CS 106AX Programming Methodologies in JavaScript and Python Autumn 2026

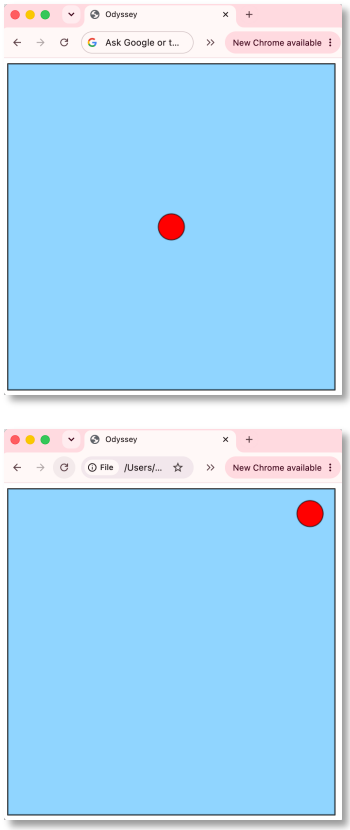
IG Problem 2A: The Odyssey

Written by Ben Yan
(this problem, not the Odyssey itself ... I'm not that old lol)

If you've been to a cinema in the past six months, chances are that you caught a premiere of Christopher Nolan's latest movie epic, *The Odyssey* (2026), an adaptation of the Greek myth, featuring monsters, giants, perilous seas, a crisis of morality and mortality, the universal poetry of a father trying to get home to his son.

If you saw *The Odyssey* in premium 70mm IMAX, I'm very, very envious of you ☺.

You are to implement a JavaScript simulation of Odysseus's long, long journey away from and back home. It is, I dare say, amateur yet auteur filmmaking.



Uh oh, where is he going?

First, **introduce a red circle with radius 20 pixels to the graphics window.** This is both a crude, low-budget rendering of his wooden ship, and a visual marker for his live location. Location-wise, **the circle's center should coincide with the window's center** (you can imagine this as the coordinates of Odysseus's island home in Ithaca). And unlike the images, **you're not required to make the canvas ocean blue** ☺. I did this purely out of the love of the game.

Next, **add a timer animation to have the ship move 20 pixels in a randomly chosen cardinal direction** (e.g., north, east, south, or west) **every 500 milliseconds**. The ship may drift out of the window, which is perfectly fine, both graphically and narratively. To choose one of 4 random directions, you may find the **`randomInteger(low, high)`** method helpful from the random library—which, as the name dutifully suggests, selects a random integer between **`low`** and **`high`**, inclusive on both ends of the spectrum.

If you're feeling particularly ambitious, you can make use of **`movePolar`** , a method that takes a distance in pixels and an angle in degrees. For instance,

- **`obj.movePolar(10, 0)`** moves the object 10 pixels **east**.
- **`obj.movePolar(10, 90)`** moves an object 10 pixels **north**.
- **`obj.movePolar(10, 180)`** moves an object 10 pixels **west**.
- **`obj.movePolar(10, 270)`** moves an object 10 pixels **south**.

Otherwise, the standard **`move`** method more than suffices ☺.

Lastly, **the ship should keep moving until it has returned home to the center, at its starting coordinates**, which is when the Odyssey at last ends. At this point, you should call **`clearInterval`** to terminate the animation—and be careful to only do this after Odysseus has departed home for the very first time.

Use the following page to present your implementation; we highly encourage helpers and decomposition!

Fun Fact: Odysseus is mathematically certain to return home eventually, as I learned back in MATH 63DM. However, it may take very, very long—as he can really get off course (which, um spoiler alert, happens).



```
/* Constants (in pixels) */
const GWINDOW_HEIGHT = 500;
const GWINDOW_WIDTH = 500;
const SHIP_RADIUS = 20;
const SHIP_DIAMETER = 2 * SHIP_RADIUS;
const DRIFT = 20; // pixels moved in each time step

/* Constants (in milliseconds) */
const TIME_STEP = 500;

/* Main program to playfully imitate the story of King Odysseus's return */
function Odyssey() {
      let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);









}
```

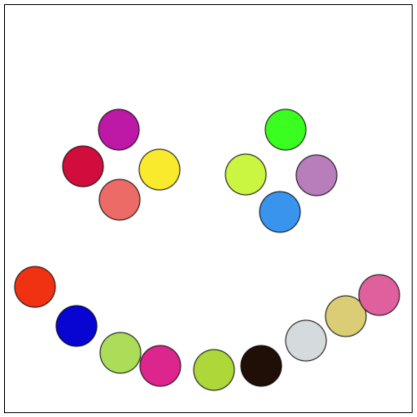
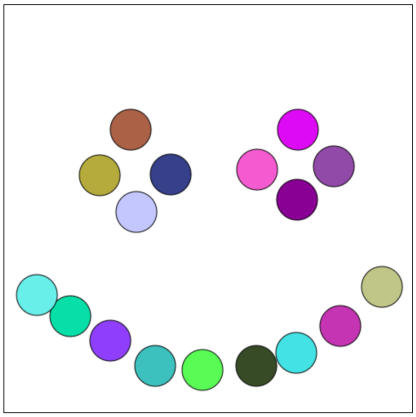
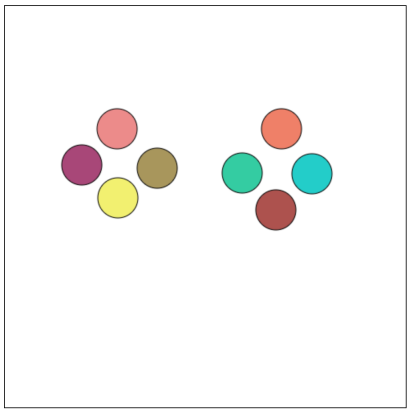
IG Problem 2B: Holiday Lights

Written by Ben Yan
Inspired by a Nick Troccoli problem, back in the old Java-based CS106A.

Implement a graphics program that responds colorfully and festively to mouse clicks, by introducing a holiday light to the graphics window where each click occurs. Each holiday light is a randomly colored circle with radius 25 pixels, and centered at the coordinates of the mouse click. **However, if the user clicks a location where a light already exists, the program should remove the existing light there, instead of adding a new light.** If there are multiple circles that coincide with the click's location, you should simply remove the topmost one, which is serendipitously what the method **`gw.getElementAt(x,y)`** detects.

Also, it would be rather boring if the lights all stayed the same and couldn't change color, so you're going to change that ☺. **Each circle, independently of others, randomly changes its color every 1000 milliseconds** once it's added to the graphics window. You can use the **`randomColor()`** method, as well as JavaScript timer animation, to make this vibrant lighting happen. When a mouse click removes a holiday light, you're surprisingly **not required** to terminate its color animation using **`clearInterval`**. It's entirely possible, but it's a bit tricky to retrieve the timer for that specific light, and requires a technique you'll learn in Assign4 ☺.

A playful example is shown below, with a trilogy of images taken in chronological order from earliest (left) to latest (right). Before taking the left image, I clicked my mouse on-screen carefully, repeatedly to form a Jack O' Lantern smile of varicolored lights ☺. I didn't click at all between taking the left and middle images, but with the lights changing color every second, all the hues ended up different. And before capturing the right image, I clicked directly on each holiday light along the bottom smile / arc, which removed them ☹.

Use the snowy expanse of the following page to illuminate your implementation of **`HolidayLights`**.



```
/* Constants (in pixels) */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 500;
const LIGHT_RADIUS = 25;
const LIGHT_DIAMETER = 2 * LIGHT_RADIUS;

/* Constants (in milliseconds) */
const COLOR_CHANGE_DELAY = 1000;

/* Main program, where you enable adding and removing holiday lights */
function HolidayLights() {
      let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);









}
```