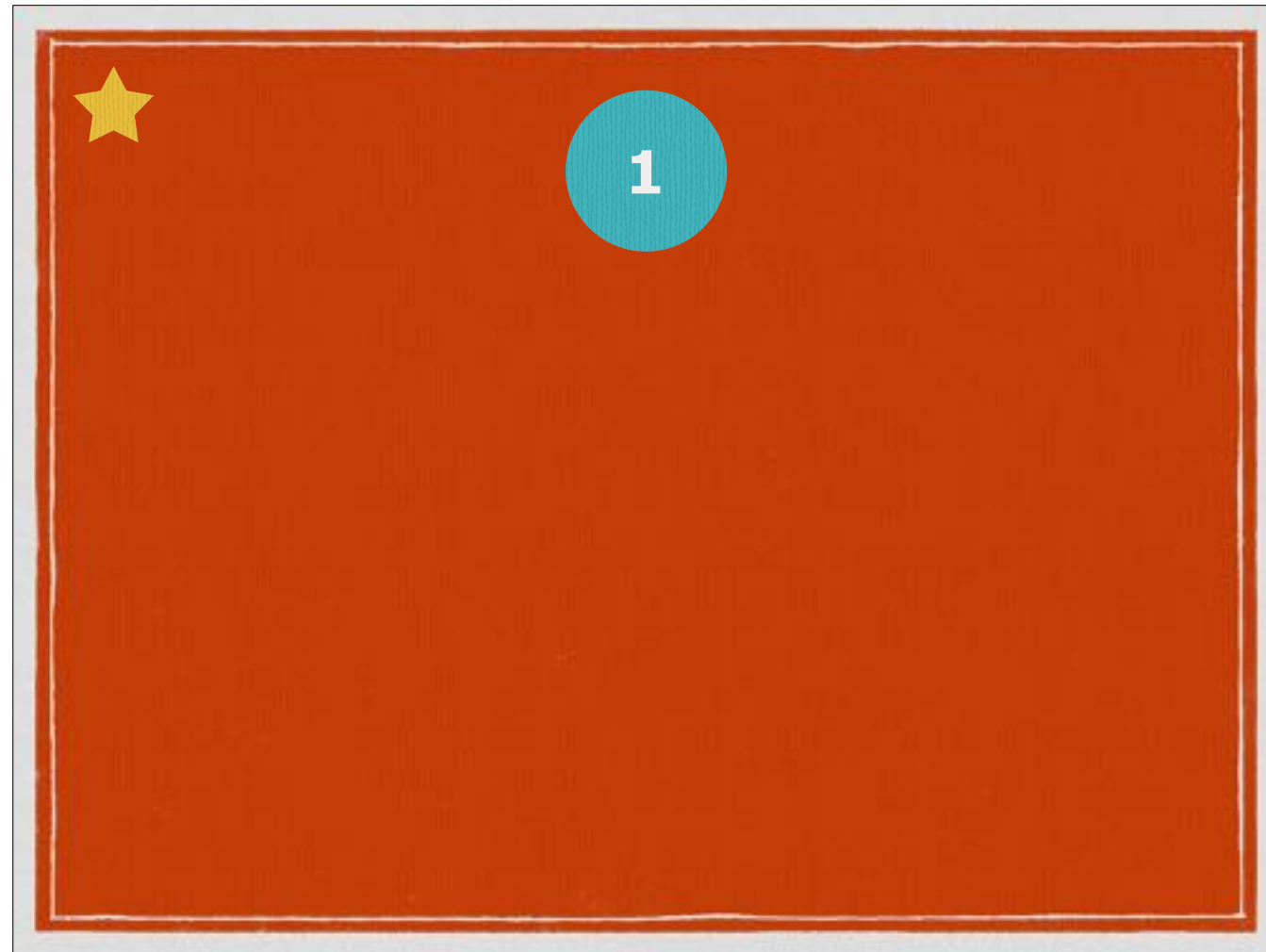


inter*feud*!

the game that gets YOU hired

A graphic featuring a teal-colored five-pointed star centered on a solid orange background. The word "idiosyncronous" is written across the middle of the star in a white, bold, sans-serif font. The letter "y" is highlighted in a yellowish-orange color. The entire composition is enclosed within a thin white border.

idiosyncronous





1

**How can you *iterate* through
an object's *key-value* pairs?**





...what is *another way* to iterate
through key-value pairs?



...what is *another way* to iterate
through key-value pairs?

```
Object.keys(x)  
for (var k in x) {x.hasOwnProperty()}
```



2



2

What is **strict mode? Name
at least 3 things it does.**



2

What is **strict mode**? Name at least 3 things it does.

- forbids global assignment
- forbids improper writes/deletes
- forbids non-unique keys OR params
- forbids octal syntax
- forbids with
- eval can't modify surrounding scope
- this defaults to undefined
- forbids fn.caller, fn.callee
- ...a few other things



3



3

**How do `==` and `===`
differ?**



3

How do `==` and `===` differ?

`==` compares only value so if the types are different, it'd still be true

`===` compares both value and type



4



4

How does prototypal inheritance work?



4

How does **prototypal inheritance** work?

A prototype for an object is simply an existing object. Any instance will have some implicit reference to the properties on its prototype. In JavaScript this implicit reference is via `__proto__`.



5



5

**How does `new Thing()`
differ from `Thing()`?**



5

How does **new Thing()** differ from **Thing()**?

`new` will make a new empty object, assign its `__proto__` to `Thing.prototype`, bind it to `this`, and then invoke the constructor. Also, it will return `this` unless the constructor returns some other non-primitive value.



6



6

**What are JavaScript's
six primitive types?**



6

What are JavaScript's **six primitive types?**

boolean, number, string, null, undefined,
and symbol (ES6)



7



7

**What are JavaScript's
six “falsy” values?**



7

What are JavaScript's **six “falsy” values?**

false, null, undefined, NaN, 0, “”



8



8

What is the difference between unary, binary, and ternary operators? Give an example of each.



8

What is the difference between unary, binary, and ternary operators? Give an example of each.

- Unary: +, -, typeof, new, instanceof
- Binary: +, -, /, *, %
- Ternary: ?:





9

![] == []
Why?



9

![] == []

Why?

Boolean coercion algorithm:

`[] == false` // convert false to Number

`[] == 0` // convert [] to primitive

`"" == 0` // convert "" to Number

`0 == 0` // true

Short answer: `[]`.toString() is empty string which is falsy.



10



10

**Describe pass-by-reference
versus pass-by-value.**



10

Describe **pass-by-reference versus **pass-by-value**.**

Objects get passed by reference, primitives get passed by value. Pass-by-value means that there is a copy of the entity; pass-by-reference means that there is no copy of the entity itself, only a copy of a reference to it.





11

**What's the difference between
host objects and native objects?**



11

**What's the difference between
host objects and native objects?**

Native objects are defined by an
ECMAScript implementation. Host
objects are defined by the environment.
E.g. Math is native, document is host.



12



12

typeof null evaluates
to what?



12

typeof null evaluates to what?

‘object’

It is a primitive of the type 'null' (the only one with this type). `typeof null === 'object'` is a known bug in JS which for which a fix was proposed, but rejected (because it would break old websites).





What is `typeof NaN`?



What is `typeof NaN`?

`'number'`



13



13

When would you use **promises
as opposed to vanilla **callbacks**?
When the other way around?**



13

When would you use **promises
as opposed to vanilla **callbacks**?
When the other way around?**

(open to interpretation)



14



14

**Explain/describe
event delegation.**



14

Explain/describe event delegation.

In event delegation, an event is registered with a parent or manager, which itself will supervise the emitting/handling or its child/managee.



15



15

**What is the difference between
the `.slice` and `.splice` Array
methods?**



15

What is the difference between the `.slice` and `.splice` Array methods?

`.slice` will return a copy of some subsection of the array; `.splice` will mutate the original array in addition to returning the subsection





16

What is middleware?



16

What is **middleware**?

A chain in a link of possibly asynchronous tasks. In express, middleware are functions registered to run upon certain requests.



17



17

**What does ES6
introduce?**



17

What does **ES6** introduce?

Arrows, classes, template strings, destructuring, param defaults, rest, spread, let, const, for...of, generators, modules, sets, proxies, symbols, promises, reflection, tail calls, various methods





**What did ES5
introduce?**



What did **ES5** introduce?

- strict mode
- getters/setters
- legal trailing commas
- Object.create, .getPrototypeOf, .defineProperty, .getOwnPropertyDescriptor, .keys, .getOwnPropertyNames, .preventExtensions, .isExtensible, .seal, .isSealed, .isFrozen, .freeze
- Function.prototype.bind
- Array.prototype.every, .filter, .forEach, .indexOf, .lastIndexOf, .map, .reduce, .some; Array.isArray
- JSON.parse, JSON.stringify





18

**What does it mean that
JavaScript is “asynchronous”?**



18

What does it mean that JavaScript is “asynchronous”?

Commands do not necessarily execute in the order in which they are written. This is possible through higher-order functions, and in particular callbacks.





19

**What are the differences
between **AMD** and **CommonJS**?**



19

**What are the differences
between **AMD** and **CommonJS**?**

CommonJS: require and exports
AMD: define and return



- Concurrency is handling with many tasks simultaneously. Concurrency can be non-blocking single-threaded (like Javascript), blocking multi-threaded, or non-blocking multi-threaded. In Javascript, concurrency is made possible through the use of the event loop, host/environment APIs, and the callback queue

20

What is **concurrency** and how does JavaScript support it?



- Concurrency is handling with many tasks simultaneously. Concurrency can be non-blocking single-threaded (like Javascript), blocking multi-threaded, or non-blocking multi-threaded. In Javascript, concurrency is made possible through the use of the event loop, host/environment APIs, and the callback queue

What is **concurrency** and how does JavaScript support it?

Event loop

Environment/host APIs

Callback queue

Many tasks at once

Non-blocking single-threaded

(Blocking multi-threaded)

(Non-blocking multi-threaded)

NOT blocking single-threaded



- Concurrency is handling with many tasks simultaneously. Concurrency can be non-blocking single-threaded (like Javascript), blocking multi-threaded, or non-blocking multi-threaded. In Javascript, concurrency is made possible through the use of the event loop, host/environment APIs, and the callback queue