Date Approved by DJUSD Board of Education: _____

Secondary Course Description

Course descriptions are updated and reviewed with all new text adoptions.

COVER PAGE

1. Course Title: Computer Science Principles

2. Transcript Title / Abbreviation: CompSciPrin

3. Transcript Course Code / Number (Office Use Only):

4. School: Da Vinci Charter Academy

5. District: Davis Joint Unified School District

6. Department: Mathematics

7. Graduation Requirement it meets: Elective

8. Length of Course: 1 Year

9. Graduation Credits: Elective

10. School / District Web Site: http://www.djusd.net

11. CBEDS Course Code:

12. School Contact: Da Vinci Charter Academy

Name: Tyler Millsap

Title/Position: Principal

Phone: (530) 757-7154 **Ext.:**

Fax:

E-mail: tmillsap@djusd.net

13. Subject Area:

- ☐ History/Social Science
- ☐ English
- ☒ Mathematics
- ☐ Science
- ☐ Language other than English
- ☐ Visual & Performing Arts
- ☐ DJUSD Graduation Elective
- ☒ College Prep Elective (will seek UC/CSU approval)

14. Grade Level(s): 10-12

15. UC/CSU Requirement:

16. Seeking "Honors" Distinction? ☐ Yes ☒ No

17. GPA Types:

18. Credit Value:

- ☐ 0.5 (half year or semester equivalent)
- ☒ 1.0 (one year equivalent)
- ☐ 2.0 (two year equivalent)
- ☐ Other: _____

19. Was this course previously approved by UC? ☐ Yes ☒ No

If so, in what year? _____ **Under what course title?** _____

20. Pre-Requisites: None

Co-Requisites: None

21. Preliminary Approval - Secondary Site Principal Signature (Must be signed before proceeding to Step 22):

_____

22. Date Course Proposal with Preliminary Approval (Step 15) sent to Associate Superintendent, Educational Services:

_____

23. Review & Approval:

Date		Signature
11/14/18	Site Curriculum and Instruction Leadership Team	Signature/Title _____
_____	Secondary Department Articulation/Collaboration	Signature/Title _____

Secondary Principal Signatures: _____ _____

Date: _____ _____ _____

BACKGROUND INFORMATION

Brief Course Description:

Code.org's Computer Science Principles (CSP) curriculum is a full-year, rigorous, entry-level course that introduces high school students to the foundations of modern computing. The course covers a broad range of foundational topics such as programming, algorithms, the Internet, big data, digital privacy and security, and the societal impacts of computing. The course is designed for typical school settings with teachers in classrooms. All teacher and student materials are provided for free online.

Context for Course:

There are no formal prerequisites for this course, though the College Board recommends that students have taken at least Algebra 1. The course requires a significant amount of expository writing (as well as writing computer code, of course).

The curriculum itself does not assume any prior knowledge of computing concepts before entering the course. It is intended to be suitable as a first course in computing though students with a variety of backgrounds and prior experiences will also find the course engaging and with plenty of challenges. While it is increasingly likely that students entering this course in high school will have had some prior experience in computer science (particularly with programming), that experience is equally likely to be highly varied both in quantity and quality. It is for this reason that the course does not start with programming, but instead with material that is much more likely to put all students on a level playing field for the first few weeks of class. Read more about this below in the description of Unit 1.

List the State/District Standards addressed in this course.

CS Discoveries was written using both the K-12 Framework for Computer Science and the newly revised 2017 CSTA standards as guidance. Currently, every lesson in CS Discoveries contains mappings to the relevant 2017 CSTA standards. The summary of of all CSTA 2017 mappings can be found at curriculum.code.org/csd/standards

History of Course Development:

CS Principles Ideas and Inspirations
This section gives attribution to the influential works in computer science education that form some of the philosophical underpinnings of the course.

Exploring Computer Science
We all owe a debt of gratitude to the Exploring Computer Science (ECS) curriculum and professional development program by Joanna Goode and Gail Chapman. Many members of the curriculum team and existing Code.org staff offspring of, and cut their teeth as part of, the ECS "movement" and much of the ECS ethos lives on in our CS Principles program as well. On the curriculum side, we wanted to write a curriculum that supported both students and teachers who were completely new to computer science. The inquiry/discovery-based model of lesson design, which strives to create as many opportunities as possible for all students in the classroom to engage with the material, as well as our understandings, biases and beliefs about how new CS teachers and students enter the field, are all heavily influenced by ECS and our previous work with that program.

Similarly, our professional development model for CS Principles attempts to mimic the success of the ECS professional development program. In particular, we carry on the effective the "teacher-learner-observer" (TLO) practice, in which teachers take the time to prepare, deliver and reflect on lessons with colleagues. The TLO practice reflects the spirit and goals of the ECS curriculum itself: if you want teachers (or students) to understand how to teach CS, all must be provided with as many opportunities as possible to engage with the material in authentic ways. This is especially true when trying to achieve the audacious twin-goals of creating new computer science teachers where there were none before and improving teaching practice in the field at the same time. Our CS Principles program attempts to achieve many of the same goals, and the ECS program should be credited with showing us all the path for how to do it.

CS Unplugged

The CS education community is completely indebted to CS Unplugged (Tim Bell and colleagues at the University of Canterbury, and Mike Fellows at the University of Bergen) for bringing computational thinking skills into classrooms in active, kinesthetic ways. The very term "unplugged" is now part of common parlance in CS classrooms all over the world and should be attributed the these fine folks.

Many of our computational widgets have their roots in CS Unplugged activities. Our goal was to use the activity and then "plug it in" to allow for further exploration of the concept and in some cases take it a step further. The “pixelation” widget was inspired by the Image Representation Activity. The text compression widget was inspired by CS Unplugged’s activity of the same name: Text Compression

Some of the unplugged activities in the curriculum itself are CS Unplugged activities that we modified to slightly "adult-ify" the context or focus it more explicitly on a computer science problem. The Minimum Spanning Tree activity has its roots in CS Unplugged’s Muddy City activity. The WiFi Hotspot problem is inspired by the Dominating Sets activity.

Logo Turtle and Karel the Robot

We have merged ideas from two giants of early teaching about programming. The turtle graphics from Logo (Pappert et. al. 1967) have been used with children for decades for the effective "body-syntonic" way it leads to reasoning about sequence, logic and procedural abstraction. Karel the Robot (Pattis et. al. 1981) is a learning construct in which the programmer controls a “robot” which initially has a very limited set of commands and as the programmer learns new language constructs, can add to the robot’s capabilities by making new commands that build on the primitive set. This allows for problem solving exercises that lead to good programming practice and good top-down design thinking skills. We use turtle graphics like Logo, but in the curriculum initially give the turtle a very limited set of commands like Karel.

COURSE GOALS AND/OR MAJOR STUDENT OUTCOMES

A central goal of Code.org's CSP curriculum is for it to be accessible to all students, especially those in groups typically underrepresented in computing. To this end, we have worked to provide examples and activities that are relevant and topical enough for students to connect back to their own interests and lives. Wherever possible, and especially in the videos that accompany the curriculum, we seek to highlight a diverse array of role models in terms of gender, race, and profession from which students can draw inspiration and "see themselves" participating in computing.

The curriculum assumes no prior knowledge of computing and is written to support both students and teachers who are new to the discipline. Activities are designed and structured in such a way that students with diverse learning needs have space to find their voice and to express their thoughts and opinions. The activities, videos, and computing tools in the curriculum strive to have a broad appeal and to be accessible to a student body diverse in background, gender, race, prior knowledge of computing, and personal interests.

Broadening student participation in computer science is a national goal, and effectively an issue of social justice. Motivational marketing messages only get you so far. We believe that the real key to attracting students to computer science and then sustaining that growth has as much to do with the teacher in the classroom as it does with anything else. The real "access" students need to computing is an opportunity to legitimately and meaningfully participate in every lesson regardless of the student's background or prior experience in computing coming into the course. For example, the course begins with material that is challenging but typically unfamiliar even to students who have some prior experience or knowledge of computer science.

COURSE OBJECTIVES

Computing affects almost all aspects of modern life and all students deserve access to a computing education that prepares them to pursue the wide array of intellectual and career opportunities that computing has made possible. This course is not a tour of current events and technologies. Rather, it seeks to provide students with a "future proof" foundation in computing principles so that they are adequately prepared with both the knowledge and skills to live and meaningfully participate in our increasingly digital society, economy, and culture.

COURSE OUTLINE

Semester 1

Unit 1 – The Internet: Learn how the multi-layered systems of the internet function as you collaboratively solve problems and puzzles about encoding and transmitting data, both 'unplugged' and using Code.org's Internet Simulator.

Unit 2 – Digital Information: Learn how computers store complex information like images, video, and sound. Use interactive widgets to explore concepts like image representation and compression.

Unit 3 – Intro to Programming: Learn the JavaScript language with turtle programming in Code.org's App Lab coding environment. Learn general principles of algorithms and program design that apply to any programming language.

Semester 2

Unit 4 – Big Data and Privacy: Research current events at the intersection of data, public policy, law, ethics, and societal impact. Learn the basics of how and why modern encryption works.

Explore PT Prep: Practice and then complete the Explore Performance Task (PT)

Unit 5 – Building Apps: Continue learning how to program in the JavaScript language. Use Code.org's App Lab environment to create a series of applications that live on the web. Each app highlights a core concept of programming.
Create PT Prep: Practice and then complete the Create Performance Task (PT)

Post AP – Data Tools: Learn how computers allow data to be collected, cleaned, analyzed, and visualized in order to find patterns and draw conclusions.

TEXTS AND SUPPLEMENTAL INSTRUCTIONAL MATERIALS

Title, Author, Publisher, Edition: All curriculum resources and tutorials will forever be free to use and openly licensed under a Creative Commons license, allowing others to make derivative education resources for non-commercial purposes. [code.org] believes that every child deserves a high-quality education and that expensive curriculum should not prevent students from the opportunity to learn computer science.

Previously Adopted? [x] **Yes** [ ] **No (If no, provide information directly below)**

Cost per book	Total Cost	Budget Source

Other:

What materials and supplies are required for CS Principles?

This course requires students have access to computers with a modern web browser. For more details, check out Code.org's technology requirements, here.

Many lessons have handouts that are designed to guide students through activities. While these handouts are not required, we highly recommend their use. In addition to handouts, you will need the following:

- Unit 1 Lesson 2 requires some craft materials for constructing physical devices. The lesson recommends items like cups, string/yarn, construction paper, flashlights, slinkies, noise makers, markers, and glue.

Optional Materials
The following items are called for in lessons, but alternatives are offered below:

- (Unit 3, Lesson 1) A handful of legos per 2-3 students. Alternatives: post-it notes, construction paper
- (Unit 3, Lesson 2 - 3) Playing cards (1 deck per 6 students). Alternatives: write numbers of post-it notes.
- (Unit 4, Lesson 8) Clear dixie cups with beans. Alternatives: Any clear container (ziplock bag, empty water bottle, etc) with any small item (beads, raisins, coffee beans, etc)

The following supplies are completely optional but will be useful to have on hand for various lessons:

- Graph paper
- Chart paper
- Markers
- Post-it notes

It is important to note that the lack of expense for this course is due to the fact that Da Vinci possesses most of the infrastructure. Students already utilize school or personal laptops and wireless access is provided. The primary cost will be the personnel for teaching the course. This will be a .2 teacher assignment. The professional learning program includes a 4 day summer workshop and 4 days throughout the school year. The professional learning program is provided free of charge from code.org

DIFFERENTIATED INSTRUCTIONAL METHODS AND/OR STRATEGIES

CS Principles is designed from the ground up to be an accessible and engaging course for all students, regardless of background or prior experience. It provides students opportunities to engage with culturally and personally relevant topics in a wide variety of contexts and aims to show all students that CS is for them.

We believe that acknowledging and shining a light on the historical inequities within the field of computer science is critical to reaching our goal of bringing computer science to all students. We provide tools and strategies to help teachers understand and address well-known equity gaps within the field. We recognize that some students and classrooms need more supports than others, and so those with the greatest needs should be prioritized. All students can succeed in computer science when given the right supports and opportunities, regardless of prior knowledge or privilege. We actively seek to eliminate and discredit stereotypes that plague computer science and lead to attrition of the very students we aim to reach.

ASSESSMENT METHODS AND/OR TOOLS

The course materials contain a number of assessment types and opportunities which can be used formatively (to check for understanding) or summatively (for evaluation).

For students, the goal of the assessments is to prepare them for the AP exam and performance tasks. For teachers, the goal is to use assessments to help guide instruction, give feedback to students, and make choices about what to emphasize in lessons.

Code Studio includes features that assist the teacher in completing formative and summative assessments:

- Multiple choice or matching questions related to questions on the chapter summative assessment.
- Free-response text fields where students may input their answer.
- Access to student work within the App Lab programming environment and other digital tools and widgets used in the curriculum.
- The ability for students to submit final versions of App Lab projects.

Fixed Response Assessments
Lesson assessment items - You will find assessment items (multi-choice, free-response text) embedded in individual lessons, typically as the last few "bubbles" for a lesson, indicated with an assessment icon. These are intended to be used as *formative* assessment items. Students can always see them and change their responses at any time.

Chapter assessments (lockable) - are typically 10-15 question multiple choice tests intended to mimic AP-style questions. They look like their own lesson and have *lock settings* as well as the usual visibility settings. These can be use for formative or summative assessments.

Lock settings enable or disable students from modifying their answers. For example, you may want to hide an assessment before students get to it, but after students take it, you might want it to remain visible but locked, while you review the answers.

Practice Performance Tasks
Each unit contains at least one project designed in the spirit of the AP Performance Tasks (PTs). These **Practice PTs** are smaller in scope, contextualized to the unit of study and are intended to help prepare students to engage in the official administration of the AP PTs at the end of the course. Practice PTs come with:

- Project description
- Written response prompts similar to AP-style
- Project rubrics modeled after the AP rubrics

Practice PTs are the most authentic way for you and your students to prepare for the *real* PTs. Use them!

Submittable Programming Projects

When the curriculum calls for a programming project for assessment the App Lab environment will show a "submit" button below the typical "Run" button. When a student submits a project it is submitted with a timestamp and locked for teacher review. It also shows up in a special area of the teacher dashboard. The teacher can release the project back to the student as well.

Worksheets and Activity Guides
Worksheets and activity guides are good opportunities for assessment. Worksheets or activity guides often ask students to write, answer questions, and respond to prompts. When available, answer keys for worksheets and activities are provided through the "teacher only" panel on Code Studio.

It is up to the classroom teacher:

- to determine the appropriateness of the assessments for their classrooms.
- to decide how to use, or not to use, the assessments for grading purposes. The curriculum and online dashboards do not provide teachers with a gradebook, and we do not provide recommendations for how to assign grades based on performance on an assessment.

ASSESSMENT CRITERIA

The existing framework/standards for computer science will serve as the assessment criteria, particularly for the first half of the course. Students will be expected to demonstrate proficiency on concepts through quizzes, tests, and activities that apply concepts. In the second semester of the course, assessment will shift towards application of concepts through real-world coding demonstrations. These will be performance-based and will be assessed using Da Vinci's Expected School-wide Learning Results.

AP ® Assessment

The AP Assessment consists of a 74-question multiple choice exam and two "through-course" assessments called the AP Performance Tasks (PTs). The tasks can be found in the official AP CS Principles Exam and Course Description.

- Explore Performance Task (p. 108)
- Create Performance Task (p. 113)

HONORS COURSES ONLY

Indicate how this honors course is different from the standard course.