Approach Lecture

...

JS Fundamentals

Objectives

In this lecture, we'll explore...

- Memoize
  - Closure
  - Rest Param/Spread Operator
  - JSON.stringify()
- cloneDeep
  - Shallow vs deep copy
- Throttle
  - Closure
  - Date.now()
  - setTimeout()

memoize

```
/**
 * Returns a function that when called, will check if it has already computed
 * the result for the given argument and return that value instead if possible.
 */
```

Why do we need memoization?

memoize

Optimization technique used to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again

```
/**
 * Returns a function that when called, will check if it has already computed
 * the result for the given argument and return that value instead if possible.
 */
```

Closure review

Why?

- Provide functions with persistent memory
- Protect data without polluting global variable environment
- e.g. counters, caching expensive functions

How?

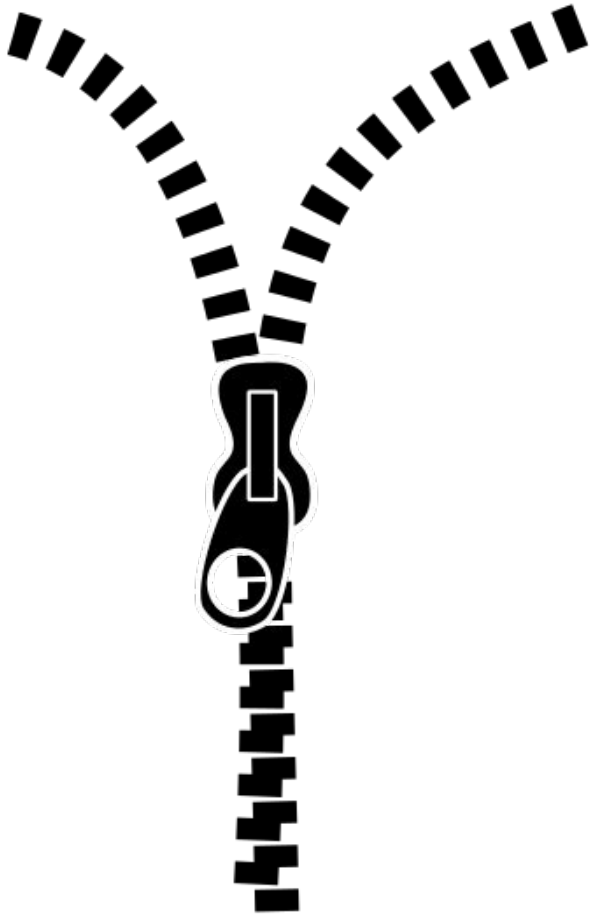
- Outer function definition stored in global memory
- Store any variables inside of outer func in local memory
- Inner function definition is stored in local memory
- On return, bring back inner function and surrounding memory (backpack)
- “Closed over variable environment”
- Only available by running inner function

Demo: memoize

Did you review stringify?

Rest Parameters

“Zips” arguments into an array



Spread Operator

“Unzips” arrays

clone vs cloneDeep

What is the difference between a shallow and deep copy?

Pass by Reference vs Value

Primitive data types are **passed by value**.

`let num = 3`

`num` refers to the number 3.

Composite data types are **passed by reference**.

`const obj = { num: 3 }`

`obj` refers to the memory address at which the object is stored.

Shallow vs Deep Clones

```
const users = [{ 'user': 'barney' },{ 'user': 'fred' }]
```

Shallow clone:

- We only copy the outermost object.
- Any objects within it are references to the original.

Deep clone:

- We copy the outermost object, as well as *all* objects nested within it.
- There is no limit to how deeply objects can be nested. Our function must take this into account.

Shallow vs Deep Clones

```
const users = [{ 'user': 'barney' },{ 'user': 'fred' }]
```

clone

```
 * Creates a shallow clone.
 * let shallowClone = clone(users);
 * shallowClone === users -> false
 * shallowClone[0] === users[0] → true
```

cloneDeep

```
 * Creates a deep clone.
 * let deepClone = cloneDeep(users)
 * deepClone === users → false
 * deepClone[0] === users[0] → false
 * deepClone[0].user === users[0].user → true
```

Demo: clone & cloneDeep

throttle overview

```
/**
 * Returns a function that only invokes func once per every wait milliseconds
 * (additional calls to func within the wait should not be invoked or queued).
 */
```

Why do we need throttling?

Source: https://levelup.gitconnected.com/throttle-in-javascript-improve-your-applications-performance-984a4e020a3f

throttle overview

Throttle is similar to a spring that throws a ball

- After the ball flies, the spring needs time to shrink back
- Cannot throw any more balls until it’s ready again

Demo: throttle

Why do we use throttle?

- Used to optimize applications by limiting the amount of times functions can be called per a set time

Summary

Today we talked about...

- memoize
  - Closure
  - Rest parameters
  - Spread operator
- cloneDeep
  - Shallow copy / Deep copy
  - Pass by reference vs value
- throttle
  - More closure
  - Date.now
  - setTimeout()