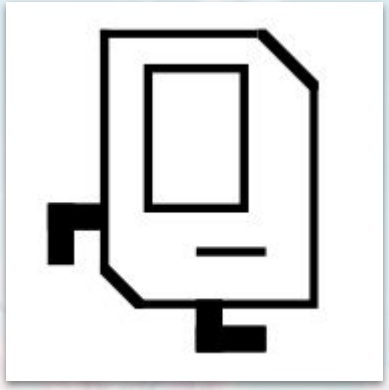
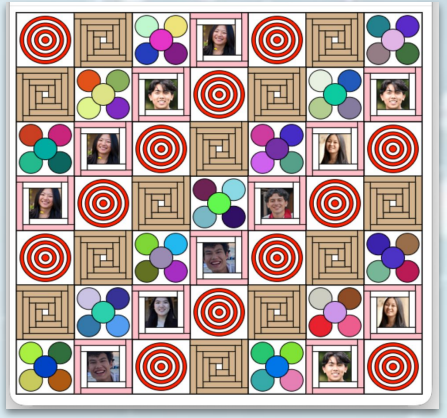
Slides at cs106ax.stanford.edu/assignments.html





CS106AX YEAH: Assign1

JavaScript Programs

Autumn 2025

cs106ax.stanford.edu

Stanford | ENGINEERING
Computer Science

Welcome to CS106AX!

Hope that you had a wonderful summer break and first week of classes!!

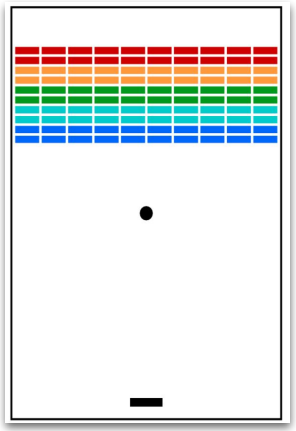
Autumn Week 2 **Winter**

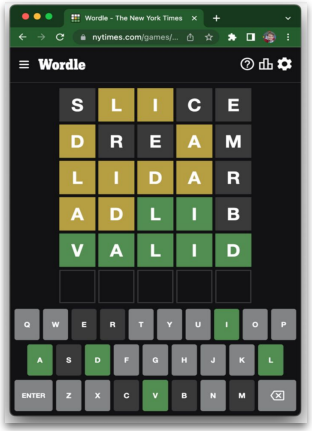
Your Early Assignment Help!

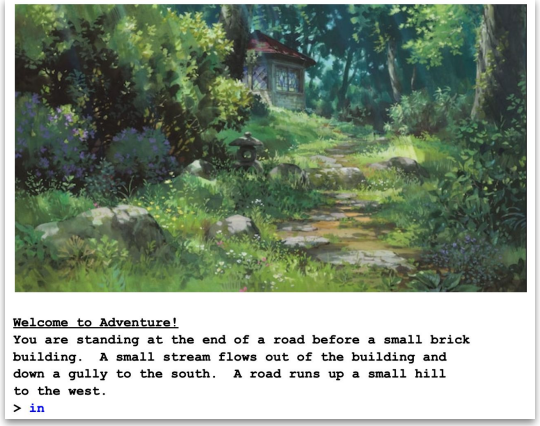
What are YEAH Hours?

- Held after assignment is released – we're piloting this for 106AX, and if interest, we'll continue for the next assignments! :)
- Early **assignment overview and tips, Q & A** for any clarifications!

A preview of future assignments!

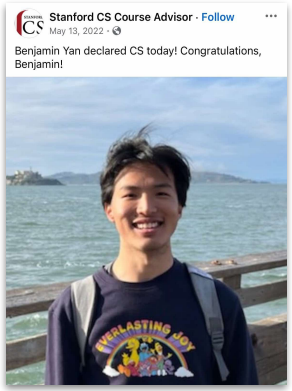


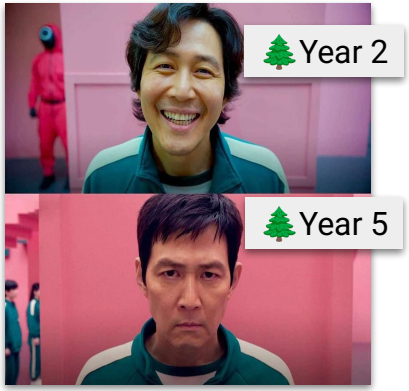


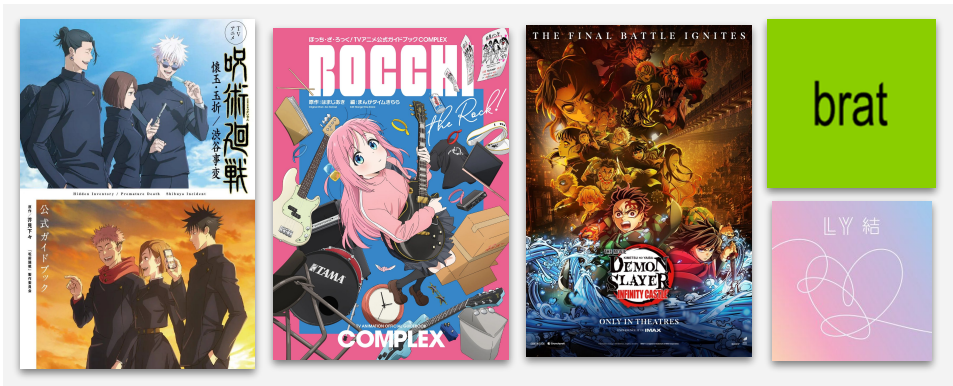


👋 Hi I'm Ben!

- '23 – '26: **CS coterm / MS student** graduating in June (nearly there! 🎓)
- '20 – '24: Just finished my undergrad here! Double Major in **CS and Math**, **Creative Writing** Minor and study abroad at Oxford (was a wild 4 years lol)
- Head TA for **CS 106AX** last year, and **elated to be back in AX this autumn**!







- Likes: manga & anime (favorite is *Jujutsu Kaisen*), K-pop, rock music 🎸, fiction writing (once wrote 2 novels in a month for a class, but have written very little since 😰), draining CS department money on boba 🧋, language learning

Intros!

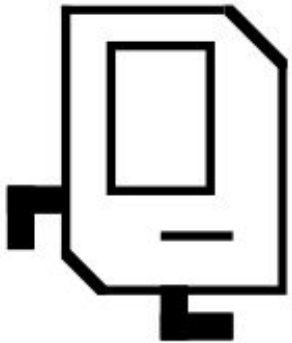
- ❑ Name & pronouns if you're comfortable sharing!
- ❑ What you're studying / thinking about studying
- ❑ Year
- ❑ Fun fact 😨😱 or **any one of the following!**

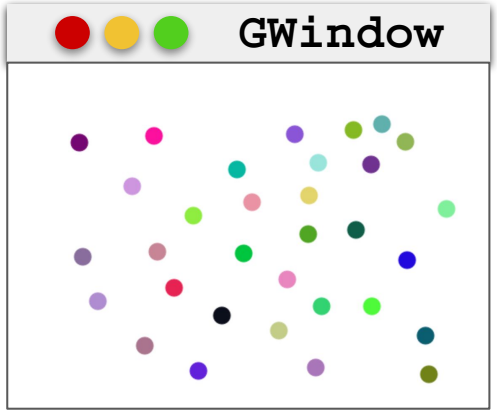
🌲 What are you looking forward to this quarter / year?
🦋 Something fun you did over the summer break?
🎶 Music, album, or book recommendations?

Or anything else you'd like to share! :)

The Map For Today

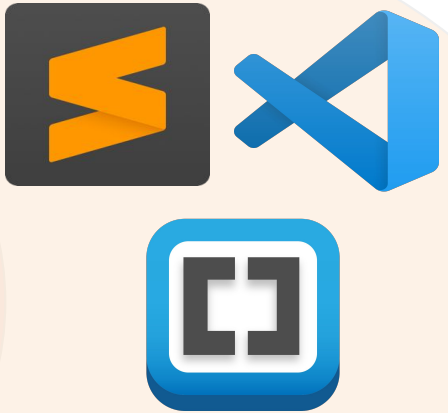
1. Getting setup, relevant tools
2. Quick recap of lecture, JavaScript
3. Assignment overview & tips
4. Questions!





Getting Set Up

Have a browser for running & debugging code, e.g. **Chrome**

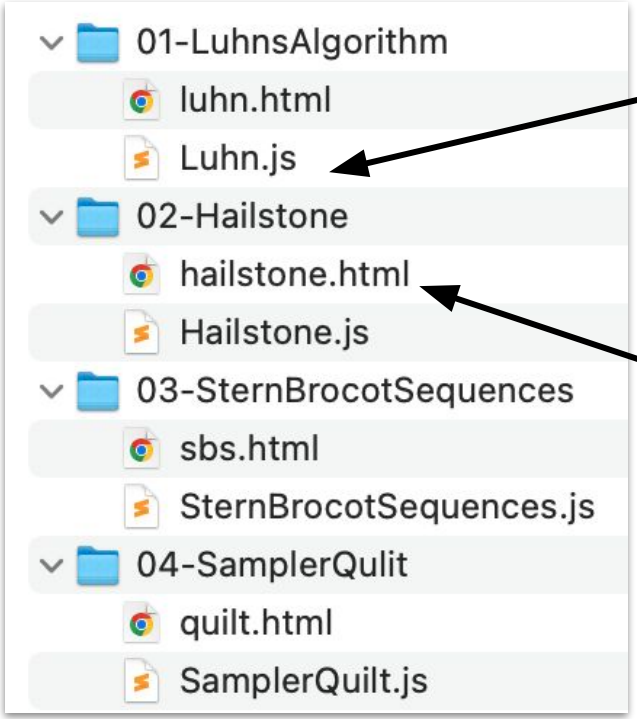


install **Sublime Text**, **VS Code**, or a text editor of your choice!

Downloading Assign1 Code

Go ahead and download the **Assign1** code from **this link** (on course website).

Then, open up the files or entire folder in your text editor!

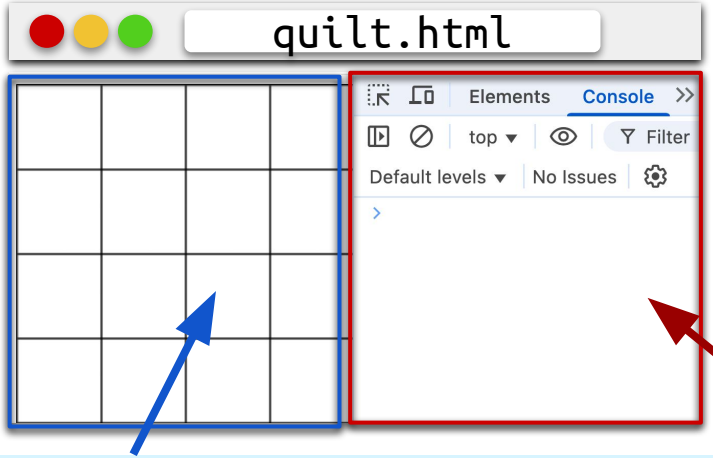


Write code in the JavaScript files (.js)

Test code by opening the corresponding HTML file in a web browser (e.g., Chrome)

Note: When editing the code file, make sure to both (1) **save the code**, and (2) **refresh the browser tab** so the changes are reflected in outputs!

Opening JavaScript Console in Chrome



1. Open relevant **html file** in Chrome
2. On Mac: Press **cmd** – **option** – **j**

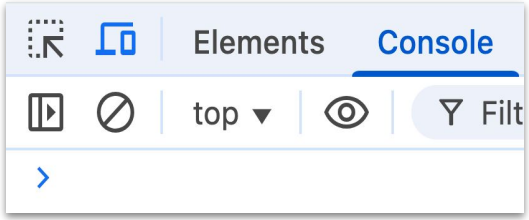
   On Windows: Press **ctrl** – **shift** – **j**

Don't let go of the previous key while pressing the next!

Can **check output** (e.g., graphics) on the body of the **webpage**

In the console that pops up, we can run JS code / **see print and error messages**

Alternatively, can **double-click** on the webpage – then press **Inspect**, and change the top tab from Elements to **Console**



Lecture Recap!

⏳ ~10 minutes

Hello JavaScript!

✨ a “vibes-based” language – we’ll see more of what means :)

my favorite programming language lol

In JavaScript, we use the **let** keyword to define variables of any type, e.g.,

```
let variableName = expression;
```

Here, **= is the assignment operator** — assigning a value from the right side to the variable on the left. It can also re-assign a new value to an existing variable.

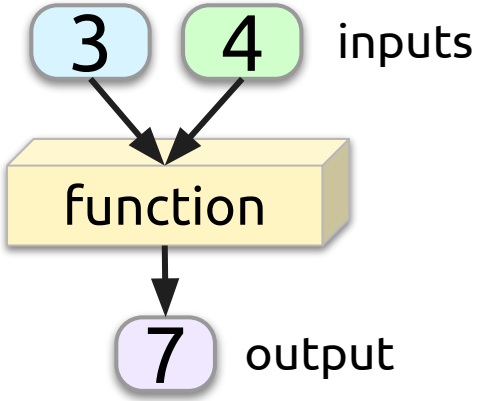
Note of caution: **=** is for assignment, **===** and **!==** for comparison

```
let numClasses = 4;
numClasses = 5; // too many! :(
numClasses = numClasses – 2; // now 3
```

```
// can use const for fixed values
// note capitalization for style
const OCTOBER_DAYS = 31;
```

P.S. The semi-colon (;) at the end of lines is optional – up to you!

JavaScript Functions



The general structure for defining a function is:

```
function functionName(list of parameters){
   list of statements in function body
}
```

You can **optionally return** a single value of any type, i.e. **return expression;** To see this in action, consider the short example:

```
function addTwoNums(x, y){
        let sum = x + y;
        return sum;
}
```

To **call / invoke a function**, you supply the function name — and pass in values for each parameter:

```
let sum1 =  addTwoNums(3, 4);
// can also compose function calls
let sum2 =  addTwoNums(
        addTwoNums(1, 2), 3);  // 3+3=6
```

🔁 Control Structures: For vs While

In addition to if / else if / else {} branching, the problems in `Assign1` will likely involve some kind of **control structure or loop**. There's 2 main types!

For loop
Generally use **when we know how many times (definite)** to iterate

```
for (init; test; step){
    statements in loop
}
```

```
for (int i = 0; i < 10; i++){
   console.log("Hello!");
} // prints 10 times
```

While loop
Generally use when **we don't know beforehand (indefinite)** how many times we will iterate

```
init
while (test){
    statements in loop
    step
}
```

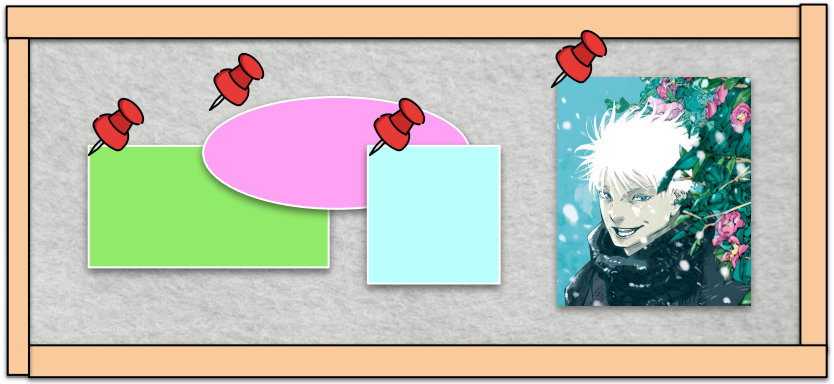
```
let roll = randomRoll(); 🎲
while (roll !== 6){
   console.log(roll);
   roll = randomRoll();
} // prints until 6 is rolled
```

Based on CS106A YEAH slides by Nick Troccoli

General JS Graphics Review

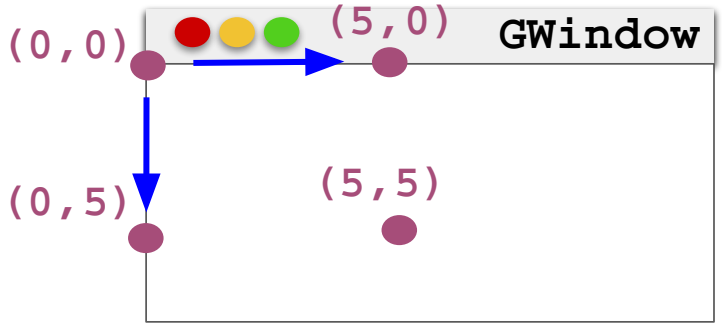
You can find library documentation at https://jdkula.github.io/jsgraphics-docs!

🖼️ **`GObjects`** usually reside in a **`GWindow`**, which is like a bulletin board for tacking on colorful objects.



📍 For most **`GObjects`**, the location coordinate **`(x,y)`** is defined as the **upper left** corner of the object.

🧭 It's quirky, but unlike in math, the **`y` coordinates go up** as we go downward.



General JS Graphics Methods

You can find library documentation at https://jdkula.github.io/jsgraphics-docs!

```
GWindow(width, height);
GLine(xStart, yStart, xEnd, yEnd);
GOval(x, y, xDiameter, yDiameter);
GRect(x, y, width, height);
GImage(image-url);
```

GObject constructor methods

any GWindow()

```
gw.add(object)
gw.add(object, x, y)
```

any GObject()

```
let width = object.getWidth()
let height = object.getHeight()
object.setColor(color)
object.setLocation(x, y)
```

any GRect() or GOval()

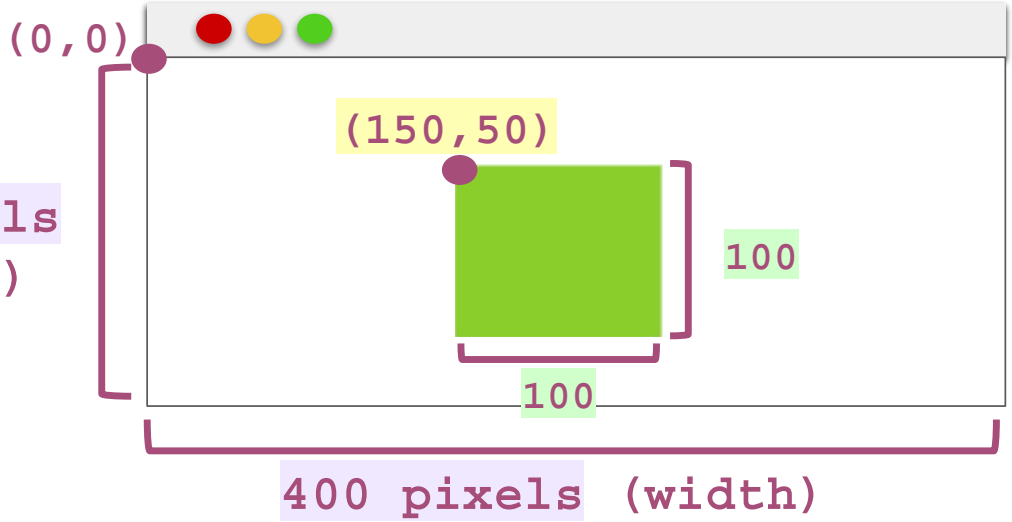
```
object.setFilled(true/false)
object.setFillColor(color)
```

Methods you may find useful!

Graphics Example: BRAT Square

```
// window of width 400, height 200
let gw = GWindow(400, 200);

// 100x100 square, top left at (150,50)
let square = GRect(150,50, 100, 100);
```

```
// fill square with brat green color
const BRAT_GREEN = "#8ACE00";
square.setColor(BRAT_GREEN);
square.setFilled(true);
gw.add(square);
```



Happy to discuss any questions!

Next Up: The first assignment yay!

1. Validating Credit Card Numbers

Many creditors ensure that new credit card numbers are valid according to **Luhn's algorithm** – a **digit manipulation** algorithm.

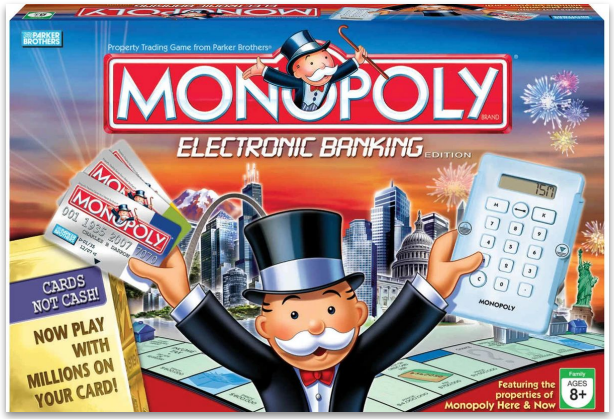
Your task is to write **`isValid(number)`**

➔ Returns **`true`** if credit number is valid

➔ Return **`false`** otherwise

Write code / additional tests in: **`luhn.js`**

Check output results in: **`luhn.html`**

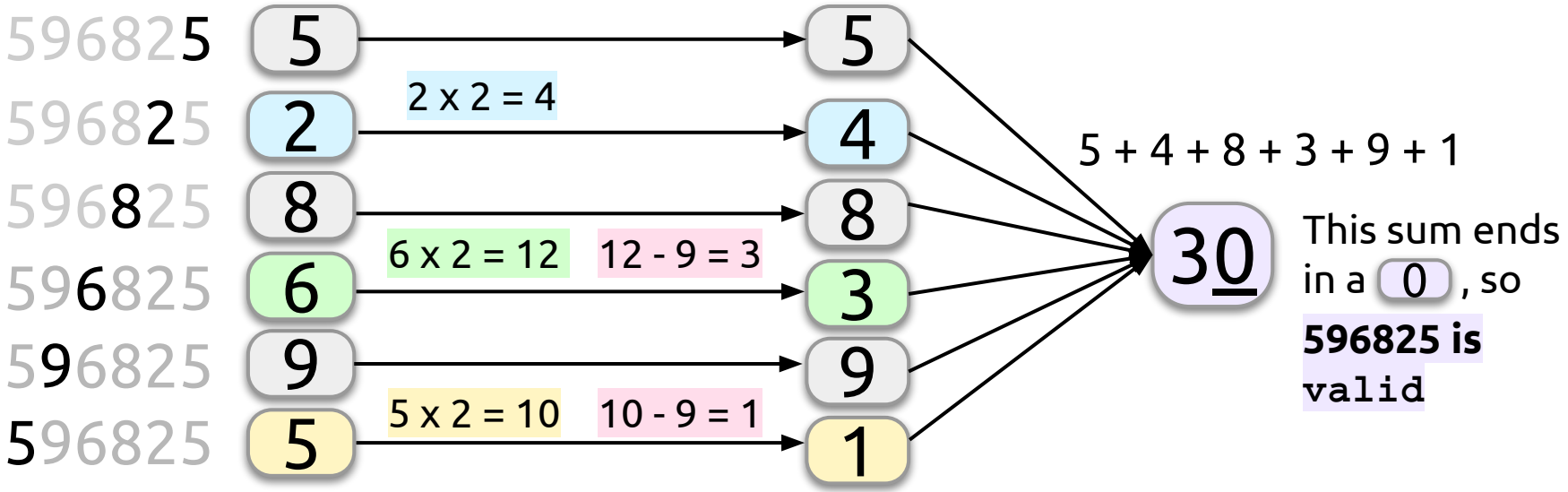


credit card declines

How do we know if a credit card number, e.g., 596825, is valid?

1. Validating Credit Card Numbers

How do we know if a credit card number, e.g., 💳 `596825`, is valid?



1. Transform each digit:
Start from right, and move to the left, doubling every 2nd digit. If doubling produces value >9, subtract 9 from it.

2. Sum up the transformed digits

3. If sum ends in <u>0</u>, card number is ✅ valid, else ❌ invalid

Luhn's Algorithm Tips

Tip: In JavaScript, **%** is the remainder or modulo operator (**n % d** is the remainder when **n** is divided by **d**).

JS Console

```
4 % 2 ⇒ 0
5 % 2 ⇒ 1
16 % 5 ⇒ 1
9 % 10 ⇒ 9
1234 % 100 ⇒ 34
```

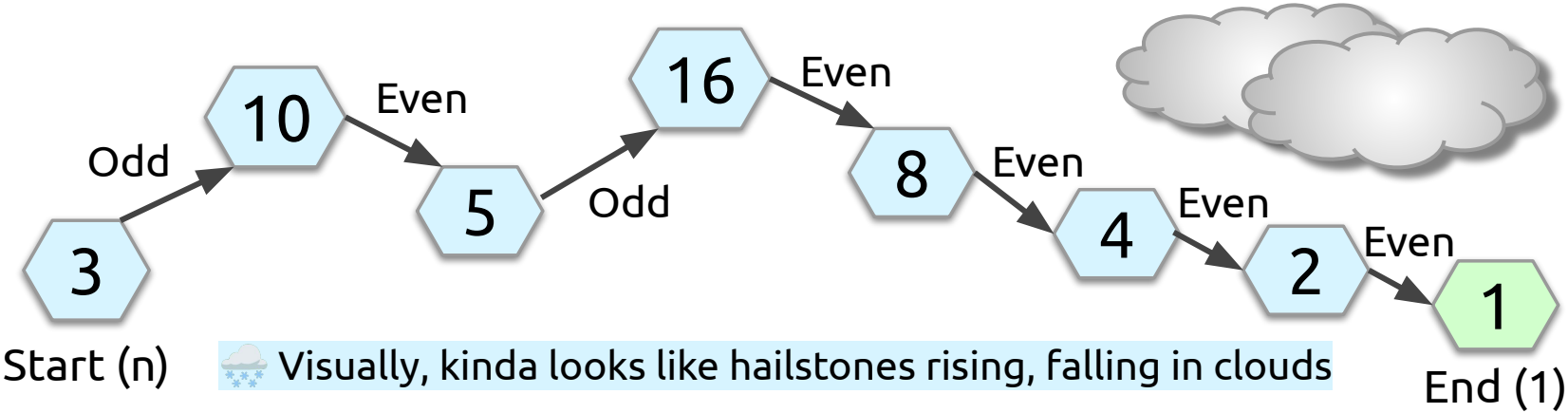
`Math.PI`	The mathematical constant π
`Math.E`	The mathematical constant e
`Math.abs(x)`	The absolute value of x
`Math.max(x, y, ...)`	The largest of the arguments
`Math.min(x, y, ...)`	The smallest of the arguments
`Math.round(x)`	The closest integer to x
`Math.floor(x)`	The largest integer not exceeding x
`Math.log(x)`	The natural logarithm of x
`Math.exp(x)`	The inverse logarithm (e^x)
`Math.pow(x, y)`	The value x raised to the y power (x^y)
`Math.sin(θ)`	The sine of θ, measured in radians
`Math.cos(θ)`	The cosine of θ, measured in radians
`Math.sqrt(x)`	The square root of x
`Math.random()`	A random value between 0 and 1

You may also find some functions in the JS **math library** applicable!

2. Hailstone Sequences

- Pick some positive integer and call it n.
  - If n is even, divide it by 2.
  - If n is odd, multiply it by 3 and add 1.
- Continue this process until n is equal to 1

Fun Fact: Whether all values of n eventually descend to 1 is an unsolved problem – the **Collatz conjecture** 😨



Visually, kinda looks like hailstones rising, falling in clouds

Hailstone Sequence Print-Out

- In `hailstone.js`, you'll write a function to generate and print out the Hailstone sequence — **matching the handout format** — starting at input n
- In `hailstone.html`, you can test it out by entering `hailstone(n)`

hailstone.html

```
Use this expression evaluator to ensure
your hailstone implementation works.
> hailstone(5)
5 is odd, so I make 3n+1: 16
16 is even, so I take half: 8
8 is even, so I take half: 4
4 is even, so I take half: 2
2 is even, so I take half: 1
The process took 5 steps to reach 1.
```

Print out each number, and how we got there

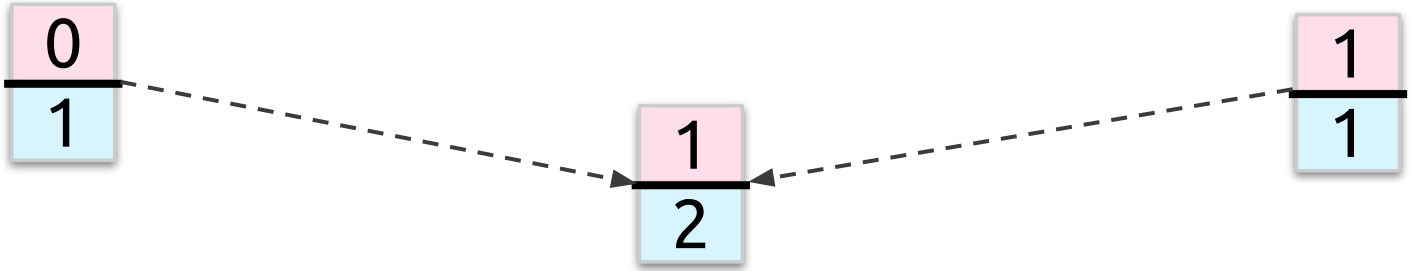
At the very end, print out how many steps it took to reach 1

definitely more algorithmically intense than P1 and P2, but you got this!

3. Stern-Brocot Tree and Sequences

The nodes of the Stern-Brocot tree are **fractions between 0 and 1**.

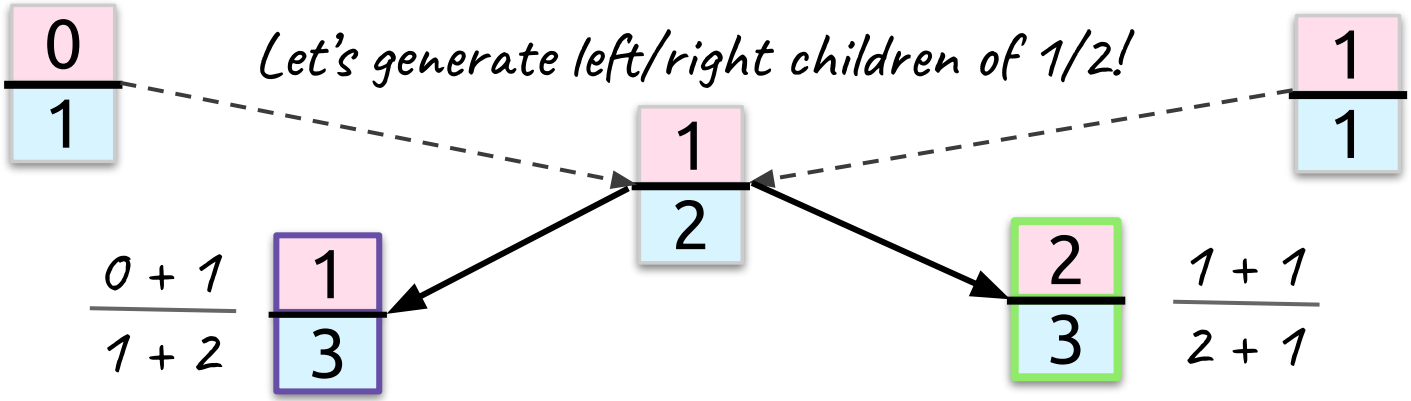
- **Starts with node 1/2**, the child of 0 (0/1) and 1 (1/1).



3. Stern-Brocot Tree and Sequences

The nodes of the Stern-Brocot tree are **fractions between 0 and 1**.

- **Starts with node 1/2**, the child of 0 (0/1) and 1 (1/1).
- From there, each node is **(nL + nR) / (dL + dR)**, where **nL / dL** is the closest ancestor up & to the **left**, and **nR / dR** is the closest ancestor up & to the **right**

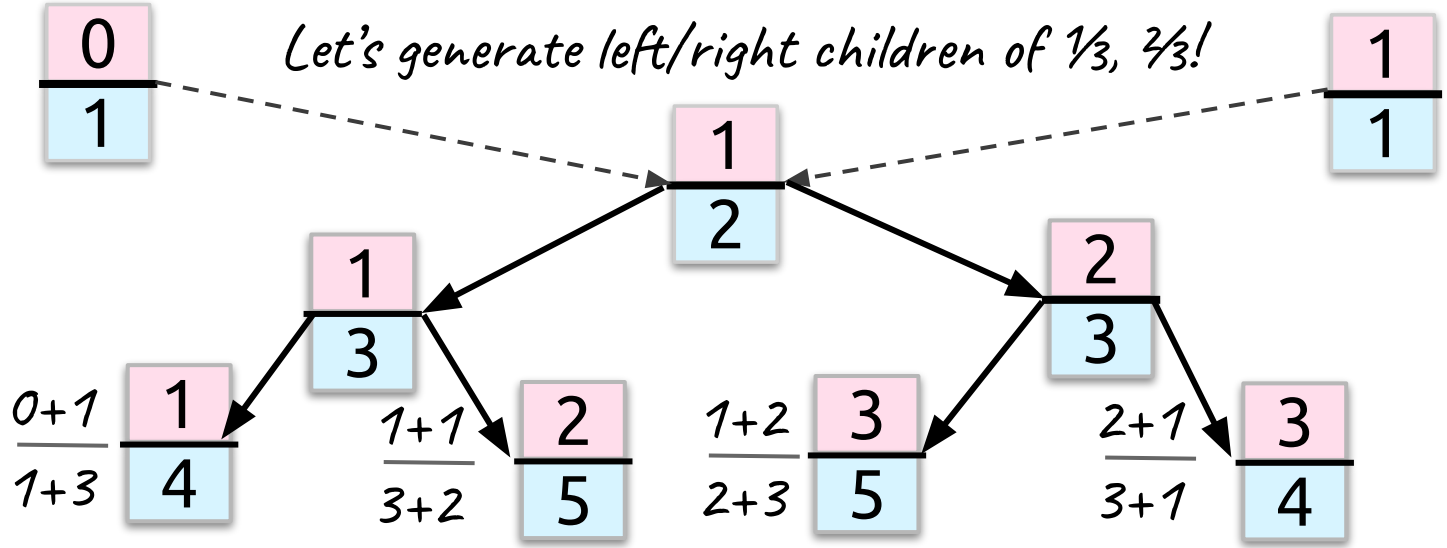


🌱 New node 1/3 is produced from 0/1 on the left and 1/2 on the right

🌱 New node 2/3 is produced from 1/2 on the left and 1/1 on the right

3. Stern-Brocot Tree and Sequences

Each node is **(nL + nR) / (dL + dR)**, where **nL / dL** is the closest ancestor up & to the **left**, and **nR / dR** is the closest ancestor up & to the **right**



🌱 Node **1/4**
L: 0/1, R: 1/3

🌱 Node **2/5**
L: 1/3, R: 1/2

🌱 Node **3/5**
L: 1/2, R: 2/3

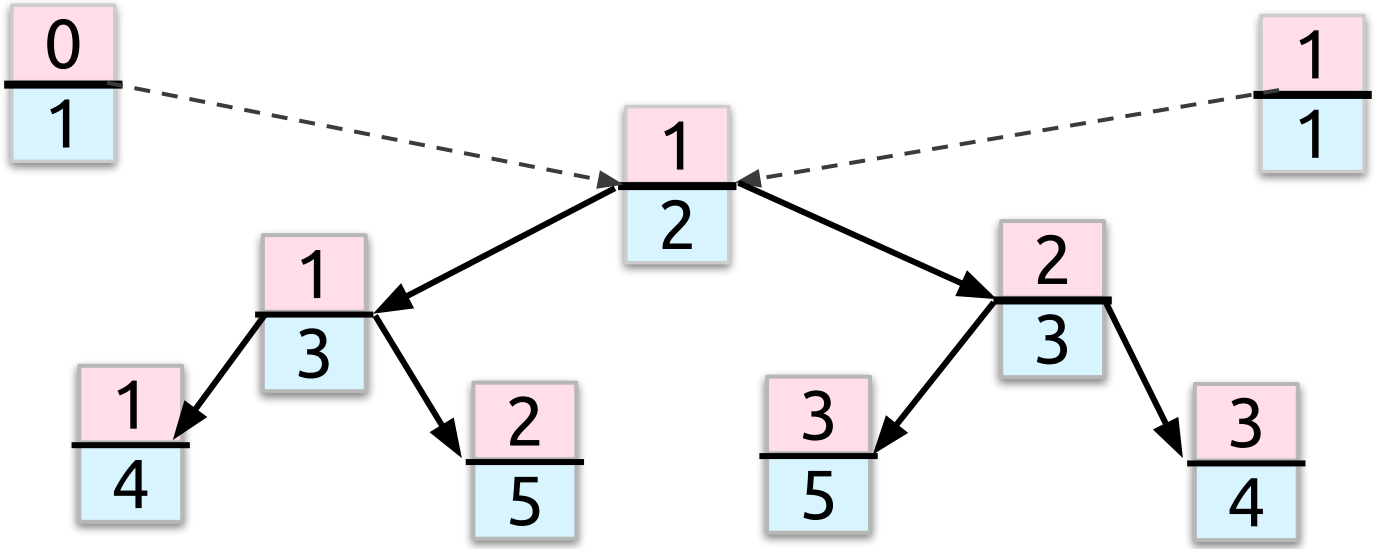
🌱 Node **3/4**
L: 2/3, R: 1/1

As we keep branching, we'll eventually reach all fractions between 0 and 1!

3. Stern-Brocot Sequence Representation

For this problem, we're more interested in **how we reach a particular fraction by navigating / descending this tree** – as opposed to building the entire tree.

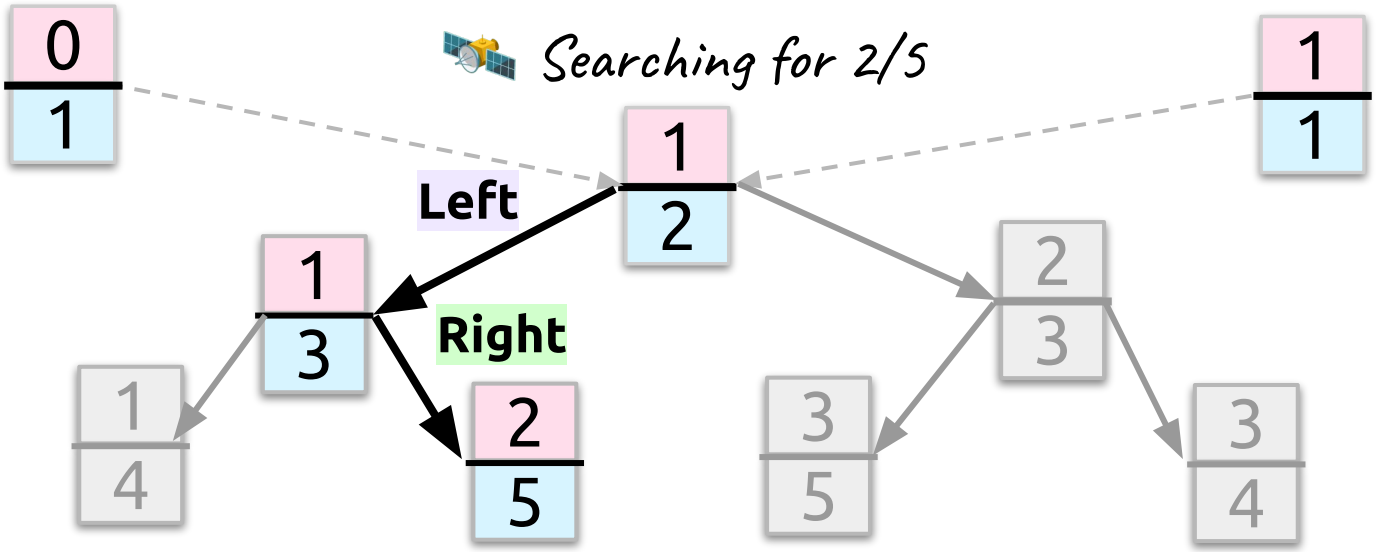
That is, a zigzag of downward **left (L) or right (R) movements** starting from 1/2!



Note: **L** moves to smaller fraction
R moves to larger fraction

3. Stern-Brocot Sequence Representation

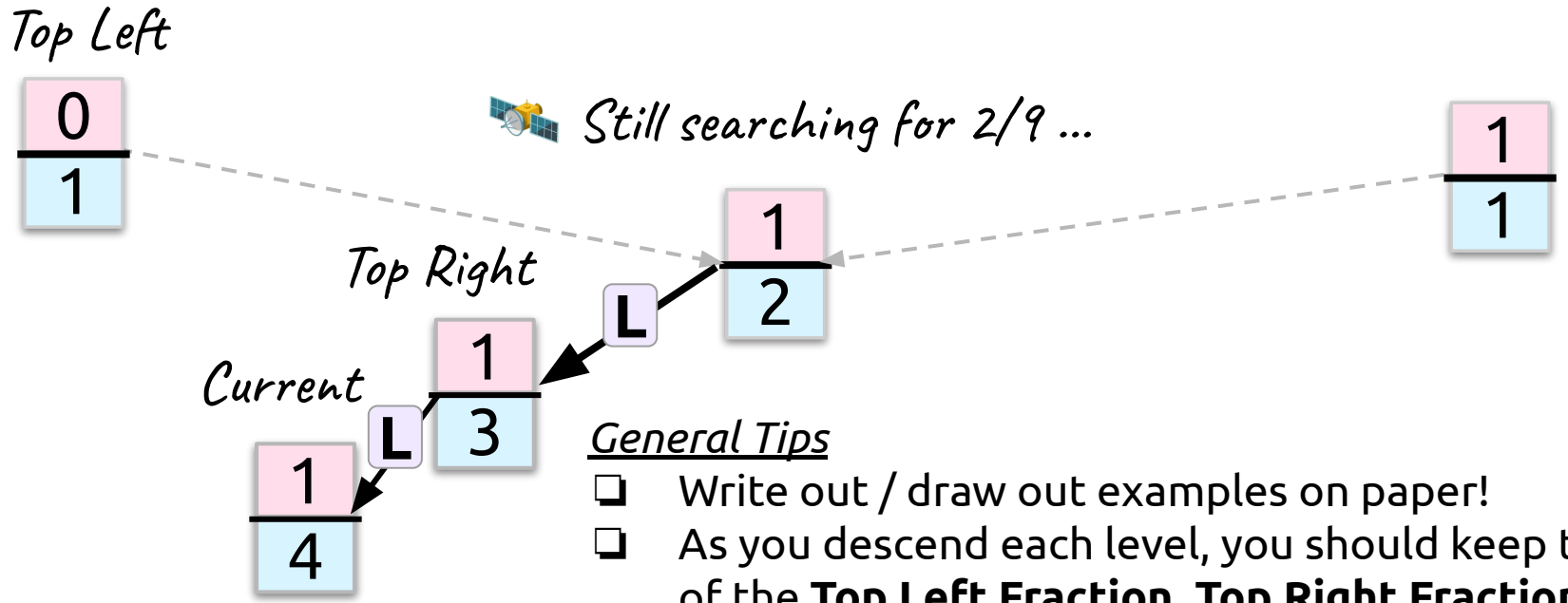
For this problem, we're more interested in **how we reach a particular fraction by navigating / descending this tree** – as opposed to building the entire tree.
That is, a zigzag of downward **left (L) or right (R) movements** starting from 1/2!



🌱 For example, to reach **2/5**, we first go **left** to 1/3, then **right** to 2/5 — so sequence is **LR**.

Note: **L** moves to smaller fraction
R moves to larger fraction

3. Stern-Brocot Sequence Representation

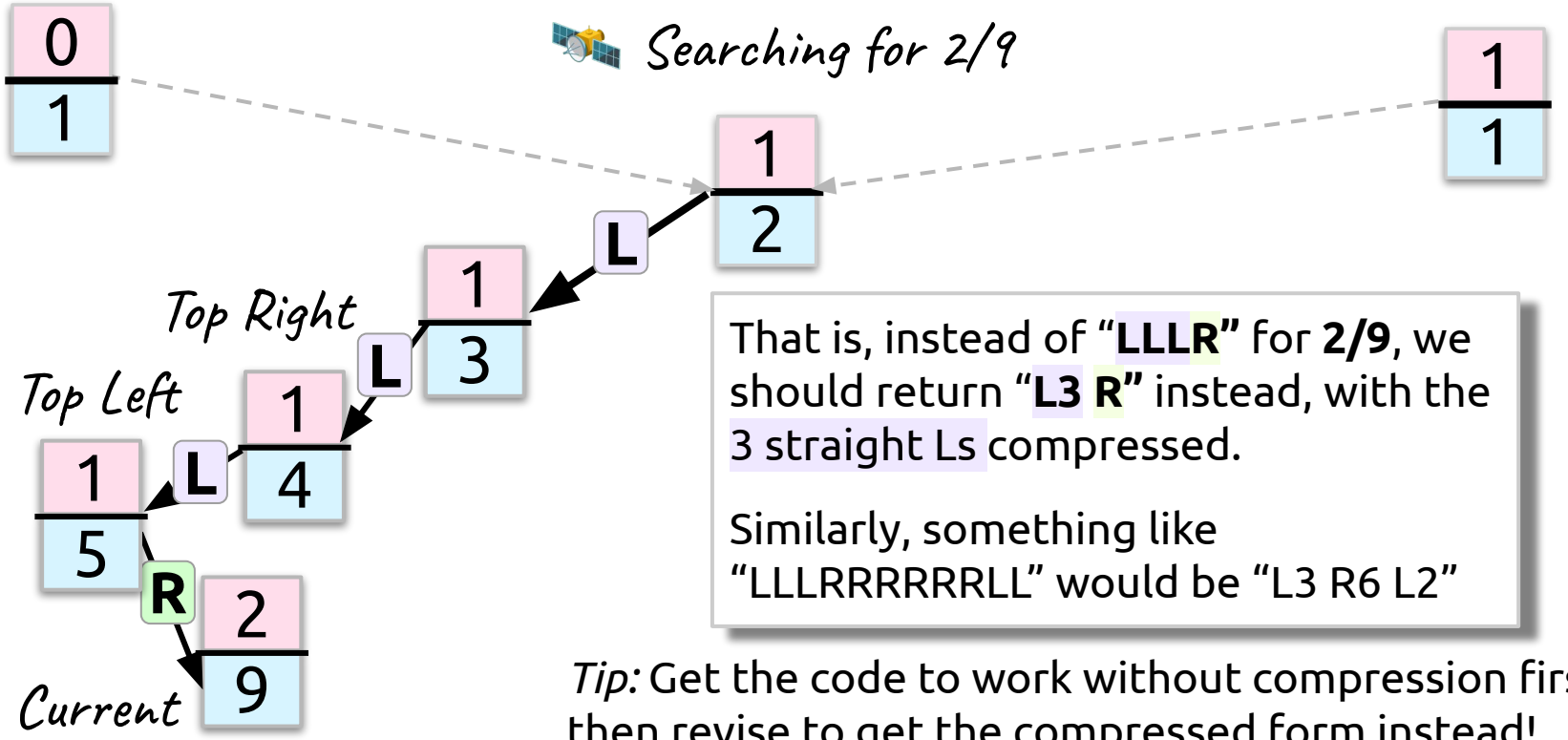


General Tips

- ❏ Write out / draw out examples on paper!
- ❏ As you descend each level, you should keep track of the **Top Left Fraction, Top Right Fraction,** and **Current Fraction.**
- ❏ Store numerator/denominator as separate integers (avoiding storing fractions as decimals)

3. Stern-Brocot Sequence Compression

Note: Whenever a letter occurs 2 or more times in a run, we should compress it



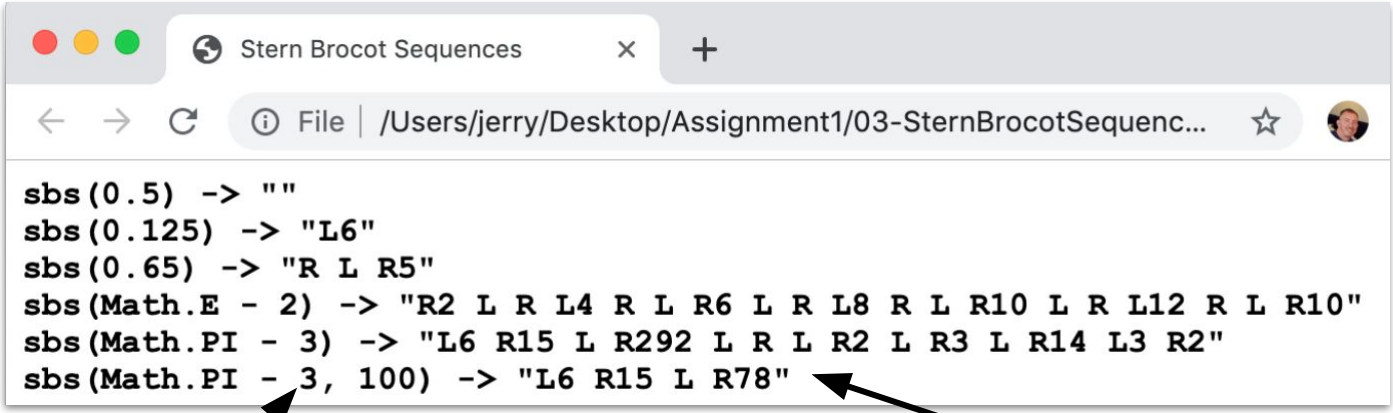
That is, instead of "**LLLR**" for **2/9**, we should return "**L3 R**" instead, with the 3 straight Ls compressed.

Similarly, something like "LLLRRRRRLL" would be "L3 R6 L2"

Tip: Get the code to work without compression first, then revise to get the compressed form instead!

Stern-Brocot Max Sequence Length

Lastly, we have a maximum sequence length (default = 500) that, once reached, we should return the sequence we have. Think about it as an **approximation.**



For the last one, **π - 3 ≈ 0.1415...** is **irrational** — no matter how long our sequence is, we'll never be able to hit it precisely :(

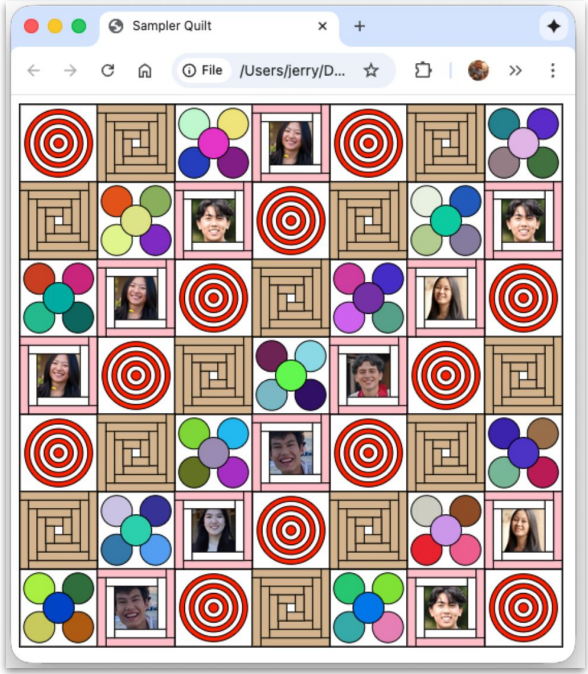
However, **L6 R15 L R78** is a decent approximation, getting very close with just **100** left/right moves.

~~notorious~~

4. Sampler Quilt (A Super Fun Classic!)

The (wow!) picture says it all pretty much – in `quilt.html`, you'll want the GWindow to appear visually as the board on the right.

Note: Don't worry about exact pixel-by-pixel matching – **we focus more on code clarity and narrative / objective.** Art reflects the individual soul after all. *If you have questions, let us know!*



Awesome 7x7 board of square patches of alternating types

Tip: Loops can be nested!

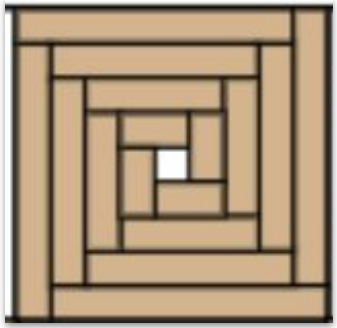
```
for (let i … ){
    for (let j … ){
    }
}
```

Tip: To think about the *alternating* pattern, I'd review the **chessboard example** from lecture

4 Different Varieties of Patches

🎯 Bullseye

7 evenly spaced and patch-centered circles, alternating red and white; black borders.



🏕️ Log Cabin Patch

4 square frames of logs, with a white square at the center. The square dimension equals the log's shorter side

💐 Flower Patch

5 circles of random color; radius ⅕ of patch dimension; arranged as shown

❤️ Section Leader Patch

2 square frames (outer pink, inner white) that are laid atop the `GImage` of a randomly selected SL

Tip: Tackle each patch one at a time — and **draw them out on paper, tracing objects and key coordinates!** That is, take advantage of your **art and geometry** skills :)

Tip: You may notice recurring shapes (circles, logs) – helper functions can really shine!

Useful for Patch: GCompound Object

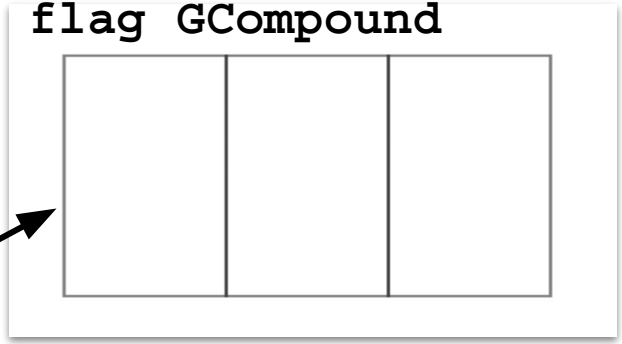
A **`GCompound` is an elegant container of multiple objects** with its own local origin, akin to a `GWindow`, but you can also move it around the screen as a whole unit.

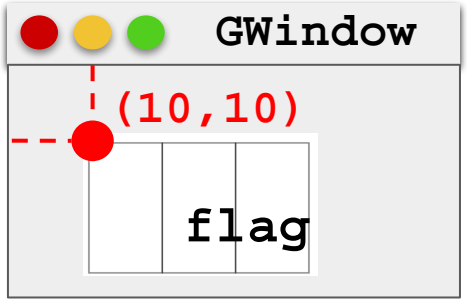
Tip – For each patch, you can create a GCompound!

Example: Create a **triband flag**, e.g. France 🇫🇷, as a compound object, and add it to the window

```
function createFlag(width, height){
   let flag = GCompound();

   // TODO

   return flag;
}

// create and add a flag to the gwindow at (10,10)
let franceFlag = createFlag(30, 20)
gw.add(franceFlag, 10, 10);
```

Note the flag is composed of **3 equally sized rectangle objects**





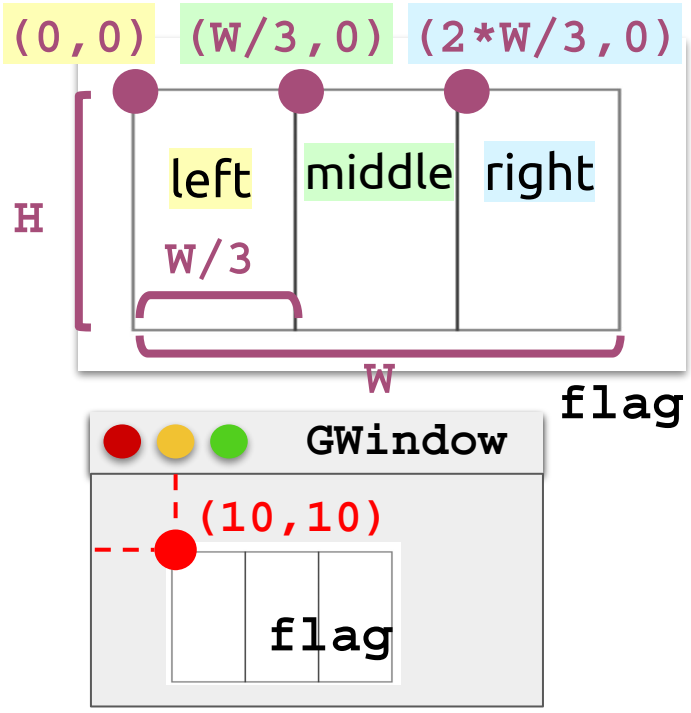
Useful for Patch: GCompound Object

A **GCompound is an elegant container of multiple objects** with its own local origin, akin to a `GWindow`, but you can also move it around the screen as a whole unit.

Tip – For each patch, you can create a GCompound!

Example: Create a **triband flag**, e.g. France 🇫🇷, as a compound object, and add it to the window

```
function createFlag(width, height){ // 3 components
   let flag = GCompound();
   let left = GRect(0, 0, width / 3, height));
   let middle = GRect(width / 3, 0, width / 3, height));
   let right = GRect(2 * width / 3, 0, width / 3, height));
   flag.add(left); flag.add(middle); flag.add(right);
   return flag;
}
// create and add a flag to the gwindow at (10,10)
let franceFlag = createFlag(30, 20)
gw.add(franceFlag, 10, 10);
```



Final First Assignment Tips

- Read the assignment carefully, and test your code incrementally
- **Draw pictures, add comments** if that helps organize your thoughts
- Try not to start the night before it's due haha
  - I know all too well, believe me :(
- If you have questions, get stuck, or find yourself in a time / debugging crunch, we'd be happy to see you in **office hours**!

*And most importantly of all, **have fun and believe in yourself!** You're going to do awesome :)*

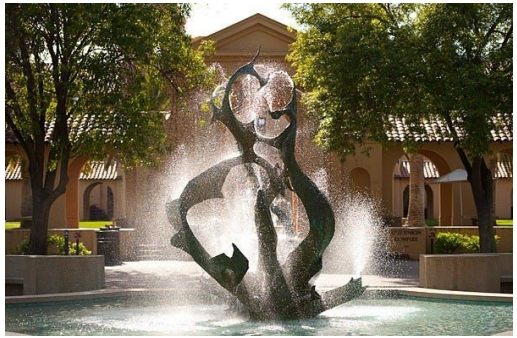
Looking Forward to this Autumn 🍁

Teaching CS has been a truly wonderful joy and privilege for me, ~~because of all the free boba over the years~~

Thank you for being here to learn with us, and I 👏 hope 👏 this 👏 will 👏 be 👏 fun 👏 for you!

Spring 2025, CS106s End-Quarter Boba Party

Jerry = best advisor 🎉


Have an awesome week!! :)

Please reach out or ask me afterward if you have any questions!
📬 Contact: **`bbyan@stanford.edu`**

I always really appreciate and value your feedback; also, it's not a bad time to exchange contact info and find study partners / groups!