CS 106AX Programming Methodologies in JavaScript and Python Autumn 2026

Ben's Practice Midterm Solutions

Written by Ben Yan

Problem 1: Tracing that Dream (JavaScript Expressions and Tracing)

1. [3 points] Compute the value of each of the following JavaScript expressions.

```
2026 + 3 % 2 * 2 - 2020
```
8

```
"IDOL" === "idol" || "YUUSHA" < "SUKI" + "DA"
```
false

```
6  + "SEIS" + 1 + 2 + 4 + "SIETE"
```
"6SEIS124SIETE"

2. [3 points] Assume the function **LoveLetter**, which accepts arrays **truth** and **obfuscation** as input, is defined as follows. Note that it returns a new array of identical length to **truth**.

```
function LoveLetter(truth, obfuscation) {
      let encrypted = [];
      let ambiguity = obfuscation.length;

      for (let i = 0; i < truth.length; i++) {
            let mySecret = truth[i];
            let feelings = i % ambiguity;

            let myWords = mySecret + obfuscation[feelings];
            if (myWords >= 26) myWords = myWords - 26;

            encrypted.push(myWords);
      }
      return encrypted;
}
```

What does the call **LoveLetter([2, 3, 5, 7, 11, 13], [17, 19, 23])** return? Work through this function trace carefully, and indicate the value of each of the elements in the returned array below.

19	22	2	24	4	10
0	1	2	3	4	5

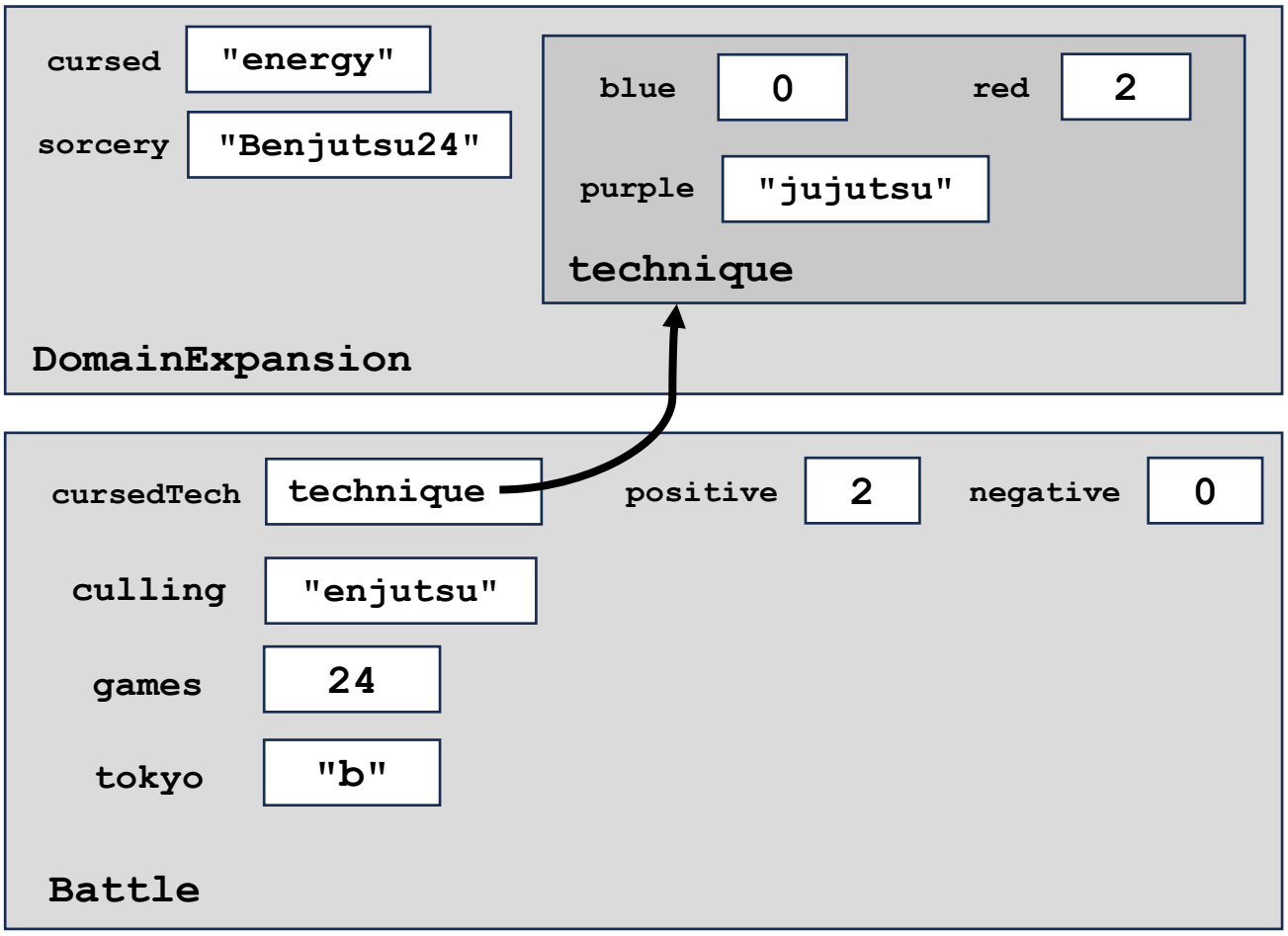
3. [4 points] What output is produced by a call to the function **`DomainExpansion()`**?

```
function DomainExpansion() {
      let cursed = "energy";
      let technique = function(blue, red, purple) {
            return cursed.substring(blue, red) + purple.substring(red);
      }
      let sorcery = Battle(technique, cursed.lastIndexOf("e"),
                                      cursed.indexOf("e"));
      sorcery.toLowerCase();
      console.log("This truly was our " + sorcery);
}
```

Solution: **This truly was our Benjutsu24**

```
function Battle(cursedTech, positive, negative) {
      let culling = cursedTech(negative, positive, "jujutsu");
      let games = "shibuya".charCodeAt(5) - "arc".charCodeAt(0);
      let tokyo = "tombsOfStar".charAt(3); // not the same as charCodeAt!
      return tokyo.toUpperCase() + culling + games;
}
```

The final stack frame of each function called—and thus, value of each variable—is shown below ☺.



Problem 2: Sparks Fly (Graphics and Animation, 15 points)

```
function sparksFly(){
      let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);

      let createSparks = false; // Boolean tracking if sparks are launching

      let wand = GRect(WAND_WIDTH, WAND_HEIGHT);
      wand.setFilled(true);
      wand.setColor("brown");
      gw.add(wand, 0, 0); // add to top-left corner of the screen

      /* the top-left coordinate of wand moves with the mouse */
      let moveWand = function(e) {
            wand.setLocation(e.getX(), e.getY());
      }
      gw.addEventListener("mousemove", moveWand);

      function mouseClickAction(e) {
            /* toggle wand between launching and not launching sparks */
            createSparks = !createSparks;
      }
      gw.addEventListener("click", mouseClickAction);

      let letSparkFly = function() {
            if (!createSparks) return; // wand not currently creating sparks

            /* creates circular spark starting at the top center of wand */
            let spark = GOval(wand.getX(), wand.getY(),
                              SPARK_DIAMETER, SPARK_DIAMETER);
            spark.setFilled(true);
            spark.setColor(randomColor());
            gw.add(spark);

            /* creates animation to send this spark to the top */
            let moveSparkUp = function() {
                  spark.move(0, SPARK_Y_VELOCITY);
                  /* remove and clear animation if at the top */
                  if (spark.getY() < 0){
                        gw.remove(spark);
                        clearInterval(flyingTimer);
                  }
            }
            let flyingTimer = setInterval(moveSparkUp, FLYING_TIME_STEP);
      }
      /* creates a new flying spark every few milliseconds */
      let creationTimer = setInterval(letSparkFly, CREATION_TIME_STEP);
}
```

To reference each of the predefined constants, e.g. **WAND_WIDTH**, see the original problem statement ☺.

Problem 3: Shape-Shifting Cipher (Strings, 15 points)

```
const PAD_LETTER = "X";

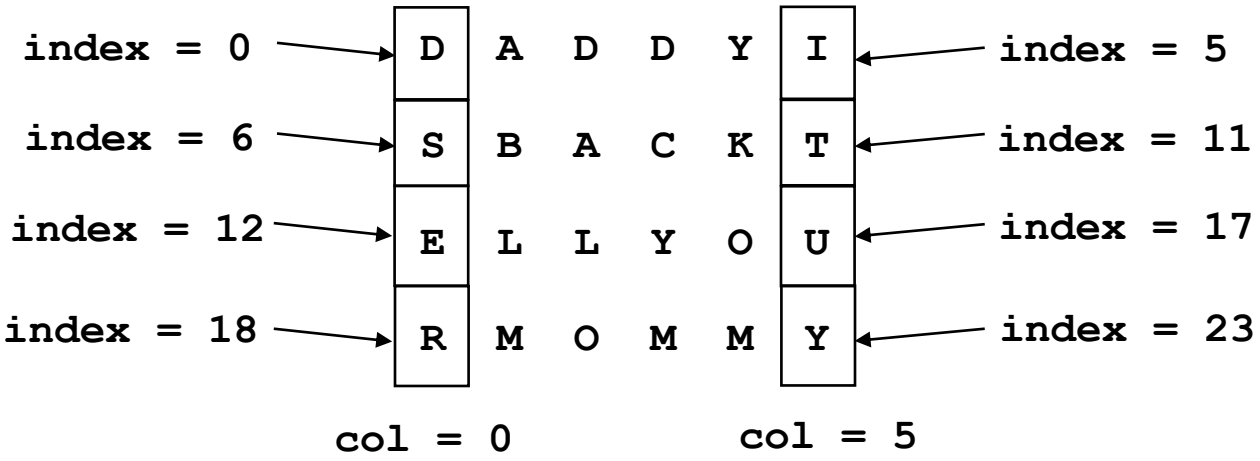
function scytale(plaintext, columns) {

      // pad the plaintext so it becomes a multiple of columns
      while (plaintext.length % columns !== 0) {
            plaintext += PAD_LETTER;
      }

      // construct the ciphertext letter by letter, by going over columns
      let ciphertext = '';
      for (let col = 0; col < columns; col++) {
            // loop over the plaintext indices for each column
            let index = col;
            while (index < plaintext.length) {
                  ciphertext += plaintext[index];
                  index += columns; // jump from one row to the next
            }
      }

      return ciphertext;
}
```

As an example, consider a plaintext **" DADDYISBACKTELLYOURMOMMY"**, which is written into a table with **6** columns, with the plaintext letters flowing from left to right, then top to bottom.



The corresponding ciphertext, reading top to bottom first, would be **"DSERABLMDALODCYMYKOMITUY"**, which is composed of columns **"DSER"**, **"ABLM"**, **"DALO"**, **"DCYM"**, **"YKOM"**, and **"ITUY"**.

Problem 4: Paperless Rings (Arrays, 15 points)

```
/* Creates pairs of star-crossed lovers / participants, and populates
 * an array of matches ☺, and an array of unmatched participants ☹.
 */
function MarriagePact(participants, compability, matches, unmatched) {
     // an array of booleans tracking if each person is matched yet
     let taken = [];
     for (let i = 0; i < participants.length; i++) {
          taken.push(false);
     }

     // iterate over each participant and search for a match
     for (let i = 0; i < participants.length; i++) {
          if (taken[i]) continue; // skip if person is already matched

          let name = participants[i][0];
          let zodiac = participants[i][1];

          // look ahead in participants array for a prospective lover :)
          for (let j = i + 1; j < participants.length; j++) {
               let loverCandidate = participants[j][0];
               let theirZodiac = participants[j][1];

               // check if candidate is not taken yet and is compatible
               if (!taken[j] && isCompatible(zodiac,
                                        theirZodiac, compability)) {
                    matches.push([name, loverCandidate]);
                    taken[i] = true;
                    taken[j] = true;
                    break; // stop searching for this participant
               }
          }

          if (!taken[i]) { // if ultimately, no match could be found :(
               unmatched.push(name);
          }
     }
}

// helper function: returns true only if the two signs are compatible
function isCompatible(zodiacOne, zodiacTwo, compatibleSigns) {
    for (let i = 0; i < compatibleSigns.length; i++) {
        let signGroup = compatibleSigns[i];
        if (signGroup.indexOf(zodiacOne) !== -1) {
            // returns true only if zodiacTwo is in the same group
            return signGroup.indexOf(zodiacTwo) !== -1;
        }
    }
    return false; // optional: you can assume each signs is in a group
}
```

Problem 5: Spotify Wrapped (Data Structures, 15 points)

Author Comment: After I wrote this problem, I realized the solution was a lot larger and more involved and difficult than I expected oops. Sowwy! ☹ If I were to write this problem for a real midterm, you'd just have to implement the helper function **`getMostStreamedAritst`**, and not need to also fetch the top song ☺.

```
/* Given a user's streaming data, returns an object with that user's
 * most-streamed song, and most-streamed artist across all their songs.
 */
function MusicWrapped(streamsByAlbum) {
      let topArtist = getMostStreamedArtist(streamsByAlbum);
      let topSong = getMostStreamedSong(streamsByAlbum);

      return {artist: topArtist, song: topSong}
}
```

As hinted above, my own solution relies on a pair of helper functions: **`getMostStreamedSong`**, to discover the top song, and **`getMostStreamedArtist`** to discover the top artist from the passed-in object.

```
function getMostStreamedSong(streamsByAlbum) {
      let mostListenedSong = null; // placeholder values
      let mostSongStreams = 0;

      // loop over every album in the song data
      for (let album in streamsByAlbum) {
            let albumData = streamsByAlbum[album];
            let songList = albumData.songs;

            // for this album, loop over every song we listened to
            for (let i = 0; i < songList.length; i++) {
                  let title = songList[i].title;
                  let numStreams = songList[i].streams;

                  // check if this is the song we've streamed the most so far
                  if (numStreams > mostSongStreams) {
                        mostSongStreams = numStreams;
                        mostListenedSong = title;
                  }
            }
      }
      return mostListenedSong;
}
```

The implementation of the helper function **`getMostStreamedArtist`** is shown on the next page ☺.

```
function getMostStreamedArtist(streamsByAlbum) {
      // build up a map of each artist to total number of streams
      let artistToStreams = {};

      for (let album in streamsByAlbum) {
            let albumData = streamsByAlbum[album];
            let artist = albumData.artist;

            if (!(artist in artistToStreams)) {
                  // if we haven't seen this artist yet
                  artistToStreams[artist] = 0;
            }

            // loop through each song, and add its streams to artist
            let songList = streamsByAlbum[album].songs;
            for (let i = 0; i < songList.length; i++) {
                  artistToStreams[artist] += songList[i].streams;
            }
      }

      // now, find out who is the most listened to artist :)
      let mostStreamedArtist = null;
      let mostStreams = 0; // tracks the current best # of streams
      for (let artist in artistToStreams) {
            if (artistToStreams[artist] > mostStreams) {
                  mostStreamedArtist = artist;
                  mostStreams = artistToStreams[artist];
            }
      }

      return mostStreamedArtist;
}
```