Weekly Planner: AP CSP week of 4.27.20

AP CSP

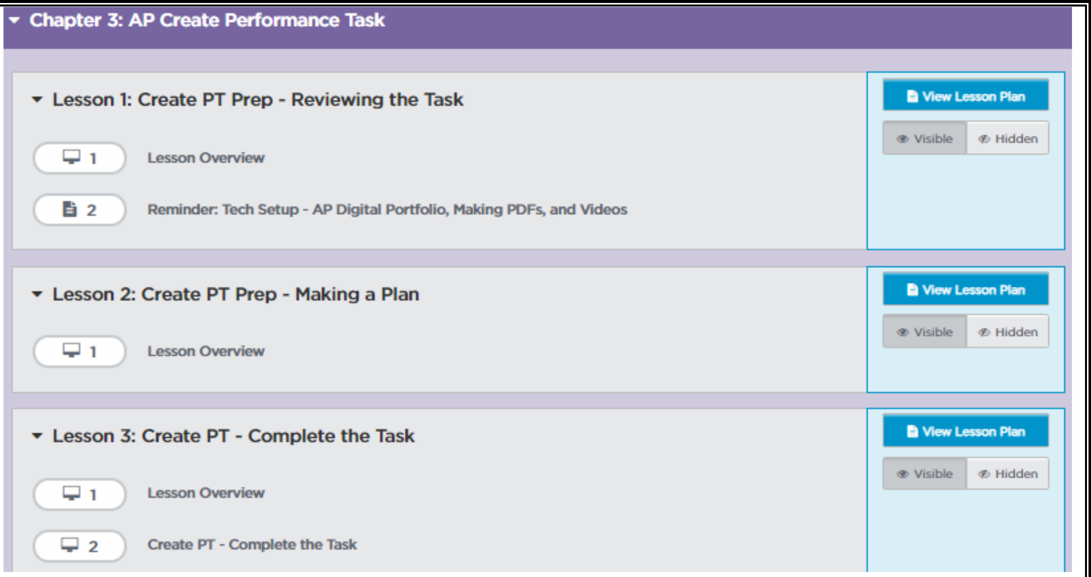
Flip Code: cccaapcsp

BIG IDEA for the week:

3- ALGORITHMS AND PROGRAMMING

At this time, I have unlocked most of Unit 5 for you and you may work ahead of schedule if you like. **LET ME KNOW if you need me to unlock an assessment for you or if I can help you with any of the programs.**

Day	
Mon 4.27	**IF you continue to check in and let me know what you are doing each day, you will NOT be given a late grade.** **Week overview: https://screencast-o-matic.com/watch/cYftivAKO7** **Once you have finished unit 5, do the video review and then schoology me to unlock test #5**
Tues 4.28	UNIT 5 Assessment #5 video review: https://screencast-o-matic.com/watch/cYfOFDAGeo Once finished, you should be able to move on to the create task unit and work on your final project. PLAN ON 2 hours per day to work on this for the first two weeks of May. Our last day of school is May 15th. CREATE TASK walkthrough: https://www.youtube.com/watch?v=rNlNFYJ9Spc 4 part CREATE TASK walkthrough!!! https://www.youtube.com/watch?v=j9HerBCGIcE
Wed 4.29	
Thurs 4.30	

Fri **5.1**	 Chapter 3: AP Create Performance Task Lesson 1: Create PT Prep - Reviewing the Task 1 Lesson Overview 2 Reminder: Tech Setup - AP Digital Portfolio, Making PDFs, and Videos Lesson 2: Create PT Prep - Making a Plan 1 Lesson Overview Lesson 3: Create PT - Complete the Task 1 Lesson Overview 2 Create PT - Complete the Task
	https://www.unicef.org/coronavirus/how-teenagers-can-protect-their-mental-health-during-coronavirus-covid-19 IF you have finished Unit 5: The world is your oyster!!!! see below :) **Login Information** Section Code: RZZNQP Ask your students to join your section by going to this link and entering the section code (above): https://studio.code.org/join Alternatively, share this section's sign in page with your students: https://studio.code.org/sections/RZZNQP

Scoring Guidelines and Notes for the 2019 Exam Administration

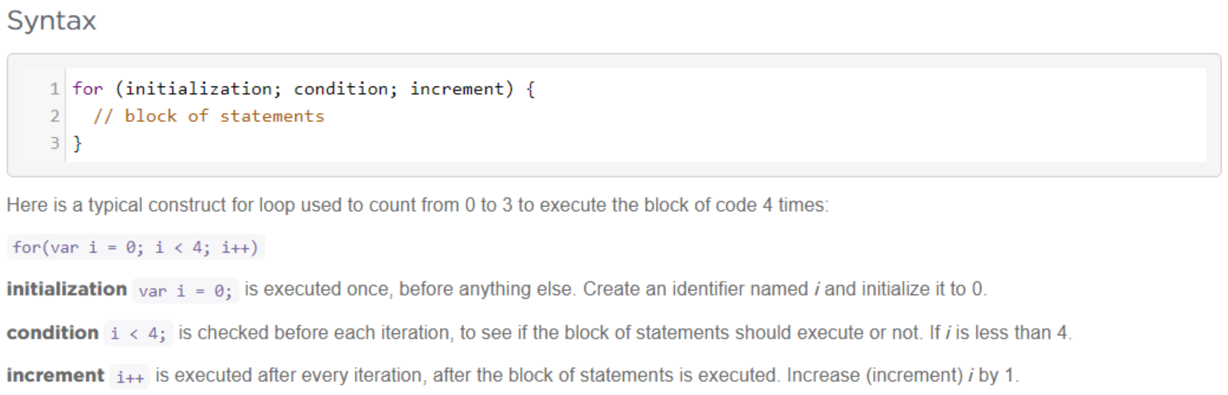
Explore – Sample Responses	Create – Sample Responses	Scoring Guidelines	Commentary
Explore A: Artifact Explore A: Written response Explore B: Artifact Explore B: Written response	Create A: Video Create A: Written response Create B: Video Create B: Written response Create C: Video	Scoring Guidelines Chief Reader Report	Explore: Commentary Create: Commentary Scoring Distribution

Explore – Sample Responses	Create – Sample Responses	Scoring Guidelines	Commentary
Explore C: Artifact Explore C: Written response Explore D: Artifact Explore D: Written response Explore E: Artifact Explore E: Written response Explore F: Artifact Explore F: Written response Explore G: Artifact Explore G: Written response Explore H: Artifact Explore H: Written response Explore I: Artifact Explore I: Written response Explore J: Artifact Explore J: Written response	Create C: Written response Create D: Video Create D: Written response Create E: Video Create E: Written response Create F: Video Create F: Written response Create G: Video Create G: Written response Create H: Video Create H: Written response Create I: Video Create I: Written response Create J: Video Create J: Written response		

TUTORING HELP!! https://www.khanacademy.org/computing/ap-computer-science-principles

https://www.khanacademy.org/computing/ap-computer-science-principles/ap-csp-exam-preparation#learn-ap-csp-exam-pseudocode

https://online-learning.harvard.edu/course/cs50s-introduction-game-development?delta=0 https://www.edx.org/course/using-python-for-research



Syntax

```
for (initialization; condition; increment) {
  // block of statements
}
```

Here is a typical construct for loop used to count from 0 to 3 to execute the block of code 4 times:

`for(var i = 0; i < 4; i++)`

initialization `var i = 0;` is executed once, before anything else. Create an identifier named *i* and initialize it to 0.

condition `i < 4;` is checked before each iteration, to see if the block of statements should execute or not. If *i* is less than 4.

increment `i++` is executed after every iteration, after the block of statements is executed. Increase (increment) *i* by 1.

AP CSP pseudocode

Assignment, display, and input

`a ← expression`

- Evaluates `expression` and assigns the result to the variable `a`.
- □Practice: Storing data in variables

`DISPLAY (expression)`

- Displays the value of `expression`, followed by a space.
- □Practice: Programming basics

`INPUT ()`

- Accepts a value from the user and returns it.

Arithmetic operators and numeric procedures

`a + b a - b a * b a / b`

- The arithmetic operators, +, -, *, and /, are used to perform arithmetic on `a` and `b`.
- □Practice: Mathematical expressions

`a MOD b`

- Evaluates to the remainder when a is divided by b. Assumes that `a` and `b` are positive integers.
- □Practice: Mathematical expressions

`RANDOM(a, b)`

- Evaluates to a random integer from `a` to `b`, including `a` and `b`.
- □Practice: Random numbers

Relational and Boolean operators

`a = b a ≠ b a > b a < b a ≥ b a ≤ b`

- The relational operators, =, ≠, >, <, ≥, and ≤ are used to test the relationship between two expressions, variables, or values.

- □Practice: Conditionals with if, else, and Booleans

```
NOT condition
```

- Evaluates to `true` if `condition` is `false`; otherwise evaluates to `false`.
- □Practice: Compound Booleans with logical operators

```
condition1 AND condition2
```

- Evaluates to `true` if both `condition1` and `condition2` are true; otherwise evaluates to `false`.
- □Practice: Compound Booleans with logical operators

```
condition1 OR condition2
```

- Evaluates to `true` if `condition1` is `true` or if `condition2` is `true` or if both `condition1` and `condition2` are true; otherwise evaluates to false.
- □Practice: Compound Booleans with logical operators

Selection

```
IF (<condition>) { <block of statements> }
```

- The code in `block of statements` is executed if the Boolean expression `condition` evaluates to `true`; no action is taken if `condition` evaluates to false.
- □Practice: Conditionals with if, else, and Booleans

```
IF (<condition>) { <first block of statements> } ELSE { <second block of
statements> }
```

- The code in `first block of statements` is executed if the Boolean expression `condition` evaluates to `true`; otherwise the code in `second block of statements` is executed.
- □Practice: Conditionals with if, else, and Booleans, Nested conditionals

Iteration

```
REPEAT n TIMES { <block of statements> }
```

- The code in `block of statements` is executed `n` times.
- □Practice: Numbered repetition of instructions

```
REPEAT UNTIL (condition) { <block of statements> }
```

- The code in `block` of statements is repeated until the Boolean expression `condition` evaluates to `true`.
- □Practice: Conditional repetition of instructions

List operations

For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates.

```
list[i]
```

- Refers to the element of `list` at index `i`. The first element of `list` is at index 1.
- □Practice: Storing and updating lists

```
list[i] ← list[j]
```

- Assigns the value of `list[j]` to `list[i]`.
- □Practice: Storing and updating lists

```
list ← [value1, value2, value3]
```

- Assigns `value1`, `value2`, and `value3` to `list[1]`, `list[2]`, `list[3]`, respectively.
- □Practice: Storing and updating lists

```
FOR EACH item IN list { <block of statements> }
```

- The variable `item` is assigned the value of each element of `list` sequentially, in order from the first element to the last element. The code in `block of statements` is executed once for each assignment of item.
- □Practice: Iterating over lists with loops

```
INSERT (list, i, value)
```

- Any values in `list` at indices greater than or equal to `i` are shifted to the right. The length of `list` is increased by 1, and `value` is placed at index `i` in `list`.
- □Practice: Storing and updating lists

```
APPEND(list, value)
```

- The length of `list` is increased by 1, and `value` is placed at the end of `list`.
- □Practice: Storing and updating lists

```
REMOVE(list, i)
```

- Removes the item at index `i` in `list` and shifts to the left any values at indices greater than `i`. The length of `list` is decreased by 1.
- □Practice: Storing and updating lists

```
LENGTH(list)
```

- Evaluates to the number of elements in list.

Procedures

```
PROCEDURE name (parameter1, parameter2, ...) { <instructions> }
```

- A procedure, `name`, takes zero or more parameters. The procedure contains programming instructions.
- □Practice: Defining a procedure, Procedures with parameters

```
PROCEDURE name (parameter1, parameter2, ...) { <instructions> RETURN
(expression) }
```

- A procedure, `name`, takes zero or more parameters. The procedure contains programming instructions and returns the value of `expression`. The `RETURN` statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling program.
- □Practice: Procedures with return values

Robot

If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.

```
MOVE_FORWARD ()
```

- The robot moves one square forward in the direction it is facing.
  `ROTATE_LEFT ()`
- The robot rotates in place 90 degrees counterclockwise (i.e. makes an in-place left turn).
  `ROTATE_RIGHT ()`
- The robot rotates in place 90 degrees clockwise (i.e. makes an in-place right turn).
  `CAN_MOVE (direction)`
- Evaluates to `true` if there is an open square one square in the `direction` relative to where the robot is facing; otherwise evaluates to `false`. The value of `direction` can be `left`, `right`, `forward`, or `backward`.
  □Practice of robot-like questions are throughout the Repetition lesson.

  There are many differences between the AP CSP pseudocode and the JavaScript language syntax.

  This table highlights the biggest differences:

Concept	**Pseudocode**	**JavaScript**
Assignment	`a ← expression`	`var a = expression;`
Equality	`a = b`	`a == b`
Inequality	`a ≠ b`	`a != b`
Repetition	`REPEAT n TIMES`	`for (var i = 0; i < n; i++)`
Repetition	`REPEAT UNTIL (condition)`	`while (condition)`
List index	`list[1]` is first item	`list[0]` is first item
List iteration	`FOR EACH item IN list`	`for (var i = 0; i < list.length; i++)`