PayPal Pre-Work Notes

JavaScript

Introduction

JavaScript(JS) is a programming language primarily used to create interactive effects within web browsers. It is used for adding **dynamic** behavior to web pages, building web applications, and server-side programming. JavaScript code runs in web browsers (client-side) and on servers (server-side) using environments like **Node.js**.

Node is a C++ program which includes Google's **v8** Javascript engine. Due to this, we can run JS code outside of a browser. This allowed possible development of the backend in JS. Node and Browser provide a runtime environment for JS code.

ECMAscript is a specification and JS is a programming language that confirms this specification. **ECMA** releases specifications which define many new features for JavaScript.

You can write JS code in Browser developer tools in the console. All statements in Javascript are terminated by **semicolon(;)**.

JS Command	Action
console.log("Message")	This prints the message in the console
alert("Message")	This gives the alert with the message

JavaScript in Node

JS Command	Action
node index.js	Node runs the javascript file.

Variables in Javascript

We follow camelCase Notation in Javascript. Also variables are case sensitive. By default, the variables are undefined.

JS Command	Action
var name = "String";	Declares and initializes a variable
let name = "String";	Declares and initializes a variable.(There are some problems with var keyword)
const name = "String"	Declares a constant and initialize it
typeof name	Prints the type of the variable in the console

You can assign string, bool, number, undefined and null values as **primitive types** to variables. Also there are **reference types** as well.

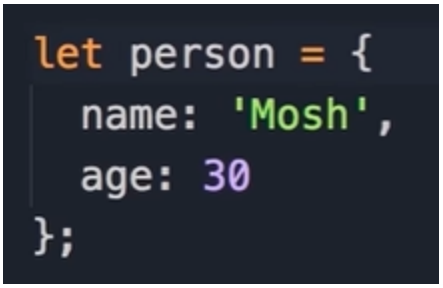
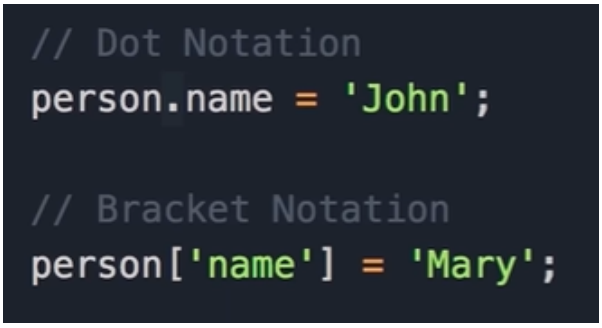
Dynamic Typing In JS

avaScript is a **dynamically typed** language, meaning variable types are determined at runtime and can change as the program executes.

Object in JS

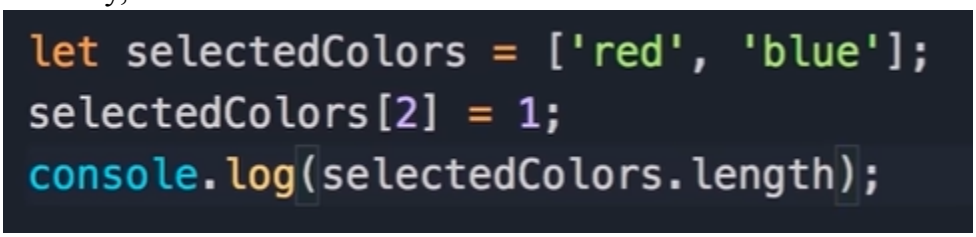
In JavaScript, an **object** is a collection of key-value pairs, where the keys are strings (or Symbols) and the values can be any data type, including other objects. It is like a struct in C++.

JS Command	Action

	Declared an object person.
	Accessing the information encompassed by an object.

Arrays in JS

In JavaScript, **arrays** are ordered collections of elements, where each element is indexed and can be accessed by its position (index) in the array. Arrays can hold elements of any data type, including other arrays. Array is also an object. And you can use many properties like key,find etc.



Functions in JS

In JavaScript, **functions** are blocks of code designed to perform a particular task, and they can be defined using the function keyword, arrow syntax, or as methods within objects.

```
function greet() {

}
```

Higher Order Functions in JS

Higher-order functions in JavaScript are functions that either take other functions as arguments or return a function as their result. They make code look much cleaner.

For example: We have a data for companies as follows:

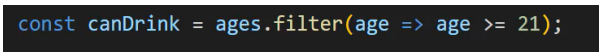
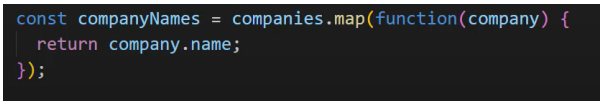
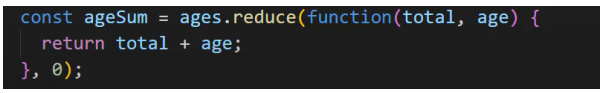
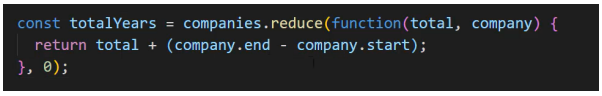
```
const companies= [
  {name: "Company One", category: "Finance", start: 1981, end: 2003},
  {name: "Company Two", category: "Retail", start: 1992, end: 2008},
  {name: "Company Three", category: "Auto", start: 1999, end: 2007},
  {name: "Company Four", category: "Retail", start: 1989, end: 2010},
  {name: "Company Five", category: "Technology", start: 2009, end: 2014},
  {name: "Company Six", category: "Finance", start: 1987, end: 2010},
  {name: "Company Seven", category: "Auto", start: 1986, end: 1996},
  {name: "Company Eight", category: "Technology", start: 2011, end: 2016},
  {name: "Company Nine", category: "Retail", start: 1981, end: 1989}
];

const ages = [33, 12, 20, 16, 5, 54, 21, 44, 61, 13, 15, 45, 25, 64, 32];
```

Note: When callback functions in higher-order functions are synchronous, it means that the callback is executed immediately and completely before the higher-order function continues executing the next line of code.

You can use the **(=>)** to shorten the length of the callback function and make code look much cleaner.

JS Command	Action

`for(let i = 0; i < companies.length; i++) {`
`  console.log(companies[i]);`
`}`
	Normal for loop

	Higher order function **forEach**(It takes a call back function which is synchronous. It can have parameters iterator, index and array)(Look into documentation)
 	Higher order function **filter**(Used to filter objects based on condition)
  	Higher order function **map**(Used to create new object array out of older ones based on condition or values)
 	Higher order function **sort**(Used to sort based on the comparator)
  	Higher order function **reduce**(Used to iterate over an array and accumulate a single value based on a reducer function)(takes accumulator and current element to produce result)

Async JS

Async JavaScript refers to techniques and patterns used to handle **asynchronous** operations, allowing code to execute without blocking the main thread. This includes using callbacks, promises, and the **async/await** syntax to manage operations like network requests, file I/O, and timers.
The following data is an example on which we will be working:

```
const posts = [
  { title: 'Post One', body: 'This is post one' },
  { title: 'Post Two', body: 'This is post two' }
];
```

Note: In JavaScript, a **Promise** represents an asynchronous operation that can complete with a value (fulfilled) or fail with an error (rejected). It allows you to chain asynchronous operations and handle their outcomes using **.then()** for success and **.catch()** for errors.

JS Command	Action
	The following function uses the setTimeout function which actually executes the function after some time in ms.
	The following function uses **callback** to push the post and display it after the data is added

```
function createPost(post) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      posts.push(post);

      const error = true;

      if (!error) {
        resolve();
      } else {
        reject('Error: Something went wrong');
      }
    }, 2000);
  });
}

createPost({ title: 'Post Three', body: 'This is post three' })
  .then(getPosts)
  .catch(err => console.log(err));
```

The same functionality can be achieved by **promises** (It uses **resolve/reject**, when the problem is resolved, or error, when something goes wrong. Then we use **then/catch** to respond to resolve/reject)

```
// Promise.all
const promise1 = Promise.resolve('Hello World');
const promise2 = 10;
const promise3 = new Promise((resolve, reject) =>
  setTimeout(resolve, 2000, 'Goodbye')
);
const promise4 = fetch
('https://jsonplaceholder.typicode.com/users').then(res =>
  res.json()
);

Promise.all([promise1, promise2, promise3, promise4])
.then(values =>
  console.log(values)
);
```

Using **Promise.all** to create an array of promise and calling functions on it. You need to call .then two times on api response as we need to map it to json as well.

```
async function init() {
  await createPost({ title: 'Post Three', body: 'This is post three' });

  getPosts();
}

init();
```

Using async/await.

The `async/await` syntax in JavaScript provides a way to write asynchronous code that looks and behaves like synchronous code, making it easier to read and maintain.

- `async`: Declares an asynchronous function that returns a Promise.
- `await`: Pauses the execution of an `async` function until the Promise is resolved or rejected.

```
async function fetchUsers() {
  const res = await fetch
  ('https://jsonplaceholder.typicode.com/users');

  const data = await res.json();

  console.log(data);
}

fetchUsers();
```

This uses the fetch function which is a promise as well. **Async/await** is a cleaner way to deal with promises rather than **.then/.catch** or **callbacks.**

Resources

Javascript	JavaScript Basics
	JavaScript Higher-Order Functions
	JavaScript Asynchronous Programming