CS 106AX Programming Methodologies in JavaScript and Python Autumn 2026

Ben's Extra Midterm Practice Solutions

Written by Ben Yan. ☺

Problem 1: Never-Ending Stories (JavaScript Expressions and Tracing)

1. Compute the value of each of the following JavaScript expressions.

```
(20 + 7) % 24 + 24 % 12
```

3

```
1 + 2 + 3 + 4 === 10 || 1234 / 0 === 1234
```

true

```
10 + 6 + "YO" + 3 * 4 + 9 + "ASOBI"
```

"16YO129ASOBI"

2. Assume the function **chaseYourDreams** is defined as presented below. The code is, of course, narratively and poetically nonsense, and you just want to do a faithful JavaScript trace.

```
function chaseYourDreams(dreamOne, dreamTwo){
      let resilience = 0;
      let failure = 1;
      while (dreamOne > 0) {
            let learning = dreamOne % 10 + dreamTwo % 10;
            for (let i = 0; i < failure; i++) {
                  resilience += learning;
            }
            failure *= 10;

            let belief = Math.floor(dreamOne / 10);
            dreamOne = Math.floor(dreamTwo / 10);
            dreamTwo = belief;
      }
      return resilience;
}
```

What's the value returned by a call to **chaseYourDreams(2048, 4096)** ? Solution: **6144**

The function **chaseYourDreams** is a complicated way of **adding up two input numbers that have the same number of digits** ☺. The return value is ultimately **dreamOne + dreamTwo**, because, as you know, dreams build on one another—and as you chase your dreams, failure, but also resilience, grows.

It adds the ones digit of **dreamOne** with the ones digit of **dreamTwo**, the tens digit of **dreamOne** with the tens digit of **dreamTwo**, and so forth. Each digit sum is weighted by its place, e.g., ones (**failure = 1**), tens (**failure = 10**), hundreds (**failure = 100**), etc. Note the switching at the end of the **while** loop does not affect summation. On the next page is my stack trace of each of its **4** iterations.

Here are the variable values in **`chaseYourDream`**'s stack frame prior to the **`while`** loop.

`dreamOne`	**`2048`**	**`dreamTwo`**	**`4096`**
`resilience`	**`0`**	**`failure`**	**`1`**

Here's the variable values in the stack frame at **the end of each iteration** of the **`while`** loop.

Iteration 0

`learning`	**`14`**	**`resilience`**	**`14`**	**`failure`**	**`10`**
	`8 + 6`		**`+ 14`**		
`belief`	**`204`**	**`dreamOne`**	**`409`**	**`dreamTwo`**	**`204`**

Iteration 1

`learning`	**`13`**	**`resilience`**	**`144`**	**`failure`**	**`100`**
	`9 + 4`		**`+ 130`**		
`belief`	**`40`**	**`dreamOne`**	**`20`**	**`dreamTwo`**	**`40`**

Iteration 2

`learning`	**`0`**	**`resilience`**	**`144`**	**`failure`**	**`1000`**
	`0 + 0`		**`+ 0`**		
`belief`	**`2`**	**`dreamOne`**	**`4`**	**`dreamTwo`**	**`2`**

Iteration 3

`learning`	**`6`**	**`resilience`**	**`6144`**	**`failure`**	**`10000`**
	`4 + 2`		**`+ 6000`**		
`belief`	**`0`**	**`dreamOne`**	**`0`**	**`dreamTwo`**	**`0`**

And so the end result returned by the call **`chaseYourDreams(2048, 4096)`** is the sum **`6144`**.

3. Assume the following (silly) definitions of the functions **sillyString** and **sillyBanz**.



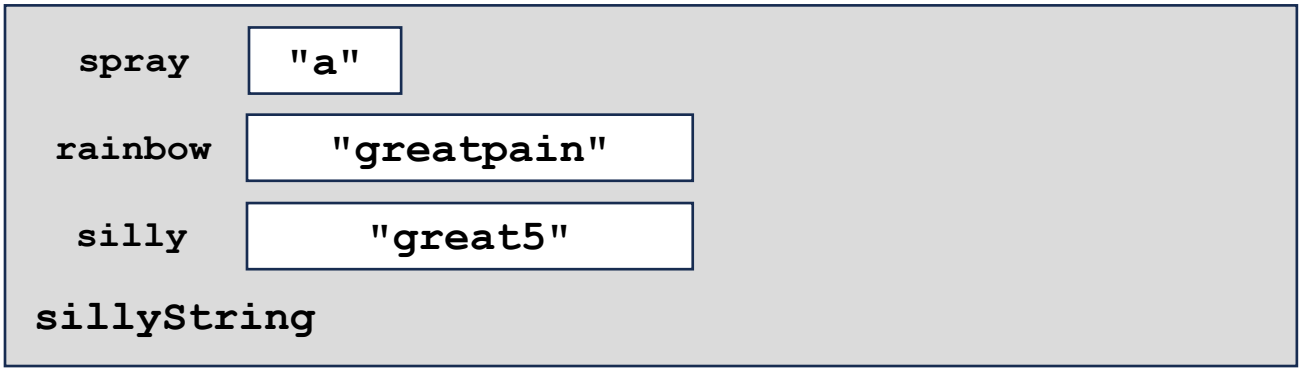
```
function sillyString(string) {
      let spray = string.charAt(0);
      let rainbow = string.substr(1, string.length - 1);
      let silly = sillyBandz(spray, rainbow);
      return silly + spray.toUpperCase();
}

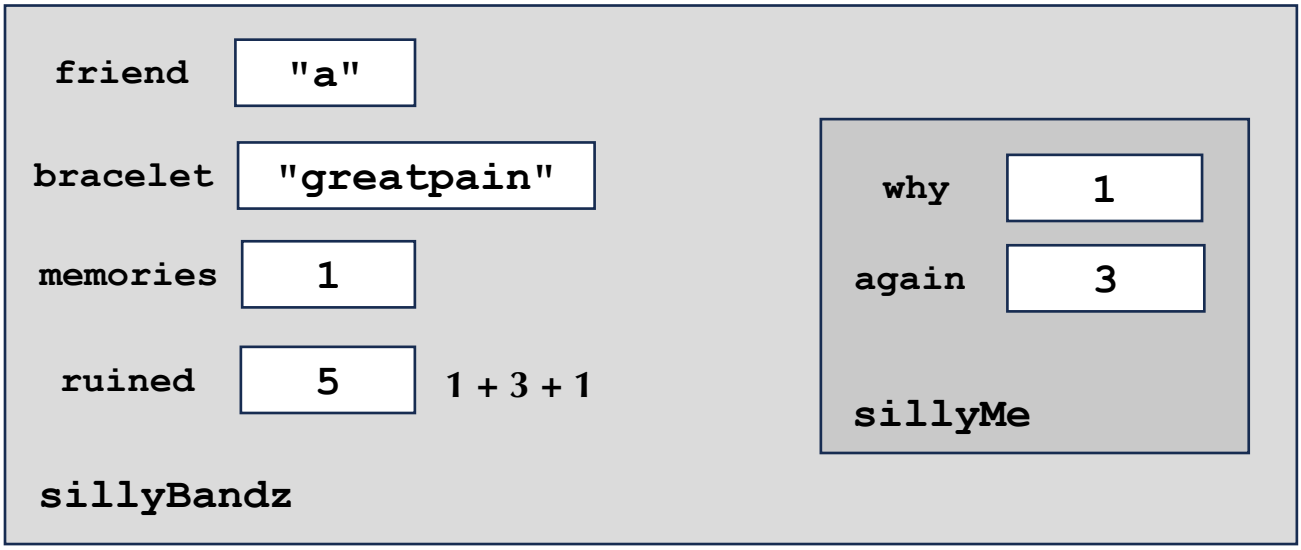
function sillyBandz(friend, bracelet) {
      let memories = "abc".charCodeAt(1) - friend.charCodeAt(0);
      let ruined = sillyMe(memories);
      function sillyMe(why){
            let again = bracelet.indexOf(friend);
            return memories + again + why;
      }
      return bracelet.substr(0, ruined) + ruined;
}
```

What output is returned by a function call to **sillyString("agreatpaint")**?

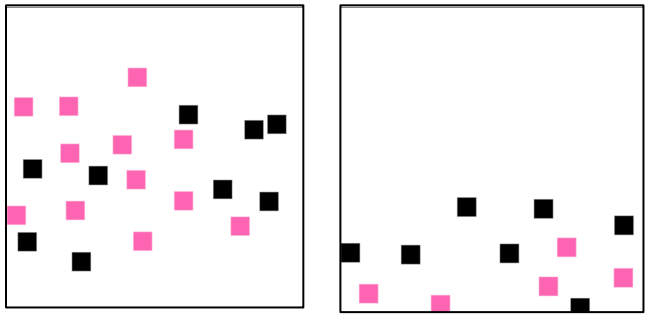
Solution: **great5A**

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





Problem 2: Blackpink In Your Area (Graphics and Animation)



```
/* Constants (in pixels) */
let GWINDOW_WIDTH = 500;
let GWINDOW_HEIGHT = 500;
let SQUARE_DIMENSION = 30;
let BLACK_SQUARE_VY = 1;
let PINK_SQUARE_VY = 2;

/* Constants (in milliseconds) */
let FALLING_TIME_STEP = 50;
let COLOR_CHANGE_TIME_STEP = 1000;

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

      let createFallingSquare = function(e) {
            // creates a black square centered at the mouse clicked
            let square = GRect(SQUARE_DIMENSION, SQUARE_DIMENSION);
            square.setColor("Black");
            square.setFilled(true);
            gw.add(square, e.getX() - SQUARE_DIMENSION / 2,
                           e.getY() - SQUARE_DIMENSION / 2);

            // for each square, add timers to move square down & change color
            let step = function() {
                  // checks if square has fallen beneath the graphics window
                  if (square.getY() > GWINDOW_HEIGHT) {
                        clearInterval(fallingTimer);
                        clearInterval(colorChangeTimer);
                  }

                  // moves the square downward, at a faster pace if pink
                  let vy = BLACK_SQUARE_VY;
                  if (square.getColor() === "Black") vy = PINK_SQUARE_VY;
                  square.move(0, vy);
            }

            let fallingTimer = setInterval(step, FALLING_TIME_STEP);

            let changeColor = function() {
                  // change Pink square to Black square, and vice versa
                  if (square.getColor() === "Black") square.setColor("Pink");
                  else square.setColor("Black");
            }

            let colorChangeTimer = setInterval(changeColor,
                                               COLOR_CHANGE_TIME_STEP);
      }

      gw.addEventListener("click", createFallingSquare);
}
```

Problem 3: Rail Fence Cipher (Strings)

```
function railFenceEncrypt(message, numRails) {
    let encrypted = "";

    for (let rail = 0; rail < numRails; rail++) {
        let pos = rail;

        if (rail === 0 || rail === numRails - 1) {
            // is the top or bottom rail, so the jumps between
            // consecutive letters on that rail are the same distance

            let jump = 2 * (numRails - 1);
            while (pos < message.length) {
                encrypted += message[pos];
                pos += jump;
            }

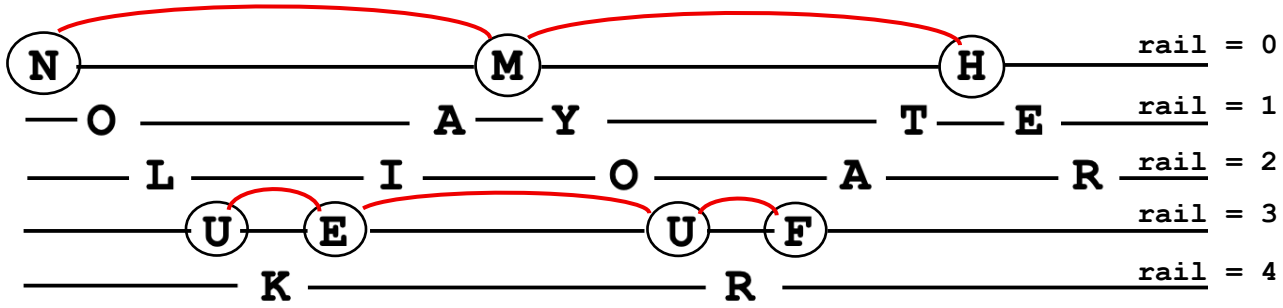
        } else {
            // is a middle rail, so jumps alternate in distance between
            // x and 2*(numRails - 1) - x, with x = 2 * (numRails - 1 - rail)

            let jump = 2 * (numRails - 1 - rail);
            while (pos < message.length) {
                encrypted += message[pos];
                pos += jump;
                jump = 2 * (numRails - 1) - jump; // alternating jumps
            }
        }
    }

    return encrypted;
}
```

`message = "NOLUKEIAMYOURFATHER"` | `numRails = 5`

Index **jump** on top and bottom rails is a fixed **8 = 2 * (numRails – 1)**



For rail **1**, the **jump** alternates between **6** and **2** For rail **3**, the **jump** alternates between **2** and **6**

For rail **2**, the **jump** "alternates" between **4** and **4**

`encrypted = "NMHOAYTELIOARUEUFKR"`

Problem 4: What the F— (Strings)

```
/* Given a text string possibly full of bad, colorful words, creates a new
 * string where all bad words are substituted for a mundane replacement 🤬.
 *
 * bleep("The name's [[bond]], James [[bond]].", "bleep")
 *    => " The name's BLEEP, James BLEEP."
 *
 * bleep("apply to [[j*bs]], [[empl*yment]], and [[w*rk]]", "coterm")
 *    => "apply to coterm, coterm, and coterm"
 *
 */
function bleep(text, replacement) {

    // keep looping until all the swear words have been replaced :)
    while (true) {
        let left = text.indexOf("[[");
        if (left === -1) break; // no more swear words, so exit loop!

        let right = text.indexOf("]]");
        // this check is optional, as we assume [[ is followed by ]]
        if (right === -1) break;

        let beforeSwear = text.substr(0, left);
        let afterSwear = text.substr(right + 2,);

        // surgically slice out this swear word, and replace it :)
        // e.g., "what the [[fsck]] Ben" to "what the BLEEP Ben"
        text = beforeSwear + replacement + afterSwear;
    }

    return text;
}
```

Problem 5: U.S. National Debt (Arrays)

```
/* Accepts two big number arrays (digits in reverse order) as input, and
 * returns a big number array that represents the sum of the input numbers.
 *
 *    >> addBigNumbers([3, 2, 1], [4, 3, 2]) => [7, 5, 3] // 123 + 234 = 357
 *
 *    >> addBigNumbers([1, 9, 9], [0, 1]) => [1, 0, 0, 1] // 991 + 10 = 1001
 */

function addBigNumbers(bigNumOne, bigNumTwo) {
      let sum = [];
      let carry = 0; // +1 if we have a carry over from adding previous digits

      for (let i = 0; i < Math.max(bigNumOne.length, bigNumTwo.length); i++) {
            // add sum of digits in current digit position
            let digitOne = getDigitSafe(bigNumOne, i);
            let digitTwo = getDigitSafe(bigNumTwo, i);
            let digitSum = digitOne + digitTwo + carry;

            // determine whether to carry over to the next digit
            if (digitSum >= 10) {
                  digitSum -= 10;
                  carry = 1;
            } else {
                  carry = 0;
            }

            // add new digit to end of array (note: reverse order of digits)
            sum.push(digitSum);
      }

      // check for a carry on the final / highest digit
      if (carry > 0) {
            sum.push(carry);
      }

      return sum;
}
```

Problem 6: Nomad Palindrome (Arrays)

```
function generatePalindromes(maxDigits) { // assume maxDigits >= 2
    let palindromes = [];
    palindromes.push([]); // 0 digit palindromes
    palindromes.push([1, 2, 3, 4, 5, 6, 7, 8, 9]); // 1 digit
    palindromes.push([11, 22, 33, 44, 55, 66, 77, 88, 99]); // 2 digit

    for (let numDigits = 3; numDigits <= maxDigits; numDigits++) {

        // generate K-digit palindromes from (K - 2)-digit palindromes
        let extensions = [];
        let generateFrom = palindromes[numDigits - 2];

        // for each smaller palindrome, add same digit to the left and right
        for (let topDigit = 1; topDigit <= 9; topDigit++) {
            for (let i = 0; i < generateFrom.length; i++) {
                let middle = generateFrom[i];
                let bottomDigit = topDigit; // to ensure palindrome

                // for instance, 232 => 12321 with topDigit = bottomDigit = 1
                let newPalindrome = topDigit * Math.pow(10, numDigits - 1)
                                    + middle * 10 + bottomDigit;
                extensions.push(newPalindrome);
            }
        }
        palindromes.push(extensions);
    }

    return palindromes; // array of arrays of palindrome numbers :)
}
```

Problem 7: Hung Parliament (Working with Data Structures)

```
function isHungParliament(electionData, numSeats) {

      // build up a map of each party to how many seats/districts they won
      let partyToSeats = {};

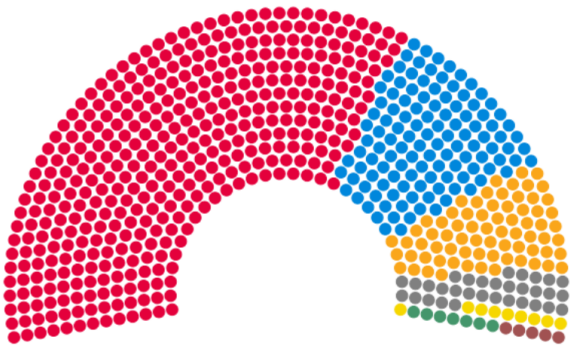
      for (let district in electionData) {
            let candidateArray = electionData[district];
            let winningParty = null; // placeholder values
            let winningVotes = -1;

            // iterate over all candidates and track the highest vote tally
            for (let i = 0; i < candidateArray.length; i++) {
                  let candidateInfo = candidateArray[i];
                  if (candidateInfo["votes"] > winningVotes) {
                        winningParty = candidateInfo["party"];
                        winningVotes = candidateInfo["votes"];
                  }
            }

            // add district the party just won to their running seats total
            if (!(winningParty in partyToSeats)) {
                  partyToSeats[winningParty] = 0; // if party not yet in map
            }
            partyToSeats[winningParty]++;
      }

      // see if any party has a strict majority (i.e. no hung parliament)
      const seatsForMajority = 0.5 * numSeats;
      for (party in partyToSeats) {
            if (partyToSeats[party] > seatsForMajority) {
                  return true;
            }
      }

      return false; // if we reach here, no party controls a majority!
}
```



Problem 8: Great CS106AX Bake Off (Working with Data Structures)

```
function attemptBakeOff(pantry, cakeOrders) {

      // build up a map of each ingredient to the amount of it we need
      let needed = {};

      for (let i = 0; i < cakeOrders.length; i++) {

            let order = cakeOrders[i];
            for (let ingredient in order) {
                  let amount = order[ingredient];

                  // add amount of the ingredient to its running total
                  if (!(ingredient in needed)) {
                        needed[ingredient] = 0;
                  }
                  needed[ingredient] += amount;
            }
      }

      // check if pantry has enough of each ingredient for all cake orders
      for (ingredient in needed) {

            // if ingredient not in the pantry, assume its amount is zero
            let supply = (ingredient in pantry) ? pantry[ingredient] : 0;

            if (needed[ingredient] > supply) {
                  return false; // not enough of this ingredient!
            }
      }

      // if we reach here, we have enough of each ingredient :), so bake!
      for (let i = 0; i < cakeOrders.length; i++) {

            let order = cakeOrders[i];
            for (let ingredient in order) {
                  pantry[ingredient] -= order[ingredient];
            }
      }

      return true;
}
```