Create PT - Written Response Template
Assessment Overview and Performance Task Directions for Students

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)*
My project, Type-o-meter, is a website application built with HTML, CSS, and JavaScript; where JavaScript is the programming language. Its purpose is to measure the user's typing speed and accuracy. It is built for personal pleasure and typing practice during the coronavirus quarantine. The video demonstrates a user testing the program. After pressing start, the user is presented with a piece of text, which he or she will type as fast as possible in the input field. During the process, the user's word per minute (wpm) is calculated and displayed. This is the main feature demonstrated in the video.

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 indicates 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)*
My project, Type-o-meter, begins with an input field and a timer. Typing speed (wpm) is the amount of characters in the input field divided by the time elapsed and then divided by a constant. This calculation forms the basis of the first function: updateStat(). Afterward, I set up a system for random texts to be displayed. At first, it is done with an array of manually inputted text; however, the process is slow and inefficient. To bypass this obstacle, I decide to implement APIs. If my program fetches data from a "random sentence" API server rather than relying on stored data, I wouldn't have to manually input sentences into the program. As such, I culled some API from the free public database by testing them individually in my program. Moving forward, I added a restart button to enable retests; where in, I discovered an opportunity. Currently, once the user restarts, the previous data is overridden by new ones. However, I can create a variable that keeps track of the best data. Called best stat, this will create an incentive for the user to keep practicing as they try to beat their previous scores. Both developments are done independently.

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3**) 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)*

Code Segment

1.

```
async function setupEvent() {
   if (type == "quote") {
      var data = await getQuote();
   }
   else if (type == "trivia") {
      var data = await getTrivia();
   }
   else if (type == "paragraph") {
      var data = await getParagraph();
   }
   else if (type == "word") {
      var data = await getWord();
   }
   else if (type == "character") {
      var data = getCharacter();
   }

   setTarget(data);
   beginEvent();
}
```

2.

```
// API source: https://github.com/lukePeavey/quotable
async function getQuote() {
   // Fetch a random quote from the Quotable API
   const response = await fetch("https://api.quotable.io/random");
   const data = await response.json();
   if (response.ok) {
      return data;
   }
   else {
      return contingencyData;
   }
}
```

3.

```
function setTarget(data) {
    target = data.content;
    targetIndex = 0;
    targetSource = data.author;
    targetAnswer = "";
    if (data.answer) {
        targetAnswer = data.answer;
    }

    if (event == "relay") {
        targetText.innerHTML = target;
        targetSourceText.innerHTML = targetSource;
        targetAnswerText.innerHTML = targetAnswer;
    }
    else if (event == "precision") {
        targetText.innerHTML = "[<span class='targetEmphasis'>" + target[targetIndex] + "</span>]" + target.substr(targetIndex +
1);
        targetSourceText.innerHTML = targetSource;
        targetAnswerText.innerHTML = targetAnswer;
    }
}
```

4.

```
function beginEvent() {
    eventActive = true;

    introScreen.style.display = "none";
    eventScreen.style.display = "block";
    continueButton.innerHTML = "Enter";

    updateBestStat();

    clearInput();
    userInput.focus();

    time.initial = new Date();
}
```

Written Response

My chosen algorithm: setupEvent(), denoted by the number 1, is responsible for the set up of the typing speed test. First, it generates a random “target” text by selectively calling one of five functions to access their corresponding “random sentence” API. Algorithm 2, which is one of the five functions, retrieves an inspirational quote from Luke Peavey’s Quotable API. Next, algorithm 3 is called to store the retrieved text in a “target” variable and display it as an HTML element. If the event of the typing test is “precision,” the content of the displayed text is modified. In this way, a raw copy of the sentence is stored as data and another, possibly modified, copy is visible to the user. Finally, algorithm 4 is called. It sets the variable “eventActive” to true, which removes the lock on user inputs and sets “initial time” to begin the timer for the typing test. Individually, algorithm 2 retrieves a random sentence from the Quotable API, algorithm 3 takes a piece of text and makes it visible to the user, and algorithm 4 enables user input.

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**). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. *(Must not exceed 200 words)*

Code Segment

1.

```
function updateStat() {
    time.end = new Date();
    stat.current[event][type].time = floor_nDecimal((time.end - time.initial) / millisecToMinute, 2);
    if (event == "relay") {
        stat.current[event][type].wpm = floor_nDecimal(userInput.value.length / (averageWordLength *
Math.max(stat.current[event][type].time, 0.01)), 2);
    }
    else if (event == "precision") {
        stat.current[event][type].wpm = floor_nDecimal(targetIndex / (averageWordLength * Math.max(stat.current[event]
[type].time, 0.01)), 2);
    }

    statText.current.wpm.innerHTML = "wpm (word/min): " + stat.current[event][type].wpm;
    statText.current.time.innerHTML = "time (min): " + stat.current[event][type].time;
    statText.current.backspace.innerHTML = "backspace (count): " + stat.current[event][type].backspace;
    statText.current.error.innerHTML = "error (count): " + stat.current[event][type].error;
}
```

2.

```
function concludeEvent() {
    if (eventActive) {
        eventActive = false;
        continueButton.innerHTML = "Next";

        updateStat();
        updateBestStat();

        clearInput();
    }
}
```

Written Response

My chosen algorithm: updateStat(), denoted by the number 1, exercises abstraction by condensing multiple lines of instruction into one readable function. Algorithm 1 calculates elapsed time and typing speed (wpm). Then, it makes the stats visible to the user as HTML elements. Rather than retyping 12 lines of instructions each time that the stat must be updated, updateStat() condensed those lines into a single function call. In this way, it saves space, reduces clusters, and increases code visibility. This allows higher-level functions, such as algorithm 2, which ends the typing test, to be concerned with “which tasks must be executed” and “when they are executed”, rather than “how they are executed.” As such, due to abstraction exercises by functions, such as algorithm 1, higher-level functions, such as algorithm 2, are streamlined and thus, easier to be modified and debugged.