AP CSP

Unit 3: Block-Based Programming

**Illustrations and some definitions come from our textbook: Computer Science Illuminated, by Nell Dale and John Lewis, © 2016 Jones & Bartlett Learning, LLC, Burlington, MA

Quote:

- “I don’t think everyone will be a coder, but the ability to speak and structure your thinking in a way a computer understands it will be one of the core future skills whatever your field.”
  - Linda Liukas

Learning Targets:

- Use a block-based programming language to write programs that solve given problems.
- Implement multiple levels of abstraction when writing a program.
- Implement an algorithm when writing a program and test the correctness of the algorithm.

Essential Questions & Why:

- Essential Questions:
  - How does abstraction help us in writing programs, creating computational artifacts, and solving problems?
  - How are algorithms implemented and executed on computers and computational devices?
  - How do computer programs implement algorithms?
  - How does abstraction make the development of computer programs possible?
  - How do people develop and test computer programs?
- Why?
  - Abstraction reduces information and detail to facilitate focus on relevant concepts.
  - Algorithms are used to develop and express solutions to computational problems.
  - Programming enables problem solving, human expression, and creation of knowledge.

  - *Questions and the reason why come from AP Computer Science Principles Course and Exam Description https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf, The College Board.

Computational Thinking Practices:

- P2: Creating Computational Artifacts
  - Students are expected to select appropriate techniques to develop a computational artifact.

Computational Thinking Practices:

- P3: Abstracting
  - Students are expected to explain how data, information, or knowledge is represented for computational use.
  - Students are expected to identify abstractions.

Computational Thinking Practices:

- P4: Analyzing Problems and Artifacts
  - Students are expected to locate and correct errors.
  - Students are expected to explain how an artifact functions.

Computational Thinking Practices:

- P5: Communicating
  - Students are expected to describe computation with accurate and precise language, notations, or visualizations.
  - Students are expected to summarize the purpose of a computational artifact.

Content Standards:

- Big Idea 2: Abstraction
  - EU 2.2: Multiple levels of abstraction are used to write programs or create other computational artifacts.
    - LO 2.2.1: Develop an abstraction when writing a program or creating other computational artifacts. [P2]
    - LO 2.2.2: Use multiple levels of abstraction to write programs. [P3]

Content Standards:

- Big Idea 4: Algorithms
  - EU 4.1: Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
    - LO 4.1.1: Develop an algorithm for implementation in a program. [P2]
    - LO 4.1.2: Express an algorithm in a language. [P5]
  - EU 4.2: Algorithms can solve many, but not all, computational problems.
    - LO 4.2.4: Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity. [P4]

Content Standards:

- Big Idea 5: Programming
  - EU 5.1: Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
    - LO 5.1.1: Develop a program for creative expression, to satisfy personal curiosity, or to create new knowledge. [P2]
    - LO 5.1.2: Develop a correct program to solve problems. [P2]

Content Standards:

- Big Idea 5: Programming
  - EU 5.2: People write programs to execute algorithms.
    - LO 5.2.1: Explain how programs implement algorithms. [P3]
  - EU 5.3: Programming is facilitated by appropriate abstractions.
    - LO 5.3.1: Use abstraction to manage complexity in programs. [P3]

Assignments for Wednesday, September 18:

- Class Work:
  - Welcome to Snap! handouts.
    - Turn in the completed handouts and submit a 30-second video of your kaleidoscope program running.
- Notes:
  - Unit 2 Quiz: Friday, September 20
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Thursday, September 19:

- Class Work:
  - Snap! handouts.
    - Turn in the completed handouts and submit parts of the assignments to Google Classroom. See Google Classroom for details on each lab.
- Notes:
  - Unit 2 Quiz: Friday, September 20
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Friday, September 20:

- Class Work:
  - Snap! handouts.
    - Turn in the completed handouts and submit parts of the assignments to Google Classroom. See Google Classroom for details on each lab.
- Note:
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Monday, September 23:

- Class Work:
  - Snap! handouts.
    - Turn in the completed handouts and submit parts of the assignments to Google Classroom. See Google Classroom for details on each lab.
- Note:
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Tuesday, September 24:

- Class Work:
  - Snap! House.
    - Use Snap! to create a house. Your house picture should include at least one window, a door, a roofline, and either the sun or the moon or a lovely cloud. Use an abstraction in your program; i.e., make a block to draw repeated shapes. Make your picture fairly plain or quite fancy. Add a tree or flowers if you wish. Submit one document with screenshots of your code and a screenshot of the stage showing the completed house.
    - See Google Classroom.
- Note:
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Wednesday, September 25:

- Class Work:
  - Snap! Spiral Maze.
    - Install a background stage for your spiral maze Create a sprite to draw spiral maze. Create a second sprite to wait at the center of the maze. Create a third sprite that moves continuously to navigate the maze. The game player should help the sprite navigate through the maze to the center by pressing the arrow keys to make the sprite turn and keep moving in the new direction. The maze navigator should keep moving until it touches the sprite at the center of the maze (win!) or the wall of the maze (not win). Include messages for the two possible outcomes. Submit a document with screenshots of your code (for each of the three sprites) and a video of your game being played. Include both possible outcomes in your video. Credit for the inspiration for this assignment will be posted after the project is completed.
    - See Google Classroom.
- Note:
  - Test on Units 1, 2, 3: Friday, September 27

Assignments for Thursday, September 26:

- Class Work:
  - Snap! Maze Project.
    - Import a 30 x 20 maze from mazegenerator.net as the stage. (Take a screenshot of the maze and import it as the stage.) Select a ball sprite and resize it to fit your maze. Drag the ball to the top opening in the maze. Check the position for the x- and y-coordinates. Set the ball to go to this position. Create variables "old x" and "old y" In a forever loop: set "old x" to x-position and "old y" to y-position, write four if-statements to use the arrow keys to move the sprite 1 or 2 steps in the given direction, and write an if-statement to check if the sprite is touching the wall of the maze. If so, send the sprite back to "old x", "old y" and make a sound. Make the program celebrate at the completion of the maze with a celebratory sound or message. Have fun! Add another sprite or an opening screen for extra credit.
    - See Google Classroom.
- Note:
  - Test on Units 1, 2, 3: next week.