COMPSCI 326 - Web Programming JavaScript Objects, Arrays, Functions, Iteration

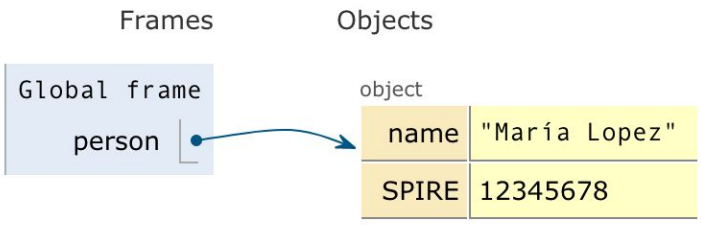
join on the Slack #q-and-a channel as well as Zoom
remember, you can ask questions of your teammates on your group Slack!
please **turn on your webcam** if you can
mute at all times when you aren't asking a question

Background resources:

(lots of details)
JavaScript data types and data structures - JavaScript | MDN
Object - JavaScript | MDN
Functions - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Today: More JavaScript: Objects, Arrays, Functions, and Iteration

```
let person = { 'name' : 'María Lopez', 'SPIRE' : 12345678 };
```



images from http://pythontutor.com/javascript.html#mode=edit - great place to try out snippets of JavaScript code!

Objects: we can access objects in two ways:

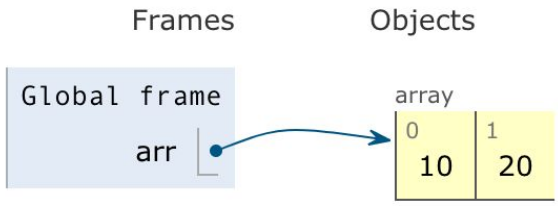
- like an array, with brackets
  - `person['name']` → `'María Lopez'`
  - `let id = 'name';`
  - `person[id]` → `'María Lopez'`
- or like an object (as in Java)
  - `person.name` → `'María Lopez'`
- you can check if something is in an object
  - `'name' in person` → `true`
  - `'age' in person` → `false`
- and you can remove a field
  - `delete person.name` → `person === { 'SPIRE' : 12345678 }`

When to use: brackets vs. dots:

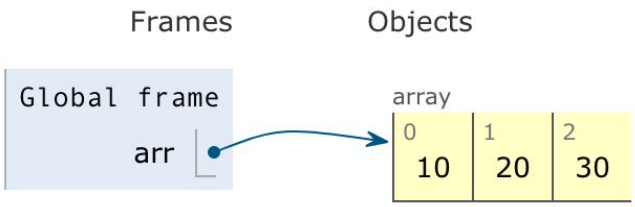
- *brackets* - [] when the field is a variable (like `person[x]`)
  - it's your only choice then
- *dots* - . when you know what the field is (like `person.name`)
  - less verbose and more JavaScripty
  - and your misspellings will be caught easier!

Arrays: using them, indexing them, iterating over them

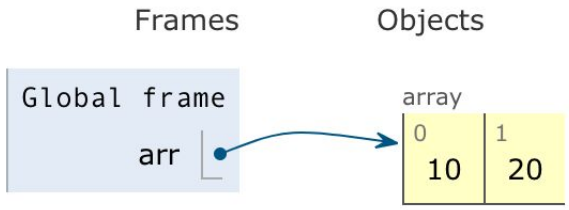
- indexing
  - `let arr = [10, 20];`



  - `arr.length` → `2`
  - `arr[0]` → `10`
  - `arr[12]` → **`undefined`**
  - `!arr[12]` → `true`
    - special values like **`undefined`** and **`null`** are *"falsy"*
    - they act like `false`, so when you negate them, they are true
    - what about things that aren't *falsy*?
    - ```
      if (0) {
        console.log('does this print anything?');
      }
      ```
- can treat like a *stack* with **push** and **pop**
  - `arr.push(30)` → returns 3 (new length of array), and arr === [10, 20, 30] // add to end

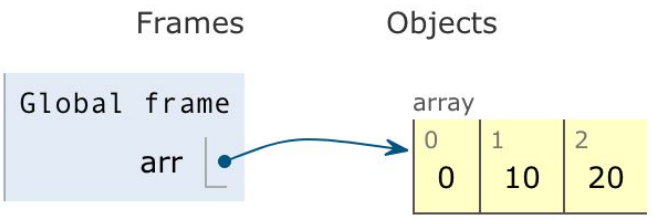


  - `arr.pop()` → returns 30, and now arr === [10, 20] // remove from end

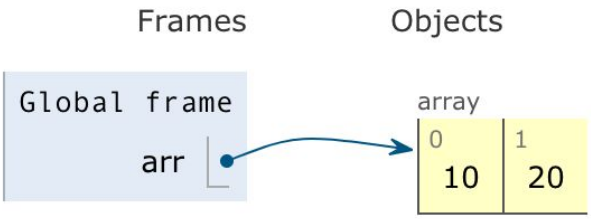


  - can also use the front of the array with **unshift** and **shift**
      ```

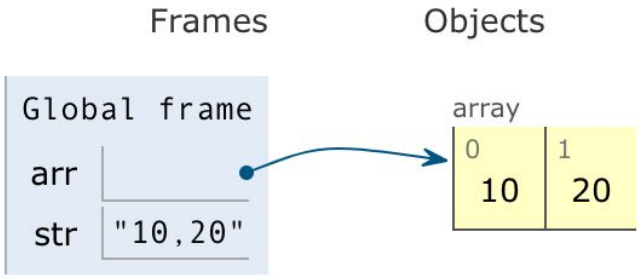
        - `arr.unshift(0)` → returns 3 (new length of list), and arr === [0, 10, 20] // add to front



        - `arr.shift()` → returns 0, and now arr === [10, 20]

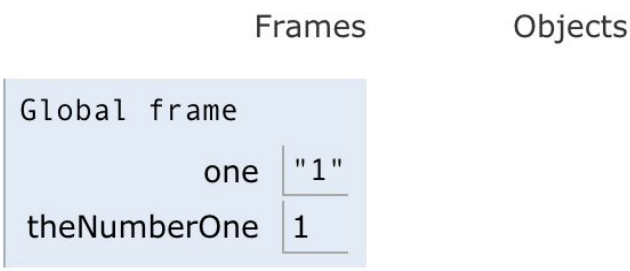


- to simulate a *queue* (first-in, first-out = "FIFO"):
    - **`unshift`** to insert into the front, **pop** to remove from the back
- converting them to *strings* and back: **join** and **split**
    - `let str = arr.join(',')` → returns "10,20"



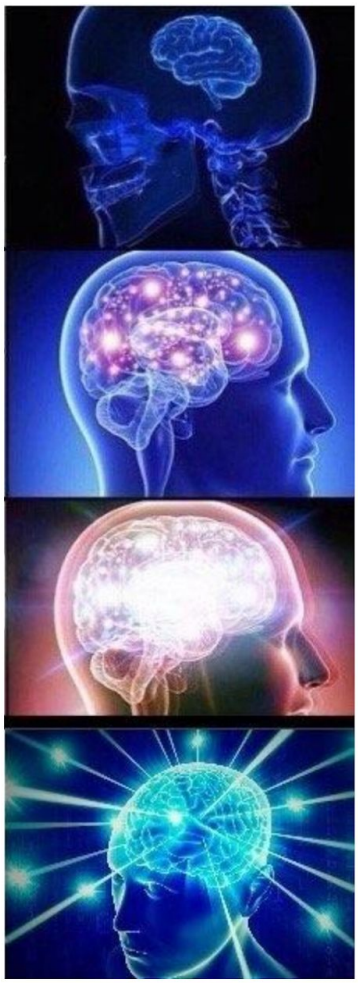
    - `str.split(',')` → returns ['10', '20'] // hmm
        - `str.split('')`
        - `str.split(' ')`
    - to convert a string back to a number, use **Number.parseInt**
        - `let one = '1';`
        - `one + 1` → `'11'`

- let theNumberOne = **Number.parseInt**(one); → 1 // a number



- theNumberOne + theNumberOne → 2
- theNumberOne.**toString()** → '1'

- **Iterating with arrays** - SO MANY WAYS!
  - you can iterate *with indices* (worst way, very unJavaScript)
    - `for (let i = 0; i < arr.length; i++) { console.log(arr[i]); }`
  - you could use **`shift`** (destroying the array - what you want sometimes)
    - `while (arr.length > 0) { console.log(arr.shift()); }`
  - you can iterate with `let` **`of`**:
    - `for (let item of arr) { console.log(item); }`
- **Functions**
  - in JavaScript, functions can be passed around and stored, just like any other type (they are "first-class")
  - you can make your own "*anonymous*" functions wherever you want by using the **`() =>`** notation
    - `(arg1, ... arg2, ...) => { ... // do something; possibly return a value }`
  - *these are the same (for our purposes right now)*:
    - `function print(item) { console.log(item); }`
    - `let print = (item) => { console.log(item); }`
- so you can iterate over arrays like this, using **`forEach`**:
  - `arr.forEach((item, index) => console.log(item));`
    or
    `arr.forEach(print)`
- **map** is when you want to do something with every element and make it into a new array
  - `let arrSquared = arr.map((item) => item * item);`
    which is an abbreviation for
    `let arrSquared = arr.map((item) => { return item * item; })`



```
for (let i = 0; i < arr.length; i++) {
  console.log(i);
}
```

```
for (let item of arr) {
  console.log(i);
}
```

```
console.log(arr.join('\n'));
```

```
arr.forEach((item) => console.log(item));
```

Exercise!

https://docs.google.com/document/d/1crze-uIiJyh9U-utTBKeufO99IlFdUsnbrpIo7QQP5c/edit?usp=sharing