? How have we been working with them up until now? What is a class?

Lesson 7 (1-2 days): Declaring classes, constructors, and instance variables

Lesson 8 (1-2 days): Writing methods, using the *self* argument

Lesson 9 (1 days): Practice making a full class from a specification with methods and instance variables

Lesson 10 (1 day): *Programming Mini Quiz 7*, review the answers

Week 4:

Lesson 11 (5 days): *Programming and Math Lab 7*, (optional) students present their findings

4.2.3 Selection of Lesson Plans

Unit 1 Lesson 7

Unit: One-Variable Statistics (1)

Grade Level: 7-9th grade

Topic: Mean, Median

Time: 1 session (60 minutes)

Essential Questions: How does programming help us perform calculations on big data sets?	**Required Prior Knowledge:** ● How to compute the mean and median of a dataset by hand ● How to run Google CoLab from a browser ● How to declare variables, do basic arithmetic, and print values in Python
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook files: Questions, Answers	**Vocabulary:** Mean, Median, Python List, Sum
Brief Descriptions of Problems Used: ● Deciding if it's possible a student cheated based on their test scores and the scores of the class	

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Calculate the mean and median of a set of numbers by hand	Informal	Visual and individual assessment during the warmup
Use Python to calculate mean and median, find the length of lists, and sort lists	Formal	Assignment due at the end of class
Describe the difference between mean and median, and when we would use each	Formal	Assignment due at the end of class

Summary of Activities

Warm-Up (10 minutes)	Students calculate the mean and median of small groups of numbers by hand. Students and teachers review the answers.

Framing (5 minutes)	Pose the question: “What if we have groups of numbers that are so big that we can’t calculate them by hand?” as motivation for using Python to code.
Activity Introduction (5 minutes)	Teacher shows where to find the base file and instructions for the exercise. The base file includes several data sets stored as lists.
Programming Lesson (10 minutes)	Teacher demonstrates how to find the length of a list, how to sort a list, and how to use the “sum” function. *Note: Students who work quickly should be allowed to continue to the next part during the programming lesson.*
Student Practice (15 minutes)	Students download the base file and complete the activity, calculating the mean and median of each data set, and comparing the results.
Activity Closing (5 minutes)	Review the answers to the questions. Students turn in their work.

Unit 1 Lesson 8

NOTE: This lesson is made for the Unit 1 programming mini-quiz, but a similar structure can be used for the programming mini-quiz days for other units.

Unit: One-Variable Statistics (1)

Grade Level: 7-9th grade

Topic: Python Variables and Arithmetic, Printing

Time: 1 session (60 minutes)

Essential Questions: How can Python be used for simple arithmetic? How can text be used to add context to printed values?	**Required Prior Knowledge:** ● Declare a variable in Python ● Perform addition, subtraction, multiplication, division, and modulo operations using Python ● Print to console in Python using the *print* function and string literals
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook files: Questions, Solution	**Vocabulary:** Print, Variable, Remainder, Sum, Product

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Declare Python variables and perform Python arithmetic	Formal	Programming Mini Quiz
Use strings along with the *print* function to add context to printed values	Formal	Programming Mini Quiz
Assess their own work as well as the work of their peers	Informal	Listening to student conversations during the *Student Practice* section of the lesson
Use assessment of their own work as well as the work of their peers to improve their original solutions	Formal	Programming Mini Quiz - Peer Work Submission

Summary of Activities

Warm-Up (10 minutes)	Students review the syntax for making variables and doing arithmetic with 3 warm-up questions that they write the answer to on a piece of a paper. Students are allowed to work with other students at this time.

Review (5 minutes)	Review the answers to the warm-up as a class. The teacher should call particular attention to the symbols used for the arithmetic operations (*, +, -, /, %) and the meaning of modulo.
Programming Mini Quiz (15 minutes)	Students download the quiz file from their learning management system (Google Classroom or similar), individually complete their Python notebook, and ***turn in their results***.
Student Practice (15 minutes)	Students compare their answers with partners. They work towards a partner set of answers. If students think they missed answers the first time, they can submit their answers from their pair-work to receive partial credit for newly correct answers.
Answer Review (10 minutes)	As a class, review the answers to the Programming Mini Quiz.
Exit Ticket (5 minutes)	To review the skills from the programming mini quiz, students perform a hand-written coding exercise where they must declare two variables, perform modulo division, and print the result. They turn this in as their exit ticket.

Unit 1 Lesson 9

NOTE: This lesson serves as a template for lessons where the students do their math and programming labs.

Unit: One-Variable Statistics (1)

Grade Level: 7-9th grade

Topic: Using single-variable statistics to make and justify decisions

Time: 5 sessions *(60x5=300 minutes)*

Essential Questions:	**Required Prior Knowledge:**
What are some real-world situations in which it is useful to use programming to calculate summary statistics? How can someone use statistics to justify their preferences? How can people use the same data to reach different conclusions?	• Declare variables in Python • Perform arithmetic in Python • Calculate the mean, median, and range of a set of numbers by hand
Materials: • Computers for each student with access to a web browser and Google CoLab not blocked on the local network • Math and Programming Lab 1 base file	**Vocabulary:** Average, Justification, List, Data Set

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Use statistics to make and justify decisions	Formal	Lab
Use Python to calculate useful statics on big data sets	Formal	Lab
Reason about open-ended problems requiring quantifiable evidence	Formal and Informal	Lab, Student-Teacher interactions during the working period

Summary of Activities

Warm-Up (15 minutes)	Students are prompted to think about what temperatures they like with a problem in which they are given two options for vacation locations. They are given the average temperature for each day of the week they

	are thinking of going on vacation. They should decide which place they'd prefer to go.
Warm-up Debrief (10 minutes)	Teacher leads a discussion about how students have decided which place they'd prefer to go.
Activity Introduction (5 minutes)	Teacher shows where to find the base file and instructions for the exercise. At this point, students who would like are allowed to start working independently.
Programming Lesson (30 minutes)	Teacher explains the exercise for the lab. Teacher also goes over three examples of using Python lists to access elements and call functions on them like *len* and *sorted*.
Student Work (3-4 hours)	For the remaining classes of this week, students are allowed to work on their projects and ask the teacher for help. Students who finish early should be prompted to create a small presentation displaying their work and the decision that they made.
(Time Permitting) Presentations (5-60 minutes)	Students present their work to the class

Unit 2 Lesson 11

Unit: Linear Equations, Inequalities, and Systems (2)

Topic: While loops, using logic and loops in practice problems

Grade Level: 7-9th grade
Time: 2 sessions *(120 minutes)*

Essential Questions:	**Required Prior Knowledge:**
What is the difference between a for and a while loop? How is each used? When should each be used?	● How to represent an iterative process using a *for* loop ● How to use *if* and *else* statements to make branching logical patterns
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook files: Questions, Answers	**Vocabulary:** Loop, Iteration, Condition
Brief Descriptions of Problems Used: ● Comparing rookie statistics of two famous NBA players ● Determining if a number is prime ● Validating the length of a user-input password ● Reversing a number and determining if it's a palindrome	

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Use a while loop with two changing variables to analyze a situation	Formal	Through the turned-in notebook at the end of the class
Use a while loop to traverse a list	Formal	Through the turned-in notebook at the end of the class
Justify evaluative opinions using statics calculated using Python	Formal	Through the turned-in notebook at the end of the class

Summary of Activities

Warm-Up (10 minutes)	Example of a problem that is inconveniently written as a for loop, validating a user password

Warm-up Debrief (5 minutes)	Teacher explains that it is difficult if not impossible to make an infinite loop using *for*, and this is the motivation for why we're going to learn the *while loop*.
Part I: While Loops (15 minutes)	As a class, the teacher goes through the exercises in part I of the exercise file, explaining what a *while* loop is and solving exercises together with the class. Note that at this point, students who are independent or move quickly may begin to work on their own.
Student Practice Time (40 minutes)	Students do their best to work through the problems in the base file. During this time they can ask the teacher and their peers for help.
Turn-in (5 minutes)	Students have time to turn in their assignments to the schools learning management platform (Google Classroom or similar).
Review (30 minutes)	Teachers and students do the exercises together.

Unit 3 Lesson 6

Unit: Two-Variable Statistics (3)

Grade Level: 7-9th grade

Topic: Plotting using Matplotlib

Time: 1 session *(60 minutes)*

Essential Questions: How can graphs be made and shown in Python? What type of graph should be made to visually determine the strength of a correlation between two variables?	**Required Prior Knowledge:** ● Python variables ● Sort and calculate the length of a Python list ● Import a package in Python
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook files: Questions, Answers	**Vocabulary:** Matplotlib, show, figure, line graph, scatter plot

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Make and show a line plot and a scatter plot using matplotlib	Formal	Through the turned-in notebook at the end of the class
Title a plot using matplotlib and add axis labels	Formal	Through the turned-in notebook at the end of the class
Determine whether to use a scatter or a line graph	Formal, Informal	Through the turned-in notebook at the end of the class, discussion during the warmup

Summary of Activities

Warm-Up (10 minutes)	Students make a scatter plot and a double line graph given two small sets of data by hand.
Warm-up Debrief (5 minutes)	Review the answers to the warm-up. Segway into doing the same thing in Python
Part I: Making Scatter and Line Plots using Matplotlib	As a class, the teacher goes through the exercises in part I of importing matplotlib, making and showing line graphs and scatter plots, and

(15 minutes)	showing the graph. Note that at this point, students who move quickly should be allowed to work on their own, going at their own pace.
Student Practice Time (20 minutes)	Students do their best to work through the problems in the base file. During this time they can ask the teacher and their peers for help.
Turn-in (3 minutes)	Students have time to turn in their assignments to the schools learning management platform (Google Classroom or similar).
Review (7 minutes)	Teachers and students do the exercises together.

Unit 4 Lesson 7

Unit: Functions (4)

Grade Level: 7-9th grade

Topic: Programming versus mathematical functions

Time: 1 session *(60 minutes)*

Essential Questions: What is a programming function? How are they similar or different from programming functions?	**Required Prior Knowledge:** ● How to write mathematical equations in function notation
Materials: ● Teacher File ● Students should have a piece of paper and a writing utensil	**Vocabulary:** Function, parameter, output, return

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Recognize the similarities and differences between mathematical and Python functions.	Informal	Through class discussions and teachers helping students during work time
Read a Python function and determine what are the inputs and outputs	Informal	Through class discussions and teachers helping students during work time

Summary of Activities

Warm-Up (10 minutes)	Given several descriptions of mathematical functions, students should write down the mathematical function in function notation. Teachers can show the function descriptions via the teacher version of the notebook for this lesson.
Warm-up Debrief (5 minutes)	Students are asked to write the answers to the questions on the board. Teacher then reveals the answers from the notebook.
Parts of a Python function (10 minutes)	Show students the functions written as Python functions, discuss the inputs and the outputs.
Introduce more than one argument (5 minutes)	Show several examples of functions with more than one input. Discuss how they work.

Student Practice Time (10 minutes)	Given several Python functions, students write the inputs and describe the outputs and behavior.
Review (10 minutes)	Students explain their answers to the class.

Unit 4 Lesson 9

Unit: Functions (4)

Topic: Calling programming functions and real-world problem practice

Grade Level: 7-9th grade

Time: 2 sessions *(120 minutes)*

Essential Questions: How can functions be used to reuse code? What types of mathematical functions can be represented with programming functions?	**Required Prior Knowledge:** ● How to write a function in Python with multiple parameters
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook file: Questions, Answers	**Vocabulary:** Function, parameter, output, return
Brief Descriptions of Problems Used: ● Calculating the total bill at a restaurant with a programming function ● Perimeter and area of regular polygons	

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Determine what functionality of a piece of code should be written as a function.	Formal	Through the turned-in notebook at the end of the class
Write a programming function given a description of a physical phenomenon	Formal	Through the turned-in notebook at the end of the class

Summary of Activities

Warm-Up (10 minutes)	Students write a function given a description.
Warm-up Debrief (5 minutes)	Review the answer to the warm-up.

Demonstration of repeated code examples (10 minutes)	Given a large code example with many repeated pieces of code, convert the repeated pieces into a function to make it easier to read.
Student Practice Time (40-60 minutes)	Students do their best to work through the problems in the base file. During this time they can ask the teacher and their peers for help.
Turn-in (5 minutes)	Students have time to turn in their assignments to the schools learning management platform (Google Classroom or similar).
Review (30 minutes)	Teachers and students do the exercises together.

Unit 5 Lesson 7

Unit: Introduction to Exponential Functions (5)

Grade Level: 7-9th grade

Topic: Useful and readable graphics

Time: 1 session *(60 minutes)*

Essential Questions: What makes a graph useful? How are graphs used to communicate information?	**Required Prior Knowledge:** ● How to read a graph
Materials: ● Student Handout ● Teacher File ● Students will need note-taking materials ● Internet access or access to this TED Talk	**Vocabulary:** Visualization, Human-Friendly, Legend, Axis Label, Title, Minimalism, Takeaway

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Determine the some of the potential goals of a graphic before beginning to make it	Informal	Class Discussion
List important pieces parts of a graphic for discerning the intended meaning	Informal	Class Discussion

Summary of Activities

Warm-Up (17 minutes)	Watch the TED Talk https://www.youtube.com/watch?v=edAf1jx1wh8&ab_channel=TEDxTalks
Warm-up Debrief (5 minutes)	Discuss the video, highlighting the three main takeaways from the video, that visualizations should be: human-friendly, ruthlessly minimalistic, and highlighting a clear takeaway
Graphic Vocabulary (10 minutes)	Review the parts of a graphic (Title, Legend, Axis Labels, Text)

Graph Analysis (5 minutes)	Given a graphics, we dissect and analyze to see what we like and what we would change
Student Practice (13 minutes)	Given several graphics, students dissect and analyze in groups to see what they like and what they would change
Review (10 minutes)	Review what people believe about the given graphs, emphasizing the fact that there can be multiple ways to tell the story of a dataset

Unit 5 Lesson 9

Unit: Introduction to Exponential Functions (5)

Grade Level: 7-9th grade

Topic: Making useful graphics using Python

Time: 1-2 sessions *(60-120 minutes)*

Essential Questions: What makes a graph useful? How are graphs used to communicate information?	**Required Prior Knowledge:** ● How to use matplotlib to make line plot and a scatter plot
Materials: ● Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● Python notebook files: Questions, Answers	**Vocabulary:** Visualization, Human-Friendly, Legend, Axis Label, Title, Minimalism, Takeaway
Brief Descriptions of Problems Used: ● Car value depreciation ● Car loan value over time	

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Decide between bar, line, and scatter plots for visualizing data	Formal	Through the turned-in notebook at the end of the class
Determine a story that a data set is telling and adjust a graph to communicate that story	Formal	Through the turned-in notebook at the end of the class
Gradually increase the complexity of a script	Formal	Through the turned-in notebook at the end of the class

Summary of Activities

Warm-Up (10 minutes)	Students make a sketch on a piece of paper of the graphic they think would best represent the situation of a car depreciating in value.
Warm-up Debrief (5 minutes)	Discuss the warm up. Students are asked to draw their sketch on the board to share with the class.

Student Practice Time (50-70 minutes)	Students do their best to work through the problems in the base file. During this time they can ask the teacher and their peers for help.
Turn-in (5 minutes)	Students have time to turn in their assignments to the schools learning management platform (Google Classroom or similar).
Review (30-40 minutes)	Teachers and students do the exercises together.

Unit 6 Lesson 6 (1-2 sessions analog and 1 digital)

Unit: Introduction to Quadratic Functions and Their Forms (6)

Topic: Interpreting intercepts and vertices of graphs in the context of free-falling objects and other real-world applications

Grade Level: 7-9th grade

Time: 2-3 sessions *(60-180 minutes)*

Essential Questions: What types of real-world situations can be modeled by quadratic equations?	**Required Prior Knowledge:** ● How to use matplotlib to make line plot and a scatter plot
Materials: ● (Days 1-2) Practice Problems: Student Handout, Answers ● (Day 3) Computers for each student with access to a web browser and Google CoLab not blocked on the local network ● (Day 3) Python notebook files: Questions, Answers	**Vocabulary:** Title, axis, axis label, legend
Brief Descriptions of Problems Used: ● The motion of a dropped and a thrown ball: *height-time*, *horizontal distance-time, height-horizontal distance* graph analysis ● Air pollutant ($PM_{2.5}$) dispersion from a wildfire: analyzing safe distances from the fire based on the Air Quality Index versus time graph ● Animal Population: Recovery of a a dying ant farm ● Optics: Finding the focal point of a parabolic mirror for using lighting the Olympic torch ● Area: Determining the sizes of kiddie pools based on the surface area and the price	

Objectives and Corresponding Assessments

Students will be better able to...	**Assessment Type**	**Assessment**
Appropriately reason the meaning of x-intercept, y-intercept, and vertex of a quadratic function that models a physical phenomenon	Formal	Through the turned-in notebook at the end of the class

Intuitively explain the existence of small deviations from the ideal in the data	Formal	Through the turned-in notebook at the end of the class

Summary of Activities: Days 1-2

Warm-Up (10 minutes)	Given a graph of a ball falling versus ***time***, students should attempt to point out the following points: *when the ball is dropped, when the ball hits the ground, when the ball is moving the fastest, when the ball is moving the slowest*
Warm-up Debrief (5 minutes)	Review the answer to the warm-up. Students should explain their reasoning to the class and be encouraged to go to the front of the class to point to the parts of the graph that they're referring to.
Ball Being Thrown (x versus y) (10 minutes)	Given a graph of a ball falling versus ***distance***, students should attempt to point out the following points: *how far the ball traveled before hitting the ground, how far the ball was when it began to roll, how far the ball was thrown, how far away the ball was when it reached its highest point*
Ball Being Thrown (x versus y) Debrief (10 minutes)	Review the answer to the Ball Being Thrown. Students should explain their reasoning to the class and be encouraged to go to the front of the class to point to the parts of the graph that they're referring to. Special attention should be paid to differentiate *distance* (this graph) and *time* (the warm-up) as well as using the terms "x-axis" and "y-axis". In this graph, we have no information about *time*.
Student Practice Time: (50 minutes)	Students work through the problem set in groups, asking the teacher for help as necessary
Turn-in (5 minutes)	Students have time to turn in their assignments to the teacher
Review (30 minutes)	Teachers and students do the problem set together.

Summary of Activities: Day 3

Warm-Up (10 minutes)	Given a time versus height graph of a thrown ball and some instruction, students attempt to write equation for the height of the ball as a function of time
Warm-up Debrief (5 minutes)	Review the answer to the warm-up.

Student Practice Time (20 minutes)	Students do their best to work through the problems in the base file. During this time they can ask the teacher and their peers for help.
Turn-in (5 minutes)	Students have time to turn in their assignments to the schools learning management platform (Google Classroom or similar).
Review (10 minutes)	Teachers and students do the exercises together.

4.4 Discussion

While this curriculum has not yet been implemented in any institute, there are a variety of challenges that the author anticipates.

Importantly, and perhaps the most challenging for a school ready to implement such a curriculum, the pacing of the course may be difficult for teachers to implement for the first time. This curriculum design in essence injects what could be another half-year course into the already crowded math curriculum. However, as discussed in the introduction, the ubiquity of programming makes it an essential skill, worth prioritizing over other thoughtfully chosen materials to make sure students are able to apply their knowledge. This means that in order to appropriately pace this course, some skills that were previously developed in depth may need to be less-deeply explored or even left behind. For example, while the Illustrative Mathematics curriculum includes students learning to use spreadsheets through programs like Excel in their first unit (Illustrative Mathematics, 2019), *Computational Algebra I* omits this skill because all the things that can be calculated in a spreadsheet can also be calculated with Python. Furthermore, the author postulates that after having learned to program, a student would be able to directly apply their programming skills to spreadsheets if an employer or teacher were to oblige the student to use them. There are other examples where manual computation is deprioritized in favor of the programmatic application. For example, significantly less time is spent in *Computational Algebra I* on methods like completing the square and factoring in favor of interpreting the physical meaning of the key features of the forms of quadratic equations as well as computing and visualizing them programmatically. Similarly, significantly less time is spent graphing functions and inequalities by hand in favor of learning to effectively communicate data through graphs generated with Python.

The author anticipates that proposing such changes to the curriculum could be met with reactionary attitudes. Some teachers, parents, administrators, or community members may say programming falls out of the scope of math class and that the things that have been taught in math classes until now are more important. Such concerns are important to keep in mind but should not be cause for wholesale rejection of changes to the curriculum. When calculators were first introduced into the classroom, many, including writer Deedee Pendelton at Science News were worried that calculators would “leave students unable to do simple math on paper” (Pendelton, 1975). While some may still be disgruntled by the fact that today’s students are not as nimble in their pen-and-paper calculations, one would be hard pressed to make the argument that students should not learn how to use calculators in math class. Acting on such a position would leave students unable to use one of the most essential mathematical tools of this era. As each new generation is faced with changing technologies, it is the job of the teachers to adapt. There is a reason schools no longer teach slide rules in favor of graphing calculators and it is the same reason that soon programming will displace these very same graphing calculators: schools best serve their students by teaching them to navigate the world and the technology of the

student's time, not the time of the teacher. So, concerns about leaving behind previously learned skills being forgotten should not be ignored, but rather used in the thoughtful decision making each teacher performs to determine what they want to prioritize in their classrooms and what could be sacrificed for time to explore the new and important tools of the modern age.

In a similar vein, at the time of the writing of this curriculum, one of the bleeding-edge technologies shaking the world of education are Large Language Models (LLMs) like ChatGPT. Reactionary attitudes towards such powerful technology must also be moderated, and future iterations of this curriculum will have to take into account this new tool. Perhaps in the future, instead of asking students to write entire programs by themselves, students will be taught how to take the questions given and form them into effective prompts for ChatGPT. This has fallen out of scope of the writing of this curriculum, but is something that teachers should keep in mind if they decide to implement this course. There are some questions that increasingly students may be able to copy and paste into an LLM and get a decent answer. Teaching expectations, styles, and evaluation must reflect this reality.

Another set of challenges may arise from making sure that the students have consistent access to the materials needed for their programming assignments. For example, this course might work best in schools where each student has their own school device, or the class has consistent access to a computer lab. In addition to physical resources, programming is heavily dependent on online resources for downloading necessary programs to run student code or for looking up how to perform specific tasks. It is essential that if a school has a firewall, the teacher takes special care to make sure that all the sites that the students need are unblocked or white-listed on the school network so students can freely access the information and programs they need. Finally, access to the appropriate human resources is also necessary to be able to offer this course. Administrations would need to find math teachers that are also proficient in Python or some other programming language, which may initially pose a challenge until this qualification becomes the new norm.

The combination of the need for rigorous pacing and dependable access to materials means that inevitably schools with more resources will have an easier time implementing this curriculum. These are schools that are also likely to have higher test scores (Mensah et al., 2013). However, the relative difficulty must not be seen as an insurmountable obstacle in the implementation of this course. As mentioned by the Burning Glass study, coding jobs pay on average higher wages, as much as $20,000 per year more ("Beyond Point and Click," 2016), meaning giving students of all socioeconomic backgrounds access to this type of education is important for enabling upwards socioeconomic mobility. Furthermore, when faced with challenges like low student achievement, it is important not to turn to the programming components of the course as the culprits or the parts to deprioritize. At the end of their studies, the student's programming skills may be much more relevant than the math skills they learned when searching for a living-wage job.

Given these challenges, implementation and trial and error are the one true way to improve the efficacy of this course and subsequently the math education of students. This curriculum design has been written in anticipation of the author's first year as a mathematics teacher in the state of Maryland. As such, while the curriculum has not yet been implemented in any institute, it is a way the author could teach courses in the immediate future. Further, while Maryland was chosen because it is the home state of the author, it was also chosen because it is one of 41 states currently using the Common Core Standards (*Common Core States*, 2023). In conforming with these standards, the author increases the potential implementability in other states, expanding the impact of the work.

5. Conclusions

While this curriculum has not been implemented yet, the author sees it as a useful framework for a curriculum that could be implemented in the state of Maryland or in other states using the Common Core standards. The author acknowledges that it will be difficult to appropriately prioritize to maintain pacing at first. However, the value that a course which incorporates programming into mathematics adds is undeniable for students navigating today's digital world. The calculator too was once feared as a radical change and potentially the end of mathematics education. Such hysteria must be overcome in order to prepare students for more comfortable, happy, and lucrative futures. Future work of the author includes implementing it in their own classrooms and continuing to adapt it as burgeoning technologies change the landscape of a student's necessary capacities.

6. Appendices

Appendix 1

All of the most up-to-date original labs, math quizzes, and programming quizzes can be found in the following github repository. An outline of and specific links to the specific materials are included below. Furthermore, in Appendix 2, there are samples of each type of material.

https://github.com/TonyTerrasa/computational-algebra-1

These materials include:

- 7 Math Quizzes and Solutions
    - Math Mini Quiz 1: Questions, Solution
    - Math Mini Quiz 2: Questions, Solution
    - Math Mini Quiz 3: Questions, Solution
    - Math Mini Quiz 4: Questions, Solution
    - Math Mini Quiz 5: Questions, Solution
    - Math Mini Quiz 6: Questions, Solution
    - Math Mini Quiz 7: Questions, Solution
- 7 Programming Quizzes and Solutions
    - Programming Mini Quiz 1: Questions, Solution
    - Programming Mini Quiz 2: Questions, Solution
    - Programming Mini Quiz 3: Questions, Solution
    - Programming Mini Quiz 4: Questions, Solution
    - Programming Mini Quiz 5: Questions, Solution
    - Programming Mini Quiz 6: Questions, Solution
    - Programming Mini Quiz 7: Questions, Solution
- 7 Base Files for Math and Programming Labs that will walk students through the application of the programming and math concepts being covered
    - Math and Programming Lab 1
    - Math and Programming Lab 2
    - Math and Programming Lab 3
    - Math and Programming Lab 4
    - Math and Programming Lab 5
    - Math and Programming Lab 6
    - Math and Programming Lab 7

- The problems, solutions and teacher materials for 8 lessons. Note that two more are included but are for class sessions with quizzes.
    - Unit 1 Lesson 7: Questions, Answers
    - Unit 2 Lesson 11: Questions, Answers
    - Unit 3 Lesson 6: Questions, Answers
    - Unit 4 Lesson 7: Teacher File
    - Unit 4 Lesson 9: Questions, Answers
    - Unit 5 Lesson 7: Student Handout, Teacher File
    - Unit 5 Lesson 9: Questions, Answers
    - Unit 6 Lesson 9 Days 1-2: Student Handout, Answers
    - Unit 6 Lesson 9 Day 3: Questions, Answers

Appendix 2

Samples of each of the types of the materials prepared for this curriculum design can be found below.

Appendix 2.1 Math Quiz Sample: Math Mini Quiz 1

Appendix 2.1.1 Math Mini Quiz 1 Questions

Name: _____ Date: _____

Class: _____ Teacher: _____

Math Mini Quiz 1

This Mini Quiz, we're going to explore the math concepts that you've learned so far in this unit. This assignment should take you about **15 minutes**.

Part 1 Freediving

Free diving is a sport where divers go under the water without the aid of an oxygen tank, like would be the case with scuba diving. To do this, freedivers have to hold their breaths for long periods of time with some of the longest free dives ever lasting over four minutes! Below is a list of numbers representing the highest **number of minutes** dived by the top freedivers[1].

3.6, 3.0, 3.1, 3.5, 4.5, 2.25, 2.1, 2.0, 1.3

1) Find the **average** of these top freediving times.

2) Find the **median** of these top freediving times.

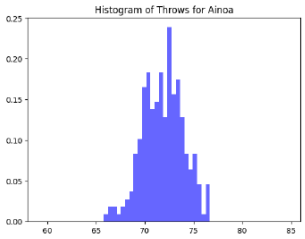
3) Which of these two would you prefer to use *to describe the center* of this data? Why?

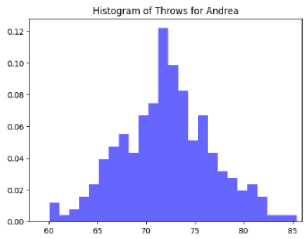
[1] Image from: https://www.youtube.com/watch?v=vWjZt-S3B54&ab_channel=LexieLimitless

(yes, there's a back, don't forget it)

Part 2 Graph Analysis

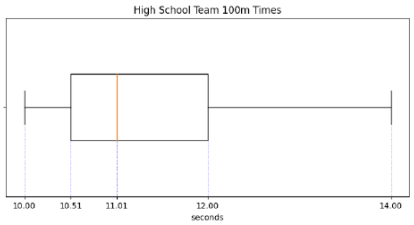
Below are the histograms of the distance thrown by two different javelin throwers[2].





4) Which of these two has the greater standard deviation? What does that tell you about the way that person throws?

Below is a box and whisker plot of the 100m times for a highschool track team[3].



5) Can we find the mean, median, or both from this plot? How do you know? What is it/what are they?

6) Name two ranges in which 75% of the data lies

The end, good job :)

---

[2] Image from: https://en.wikipedia.org/wiki/Javelin_throw
[3] Image from: https://blog.orthoindy.com/2020/06/24/what-are-the-most-common-injuries-in-track-and-field/

Appendix 2.1.2 Math Mini Quiz 1 Answers

Name: _____ Date: _____

Class: _____ Teacher: _____

Math Mini Quiz 1

This Mini Quiz, we're going to explore the math concepts that you've learned so far in this unit. This assignment should take you about **15 minutes**.

Part 1 Freediving

Free diving is a sport where divers go under the water without the aid of an oxygen tank, like would be the case with scuba diving. To do this, freedivers have to hold their breaths for long periods of time with some of the longest free dives ever lasting over four minutes! Below is a list of numbers representing the highest **number of minutes** dived by the top freedivers[1].

3.6, 3.0, 3.1, 3.5, 4.5, 2.9, 2.8, 2.9, 2.7

1) Find the **average** of these top freediving times.

Answer: 3.22

3.6 + 3.0 + 3.1 + 3.5 + 4.5 + 2.9 + 2.8 + 2.9 + 2.7 = 29

25.35 / 9 = 3.22

2) Find the **median** of these top freediving times.

Answer: 3.0

First put them in order: 2.7, 2.8, 2.9, 2.9, 3.0, 3.1, 3.5, 3.6, 4.5

Find the middle: ~~2.7~~, ~~2.8~~, ~~2.9~~, ~~2.9~~, 3.0, ~~3.1~~, ~~3.5~~, ~~3.6~~, ~~4.5~~

3) Which of these two would you prefer to use *to describe the center* of this data? Why?

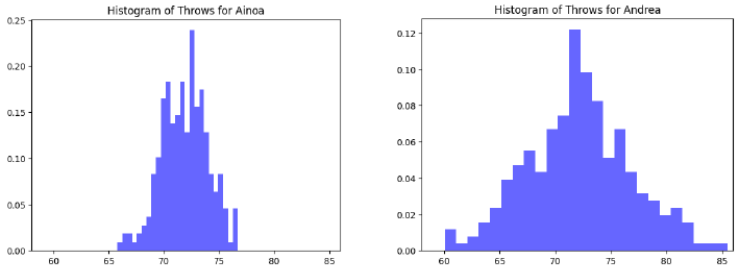
The median in this case will best describe the center. This is because we have at least one outlier in the data, the point of 4.5 minutes. This high value will throw off the mean, but the median will be relatively unaffected.

(yes, there's a back, don't forget it)

[1] Image from: https://www.youtube.com/watch?v=vWjZt-S3B54&ab_channel=LexieLimitless

Part 2 Graph Analysis

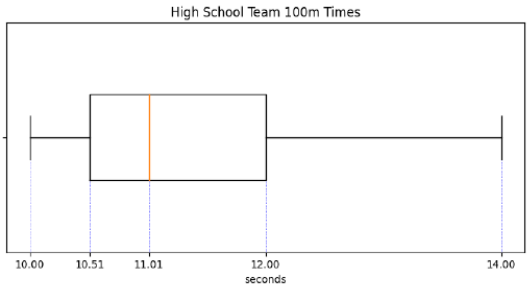
Below are the histograms of the distance thrown by two different javelin throwers[2].



4) Which of these two has the greater standard deviation? What does that tell you about the way that person throws?

Andrea's throws have a higher standard deviation. This means she has a higher variety in the distances she throws. That is to say, while it's possible for Andrea to throw her max higher than Ainoa, it's also possible for Andrea to throw lower than Ainoa's min. Ainoa throws more consistently within a smaller range.

Below is a box and whisker plot of the 100m times for a highschool track team[3].



5) Can we find the mean, median, or both from this plot? How do you know? What is it/what are they?

We can find the median, but not the mean. This is because a box and whisker plot shows us the quartiles, the middle of which would be the median not the mean. The median is 11.01 seconds.

6) Name two ranges in which 75% of the data lies

10.00 - 12.00, 10.51 - 14.00 ---- These are found by taking minimum up to the third quartile boundary, and the first quartile boundary up to the maximum

The end, good job :)

---

[2] Image from: https://en.wikipedia.org/wiki/Javelin_throw

[3] Image from: https://blog.orthoindy.com/2020/06/24/what-are-the-most-common-injuries-in-track-and-field/

Appendix 2.2 Programming Quiz Sample: Programming Mini Quiz 2

Appendix 2.2.1 Programming Mini Quiz 2 Questions (Python notebook)

Name and Date

Name:

Date:

Programming Mini Quiz Number 2

In this mini quiz you will demonstrate your understanding of the most important programming skills we've learned so far in this unit.

Please remember that outside resources are not allowed while you write your code.

This assignment should take **25 minutes**.

The Fibonacci Sequence

The Fibonacci sequence is an ordered set of numbers in which each number is equal to the sum of the previous two. The first thriteen numbers are:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```

It is a sequence of numbers which beautifully shows up in a variety of mathematical and natural ways. For example, in the number of florets in the center of a sunflower.

Image from Ask Nature[1]

Don't worrv. let's break it down into more detail.

How it works

The first two numbers are: `0` and `1`. How do we get to the third number? Well we add the first two! So: `0 + 1 = 1` and notice that `1` is the third number in our list!

So, how do we calculate the fourth number? Well we add the **second** and **third** numbers in the sequence. The second number is `1` and the third number is `1`, so we get `1 + 1 = 2`. And look, the number `2` is the fourth number!

One more example. How would we calculate the 13th number? Well, we add the 11th and the 12th numbers. The 11th number is `55` and the 12th is `89`. So the thirteenth number should be: `55 + 89 = 144`. This is exactly how we calculated the last of the numbers given above!

Let's Make the Fibonocci Sequence

Variables

To make the fibonocci sequence, we're going to need three variables (though you can technically do it in with two!).

Make the following variables:

Variable	Value
last_number	1
second_to_last	0
current_number	-1

In [ ]:

One step of the fibbonocci sequence

Based on the rules we have above, our `current_number` shouldn't be `-1` though, it should be the sum of `last_numer` and `second_to_last`!

Set `current_number` correctly below as the sum of `last_numer` and `second_to_last`

In [ ]:

Looping the Fibbonocci sequence

When we think of what needs to happen in each step of the Fibonocci sequence, it's easy to think that we're only adding two numbers, but in reality we are doing more than that. When we calculate a new number we also move along to the next step in the sequence. Why is this important? Well, when we move to the next step in the sequence, the `last_number` and `second_to_last` must also change! If we calculate `current_number` as `last_number + second_to_last` without changing `last_number` or `second_to_last`, the `current_number` will never change. See the example below.

In [ ]:
```
a = 0
b = 1
c = -1
for i in range(13):
  c = a + b
  print("C is currently: ", c)
```

```
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
```

What do we need to add? **It's your job to figure ir out in the next step!**

Implementing the Fibonocci Sequence

Below, use your variables declared above as well as a `for` or a `while` loop to print the first 13 numbers of the Fibonnoci sequence. As you print each number, be sure to include index, for example:

```
print("Number", 13, "is:", 144)
```

In [ ]:

References

[1]Sunflowers' Fibonacci Secrets (https://asknature.org/strategy/fibonacci-sequence-optimizes-packing/)

In [ ]:

Appendix 2.2.2 Programming Mini Quiz 2 Answers (Python notebook)

Name and Date

Name: Stew Dent

Date: Spring 2023

Programming Mini Quiz Number 2

In this mini quiz you will demonstrate your understanding of the most important programming skills we've learned so far in this unit.

Please remember that outside resources are not allowed while you write your code.

This assignment should take **25 minutes**.

The Fibonacci Sequence

The Fibonacci sequence is an ordered set of numbers in which each number is equal to the sum of the previous two. The first thriteen numbers are:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```

It is a sequence of numbers which beautifully shows up in a variety of mathematical and natural ways. For example, in the number of florets in the center of a sunflower.

Image from Ask Nature[1]

Don't worry, let's break it down into more detail.

How it works

The first two numbers are: `0` and `1`. How do we get to the third number? Well we add the first two! So: `0 + 1 = 1` and notice that `1` is the third number in our list!

So, how do we calculate the fourth number? Well we add the **second** and **third** numbers in the sequence. The second number is `1` and the third number is `1`, so we get `1 + 1 = 2`. And look, the number `2` is the fourth number!

Let's Make the Fibonocci Sequence

Variables

To make the fibonocci sequence, we're going to need three variables (though you can technically do it in with two!).

Make the following variables:

Variable	Value
last_number	1
second_to_last	0
current_number	-1

In [1]:
```
last_number = 1
second_to_last = 0
current_number = -1
```

One step of the fibbonocci sequence

Based on the rules we have above, our `current_number` shouldn't be `-1` though, it should be the sum of `last_numer` and `second_to_last` !

Set `current_number` correctly below as the sum of `last_numer` and `second_to_last`

In [2]:
```
current_number = last_number + second_to_last
```

Looping the Fibbonocci sequence

When we think of what needs to happen in each step of the Fibonocci sequence, it's easy to think that we're only adding two numbers, but in reality we are doing more than that. When we calculate a new number we also move along to the next step in the sequence. Why is this important? Well, when we move to the next step in the sequence, the `last_number` and `second_to_last` must also change! If we calculate `current_number` as `last_number + second_to_last` without changing `last_number` or `second_to_last` , the `current_number` will never change. See the example below.

In [3]:
```
a = 0
b = 1
c = -1
for i in range(13):
  c = a + b
  print("C is currently: ", c)
```

```
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
C is currently:  1
```

What do we need to add? **It's your job to figure ir out in the next step!**

Implementing the Fibonocci Sequence

Below, use your variables declared above as well as a `for` or a `while` loop to print the first 13 numbers of the Fibonnoci sequence. As you print each number, be sure to include index, for example:

```
print("Number", 13, "is:", 144)
```

In [7]:
```
last_number = 1
second_to_last = 0
current_number = -1

# First we print out the first two numbers which we already have
print("Number", 1, "is:", second_to_last)
print("Number", 2, "is:", last_number)

# Use this to keep track of what number we are in the sequence
number = 3

while number <= 13:

  # Calculate the current number
  current_number = last_number + second_to_last
  print("Number", number, "is:", current_number)

  # prepare for the NEXT iteration by shifting
  # second_to_last to the previous last_number
  # and last_number to the current_number
  # the order of these statements is important
  # putting them out of order will result in
  # incorrect answers
  second_to_last = last_number
  last_number = current_number
  number += 1
```

```
Number 1 is: 0
Number 2 is: 1
Number 3 is: 1
Number 4 is: 2
Number 5 is: 3
Number 6 is: 5
Number 7 is: 8
Number 8 is: 13
Number 9 is: 21
Number 10 is: 34
Number 11 is: 55
Number 12 is: 89
Number 13 is: 144
```

Appendix 2.3 Programming and Math Lab Sample: Lab 2

Name:

Date:

Lab 2

In this lab, we're going to be used the programming and the math skills we've been using in class to determine where we want to live!

User Input

First, let's review how to take user input. To recieve use input, you use the `input` function. For example, see the simple program below.

In [ ]:
```
users_name = input("What is your name? ")
print("Hello", users_name)
```

```
What is your name? Kelsey
Hello Kelsey
```

But what if we want to take in a number? Let's try:

In [ ]:
```
current_year = 2023
birth_year = input("What year were you born? ")
year_difference = current_year - birth_year
print("You were born", year_difference, "years ago")
```

```
What year were you born? 1999

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-8cb52b47481e> in <cell line: 3>()
      1 current_year = 2023
      2 birth_year = input("What year were you born? ")
----> 3 year_difference = current_year - birth_year
      4 print("You were born", year_difference, "years ago")

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

What happened? Let's look at the error message. First it tells us the type of error. That's this part.

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)
```

Then it tells us where:

```
<ipython-input-6-8cb52b47481e> in <cell line: 3>()
      1 current_year = 2023
      2 birth_year = input("What year were you born? ")
----> 3 year_difference = current_year - birth_year
      4 print("You were born", year_difference, "years ago")
```

And finally it gives us more detail:

```
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

So first, we know it's a `TypeError` that means that we tried to do something that was not allowed by the *type* of our variables. Image dividing a string by three:

```
"hello" / 3
```

What does it mean to divide the word `hello` by three? This is not a valid operation. This would also give a `TypeError`.

If we look at **where**, it seems to be specifically in the line where we calculate the year difference (`year_difference = current_year - birth_year`).

Our detailed message says `unsupported operand type(s) for -: 'int' and 'str'`. That means it doesn't know how to subtract an integer minus a string. But what happened? We gave the `input` a number right? Yes, but `input` always reads as a string. So, when we take the user input, we need to tell it to convert to an integer like this:

In [ ]:
```
fav_number = input("What is your favorite number? ")
print("fav_number is a", type(fav_number))

# convert to an integer so we can use it in math
fav_number = int(fav_number)
print("fav_number is a", type(fav_number))

# now we can do math
big_fav = fav_number * 100
print("your favorite number is", fav_number)
print("your BIG favorite number is", big_fav)
```

```
What is your favorite number? 5
fav_number is a <class 'str'>
fav_number is a <class 'int'>
your favorite number is 5
your BIG favorite number is 500
```

1. Below, fix the code so that it does not give an error and computes succesfully.

In [ ]:
```
# fix this code!
current_year = 2023
birth_year = input("What year were you born? ")
year_difference = current_year - birth_year
print("You were born", year_difference, "years ago")
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-11-ff20ab8d6039> in <cell line: 3>()
      1 # fix this code!
      2 current_year = 2023
----> 3 birth_year = input("What year were you born? ")
      4 year_difference = current_year - birth_year
      5 print("You were born", year_difference, "years ago")

/usr/local/lib/python3.9/dist-packages/ipykernel/kernelbase.py in raw_input(sel
f, prompt)
    849                 "raw_input was called, but this frontend does not suppor
t input requests."
    850             )
--> 851         return self._input_request(str(prompt),
    852             self._parent_ident,
    853             self._parent_header,

/usr/local/lib/python3.9/dist-packages/ipykernel/kernelbase.py in _input_request
(self, prompt, ident, parent, password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from None
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

Your Turn

Now, it's going to be your turn to make a small script that takes user input, calculates a value based on that input. As you do you analysis write your answers to these questions.

- What is the equation that you chose to use? Why do you find that one interesting?
- Try a variety of inputs. Did it give the answers that you expected?
- Write at least 1 equivalent equations to the one that chose. What does it tell you about the relationship between the variables involved?

It can be anything you want, but some ideas could be:

- Convert Between Fahrenheit and Celcius (https://www.rapidtables.com/convert/temperature/celsius-to-fahrenheit.html)
- Convert Between Dollars and Euros (or other currency) (https://www.x-rates.com/table/?from=USD&amount=1)
- Baking Conversions (liters and cups, etc.) (https://annaolson.ca/baking-conversions/)
- Calculating Tax and Tip at a Restaurant (https://www.wikihow.com/Tip-Your-Server-at-a-Restaurant)

In [ ]:
```
# Your Code Goes Here
```

Your Response

Double click here to add you response. Press Shift+Enter when done (just like running a code cell).

Appendix 2.4 Lesson Sample: Unit 2 Lesson 11

Appendix 2.4.1 Unit 2 Lesson 11 Questions (Python notebook)

Name:

Date:

Warm-up

Consider the following example. How would you write the program to solve this problem?
We want to ask the user for a password, but make sure that the password is given is valid. The password must be at least 8 characters. If they don't give a valid password, they should be reprompted. We want to user to be able to try an infinite number of times until they give a valid password.

Part 1: While Loops

While loops, much like for loops can be used to execute code repeatedly. Check out the explanation and example from W3Schools[1]

> With the `while` loop we can execute a set of statements as long as a condition is true.
>
> Print i as long as i is less than 6:
>
> ```
> i = 1
> while i < 6:
>   print(i)
>   i += 1
> ```
>
> Note: remember to increment i, or else the loop will continue forever.
>
> The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

In this case the **condition** is `i < 6` . So, as long as `i < 6` is `True` , the coded will run. As soon as `i < 6` becomes `False` , it will stop and move on with the code. Now, run the code in the example to see the output

In [ ]:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

1. How many times does the loop execute? Write your answer below

Your Response

2. Now, what happens if we add a `print(i)` to the end? Think about your answer before running the code.

In [ ]:
```
i = 1
while i < 6:
  print(i)
  i += 1

# here we are, after the while loop
print(i)
```

Your Answer Here

Type *Markdown* and LaTeX: α^2

3. Now, write a small script using a `while` loop to count down from 10. Start with a variable `i` at `10` and print each value of `i` as it counts down to `1`. The last number to be printed should be `1`.

In [ ]:
```
# Your Code Here
```

4. Now, declare a variable `a` with value `3` and another variable `b` with value `2`. Using a `while` loop, determine how many times you can add `a` to `b` before `b` reaches `100`.

HINT: you may need to use another variable, `num_additions`

In [ ]:
```
# Your Code Here
```

Part 2: Problems

Basketball Stars

One common use of coding is in the word of sports statistics. Here we have some data[2] about two NBA starts: Steph Curry and LeBron James. Below, we have the **number of points** each played scored in each game of their first season (2003 for LeBron and 2009 for Steph).

This data can be accessed similar to the way we accessed the daeta in lab 1:

```
# prints the number of points LeBron scored
# in his FIRST game
print(LeBron[0])

# prints the number of points LeBron scored
# in his SECOND game
print(LeBron[1])

# prints the number of points LeBron scored
# in his 50th game
print(LeBron[49])
```

In [ ]:
```
# data, MAKE SURE TO RUN THIS CELL
LeBron_first_season = [25, 21, 8, 7, 23, 17, 17, 18, 10, 22, 14, 28, 19, 15, 15, 6, 3
Steph_first_season = [38, 23, 24, 14, 27, 25, 33, 36, 34, 21, 31, 34, 40, 32, 34, 25,
```

1. Using a `while` loop, determine how many games it took LeBron to reach `500` total points.

In [ ]:
```
# Your Code Here
```

2. Using a `while` loop, determine how many games it took Steph to reach `500` total points.

In [ ]:
```
# Your Code Here
```

3. Who was reached 500 faster? Can you calculate the average points per game scored for each person up until the game where they reached 500 points?

In [ ]:
```
# Your Code Here
```

Your Response

4. The 2022-2023 NBA season recieved a lot of attention for the high number of games were a single player scored 40 or more points. This is something both LeBron and Steph have been doing for a long time now. Using `while` loops, find the number of the **first** time each of these players scored at least 40 points in one game.

In [ ]:
```
# Your Code Here
```

5. How many 40+ point games did each player have in their first season? You may use a `while` or a `for` loop for this, your choice.

In [ ]:
```
# Your Code Here
```

6. Who would you say had the stronger first season **in terms of their scoring**? Why? You are welcome to do more calculations to justify your answer.

In [ ]:
```
# Your Code Here
```

Your Response Here

Type *Markdown* and LaTeX: α^2

Prime Numbers

Recall that prime numbers are really special in that they have factors other than 1 and themselves.

We're to write code to decide if a number is prime.

How can we tell if a number is prime?

The process for determining if number `n` is prime will be something like this:

1. Check if `n` is divisible by `2`, if it is, then `n` isn't prime
2. Check if `n` is divisible by `3`, if it is, then `n` isn't prime
3. Check if `n` is divisible by `4`, if it is, then `n` isn't prime

   ...

   Check if `n` ia divisible by `int(n/2)`, if it ISNT, then `n` IS prime

But how can we tell if a number number is divisible by another? Well, remember that in coding we have **modulo** division which finds the remainder of the division of two numbers. Note that if a number is divisible by another, than the remainder of the division of these two numbers will be 0.

1. Complete the following code to determine if `n` is divisible by `x`.

In [ ]:
```
x = 13
n = 5356
if :
  print(n, "is divisible by", x)
else:
  print(n, "is NOT divisible by", x)
```

2. Write a script to determine if the number `n` is prime.

In [ ]:
```
n = 313
# Your Code Here
```

3. Calculate the number of prime numbers under 100?

In [ ]:
```
# Your Code Here
```

Password Validation

Now, we return to the problem from the beginning of class. Ask the user for a password. The password must be at least 8 characters long. If the password is not at least 8 characters long, reject the password and ask again.

In [ ]:
```
# Your Code Here
```

Bonus: Palindrome Numbers

Imagine you're really interested in palidrome numbers, that is numbers which are the same forwards and backwards, for example `303` , `1221` , or `1246421` . To do this, we'd need to first think about how you would reverse the digits of a number. One way would be to turn it into a string and reverse the string:

```
a = 12454132
a_str = str(a)
# reverses the string
a_str = a_str[::-1]
print(a_str)
```

However, this is a little bit confusing and the conversion between integer and string isn't terribly efficient. There's a way to do it with just math and loops. Can you figure it out?

In [ ]:
```
a = 12454132
# Your code here
```

```
23145421
```

Now, write a script which determines if a number is a palindrome.

In [ ]:
```
# Your code here
```

References

[1]W3 Schools `while` Loop Page (https://www.w3schools.com/python/python_while_loops.asp)

[2]The NBA has a way for accessing states using Python, called the nba_api (https://pypi.org/project/nba-api/)

Appendix 2.4.2 Unit 2 Lesson 11 Answers (Python notebook)

Teacher version

Warm-up

Consider the following example. How would you write the program to solve this problem?
We want to ask the user for a password, but make sure that the password is given is valid. The password must be at least 8 characters. If they don't give a valid password, they should be reprompted. We want to user to be able to try an infinite number of times until they give a valid password.

Part 1: While Loops

While loops, much like for loops can be used to execute code repeatedly. Check out the explanation and example from W3Schools[1]

> With the `while` loop we can execute a set of statements as long as a condition is true.
>
> Print i as long as i is less than 6:
>
> ```
> i = 1
> while i < 6:
>   print(i)
>   i += 1
> ```
>
> Note: remember to increment i, or else the loop will continue forever.
>
> The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

In this case the **condition** is `i < 6` . So, as long as `i < 6` is `True` , the coded will run. As soon as `i < 6` becomes `False` , it will stop and move on with the code. Now, run the code in the example to see the output

In [1]:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

1. How many times does the loop execute? Write your answer below

Your Response

The **condition** is evaluated 6 times, and the loop is **executed** 5 times.

1. `i` is 1, `i < 6` is `True` , `1` is printed, `i` becomes `2`
2. `i` is 2, `i < 6` is `True` , `2` is printed, `i` becomes `3`

3. `i` is 3, `i < 6` is `True`, `3` is printed, `i` becomes `4`
4. `i` is 4, `i < 6` is `True`, `4` is printed, `i` becomes `5`
5. `i` is 5, `i < 6` is `True`, `5` is printed, `i` becomes `6`
6. `i` is 6, `i < 6` is `False` (`6` is not less than `6`), the inside of the loop is not executed, nothing is printed and the program continues

2. Now, what happens if we add a `print(i)` to the end? Think about your answer before running the code.

In [2]:
```
i = 1
while i < 6:
  print(i)
  i += 1

# here we are, after the while loop
print(i)
```

```
1
2
3
4
5
6
```

Your Answer Here

6 will be printed at the end

3. Now, write a small script using a `while` loop to count down from 10. Start with a variable `i` at `10` and print each value of `i` as it counts down to `1`. The last number to be printed should be `1`.

In [3]:
```
# Your Code Here
i = 10
while i > 0:
  print(i)
  i -= 1
```

```
10
9
8
7
6
5
4
3
2
1
```

4. Now, declare a variable `a` with value `3` and another variable `b` with value `2`. Using a `while` loop, determine how many times you can add `a` to `b` before `b` reaches `100`.

HINT: you may need to use another variable, `num_additions`

In [4]:
```
# Your Code Here
a = 4
b = 3
num_additions = 0
while b < 100:
  b += a
  num_additions += 1

print("b reached 100 after:", num_additions, "additions of a")
```

```
b reached 100 after: 25 additions of a
```

Part 2: Problems

Basketball Stars

One common use of coding is in the word of sports statistics. Here we have some data[2] about two NBA starts: Steph Curry and LeBron James. Below, we have the **number of points** each played scored in each game of their first season (2003 for LeBron and 2009 for Steph).

This data can be accessed similar to the way we accessed the daeta in lab 1:

```
# prints the number of points LeBron scored
# in his FIRST game
print(LeBron[0])

# prints the number of points LeBron scored
# in his SECOND game
print(LeBron[1])

# prints the number of points LeBron scored
# in his 50th game
print(LeBron[49])
```

In [5]:
```
# data, MAKE SURE TO RUN THIS CELL
LeBron_first_season = [25, 21, 8, 7, 23, 17, 17, 18, 10, 22, 14, 28, 19, 15, 15, 6, 3
Steph_first_season = [38, 23, 24, 14, 27, 25, 33, 36, 34, 21, 31, 34, 40, 32, 34, 25,
```

1. Using a `while` loop, determine how many games it took LeBron to reach `500` total points.

In [6]:
```
# Your Code Here
LeBron_total_points = 0
LeBron_game = 0
while LeBron_total_points < 500:
  LeBron_total_points += LeBron_first_season[LeBron_game]
  LeBron_game += 1

# important: why LeBron_game and not LeBron_game + 1?
print("LeBron reached 300 points in his game number", LeBron_game, "after which he h
```

```
LeBron reached 300 points in his game number 27 after which he had 519
```

2. Using a `while` loop, determine how many games it took Steph to reach `500` total points.

In [7]:
```
# Your Code Here
# How many did it take for LeBron to reach 500?
Steph_total_points = 0
Steph_game = 0
while Steph_total_points < 500:
  Steph_total_points += Steph_first_season[Steph_game]
  Steph_game += 1

# important: why Steph_game and not Steph_game + 1?
print("Steph reached 300 points in his game number", Steph_game, "after which he hac
```

```
Steph reached 300 points in his game number 18 after which he had 508
```

3. Who was reached 500 faster? Can you calculate the average points per game scored for each person up until the game where they reached 500 points?

In [8]:
```
# Your Code Here
first_reached = "Steph" if Steph_game < LeBron_game else "Lebron"
print("Therefore,", first_reached,"reached first.")

# Calculate the average points per game
LeBron_average = LeBron_total_points / (LeBron_game)
Steph_average = Steph_total_points / (Steph_game)
print("Lebron scored an average of,", LeBron_average, "points per game up until this
print("Steph scored an average of,", Steph_average, "points per game up until this p
```

```
Therefore, Steph reached first.
Lebron scored an average of, 19.22222222222222 points per game up until this poi
nt
Steph scored an average of, 28.22222222222222 points per game up until this poin
t
```

4. The 2022-2023 NBA season recieved a lot of attention for the high number of games were a single player scored 40 or more points. This is something both LeBron and Steph have been doing for a long time now. Using `while` loops, find the number of the **first** time each of these players scored at least 40 points in one game.

In [9]:
```
# Your Code Here
LeBron_game = 0
while LeBron_first_season[LeBron_game] < 40:
  LeBron_game += 1

Steph_game = 0
while Steph_first_season[Steph_game] < 40:
  Steph_game += 1

print("LeBron's first 40+ point game was his game number", LeBron_game)
print("Steph's first 40+ point game was his game number", Steph_game)
```

```
LeBron's first 40+ point game was his game number 68
Steph's first 40+ point game was his game number 12
```

5. How many 40+ point games did each player have in their first season? You may use a `while` or a `for` loop for this, your choice.

In [10]:
```
# Your Code Here
LeBron_40_games = 0
for game in LeBron_first_season:
  if game >= 40:
    LeBron_40_games += 1

Steph_40_games = 0
for game in Steph_first_season:
  if game >= 40:
    Steph_40_games += 1

print("LeBron's first season, he had", LeBron_40_games, "40+ point games")
print("Steph's first season, he had", Steph_40_games, "40+ point games")
```

```
LeBron's first season, he had 1 40+ point games
Steph's first season, he had 9 40+ point games
```

6. Who would you say had the stronger first season **in terms of their scoring**? Why? You are welcome to do more calculations to justify your answer.

In [11]:
```
# Your Code Here
LeBron_mean = sum(LeBron_first_season)/len(LeBron_first_season)
Steph_mean = sum(Steph_first_season)/len(Steph_first_season)
print("LeBron's average points per games in his first season:", LeBron_mean)
print("Steph's average points per games in his first season:", Steph_mean)
```

```
LeBron's average points per games in his first season: 20.936708860759495
Steph's average points per games in his first season: 29.710526315789473
```

Your Response Here

From the metrics we've calculated already, we could say that Steph had a more dominant first season. He had 8 more 40+ point games and up to 500+ points seemed to be scoring points faster. If we want, we could calculate the average number of points per game for each player for the whole season, which would also show Steph scoring more per game than LeBron in his first season.

Prime Numbers

Recall that prime numbers are really special in that they have factors other than 1 and themselves.

We're to write code to decide if a number is prime.

How can we tell if a number is prime?

The process for determining if number `n` is prime will be something like this:

1. Check if `n` is divisible by `2`, if it is, then `n` isn't prime
2. Check if `n` is divisible by `3`, if it is, then `n` isn't prime
3. Check if `n` is divisible by `4`, if it is, then `n` isn't prime

   ...

Check if `n` ia divisible by `int(n/2)` , if it ISNT, then `n` IS prime

But how can we tell if a number number is divisible by another? Well, remember that in coding we have **modulo** division which finds the remainder of the division of two numbers. Note that if a number is divisible by another, than the remainder of the division of these two numbers will be 0.

1. Complete the following code to determine if `n` is divisible by `x` .

In [12]:
```
x = 13
n = 5356
if n % x == 0:
  print(n, "is divisible by", x)
else:
  print(n, "is NOT divisible by", x)
```

```
5356 is divisible by 13
```

2. Write a script to determine if the number `n` is prime.

In [13]:
```
n = 313
# Your Code Here
prime = True
for i in range(2, int(n/2)):
  if n % i == 0:
    prime = False

if prime:
  print(n, "is prime")
else:
  print(n, "is NOT prime")
```

```
313 is prime
```

3. Calculate the number of prime numbers under 100?

In [14]:
```
# Your Code Here
num_primes = 0

# 1 is not a prime number, so we don't test it
for n in range(2, 100):
  prime = True
  for i in range(2, int(n/2)):
    if n % i == 0:
      prime = False
  if prime:
    num_primes += 1

print("There are", num_primes, "primes under 100")
```

```
There are 26 primes under 100
```

Password Validation

Now, we return to the problem from the beginning of class. Ask the user for a password. The password must be at least 8 characters long. If the password is not at least 8 characters long, reject the password and ask again.

In [15]:
```
# Your Code Here

password = input("Please enter a password:")
while len(password) < 8:
  print("Error, your password must be at least 8 characters long, yours was", len(pa
  password = input("Please enter a password:")

print("Password Saved!")
```

```
Please enter a password:asdfasfh
Password Saved!
```

Bonus: Palindrome Numbers

Imagine you're really interested in palidrome numbers, that is numbers which are the same forwards and backwards, for example `303` , `1221` , or `1246421` . To do this, we'd need to first think about how you would reverse the digits of a number. One way would be to turn it into a string and reverse the string:

```
a = 12454132
a_str = str(a)
# reverses the string
a_str = a_str[::-1]
print(a_str)
```

However, this is a little bit confusing and the conversion between integer and string isn't terribly efficient. There's a way to do it with just math and loops. Can you figure it out?

In [16]:
```
a = 12454132
# Your code here
a_copy = a
a_reversed = 0

while a_copy > 0:
  # "move" the last digit to the left
  a_reversed *= 10

  # make the last digit of a_copy the last
  # digit of a_reversed
  a_reversed += a_copy % 10

  # "remove" the last digit
  a_copy = int(a_copy/10)

print(a_reversed)
```

```
23145421
```

Now, write a script which determines if a number is a palindrome.

In [17]:
```
# Your code here
a = 1331
a_copy = a
a_reversed = 0

while a_copy > 0:
  # "move" the last digit to the left
  a_reversed *= 10

  # make the last digit of a_copy the last
  # digit of a_reversed
  a_reversed += a_copy % 10

  # "remove" the last digit
  a_copy = int(a_copy/10)

if a_reversed == a:
  print(a, "is a palindrome")
else:
  print(a, "is NOT palindrome")
```

```
1331 is a palindrome
```

References

[1]W3 Schools `while` Loop Page (https://www.w3schools.com/python/python_while_loops.asp)

[2]The NBA has a way for accessing states using Python, called the nba_api (https://pypi.org/project/nba-api/)

In [ ]:

7. Works Cited

7 Reasons Why Kids Should Learn to Code. (2020, November 11). *UT Austin Boot Camps*. https://techbootcamps.utexas.edu/blog/why-kids-should-learn-to-code/

21K School. (2022, August 12). The Importance of Coding in 21st Century Education. *21K School*. https://www.21kschool.com/blog/the-importance-of-coding-in-21st-century-education/

Algebra I Year at a Glance Baltimore County. (2022). Baltimore County Public Schools Office of Mathematics PreK-12. https://cdnsm5-ss3.sharpschool.com/UserFiles/Servers/Server_9046958/File/Mathematics/22-23%20Year%20At%20A%20Glance/Algebra%201%20Year%20at%20a%20Glance.pdf

Bell, T. (2016). *What's all the fuss about coding?*

Beyond Point and Click. (2016). *BURNING GLASS TECHNOLOGIES*. https://www.burning-glass.com/wp-content/uploads/Beyond_Point_Click_final.pdf

Börner, K., Bueckle, A., & Ginda, M. (2019). Data visualization literacy: Definitions, conceptual frameworks, exercises, and assessments. *Proceedings of the National Academy of Sciences*, *116*(6), 1857–1864. https://doi.org/10.1073/pnas.1807180116

Brooks, A. (2020, June 28). *What Kinds of Math Do I Need to Know for Coding? - Wyzant Blog*. https://www.wyzant.com/blog/math-for-coding/

Canales Luna, J. (2023, March). *Top programming languages for data scientists in 2023*. https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022

Carroll College Library & Learning Commons (Director). (2021, April 23). *6 Ways to Implement Coding in a Math Class*. https://www.youtube.com/watch?v=G8H7AM6-jP4

Coding in Math Syllabus. (n.d.). CodeHS. https://codehs.com/uploads/07697aed5b957b14873c045f3d5bc2ed

College Board. (n.d.). *AP Computer Science A*. Retrieved March 21, 2023, from https://apstudents.collegeboard.org/courses/ap-computer-science-a

Common Core State Standards for Mathematics. (n.d.).

Computational Biology. (2019). MIT Department of Biology. https://biology.mit.edu/faculty-and-research/areas-of-research/computational-biology/

Corradini, I., Lodi, M., & Nardelli, E. (2018). An Investigation of Italian Primary School Teachers' View on Coding and Programming. In S. N. Pozdniakov & V. Dagienė (Eds.), *Informatics in Schools. Fundamentals of Computer Science and Software Engineering* (pp. 228–243). Springer International Publishing. https://doi.org/10.1007/978-3-030-02750-6_18

Craig, T. T., & Marshall, J. (2019). Effect of project-based learning on high school students' state-mandated, standardized math and science exam performance. *Journal of Research in Science Teaching*, *56*(10), 1461–1488. https://doi.org/10.1002/tea.21582

edX team. (2021, November 3). *How Is Math Used in Computer Science?* https://blog.edx.org/how-is-math-used-in-computer-science

Engebretsen, M., & Kennedy, H. (2020). *Data Visualization in Society*. Amsterdam University Press. http://ebookcentral.proquest.com/lib/bibalcala/detail.action?docID=6637413

Faller, A., Herwehe, L., & Nagy, C. (2021, January 12). *What is Scientific Programming (And Why It Rocks) | Earthlab*. https://earthlab.colorado.edu/blog/what-scientific-programming-and-why-it-rocks

Fan, X., Yuan, Y., & Liu, J. S. (2010). The EM Algorithm and the Rise of Computational Biology. *Statistical Science*, *25*(4), 476–491. https://doi.org/10.1214/09-STS312

Félix-Herrán, L. C., Rendon-Nava, A. E., & Nieto Jalil, J. M. (2019). Challenge-based learning: An I-semester for experiential learning in Mechatronics Engineering. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, *13*(4), 1367–1383. https://doi.org/10.1007/s12008-019-00602-6

Fowler, T. (2020, August 12). *High-level and Low-level Programming Languages*. Career Karma. https://careerkarma.com/blog/high-level-and-low-level-languages/

freeCodeCamp. (n.d.). *Scientific Computing with Python*. Retrieved April 25, 2023, from https://www.freecodecamp.org/learn/scientific-computing-with-python/

Gallagher, J. (2020, December 11). How to Write Python Functions. *Career Karma*. https://careerkarma.com/blog/python-functions/

Grade 8 Mathematics: Performance Level Descriptors. (2022). Maryland State Department of Education. https://marylandpublicschools.org/about/Documents/DCAA/Math/PLD/MCAPMathGrade8ContentPLDs.pdf

Hemmendinger, D. (2022, December 2). *Computer programming language*. https://www.britannica.com/technology/computer-programming-language

Hewer, R. (2022, March 20). *Mathematics is the Key to a STEM Major: Why Students Need to Excel in This Field*. DropKick Math. https://dropkickmath.com/blog/mathematics-is-the-key-to-a-stem-major/

High and Low Level Languages. (n.d.). Computer Science GCSE GURU. Retrieved March 23, 2023, from https://www.computerscience.gcse.guru/theory/high-low-level-languages

Hobenshield Tepylo, D., & Floyd, L. (2016, June 4). Learning Math Through Coding. *Math + Code 'Zine*. https://researchideas.ca/mc/learning-math-through-coding/

Howard County Public Schools. (n.d.). *HCPSS Algebra I/Algebra I GT Essential Curriculum*. https://www.hcpss.org/f/academics/math/algebra-one-gt-curriculum.pdf

Illustrative Mathematics. (2019). *Algebra I Units*. Illustrative Mathematics.

https://curriculum.illustrativemathematics.org/HS/teachers/1/index.html

Kay, R. M. (2022, December 21). *What is Abstraction in Programming? Explained for Beginners*. FreeCodeCamp.Org. https://www.freecodecamp.org/news/what-is-abstraction-in-programming-for-beginners/

Kolb, D. (2014). *Experiential Learning: Experience as the Source of Learning and Development, Second Edition*. https://learning.oreilly.com/library/view/experiential-learning-experience/9780133892512/

KS3. (n.d.). *Iteration in programming*. BBC Bitesize. Retrieved March 24, 2023, from https://www.bbc.co.uk/bitesize/guides/z3khpv4/revision/1

Larson, M., & Goetz, A. (n.d.). *CS and Math*. Retrieved March 21, 2023, from https://csandmath.org/blog/

Lee, Y.-J. (2022). Promoting social and emotional learning competencies in science, technology, engineering, and mathematics project-based mathematics classrooms. *School Science and Mathematics*, *122*(8), 429–434. https://doi.org/10.1111/ssm.12557

Lohr, S. (2017, April 4). Where Non-Techies Can Get With the Programming. *The New York Times*. https://www.nytimes.com/2017/04/04/education/edlife/where-non-techies-computer-programming-coding.html

Maryland Department of Education. (2020). *Maryland College and Career Ready Standards: Algebra 1*. https://www.marylandpublicschools.org/about/Documents/DCAA/Math/MGLCR/HSR/HSR-Algebra/HS.Algebra1.MCCRS.pdf

Maryland State Department of Education. (2012). *Maryland's New Common Core State Standards And PARENTS*.

Maryland State Department of Education. (2023a). *Maryland Comprehensive Assessment Program (MCAP) Mathematics*. https://marylandpublicschools.org/about/Pages/DAAIT/Assessment/MCAP/Math.aspx

Maryland State Department of Education. (2023b). *Maryland Public School Openings And Closings Dates*. https://marylandpublicschools.org/about/pages/school-systems/open-closing-dates.aspx

Mensah, Y. M., Schoderbek, M. P., & Sahay, S. P. (2013). The effect of administrative pay and local property taxes on student achievement scores: Evidence from New Jersey public schools. *Economics of Education Review*, *34*, 1–16. https://doi.org/10.1016/j.econedurev.2013.01.005

Montgomery County Public Schools. (n.d.). *Algebra 1*. Retrieved March 23, 2023, from https://www2.montgomeryschoolsmd.org/curriculum/math/high/algebra1/

Murphy, R. (2023). *What is Computational Biology?* http://cbd.cmu.edu/about-us/what-is-computational-biology.html

National Institute of Standards and Technology. (2016). Scientific Computing. *National Institute of Standards and Technology*. https://www.nist.gov/itl/math/scientific-computing

Noble, D. (2002). The rise of computational biology. *Nature Reviews. Molecular Cell Biology*, *3*(6), 459–463.

https://doi.org/10.1038/nrm810

O'Brien, W., Doré, N., Campbell-Templeman, K., Lowcay, D., & Derakhti, M. (2021). Living labs as an opportunity for experiential learning in building engineering education. *Advanced Engineering Informatics*, *50*, 101440. https://doi.org/10.1016/j.aei.2021.101440

Pappano, L. (2017, April 4). Learning to Think Like a Computer. *The New York Times*. https://www.nytimes.com/2017/04/04/education/edlife/teaching-students-computer-code.html

Pendelton, D. (1975, March 15). Calculators in the Classroom. *Science News*, *107*(11), 175–181.

PLOS Blogs. (2016, November 29). *Computer coding in science*. PHYS.ORG. https://phys.org/news/2016-11-coding-science.html

Programiz. (n.d.). *Python Functions*. Retrieved March 22, 2023, from https://www.programiz.com/python-programming/function

Python Software Foundation. (2023a). *What is Python? Executive Summary*. Python.Org. https://www.python.org/doc/essays/blurb/

Python Software Foundation. (2023b, June 4). *Data model*. Python Documentation. https://docs.python.org/3/reference/datamodel.html

Savery, J. R. (2006). Overview of Problem-based Learning: Definitions and Distinctions. *Interdisciplinary Journal of Problem-Based Learning*, *1*(1). https://doi.org/10.7771/1541-5015.1002

Secondary Mathematics. (n.d.). Retrieved March 21, 2023, from https://dci.bcps.org/department/academics/mathematics_pre_k-12/secondary_mathematics

Shumow, L. (2023). *Math Matters for Careers and Jobs*. Northern Illinois University. https://www.niu.edu/mathmatters/careers-jobs/index.shtml

Sukma, Y., & Priatna, N. (2021). The effectiveness of blended learning on students' critical thinking skills in mathematics education: A literature review. *Journal of Physics: Conference Series*, *1806*(1), 012071. https://doi.org/10.1088/1742-6596/1806/1/012071

Sundnes, J. (2020). *Introduction to Scientific Programming with Python*. Springer International Publishing. https://doi.org/10.1007/978-3-030-50356-7

Sweigart, A. (2012, March 17). *"How much math do I need to know to program?" Not That Much, Actually.* https://inventwithpython.com/blog/2012/03/18/how-much-math-do-i-need-to-know-to-program-not-that-much-actually/

Tutorials Point. (n.d.). *Python—Variables*. Retrieved March 24, 2023, from https://www.tutorialspoint.com/python/python_variables.htm

U.S. Bureau Of Labor Statistics. (2023, February 6). *Software Developers, Quality Assurance Analysts, and Testers: Occupational Outlook Handbook: : U.S. Bureau of Labor Statistics*. U.S. Bureau Of Labor Statistics.

https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-1

Utah Valley University Office of Teaching and Learning. (n.d.). *Project-Based Learning and Problem-Based Learning (x-BL)*. Retrieved June 3, 2023, from https://www.uvu.edu/otl/resources/group_work/pbl.html

Uyen, B. P., Tong, D. H., & Lien, N. B. (2022). The Effectiveness of Experiential Learning in Teaching Arithmetic and Geometry in Sixth Grade. *Frontiers in Education*, *7*. https://www.frontiersin.org/articles/10.3389/feduc.2022.858631

Vaidyanathan, S. (n.d.). *Scratch Coding projects in Math class*. Computers For Creativity. Retrieved April 25, 2023, from https://www.computersforcreativity.com/coding-in-math-class

W3 Schools. (2023a). *Python Functions*. https://www.w3schools.com/python/python_functions.asp

W3 Schools. (2023b). *Python Variables*. https://www.w3schools.com/python/python_variables.asp

Wolff, A., Gooch, D., Montaner, J. J. C., Rashid, U., & Kortuem, G. (2016). Creating an Understanding of Data Literacy for a Data-driven Society. *The Journal of Community Informatics*, *12*(3), Article 3. https://doi.org/10.15353/joci.v12i3.3275

Wolff, A., Wermelinger, M., & Petre, M. (2019). Exploring design principles for data literacy activities to support children's inquiries from complex data. *International Journal of Human-Computer Studies*, *129*, 41–54. https://doi.org/10.1016/j.ijhcs.2019.03.006

Common Core States, (May 2023). https://worldpopulationreview.com/state-rankings/common-core-states

Zingale, M., Timmes, F. X., Fisher, R., & O'Shea, B. W. (2016). *The Importance of Computation in Astronomy Education* (arXiv:1606.02242). arXiv. http://arxiv.org/abs/1606.02242