Create PT Survival Guide[1] [2]

Create PT Overview

Goal of the Task: Create a programming project of your own design and then explain the **purpose, process, algorithms and abstractions** used to build it. You have **12 hours** to complete this task.

What you Submit: (1) Video of running program (2) Written Responses to prompts 2a-d (3) PDF of program code

How you get a good score: The AP committee wants to see that you can:

- Design and write the code for a computer program with a topic of your choosing
- Describe how you identified and solved problems as you developed your program
- Write code for an algorithm and describe its purpose within your program
- Write code for an abstraction and describe its purpose within your program

Suggested Process in a Nutshell (see also: Sample Timeline on following pages)**:**

Pick your project… Something small enough that you can design and write it in only a few hours. You should basically know ahead of time what the core algorithm will be

Hours 1-2 **Develop a "good enough" program / prototype within 2 hours**

- ❑ Evaluate whether you can finish what you set out to do, and adjust as necessary.
- ❑ Check progress for responses 2c and 2d - algorithms and abstraction
- ❑ You should **know** what **your algorithm and abstraction** will be at this point.

Hours 3-7 **Keep coding and get to a stopping point with ~5 hours to go**

- ❑ Your video and written responses will take time to make - you'll want to make last minute improvements

Hour 8 **Record your video and respond to 2a**

Hours 9-10 **Write responses to 2b, 2c, 2d**

Hours 11-12 **Prepare to submit**

- ❑ Finalize program code and written responses and submit on the digital portfolio.

[1] *Much of the content of this this guide was inspired by Jill Westerlund at the Abstracting CS blog. We are grateful for Jill's ingenuity and and generosity.*

[2] *The code.org version of this survival guide was edited / adapted by Katie O'Shaughnessey to use content created by Harvard's CS50 and Margaret Tanzosh and apply it to the CS50 AP curriculum.*

Algorithms on the Create PT (20 mins)

Is It a Good Algorithm?

Algorithm - College Board Definitions: Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages. Algorithms make use of sequencing, selection or iteration. People write programs to execute algorithms. Your algorithm must include two or more algorithms that work in combination to achieve some desired result. Finally it must integrate mathematical and/or logical concepts.

What it means: Algorithms are chunks of code that accomplish a task. To make sure your algorithm is a little complex and that you demonstrate your programming abilities the College Board has a few specific requirements.

- **Something You Wrote:** The entirety of the code you submit as your algorithm should be something that you wrote entirely on your own. You cannot submit code that a partner helped you write.

- **Mathematical and/or Logical Concepts:** Mathematical concepts means code where your app does some sort of mathematical computation (+, -, *, /, %). These are typically used when your program is making a calculation.

  Logical concepts means code where you use logical or comparison operators (&&, ||, !, ==, !=, <, > <= >=). These are typically used in combination with if-statements where your program is making a decision.

- **A Parent and Two Children:** Your algorithm must have two "included" algorithms that work in combination to achieve some result. The best way to think about this is that you have a "parent" algorithm that requires two or more components that can stand on their own as independent "child" algorithms.

  You should NOT submit three separate algorithms. Instead you should find one large task in your program that can actually be divided into two or more sub-tasks (for example: a large task might be making a user login screen; subtasks are (1) checking their login information (2) updating the screen appropriately). The code to accomplish each sub-task are the "child" algorithms that can work independently to solve each sub-task, but can are combined to work together as a "parent" algorithm to solve the overall task.

- **Talking About Your Algorithm:** It is recommended that you place your main algorithm and included algorithms in separate functions. This will make it easier for you to talk about your algorithms in your responses. You may also, however, simply place rectangles around them and then use line numbers to identify your algorithms.

Does It Count? - Algorithm Edition

The AP reader has to judge strictly from the code you include in your response to 2c whether it meets the requirements. They will assume that all of the variables and function declarations that the code refers to exist, and that the code is the working code from your video.

Row 6 - Response 2C: The points for Row 6 of the scoring guidelines are awarded strictly for the code segment selected.

Criteria	Decision Rules	Scoring Notes
Selected code segment implements an algorithm that includes at least two or more algorithms. **AND** At least one of the included algorithms uses mathematical or logical concepts. **AND** Explains how one of the included algorithms functions independently.	**Do NOT award a point if any one of the following is true:** • the selected algorithm consists of a single instruction; • the selected algorithm consists solely of library calls to existing language functionality; • neither of the included algorithms nor the selected algorithm that includes two or more algorithms uses mathematical or logical concepts; • the code segment consisting of the algorithm is not included in the written responses section or is not explicitly identified in the program code section; or • the algorithm is not explicitly identified (i.e., the entire program is selected as an algorithm, without explicitly identifying the code segment containing the algorithm).	• Algorithms make use of sequencing, selection or iteration. • Mathematical concepts include mathematical expressions using arithmetic operators and mathematical functions. • Logical concepts include Boolean algebra and compound expressions. • Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times. • Selection uses a Boolean condition to determine which of two parts of an algorithm is used.

You be the AP Reader! You are are the AP reader trying to determine if they get the point for Row 6. Assume each algorithm snippet below was submitted as part of written response 2c. For each, select yes or no - should the point be awarded, and explain why. (It doesn't have to be a written explanation. You can draw an arrow or circle something in the code with a brief word or label about why.)

Example Algorithm 1

```
6     // prompt user for x
7     int x = get_int("x is: ");
8
9     // prompt user for y
10    int y = get_int("y is: ");
11
```

Earn Point? Yes / No

Why?

No. Take your pick of criteria this doesn't meet. No math. No logic. Basically single instruction. No interrelation between them.

Example Algorithm 2

```
1 import cs50
2
3 f = cs50.get_float()
4
5 c = 5 / 9 * (f - 32)
6
7 print("{:.1f}".format(c))
```

Earn Point? Yes / No

Why?

No. This alone does not have two subcomponents that come together to form a single, more complex algorithm. The use of calculations for c would likely meet the bar for mathematical concepts, but there are not two clearly articulated sub-algorithms making up this larger algorithm.

Example Algorithm 3

```c
#include <cs50.h>
#include <stdio.h>

void cough(int n);
void say(string word, int n);
void sneeze(int n);

int main(void)
{
    for (int i =0 ; i < 3; i++)
    {
        say ("Hoi! I\'m Temmie!", 1);
        say ("and this is my friend, Temmie.", 1);
    }
    say ("Hi. I'm bob.", 1);
    cough(3);
    sneeze(3);
}

void cough(int n)
{
    say("cough", n);
}

void say(string word, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%s\n", word);
    }
}

void sneeze(int n)
{
    say("achoo", n);
}

```

Earn Point? Yes / No

Why?

Probably. The main algorithm could be "main", which has at least two included algorithms. Should you choose to use “say” and “sneeze” as the two inner algorithms, that would likely earn this point. "Say" includes logical concepts because it has a parameter and a loop that uses the parameter. "Sneeze" calls "say" and uses the parameter, which indicates another piece of logic. This would need to have a great explanation along with it to earn this point.

Example Algorithm 4

```c
#include <stdio.h>

int decrement_passed_by_value(int a);

int main(void)
{
    int a = 4;
    a = decrement_passed_by_value(a);
    printf("Passed by value: %d\n", a);

}

int decrement_passed_by_value(int a)
{
    a--;
    return a;
}
```

Earn Point? Yes / No

Why?

No. While the decrement_passed_by_value function might qualify as having mathematical or logical concepts because it is a function with a parameter and uses subtraction, there are not two separate and distinct algorithms in this snippet of code that make up this larger algorithm.

Example Algorithm 5

```c
    // Add each semitone
    for (int i = 0, n = sizeof(NOTES) / sizeof(string); i < n; i++)
    {
        // Append octave to note
        char note[4];
        sprintf(note, "%s%i", NOTES[i], octave);

        // Calculate frequency of note
        int f = frequency(note);

        // Print note to screen
        printf("%3s: %i\n", note, f);

        // Write (eighth) note to file
        note_write(s, f, 1);
    }
```

```c
// Calculates frequency (in Hz) of a note
int frequency(string note)
{
    // TODO
    //printf ("DEBUG: note = %s\n", note);
    int lenNote = strlen(note);
    char letter;
    char sharpFlat = '-';
    int octave;
    float freq = -1;
    // Parse note into note, octave and sharp/flat
    // if string contains 3 characters
    if (lenNote == 3)
    {
        letter = note[0];
        sharpFlat = note[1];
        // get digit equivalent for octave
        octave = note[2] - '0';
        if (!(sharpFlat == '#' || sharpFlat == 'b'))
        {
            // not correct symbol for sharp or flat
            eprintf("INCORRECT SHARP OR FLAT\n");
            return -1;
        }
        //printf("DEBUG: letter = %c, sharpFlat = %c, octave = %i\n", letter, sharpFlat, octave);

    }
    else if (lenNote == 2)
    {
        letter = note[0];
        // Find out how many characters past 0 the octave is
        octave = note[1] - '0';
        //printf("DEBUG: letter = %c,  octave = %i\n", letter, octave);
    }
    else
    {
        eprintf("wrong note input\n");
        // incorrect input for number
        return -1;
    }

    // parse A for each octave knowing A4 = 440
    // round to nearest integer
    switch (letter)
    {
```

Earn Point? Yes / No

Why?

Yes. The main algorithm is the for loop - lines 41-55. It repeatedly reads a file to get a note, and then calls the frequency function. This function is a smaller algorithm within the bigger algorithm that contains logic. The loop then calls note_write, another function that contains logic. While this code is a bit more complex than required, it is an example of a very strong algorithm that contains two smaller algorithms, which each have a clear purpose, and work together to achieve a larger purpose together.

We think that breaking algorithmic tasks into separate, named functions will help the writing process, but it should be clear that is not a requirement, only a recommendation.

The rest of frequency has been omitted for length…

```
// Append note to end of array of notes in song
// Each duration unit is an eighth note at 120bpm
bool note_write(song s, int frequency, int duration)
{
    // Increase size of song if necessary
    if (s->size == s->capacity)
    {
        if (s->capacity > SIZE_MAX / sizeof(*s->notes) / 2)
        {
            return false;
        }
        s->capacity *= 2;
        note **temp = realloc(s->notes, sizeof(*s->notes) * s->capacity);
        if (!temp)
        {
            return false;
        }
        s->notes = temp;
    }

    // Create a new note
    note *n = calloc(1, sizeof(note));
    if (!n)
    {
        return false;
    }
    n->frequency = frequency;
    n->duration = duration;

    // Add note to song
    s->notes[s->size] = n;
    s->duration += duration;
    s->size++;

    return true;
}
```

Abstraction on the Create PT (20 mins)

Is It a Good Abstraction?

Abstraction - College Board Definitions The AP course framework includes the following statements related to abstraction and programming that are relevant for the Create PT:

- *Multiple levels of abstraction are used to write programs (EU 2.2)*
- *The process of developing an abstraction involves removing detail and generalizing functionality. (EK 2.2.1A)*
- *(Student can) Use abstraction to manage complexity in programs. (LO 5.3.1 [P3])*

What it means: Abstractions manage complexity. Programming abstractions make it easier to write complex programs. The AP reader is checking that you can **create, identify, and describe an abstraction** that manages complexity in your code.

- **Something You Wrote:** The code you submit as your abstraction should be something that you wrote entirely on your own. You cannot submit code that a partner helped you write.

- **Good Abstraction Choice - Functions that you wrote:** A function that you wrote means a function that you defined, named, and wrote code for. (i.e code that starts `void myFunc(){ ... }` or `def myFunc())`. Your function can demonstrate *managing complexity* if it either (or both):
  - gets called from multiple different places in your code
  - has a parameter that generalizes some behavior (and also probably called from several places)

- **Good Abstraction Choice – Class/Struct that you wrote:** A class that you wrote means a class that you defined, named, and wrote code for. (i.e code that starts `typedef struct {…}` or `class MyClass:…)`. Your class/struct can demonstrate *managing complexity* if it either (or both):
  - gets used in multiple different places in your code
  - has a fields or member variables that simplify data storage and access

- **Bad Abstraction Choice: built-in programming language/environment features**
  - **Variables** by themselves are not a data abstraction
  - Anything that is an existing abstraction provided by the language that you are simply using. For example: loops, logical structures, and functions in the cs50 library or graphics.py library are not student developed abstractions

Does It Count? - Abstraction Edition

The AP reader has to judge strictly from the code you include in your response to 2d whether it meets the requirements. They will assume that all of the screen elements and variables the code refers to exist, and that the code is the working code from your video.

Row 7 - Response 2D: The points for Row 7 of the scoring guidelines are awarded strictly for the code segment selected.

Criteria	Decision Rules	Scoring Notes

• Selected code segment is a student-developed abstraction.	Responses that use existing abstractions to create a new abstraction, such as creating a list to represent a collection (e.g., a classroom, an inventory), would earn this point.
	
	Do NOT award a point if any one of the following is true:
	• the response is an **existing** abstraction such as variables, existing control structures, event handlers, APIs;
	• the code segment consisting of the abstraction is not included in the written responses section or is not explicitly identified in the program code section; or
	• the abstraction is not explicitly identified (i.e., the entire program is selected as an abstraction, without explicitly identifying the code segment containing the abstraction).
		• The following are examples of abstractions (EK 5.3.1):
		  ○ Procedures
		  ○ Parameters
		  ○ Lists
		  ○ Application program interfaces (APIs)
		• Libraries
		• Lists and other collections can be treated as abstract data types (ADTs) in developing programs. (EK 5.5.1I)

Evaluate Abstractions! Each of the code segments below shows a portion of code with a rectangle around it. Sometimes additional code is included to help you understand the context of that abstraction, for example to determine whether the abstraction helps manage complexity.

For each example below respond to 3 things:

- **Earn Point? Yes / No** - Would the selected code segment earn the point for row 7?
- **Why?** - note why it does or doesn't earn the point
- **Manages Complexity? Yes / No** - based on what you can see, are you able to argue that the abstraction manages complexity?

<table>
<tr>
<td>

Example 1

```
26      // Expect note
27      string line = get_string("");
```

</td>
<td>

Earn Point? Yes / No

Why?

No. Not a student-developed abstraction.

Manages Complexity? Yes / No

No, code is used in a single place in program. It is calling a function developed by cs50, not the student.

</td>
</tr>
<tr>
<td>

Example 2

```
7       if (i < 0)
8       {
9           printf("negative\n");
10      }
```

</td>
<td>

Earn Point? Yes / No

Why?

No. Not a student-developed abstraction. This is the same as the previous example even though it includes an if-statement.

Manages Complexity? Yes / No

No, this code is only used in this single portion of the program, and printf is not a student-created function.

</td>
</tr>
<tr>
<td>

Example 3

```
35      // Check if line is rest
36      if (is_rest(line))
37      {
38          rest_write(s, 1);
39      }
```

</td>
<td>

Earn Point? Yes / No

Why?

No. The code identified in the rectangle doesn't show a student-developed abstraction. It shows a call to what might possibly be a student-developed abstraction (e.g. rest_write(s, 1);) but since those weren't inside the rectangle you don't get credit for them.

Manages Complexity? Yes / No

The functions referenced MAY manage complexity, but the code in the rectangle DOES NOT. This student captured the wrong thing.

</td>
</tr>
</table>

Example 4

```
124 // Determines whether a string represents a rest
125 bool is_rest(string s)
126 {
127     /* FROM GET_STRING cs50.h: If user
128      * inputs only a line ending, returns "", not NULL. Returns NULL
129      * upon error or no input whatsoever (i.e., just EOF). Stores string
130      * on heap, but library's destructor frees memory on program's exit.
131     */
132     if (strcmp(s, "") == 0)
133     {
134         //printf ("DEBUG: IN is_rest returning true\n");
135         return true;
136     }
137     return false;
138 }
139
```

```
35         // Check if line is rest
36         if (is_rest(line))
37         {
38             rest_write(s, 1);
39         }
```

Earn Point? Yes / No

Why?

Yes. The code outlined in red is a student-developed abstraction. The function call to is_rest is in the code below it. Assuming is_rest was written by the student, it would be considered a student-developed abstraction. It contains logical components due to the if statement, and is a new function defined by the student.

Manages Complexity? Yes / No

It helps to manage complexity because it is a parameterized function, that can be called with a variety of values.

Example 5

```
1  class Dictionary:
2      def __init__(self):
3          self.words = set()
4      def check(self, word):
5          return word.lower() in self.words
6
7      def load(self, dictionary):
8          file = open(dictionary, "r")
9          for line in file:
10             self.words.add(line.rstrip("\n"))
11         file.close()
12         return True
13
```

Earn Point? Yes / No

Why?

Yes. The student has defined a Dictionary class, which is inside the rectangle. The class contains three methods that can be run on any instances of the class.

Manages Complexity? Yes / No

If the student uses this class throughout the code to add words to the dictionary, this does manage complexity.

Example 6

```
1 class Student:
2     def __init__(self, name, dorm):
3         self.name = name
4         self.dorm = dorm
```

Earn Point? Yes / No

Why?

Yes. The student developed abstraction is inside the rectangle. The Student class is an abstraction. Furthermore the written response can make reference to places in the code where the class is used with different parameters to demonstrate how it manages complexity.

Manages Complexity? Yes / No

Yes. The snippet of code that is not in red is where this Student class is used, which demonstrates how having a Student class allows the program to have a list of students.

```python
import csv
from student import Student

students = []

for i in range(3):
    print("name: ", end="")
    name = input()

    print("dorm: ", end="")
    dorm = input()

    students.append(Student(name, dorm))

```

"Narrow it Down" (20 mins)

You should assume that you're not going to have enough time to complete the "perfect" project for the Create PT. This is ok because **you can get full credit for a programming project that feels incomplete**. While a large or more complete project is satisfying, your score is based mostly on the written responses which is how you demonstrate that you understand the concepts covered on the Create PT. All your project actually needs to include is:

- One working feature that you can demonstrate in your video
- An algorithm
- An abstraction

You will make the project much easier if you narrow down your original idea into a simpler target project. This will give you more time to complete the written responses or make smaller improvements.

How to Narrow It Down: Narrowing it down means identifying the sub-tasks or sub-problems that meet the PT requirements. Here's a few strategies to help you do that:

- **Get to the Algorithm:** Split your project into individual components that will likely require an algorithm that meets the Create PT requirements. Each of these components individually could be your entire project.

- **Pick One Part of a Bigger Idea:** Think about your project as a set of programming tasks that each solve part of a larger problem. Some of these individual tasks might suffice for the Create PT. You can just pick one part of a big idea that meets the requirements that you think you can program in the time allotted.

- **Minimal Design Mode - looks don't matter:** Complex visual design work (setting colors, fonts, spacing, etc.) will likely NOT meet any of the requirements for the Create PT. Don't worry about how your app looks until after you already have code that will let you complete the written responses.

Practice Narrowing It Down

Below are three descriptions of potential projects that another CS Principles student is considering. For each write:

- Two or three ways they could narrow down the project using the tips above
- Opportunities to write an algorithm in their project even after it's been narrowed down.

Project 1: Tic-Tac-Toe

"Here's my idea: I want to build a tic-tac-toe game. The user creates an account if they don't already have one and are taken to the main game board. From there the player will play against the computer in either easy, intermediate, or advanced mode, so I will need to write the code for the computer player. When the game is over their lifetime win total is updated. I will also keep track of how long the game took."

Ways to narrow down the project (2 or 3)	Algorithm opportunities
• *Build just the login screen* • *Build an app that can just detect a win/loss/continue condition in tic tac toe* • *Design a way to keep track of scores over time* • *Demonstrate a single example of how a computer would choose its next move based on a board setup.* • *Don't use tic-tac-toe, but a simpler game (pick a number)* • *Build the user tools that would appear around the game, not the game itself.*	*Possible algorithm choices:* • *verifying a user is in a list of accepted users.* • *Given a way to represent the state of a tic-tac-toe board (variables, lists, UI elements) write code to determine if the game is over or should continue. If the game is over did someone win? A draw?* • *Write code to determine if, say, X's is one move away from a win based on rows, and indicate that somehow (could do for columns and diags too).* • *Making a high score screen that maintains lists of player names and scores, that can be displayed, or ordered differently, or searched.*

Project 2: Health App
"I volunteer at my local health clinic so I want to build a health app. The user can record information about what they eat, how much they sleep, how much they exercise, and information like their blood pressure and weight. Based on the information provided the app will provide recommendations to the user about how they can improve their health for both diet and exercise. Users can also personalize the look of the app with different theme colors."

Ways to narrow down the project (2 or 3)	Algorithm opportunities
• *Don't track as much information. Ask for a couple pieces of information and make a single recommendation.* • *Don't personalize themes or colors* • *Only provide recommendations for Diet or for Exercise, there's no need to provide recommendations for both.* • *Focus on the core recommendation portion of the app first. Don't worry about other screens until that piece of code is written.*	*The algorithms will likely be the part of the app that uses the users' health information in order to make recommendations. Some kind of mathematical or logical concept will almost certainly need to be used to parse this user information.*

Project 3: Sports Stats
"I think that I'll build an app that allows the user to quickly record stats during a basketball game. The app will show a picture of the court. The user taps on the court to indicate something happened there. They are presented with a quick menu of options like: shot attempt, foul, steal, rebound, etc. then they select from another list which player did it. At the end of the game it displays a stat sheet for all of the players and the stats for that game."

Ways to narrow down the project (2 or 3)	Algorithm opportunities
• *Only track a single statistic at first (e.g. shots, or fouls) as a proof of concept.* • *Keep the total number of statistics simple (e.g. just a count of shots)* • *Only keep track of statistics for one team, or even on one side of the court (e.g. offense or defense).* • *Keep the app to two screens. One for showing shots/fouls/etc. so far and one for showing summary stats.* • *Avoid adding extra features at first, like an ability to delete statistics added incorrectly.*	*You may need an algorithm for updating a list of stats recorded thus far. This algorithm could both add the new stat to the list and keep track of the ongoing summary statistics.* *You may need an algorithm to calculate and display summary stats at the end of the game.* *Any code in which you're managing a list (in this case of players, stats, etc) is a good candidate for algorithms, especially if you have some feature that requires processing the list in some fashion - searching for a player, counting something, and so on.*

Bring It All Together

With an understanding of the major components of the Create PT you are ready to start brainstorming projects. While you have at least 12 class hours to complete the task, keep in mind that those 12 hours also must account for time to make your video and complete the written responses. We recommend budgeting at least 5 hours to complete the video and written responses, and so it is highly recommended that you prepare for doing a project in which the programming / coding can be completed in 6-7 hours. You want projects with the following features.

- **Personally Relevant:** Pick projects you'll actually be interested in building.
- **Clear Purpose:** Aim for a simple program whose purpose can be stated in one sentence. For example:
  - The purpose of my program is _____.
  - My app does _____.
  - My program lets a user _____.
  - My app is a _____.
- **Narrowed Down:** Repeat the "Narrow It Down" process with your own ideas. A good rule of thumb is that you'll want to be able to have a first draft of your algorithm within two hours of starting to program.
- **No New Programming Skills:** Make sure you *already* have the programming skills necessary to complete the project. Be flexible. With some creativity you can likely use the skills you've already learned to make many different types of projects. Avoid taking on new programming environments or concepts as part of the Create PT.

Brainstorm Project Ideas

Brainstorm one or two project ideas for the Create PT. Afterwards you'll share ideas with a classmate for feedback.

Project Idea	Classmate Feedback
Purpose: *Ways to Narrow It Down:* *Algorithm Opportunities:* *Confidence you have skills to do this it in time allotted?:*	*Use the list above to give feedback on the idea.*
Purpose: *Ways to Narrow It Down:* *Algorithm Opportunities:* *Confidence you have skills to do this it in time allotted?:*	*Use the list above to give feedback on the idea.*

Create PT Progress / Check-in organizer

2

Program **purpose**
(remember: your video should show this)

Row 1

2

Development **process** overall

Row 2

Difficulty/opportunity #1
Circle one: feedback • testing • reflection
(how it was resolved, incorporated into project)

Row 3

Difficulty/opportunity #2
Circle one: feedback • testing • reflection
(how it was resolved, incorporated into project)

Row 3

2

Identify main **algorithm** (name of function, location in code, etc.)

Rows 4, 5

How does it function? (explain any mathematical or logical concepts used).

Row 5

How does it relate to overall purpose?

Row 5

Algorithm Includes two or more algorithms….

Sub-algorithm 1

Sub-algorithm 2

Explain how at least one of the two (1) uses Mathematical or Logical concepts (2) can explain how it functions independently.

Row 6

2

Your developed **abstraction**
(name of function(s) *you wrote*)

How does it manage complexity?

Examples:
• Able to reuse functionality?
• Problem broken down into functions and sub-functions?
• Generalizes specific behavior?

Rows 7,8

This organizer is inspired by the work of Jill Westerlund at abstractingCS.com. Recreated and modified with permission

Create PT Completion Timeline

Before you start you should think about how you are going to allocate your time for the 12 hours provided for the task. Below is a sample timeline that you can use to plan out how you will complete the Create Performance Task.

Hour	Suggested Activity	Your Plan
1 - 2	Begin building a program for a project you brainstormed. Carefully monitor whether you will finish enough of your project in time. Write down one opportunity or difficulty you've encountered for prompt **2b**. **Goal:** you should be confident after this first round of development that you'll be able to meet the requirements for algorithms (2c) and abstraction (2d). **Use the *Create PT Progress / Check-in organizer* to test yourself about whether you are on track.**	
3 - 4	Keep working. Check in after hour 4 once again on whether you are on track to complete responses. You should ideally know: ● The abstraction you will write about ● The algorithm you will write about Write down a second opportunity or difficulty you've encountered for prompt **2b**. **Use the *Create PT Progress / Check-in organizer* to test yourself about whether you are on track.**	
5 - 7	Finalize all programming. After this point you shouldn't be writing more code (beyond simple touch ups)	
8	Complete prompt **2b**, using the notes you took as your were programming.	
9	Record **video** of your program running and complete response **2a**	
10	Complete **2c** describing your algorithm	
11	Complete **2d** describing your abstraction	
12	Complete the task ● Review the submission materials ● Check your responses against the scoring guidelines ● Enter your responses into the digital portfolio ● Upload your computational artifact (and/or PDF of written responses to the digital portfolio) **Goal:** At the end of this day, your Create PT is submitted!	

Note: The timeline above is just a guideline. You may complete the performance task on a different schedule. Make sure to leave enough time to complete your computational artifact and write-up.

Written Response Templates

Video Submit one video in .mp4, .wmv, .avi, or .mov format that demonstrates the running of at least one significant feature of your program. Your video must not exceed 1 minute in length and must not exceed 30MB in size

Prompt 2a. Provide a written response or audio narration in your video that:
- identifies the programming language;
- identifies the purpose of your program; and
- explains what the video illustrates.

(Must not exceed 150 words)

Advice: For resources on how to make your video head to https://studio.code.org/s/csp-create/stage/1/puzzle/2. Here's the most important things to remember for your video and prompt 2a.

- **Video Runs Continuously:** Your video must run continuously and show your actual code running. It can't just be a series of screenshots.
- **Show One Feature:** Your program does NOT need to be complete so long as you can demonstrate one major feature that's running.
- **Describe the Purpose:** The purpose of your program is the intended goal or objective of the program. In other words, it's "what" the program is supposed to do. If you made a game, an app, or some other kind of project, just quickly describe "what" kind of program it is and how it would be used / played.
- **Connection to Video:** Make sure that you can connect the purpose of your program to what is shown in the video. If you only have one feature working then describe the purpose of the feature.

Sentence Starters
- *My program is written in C / python using the cs50.io programming environment.*
- *The purpose of my program is… (describe what it was meant to do)*
- *In the video you can see (how a piece of functionality works that's directly tied to the purpose).*

Draft Your Response Here:

(must not exceed 150 words)

Response 2a Checklist

Video
- ❑ Video runs continuously (it cannot be a series of screenshots)
- ❑ Video is less than 60 seconds long and less than 30MB in size
- ❑ Video demonstrates one running feature of the program

Written Response
- ❑ Response identifies the programming language used
- ❑ Identifies the purpose of the program
- ❑ Describes the feature(s) shown in the video and their connection to the purpose of the program
- ❑ May be audio commentary in your video. Carefully follow this checklist even if you use audio commentary.

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and / or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. *(Must not exceed 200 words)*

Advice: There are *many* individual pieces of information you need to fit in this response. Use the checklist at the bottom carefully to make sure you don't miss any. Here's the most important pieces to remember:

- **Don't Forget the Overall Process:** Your response should reference your development process as a whole, NOT just the two points in time. Your response should reflect an iterative process of identifying and solving problems.
- **Difficulties / Opportunities:** You need to describe *two* distinct difficulties / opportunities you encountered
  - A **difficulty** is likely either a bug in your code or a difficult design problem you needed to work out.
  - An **opportunity** is likely an idea or realization you had as you developed your code.
- **Feedback / Testing / Reflection:** You should clearly describe HOW you identified the two difficulties / opportunities - was it through testing your program out? Personal reflection? Through feedback from a peer?
- **Independent or Collaborative:** If you developed your program completely independently, you need to say so - don't assume the reader will know. If you developed your program collaboratively, some parts need to be done on your own. For this response *make sure*:
  - You clearly indicate which parts were done collaboratively
  - You clearly state which of the difficulties/opportunities described here was done *independently on your own* (could be both, but at least one).

Sentence Starters

- *I completed this project (independently / collaboratively).*
- *I began developing my program by (describe beginning of process, what did you do / decide first)*
- *Then I worked iteratively to (how you continued building your project)*
- *Early on through (feedback / testing / reflection) I identified a (problem / opportunity) which was …*
- *I solved this problem (independently / with my partner) by …*

Draft Your Response Here:

(must not exceed 200 words)

Response 2b Checklist

Overall Development

- ❑ Response describes the *overall* development process, *not only* two key points.
- ❑ Response indicates whether you completed the project independently or with a partner. (note: this indication can be incorporated throughout your response *and* in comments within your code as well).

First Difficulty / Opportunity

- ❑ Response describes one difficulty / opportunity encountered early in the development process
- ❑ Response describes source of difficulty / opportunity as either feedback, testing, or reflection
- ❑ Response indicates how it was incorporated / solved, including whether you wrote the code independently.

Second Difficulty / Opportunity

- ❑ Response describes one difficulty / opportunity encountered later in the development process
- ❑ Response describes source of difficulty / opportunity as either feedback, testing, or reflection
- ❑ Response indicates how it was incorporated / solved, including whether you wrote the code independently.
- ❑ If first Difficulty / Opportunity WAS NOT solved independently, then this one must be

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3** below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Must not exceed 200 words)*

Advice: Review the "Is It a Good Algorithm" section above for lots of helpful tips on how to choose your algorithm. Here's the most important points.

- **You Wrote It:** You need to have written the code of your algorithm entirely on your own (not with a partner)
- **Copy and Paste It:** You must paste your actual algorithm code as part of this response.
- **A Parent and Two Children:** Your main algorithm (the "parent") needs to have two sub-algorithms (the "children"). See Example Algorithms 4 and 5 for ideas on how this might look.
- **Mathematical / Logic Concepts:** At least one sub-algorithm needs to use mathematical and/or logical concepts.
- **Break Into Functions:** To make it easier to refer to individual parts of your algorithm give the parent and child algorithms their own functions. Example Algorithm 4 is written in this way.
- **Describe "how", not just "what":** You need to talk about how your code works, not just what the user will see when it runs. Do this by referring to the actual variables names, programming constructs, strings, and so on, that are visible in your code snippet. For example:

> *"The algorithm I selected is signInUser() which handles the user login process in my app which has two key parts: checkName() and startHomeScreen(). checkName has an if-statement that checks to see whether the parameter usernameTxt is equal to 'MrSillyMan' or 'MsFunnyGal'. If it is then it sets the accessGranted variable to true, otherwise false. The startHomeScreen() function checks the accessGranted variable and returns the login screen if false, otherwise it proceeds to show the home screen for the user."*

Sentence Starters

- *The main algorithm that I selected is (main-algorithm name).*
- *This algorithm has two key parts, (sub-algorithm 1) and (sub-algorithm 2).*
- *(Sub-algorithm 1) is designed to (what it does). It does this by (how the code of sub-algorithm 1 works).*
- *My (main-algorithm) combines (sub-algorithm 1) and (sub-algorithm 2) to (what main-algorithm does).*
- *Together these algorithms help achieve the purpose of my program by (how these algorithms are tied to program purpose).*

Draft Your Response Here:

[don't forget - paste the algorithm code snippet here - the same one you put an oval around in the whole program code]

[write your response]

(must not exceed 250 words)

Response 2c Checklist

Overall

- ❑ You wrote all algorithm code yourself
- ❑ Response includes copy-pasted versions of code for main and sub-algorithms with ovals around them
- ❑ Response identifies the main algorithm and at least two sub-algorithms

Sub-algorithm 1

- ❑ Clearly identifies the code for the algorithm (where in the code, function name, line numbers, etc)
- ❑ Explains what the algorithm does independently
- ❑ Describes how the code of the algorithm works
- ❑ Uses mathematical or logical concepts

Sub-algorithm 2

- ❑ Clearly identifies the code for the algorithm (where in the code, function name, line numbers, etc).
- ❑ Explains what the algorithm does independently
- ❑ Describes how the code of the algorithm works
- ❑ Uses mathematical or logical concepts

Main Algorithm

- ❑ Clearly identifies the code for the algorithm (where in the code, function name, line numbers, etc).
- ❑ Describes how main algorithm combines sub-algorithms
- ❑ Explains how main algorithm helps to achieve the overall purpose of the program

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a **rectangle** in **section 3** below). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. *(Must not exceed 200 words)*

Advice: Review the "*Is It a Good Abstraction?*" section above for tips on how to choose your abstraction. Here's the most important points.

- **Choose a Function:** Unless you feel confident about another abstraction, choose a function or class/ struct, not one written by cs50 or part of the graphics.py library, but a function or class you defined and named yourself.
- **You Wrote It:** You need to have written the code of your abstraction entirely on your own (not with a partner)
- **Copy and Paste It:** You must paste your actual abstraction code as part of this question submission.
- **Manages Complexity:** Make sure you can describe how your abstraction helps manage complexity in your program.
- **Make a contrasting argument:** explain how your program would be *more* complex to read, write, or reason about if you had *not* created your abstraction.
- **Mathematical and Logical Concepts:** While you should aim to include these in your abstraction, this is NOT explicitly assessed by the Scoring Guidelines for 2018.

Sentence Starters

- *The abstraction I selected is (name of abstraction)*
- *My abstraction manages complexity in my program by…*
- *Without my abstraction my program would be more difficult to (read / write / understand) because...*

Draft Your Response Here:

[Don't forget - paste your abstraction code snippet here - the same one you put a rectangle around in the program code]

[Write your response]

(must not exceed 250 words)

Response 2d Checklist

Overall

- ❑ You wrote all abstraction code yourself (it's not a function written by someone else, but a function, class, or list you defined and named)
- ❑ Response includes copy-pasted versions of code for abstraction with a rectangle around it
- ❑ Response identifies the abstraction by name
- ❑ You explicitly describe HOW the abstraction manages complexity (e.g. by explaining how your code would be more complex to write or reason about without the abstraction)

3. Program Code

Capture and paste your entire program code in this section.

› Mark with an oval the segment of program code that implements the algorithm you created for your program that integrates other algorithms and integrates mathematical and/or logical concepts.

› Mark with a rectangle the segment of program code that represents an abstraction you developed.

› Include comments or acknowledgments for program code that has been written by someone else.

Advice: For resources on how to make a PDF of your program code head to https://studio.code.org/s/csp-create/stage/1/puzzle/2. Here's the most important things to remember:

- **Making Your PDF:** Use CodePrint to make a PDF of your program. It's designed specifically for the Create PT. You can find it from the link given above.
- **Marking Your Algorithm:** Make sure you place an oval around all parts of your algorithm (parent and children).
- **Marking Your Abstraction:** Place your rectangle around the code where you create your abstraction (e.g. define a function) not where you use the abstraction (e.g. call a function).
- **Commenting and Collaboration:** You may work with a partner on the Create PT, but you must clearly indicate which parts you completed independently and which you completed together by using comments. For example:

```
// I completed the section below with my collaborative partner
// I completed this section independently
// I have extended code found at [URL]. The code below is my additions
```

  Remember that your algorithm and abstraction need to be created entirely independently.

- **Citing Images:** If you use code or images made by someone else (for example that you found online) then you should cite those resources as well. Again you can use comments.

```
// The images used in this app came from:
// [1] bird image - http://name-of-site.com/path/to/image.jpg
// [2] flower image - http://site.com/path/to/flower.jpg
```