**Create – Applications From Ideas  
Written Response Submission Template**

Please see [Assessment Overview and Performance Task Directions for Student](https://apcentral.collegeboard.org/pdf/ap-csp-student-task-directions.pdf?course=ap-computer-science-principles) for the task directions and recommended word counts.

**Program Purpose and Development**

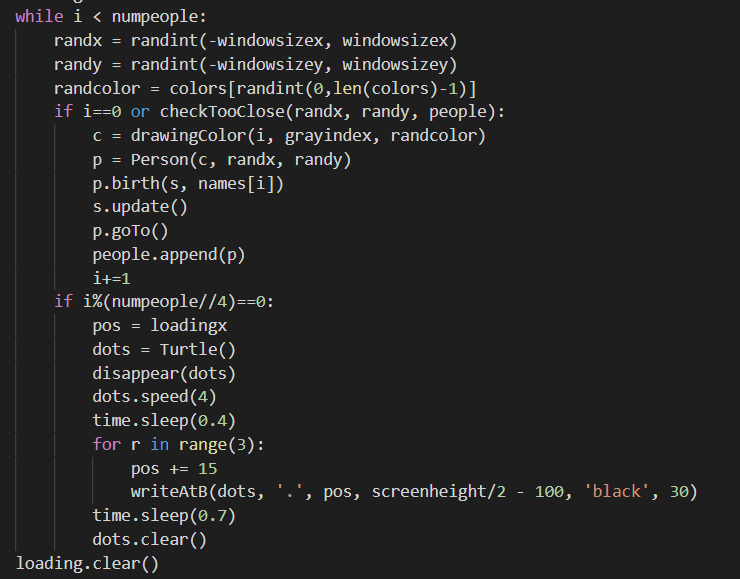
2a)

|  |
| --- |
| The programming language I used is Python. My project is a game that involves generating a city of a number of people. The user will input a number and the program will generate that number of people randomly on the screen. The colors of the people are also random, consisting of red, orange, yellow, green, blue, and purple. My video is showing a sample round. My program generates 30 people. |

2b)

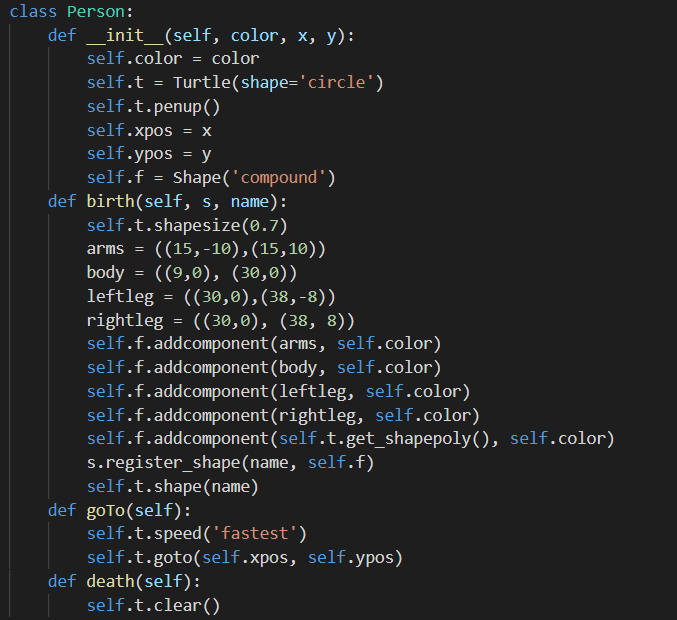
|  |
| --- |
| Some difficulties I encountered while creating my program were making the appearance of the people and making sure that each individual person was a single instance and not the reference to the same memory location. The stick figure people were made with circles and lines. Those people are Turtle objects and not Turtle drawings, so I had to use a new function called a complex shape. This led to the second difficulty. When I first started with the complex shape, the people could all generate as different colors, but after they finish being made, they would all become the color of the last person generated. I found out that this issue occurred because the design of this complex shape saved as one single type in the shapes list of the Turtle object. In order to solve this issue, I had to make each identical shape saved as a different number in the shapes list of the Turtle object so each person was a separate instance. |

2c)



|  |
| --- |
| My algorithm helps ensure that the random locations of people are not too close to each other and adds visual appeal to the loading process while the city is being generated. There are 2 subalgorithms: making sure people aren't too close and when to show the ellipsis. The first algorithm is both a "selection" and "iteration" algorithm. It verifies that either the person being generated is the first one, or has been approved. The "approval" algorithm involves saving all of the previously validated people into a Python list and checking the new person's x and y distance from every other generated person (in the list). The first person does not need to be checked. The second algorithm's type is "selection." It involves the "if" statement that checks if the number of people generated is divisible by the whole number quotient of the total number of people divided by 4. For example, the whole number quotient of 150 divided by 4 is 37 because it is the actual quotient 37.5 rounded down. Then, the loading dots would appear a minimum of 4 times. The position of the dots is determined simply by adding 15 pixels to the right. |

2d)



|  |
| --- |
| My abstraction is a self-created Python function (birth()) from a self-created Python object (Person()). This makes the project simpler to read as a whole, while also managing the complexity of the program. Here, I have the birth() method from the Person class. There are 2 abstractions at play here. The first is that each stick figure person on the screen is defined by a “Person” object. This makes it easier to manage because I am not using multiple Python Turtle commands to generate each body part of the stick figure, one line to initialize a Person object. The second abstraction is the Person object’s “birth()” function, which contains the specific steps to create/draw a Person object from a Python Turtle. |