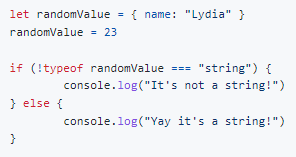
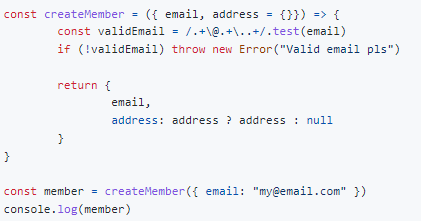
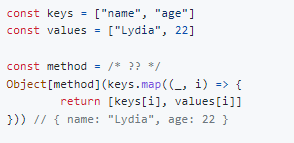
1. 

Yay it's a string!

1. 

**o/p : {email: 'my@email.com', address: {}}**

The default value of address is an empty object {}. When we set the variable member equal to the object returned by the createMember function, we didn't pass a value for address, which means that the value of address is the default empty object {}. An empty object is a truthy value, which means that the condition of the address ? address : null conditional returns true. The value of address is the empty object {}.

1. **What should the value of method be to log { name: "Lydia", age: 22 }?**

Ans: fromEntries

The fromEntries method turns a 2d array into an object. The first element in each subarray will be the key, and the second element in each subarray will be the value. In this case, we’re mapping over the keys array, which returns an array which first element is the item on the key array on the current index, and the second element is the item of the values array on the current index.

This creates an array of subarrays containing the correct keys and values, which results in { name: "Lydia", age: 22 }

1. 

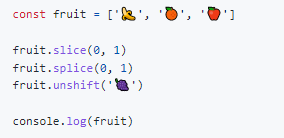
Ans: Third

The Promise.all method runs the passed promises in parallel. If one promise fails, the Promise.all method rejects with the value of the rejected promise. In this case, promise3 rejected with the value "Third". We’re catching the rejected value in the chained catch method on the runPromises invocation to catch any errors within the runPromises function. Only "Third" gets logged, since promise3 rejected with this value.

1. 

Ans: [my@email.com](mailto:my@email.com)

The updateEmail function is an arrow function, and is not bound to the user object. This means that the this keyword is not referring to the user object, but refers to the global scope in this case. The value of email within the user object does not get updated. When logging the value of user.email, the original value of my@email.com gets returned.

1. 

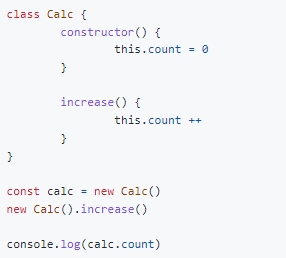
 Ans: ['🍇', '🍊', '🍎']

First, we invoke the slice method on the fruit array. The slice method does not modify the original array, but returns the value that it sliced off the array: the banana emoji. Then, we invoke the splice method on the fruit array. The splice method does modify the original array, which means that the fruit array now consists of ['🍊', '🍎']. At last, we invoke the unshift method on the fruit array, which modifies the original array by adding the provided value, ‘🍇’ in this case, as the first element in the array. The fruit array now consists of ['🍇', '🍊', '🍎'].

1. 

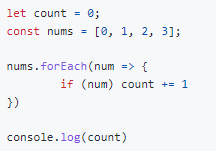
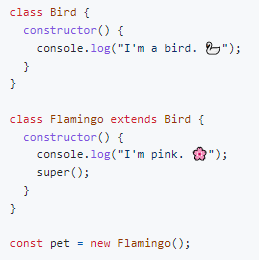
Ans : true

The updateUser function updates the values of the email and password properties on user, if their values are passed to the function, after which the function returns the user object. The returned value of the updateUser function is the user object, which means that the value of updatedUser is a reference to the same user object that user points to. updatedUser === user equals true.

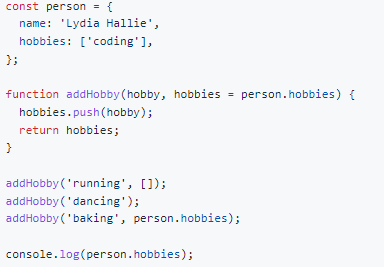
1. 

Ans: 0

We set the variable calc equal to a new instance of the Calc class. Then, we instantiate a new instance of Calc, and invoke the increase method on this instance. Since the count property is within the constructor of the Calc class, the count property is not shared on the prototype of Calc. This means that the value of count has not been updated for the instance calc points to, count is still 0.

1. A)  

Ans: 3

1. 

["coding", "dancing", "baking"]

The addHobby function receives two arguments, hobby and hobbies with the default value of the hobbies array on the person object.

First, we invoke the addHobby function, and pass "running" as the value for hobby and an empty array as the value for hobbies. Since we pass an empty array as the value for hobbies, "running" gets added to this empty array.

Then, we invoke the addHobby function, and pass "dancing" as the value for hobby. We didn't pass a value for hobbies, so it gets the default value, the hobbies property on the person object. We push the hobby dancing to the person.hobbies array.

Last, we invoke the addHobby function, and pass "baking" as the value for hobby, and the person.hobbies array as the value for hobbies. We push the hobby baking to the person.hobbies array.

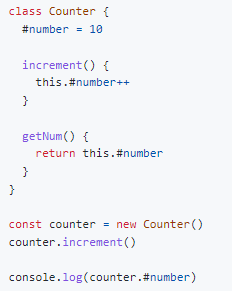
After pushing dancing and baking, the value of person.hobbies is ["coding", "dancing", "baking"]

1. 

yield\* getMembers(teams[i].members)

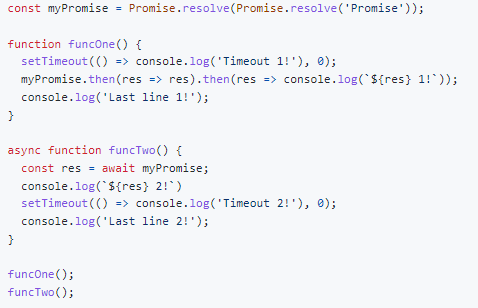
In order to iterate over the members in each element in the teams array, we need to pass teams[i].members to the getMembers generator function. The generator function returns a generator object. In order to iterate over each element in this generator object, we need to use yield\*.

If we would've written yield, return yield, or return, the entire generator function would've gotten returned the first time we called the next method.

1. 

op: SyntaxError

In ES2020, we can add private variables in classes by using the #. We cannot access these variables outside of the class. When we try to log counter.#number, a SyntaxError gets thrown: we cannot acccess it outside the Counter class!

1. 

Last line 1! Promise 2! Last line 2! Promise 1! Timeout 1! Timeout 2!

First, we invoke funcOne. On the first line of funcOne, we call the asynchronous setTimeout function, from which the callback is sent to the Web API. (see my article on the event loop [here](https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif).)

Then we call the myPromise promise, which is an asynchronous operation.

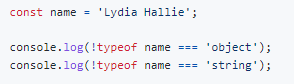
Both the promise and the timeout are asynchronous operations, the function keeps on running while it's busy completing the promise and handling the setTimeout callback. This means that Last line 1! gets logged first, since this is not an asynchonous operation.

Since the callstack is not empty yet, the setTimeout function and promise in funcOne cannot get added to the callstack yet.

In funcTwo, the variable res gets Promise because Promise.resolve(Promise.resolve('Promise')) is equivalent to Promise.resolve('Promise') since resolving a promise just resolves it's value. The await in this line stops the execution of the function until it receives the resolution of the promise and then keeps on running synchronously until completion, so Promise 2! and then Last line 2! are logged and the setTimeout is sent to the Web API.

Then the call stack is empty. Promises are microtasks so they are resolved first when the call stack is empty so Promise 1! gets to be logged.

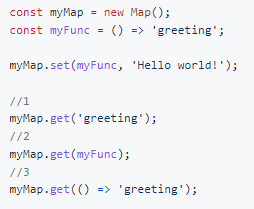
Now, since funcTwo popped off the call stack, the call stack is empty. The callbacks waiting in the queue (() => console.log("Timeout 1!") from funcOne, and () => console.log("Timeout 2!") from funcTwo) get added to the call stack one by one. The first callback logs Timeout 1!, and gets popped off the stack. Then, the second callback logs Timeout 2!, and gets popped off the stack.

1. 

Ans : false false

typeof name returns "string". The string "string" is a truthy value, so !typeof name returns the boolean value false. false === "object" and false === "string" both returnfalse.

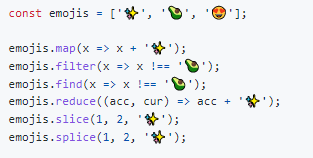
(If we wanted to check whether the type was (un)equal to a certain type, we should've written !== instead of !typeof)



Ans : 2

When adding a key/value pair using the set method, the key will be the value of the first argument passed to the set function, and the value will be the second argument passed to the set function. The key is the function () => 'greeting' in this case, and the value 'Hello world'. myMap is now { () => 'greeting' => 'Hello world!' }.

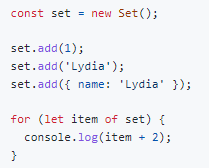
1 is wrong, since the key is not 'greeting' but () => 'greeting'. 3 is wrong, since we're creating a new function by passing it as a parameter to the get method. Object interact by reference. Functions are objects, which is why two functions are never strictly equal, even if they are identical: they have a reference to a different spot in memory.

1. 

Ans: splice

With splice method, we modify the original array by deleting, replacing or adding elements. In this case, we removed 2 items from index 1 (we removed '🥑' and '😍') and added the ✨ emoji instead.

map, filter and slice return a new array, find returns an element, and reduce returns a reduced value.

1. 

Ans: 3,Lydia,[object Object]2

The + operator is not only used for adding numerical values, but we can also use it to concatenate strings. Whenever the JavaScript engine sees that one or more values are not a number, it coerces the number into a string.

The first one is 1, which is a numerical value. 1 + 2 returns the number 3.

However, the second one is a string "Lydia". "Lydia" is a string and 2 is a number: 2 gets coerced into a string. "Lydia" and "2" get concatenated, which results in the string "Lydia2".

{ name: "Lydia" } is an object. Neither a number nor an object is a string, so it stringifies both. Whenever we stringify a regular object, it becomes "[object Object]". "[object Object]" concatenated with "2" becomes "[object Object]2".

17.

function getItems(fruitList, ...args, favoriteFruit) {

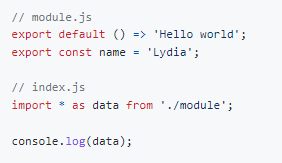
return [...fruitList, ...args, favoriteFruit]

}

getItems(["banana", "apple"], "pear", "orange")

ans : SyntaxError

...args is a rest parameter. The rest parameter's value is an array containing all remaining arguments, **and can only be the last parameter**! In this example, the rest parameter was the second parameter. This is not possible, and will throw a syntax error.

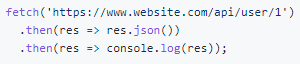
1. 

Ans : { default: function default(), name: "Lydia" }

With the import \* as name syntax, we import all exports from the module.js file into the index.js file as a new object called data is created. In the module.js file, there are two exports: the default export, and a named export. The default export is a function which returns the string "Hello World", and the named export is a variable called name which has the value of the string "Lydia".

The data object has a default property for the default export, other properties have the names of the named exports and their corresponding values.

1. What kind of information is logged?



1. 