

Session 1

Activity Packet

2019

Contents

[Pre-Training Reflection 3](#_Toc4144063)

[Memory Diagrams 4](#_Toc4144064)

Co-Teaching Model Worksheet [5](#_Toc4144065)

[Student-Centered Learning 6](#_Toc4144066)

[Learning Objectives 6](#_Toc4144067)

[Seeing the Pieces 9](#_Toc4144068)

[4 Steps to Solve Any (CS) Problem 12](#_Toc4144069)2

[1. Understand the problem 12](#_Toc4144070)2

[2. Make a plan 12](#_Toc4144071)2

[3. Implement the solution 13](#_Toc4144072)3

[4. Reflect on your work 13](#_Toc4144073)3

[Using the Four Steps to Solve Any Problem 14](#_Toc4144074)4

[Question Generator 15](#_Toc4144075)5

[The Misconception Game: Round 1 16](#_Toc4144076)6

[The Misconception Game: Round 2 18](#_Toc4144077)8

# Pre-Training Reflection

Take a few minutes to gather your thoughts and think about the experience ahead of you.

1. What do you think are the greatest assets you will bring to your classroom and teaching team?
2. What worries you most about participating in the program?
3. How does your own background influence how you may approach your work in the classroom? Consider your family’s ethnic or racial identity; your typical classroom experience growing up; and expectations your family had on interacting with authority figures.

Puzzle with 15 pieces put together in a horizontal rectangle that is 4 x 4 pieces.
Squares read, from top left to bottom right in rows first order: Thinking Styles, Language, Ethnicity, Religion, Perspectives, Experiences, Nationality, Job Level, Race, Culture, Skills, Gender, Physical Abilities, Sexual Orientation, Age.
Img. Source: data:image/jpeg;base64,

1. Consider the following quotation:

*“What matters is not what the teacher teaches, but what the student learns.”*

How does this statement relate to computer science? How does it relate to your own personal experiences learning computer science? How might it impact the way you work with your teaching team?

# Memory Diagrams

As a group, trace through the code sample below and create a memory diagram using the figure at right.

Identifiers

Values

1> name = “Denise”

2> age = 15

3> if age < 16:

4> canDrive = false

5> if age >= 16:

6> canDrive = true

7> output = name + “ is “

8> if not canDrive:

9> output = output + “not “

10> output = output + “eligible to drive.”

11> print(output)

Output

# Co-Teaching Models worksheet. Column headings: Model Name, Diagram, Description, When to Use, Risks/Challenges. Under Diagram graphic of one teach one support, team teach, parallel teaching, station teach, alternative teach

# Student-Centered Learning

As you participate in the lesson, in addition to focusing on the content, think about answers to these questions (write your thoughts and responses in your notebook):

1. How is the lesson **structured**? What parts or phases of the lesson can you identify? Roughly how much time was spent in each phase?
2. What methods for presenting new information do you notice? Which of these did you find most **engaging**? Why?
3. What types of **questions** are asked? How are these questions asked? How was someone chosen to answer the questions?

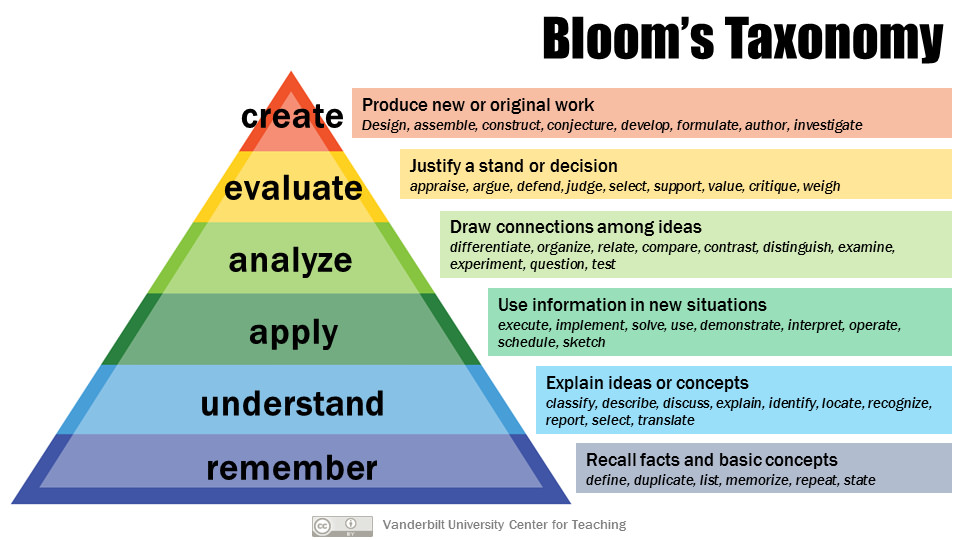
# Learning Objectives

For each of the following Learning Objectives in the table below:

1. Choose which level of Bloom’s Taxonomy you think the objective *best* applies to (see the graphic on the next page).
2. Choose which of the activities or tasks from this “Activity Bank” would be *most effective* for a lesson with the given objective.

|  |  |
| --- | --- |
| ***Activity Bank*** | |
| 1. Write a program/method individually | 1. Write a program/method as a class |
| 1. Trace the execution of a program/method as a class | 1. Observe a demo of a program without looking at code |
| 1. Act out a program/method in small groups | 1. Have a discussion/debate as a class or in small groups |
| 1. Write pseudocode in small groups | 1. Listen to a lecture/presentation |
| 1. Perform research online | 1. Read a textbook/article |

|  |  |  |
| --- | --- | --- |
| Learning Objective | Bloom’s Taxonomy Level | Effective Activity/ies |
| SWBAT explain how to use procedural decomposition to plan complex programs. (CSA Lesson 1.05) |  |  |
| SWBAT use simple conditional blocks to alter control flow. (Intro Lesson 2.3) |  |  |
| SWBAT write code that traverses a list. (Intro Lesson 4.4) |  |  |
| SWBAT give 2-3 examples of abstractions in everyday life. (CSP) |  |  |



This page intentionally left blank.   
(well, almost blank.)

# Seeing the Pieces

Look at the back of this page to find the given lesson plan for the curriculum you will be teaching.

Read that lesson plan and identify which part or activity corresponds to each of the following components of a standard TEALS lesson. If a part is missing, either give a reason why you think it is missing or provide a suggestion to incorporate.

|  |
| --- |
|  |
| *Warm-up:* |
| *Hook:* |
| *Instruction:* |
| *Practice:* |
| *Assessment:* |

**Intro: Lesson 2.2 – Nested Loops**

**APCSA: Lesson 2.02 – Declaring and Assigning Variables**

**AP CS Principles:**

**Beauty & Joy of Computing (BJC):** **Unit 2 Lab 2, Mandala Design (page 4)**

**Code.org**: **Unit 1, Lesson 7 – Encoding and Sending Formatted Text**

**UTeach CS Principles - Unit2 – Programming Repetition**

*NOTE: you will need to register with UTeach (follow the above link and select* [*student-facing curriculum*](https://docs.google.com/forms/d/1DJ0nAwum80z-EdLepK9zg0LSqagHpM5t4Hiz7jbZ8Ro/viewform?edit_requested=true)*) to access the curriculum material which is sent to you once registered. If you don’t have access to this or never registered with them, use one of the other AP CSP curricula.*

**Mobile CSP: Unit 5, Lesson 2**

*NOTE: you will need to login/register with MobileCSP to access. If you don’t have access, use one of the other AP CSP curricula above.*

Links to the Seeing the Pieces Exercise

*(please cross check them to see if you have right links)*

<https://aka.ms/LessonStructure-Intro>

<https://aka.ms/LessonStructure-APCSA>

<https://aka.ms/LessonStructure-BJC>

<https://aka.ms/LessonStructure-code>

<https://aka.ms/LessonStructure-UTeach>

<https://aka.ms/LessonStructure-Mobile>

# 4 Steps to Solve Any (CS) Problem

Adapted from George Polyá’s *How To Solve It* by Nathaniel Granor

## 1. Understand the problem

*At the end of this step, you should be able to restate the problem in your own words. Along the way you might need to draw a picture, research unfamiliar terms or syntax, observe a working demo, or ask clarifying questions.*

Can you restate the problem in your own words?

Can you draw a picture or diagram to make the problem clearer?

What are you asked to find or solve?

What can you assume to be true when your code is called (pre-conditions)?

What must be true when your code returns (post-conditions)?

What must remain true throughout code execution (invariants)?

Are there any unfamiliar terms or pieces of syntax that you should research?

Do you need to ask a question to get started?

## 2. Make a plan

*You can probably solve small problems in your head without thinking too much about them. But when it comes to bigger or harder problems, planning is the most important step! Your plan might be on paper or in a notebook. A complete plan usually takes the form of pseudocode or a diagram. Figuring out a correct plan sometimes seems like magic: you need that “special flash of insight” to see the way to a solution. But those flashes of insight come about by methodically applying questions and strategies like the ones below.*

Can you find a special case of this problem that you know how to solve?

Can you find and solve a mathematical equation that represents your problem?

Can you find a pattern in the input, output, or code you need to write?

Can you work backwards?

Can you work around the problem by creating a "stub" function or method that you implement later?

Can you think of an analogous problem that you know how to solve, or that others have solved?

Can you think of a related problem that has already been solved?

Can you find a more general problem that you can solve?

Can you derive a solution to your general problem by looking at a few specific cases?

Can you change/vary the problem to create a new problem whose solution would help you with the original problem?

Can you break the problem into smaller sub-problems and solve/recombine those problems?

Would the use of a well-known data structure help you solve the problem?

What *won't* work?

Is there a CS concept that would help you solve this problem?

Is there a library or method that would help?

What variables do I need?

Do I need any IF statements or LOOPs?

What data-types should I use?

Should I build an object?

Should my methods be public or private? Static?

What if I inherit behaviors from a superclass?

Did you use all the data? Did you use the whole condition? Have you taken into account all essential notions involved in the problem?

## 3. Implement the solution

*Once you have a correct plan, you can build the solution. Often, you’ll start building the solution only to learn that your plan wasn’t complete or correct. That’s OK, problem-solving is an iterative process. Sometimes you’ll run into translation issues: trying to figure out how to do in code what you wrote in words; other times, you’ll run into problems with your plan.*

Did you consider boundary cases?

Did you cover every possible code-path?

Are you using the correct conditions in conditional statements?

Are you using the right conditions/bounds in loops?

Do recursive functions have a base case?

Have you captured/respected the pre- and post-conditions?

Are your variables using the correct data-type?

Will your solution run in an acceptable amount of time/with acceptable memory use?

Have you reinvented a wheel? (Is there something already implemented that can replace part of your plan?)

Did you implement all the steps of your plan?

Does each piece of code actually do what your plan says it should do?

Can you explain what each line of code you've written does?

Did you include code comments to relate your code to your plan?

## 4. Reflect on your work

*Even after you’ve finished solving the problem and verifying your work, there’s an opportunity to learn more and prepare for the future by thinking about the solution you built and the process you used to get there.*

Can you check the result?  
Can you explain why it’s correct?  
Can you think of another way to solve this problem?

What other problems could you solve with this code?

What would you do the same/differently next time?

# Using the Four Steps to Solve Any Problem

*Write a program that calculates n factorial where n is between 1 and 10 inclusive.*

*If the number n is within range, calculate and output its factorial and exit the program.*

*If the number n is not in range, tell the user that the input was out of range and exit the program.*

You can apply the Four Steps to Solve Any Problem. Write or draw examples of the type of evidence that indicates completion of that step of the problem-solving process.

1. *Understand the Problem*
2. *Plan a solution*
3. *Implement the solution*
4. *Reflect on the solution*

# Question Generator

In your groups, you will work together to come up with three targeted questions that become progressively more specific, addressing the student’s problem. Then, write one actionable next step based on the questions.   
Once you’ve created one complete set of questions (Diagnosis, Leading Questions, Next Step) start over using a different approach.

|  |
| --- |
| **Intro and APCSP:**  The program is supposed to count down and say “Blast off!” when the count gets to 1. |
| Sample Snap code |

|  |
| --- |
| **AP CS A (Java)**:  The program is supposed to count down and say “Blast off!” when the count gets to 1. |
| for (int i = 10; i > 0; i++) {  System.out.println(i);  }  System.out.println(“Blast off!”); |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | **First Attempt** | **Second Attempt** | | **Diagnosis Question:** |  |  | | **Leading Question 1:** |  |  | | **Leading Question 2:** |  |  | | **Leading Question 3:** |  |  | | **Actionable Next Step:** |  |  | |
|  |

# The Misconception Game: Round 1

You and your partner will take turns pretending to be a student struggling with an assignment and a TA attempting to help. The TA will only have a few minutes to unblock the student. TA, don’t tell the student the answer and don’t worry about getting completely resolved; focus on giving the student a clear next step to try.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **AP CSP Instructions** | **Student Code** | **Student Code Output** |
| **AP CS Principles** | Draw an equilateral triangle with sides that are 75 pixels long. Remember that each interior angle of an equilateral triangle is 60o  *from Code.org 3.7; similar to MobileCSP 5.2 or BJC 1.3 (see below)* | Sample code.org app studio code | 33 sides of a hexigon |
|  | **TEALS Intro Instructions** | **Student Code** | **Student Code Output** |
| **TEALS Intro** | Draw an equilateral triangle when the number 1 is pressed on the keyboard. Remember that each interior angle of an equilateral triangle is 60°  *TEALS Intro Lab 1.3 (similar to CSP BJC 1.3)* | Sample Snap code | 3 sides of a hexigon |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **AP CS A Instructions** | **Student Code** | **Student Code Output** |
| **TEALS AP CS A** | Write a complete Java program to produce the following output.  \*\*\*\*\*  **\*\*\*\*\***  **\*\*\*\*\***  **\*\*\*\*\***  *Lesson 2.06,*  *Practice-It 2.4* | **public** **class** Square {  **public** **static** **void** main(String[] args) {  **for** (**int** i = 1; i <= 4; i++); {  **for** (**int** j = 1; j <= 5; j++) {  System.***out***.print("\*");  }  System.***out***.println();  }  }  } | \*\*\*\*\* |

**Student Thinking (ONLY STUDENT READS)**: **AP CSP & Intro**: I know an equilateral triangle has angles of 60o, and that’s how much I’m turning right but my turtle is clearly not drawing a triangle.   
**AP CS A**: It prints out five stars, so the j loop is working but I don’t know why it stops after only one line

**Student Mistake (ONLY READ AFTERWARDS)**:   
**AP CSP & Intro**: Student is using the interior angle (60o) to turn the turtle but should be using the exterior angle (180o - 60o = 120o).   
**AP CS A**: Student has a misplaced semicolon on the first for loop line

# The Misconception Game: Round 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Instructions** | | **Student Code** | | **Student Code Output** | | |
| **AP CS Principles** | Design an algorithm for drawing an equilateral triangle -- i.e., a triangle with equal sides and equal angles. Because this is another example of a repetition, you can use the for-each block in your algorithm. Because you might want to use your triangle algorithm again, define it into a procedure with a parameter.  *MobileCSP 5.2* | | | | | | |
| **TEALS Intro** *(Lab 3.2)* | Take the script you wrote in the last lesson that draws a square. Modify your script to ask the user input for a number of sides and a side length.  Part a) Adjust the script so that the size of the square is the size given by the user.  Part b) Adjust the script so that instead of a square, it draws a polygon with the numbers of sides given by the user. | | | Turtle from SnapSample Snap code | | | |
| **TEALS AP CS A** | Write a complete Java program to produce the following output.     /\       /\    /  \     /  \  +------+ +------+  |      | |      |  +------+ +------+  |United| |United|  |States| |States|  +------+ +------+  |      | |      |  +------+ +------+     /\       /\    /  \     /  \ | **public** **class** TwoRockets {  **public** **static** **void** main(String[] args) {  *printCap*();*printCap*();  *printBox*();*printBox*();  *printUS*();*printUS*();  *printBox*();*printBox*();  *printCap*();*printCap*();  }    **public** **static** **void** printCap() {  System.***out***.println("   /\\   ");  System.***out***.println("  /  \\  ");  }    **public** **static** **void** printBox() {  System.***out***.println("+------+");  System.***out***.println("|      |");  System.***out***.println("+------+");  }    **public** **static** **void** printUS() {  System.***out***.println("|United|");  System.***out***.println("|States|");  }  } | | | | Some output     /\    /  \     /\    /  \  +------+  |      |  +------+  +------+  |      |  +------+  |United|  |States|  |United|  |States|  +------+  |      |  +------+  +------+  |      |  +------+     /\    /  \     /\    /  \ |

**Student Thinking (ONLY STUDENT READS)**: **AP CSP:** Whenever I call triangle, it doesn’t matter if I give it 100 or 20, it’s always the same size. **Intro**: I made this polygon block so I could change the size and the number of sides, but no matter what numbers I put in, it’s not working right. Here it should be a size 50 square, but it’s not.   
**AP CS A**: I printed the rocket twice, but it didn't show up like it should.

**Student Mistake (ONLY READ AFTERWARDS)**:   
**AP CSP:** Student put in a variable but is not using it in the function.  
**Intro**: Student is moving “sides” instead of “size” also should be using “repeat (sides)” not repeat(4). Moving forward, they also need to think about how they’ll make this work correctly for any number greater than 2.   
**AP CS A**: Students put functions side by side thinking that's how the picture would show up.