

Linking JS file

Inline

```
<head><script>const role="developer"
</script></head>
```

Linking

```
<body> <script src="script.js"></script>
</body>
```

Primitive Data Types

string const role="dev"

number const years=10

boolean const isTrue=true

undefined let children;

null let empty=null

symbol unique value

bigint large integers that number type can't hold

Truthy and Falsy

falsy values 0, "", undefined, null, NaN

truthy values everything else

let vs. const vs. var

let - can be reassigned
 - block scoped

const - cannot be reassigned
 - block scoped
 - best practice

var - hoisted(can be referenced before being declared)
 - can be redeclared
 - do not use in modern JS

Conditionals

```
// If else statement
if(age >= 18) {
  status = " adult"
} else {
  status = " child"
}
```

Conditionals (cont)

```
// Switch statement
switch (day) {
  case " mon day ": // if(day -
=== " mon day ")
    con sol e.l og( "It's
Monday ")
    break; // exits out of the
switch statement
  case " sun day ":
  case " sat urd ay": // if(day -
=== " sat urd ay" || day === " -
sun day ")
    con sol e.l og( "It's the
weeken d")
    break;
  def ault: // else ...
    con sol e.l og( "Not a
valid day")
}
// Ternary Operator
const drink = age >= 18 ? " -
bee r" : " bubble tea"
```

Basic Operators

add/subtract a + b - c

multiply/divide a * b / 100

power: 2x2x2 2 ** 3

remainder: 13/5 13 % 5

remainder is 3

postfix increment/decre- a++, a--
ment: returns value
before increment

prefix increment/decre- ++a, --a
ment: returns value after
increment

find type of variable typeof a

assignment a = 'string'

a = a + b a+=b (*, /, -=)

less and greater than a < b, a > b

less or equal, greater or a <= b, a >= b
equal

logical and, or a && b, a || b

logical not !(a === b)

Basic Operators (cont)

strict equal, not equal: no a === b, a !== b
type coercion

loose equal, not equal: a == b, a != b
type coercion (do not
use)

Type Conversion and Coercion

convert to number Number("20-
00")

convert to string String(2000)

convert to boolean Boolean("hell-
o")

coerced to string: "a is 16" "a is" + 16

coerced to number: 80 "100" - 10 - "-
10"

coerced to boolean: if (hasMoney)
if(true) {} {}

Objects

definition - a data structure where
properties are stored in
key-value pairs
- order does not matter
ex. const student =
{ firstName: " -
She ldo n",
lastNa me= " Coo -
per ", age: 10,
fullName: () =>
this.f irs tName +
' ' + this.l -
astName }

getting the studen t.f irs -
value tName or
studen t["f irs -
tNa me"]

setting the studen t.i sSm -
value art =true

object methods a function attached to an
object

calling a studen t.f ull -
method Name ()



Iteration

for Loop

```
// loops forward
for (let i = 1; i <= 10; i++) {
  console.log(i)
}
```

```
// loops backward
```

```
for (let i = 10; i > 0; i--) {
  console.log(i)
}
```

break statement: completely terminates the loop

```
for (let i = 1; i <= 10; i++) {
  if(i === 3) break;
  console.log(i); // 1, 2
}
```

continue statement: terminates the current iteration

```
for (let i = 1; i <= 10; i++) {
  if(i === 3) continue;
  console.log(i); // 1, 2, 4, 5, 6, 7, 8, 9, 10
}
```

while loop

```
let i = 1; // initialize counter
while(i <= 10) { // condition
  console.log(i);
  i++; // update counter
}
```

Functions

purpose

- a block of code that performs certain tasks
- useful for repeated tasks
- either does something or returns value(s)

parameters: variable(s) functions accept

```
function fullName( first, last, ...params) {}
```

arguments: values passed in when function is invoked

```
fullName( " Sheldon", " Cooper ", ...arguments)
```

Functions (cont)

function declaration

- define function with a function name
 - can be invoked before they are defined
- ```
ex. function foo(bar) {}
```

#### function expression

- anonymous function stored in a variable
  - cannot be invoked before they are defined
- ```
ex. const foo = function (bar) {}
```

arrow functions

- has no this keyword
- ```
ex. const foo = bar => bar + a
```

#### default parameters

- allow params to be initialized with default values if no value/undefined is passed
- ```
ex. function foo (bar = 0) {}
```

primitive type argument

- function makes a copy of the original value

reference type argument

- functions makes a copy of the reference (the copy is still a value)
- JS, arguments can only be passed by value

first-class function

- a concept in programming where functions are treated as first-class citizens meaning they can behave like variables

higher-order function

- a function that can be passed as an argument to other functions, or one that can be returned by another function or do both
- ```
ex. bar.addEventListener('click', foo)
ex2. const foo = () => () => console.log('hello')
```

### Functions (cont)

#### .call( object, arg1, arg2, ...)

- calls the function with a given this value and arguments provided individually
  - remember this is undefined on regular function call
- ```
ex. foo(num, str) {return this.name + str}
foo(12, " hello") // 'this' keyword is undefined
foo.call( bar Object, 12, " hello")
```

.apply(object, [arg1, arg2, ...])

- calls the function with a given this value, and arguments provided as an array
- ```
ex. const arguments = [12, " hello"]
foo.apply(bar Object, arguments)
foo.call(bar Object, ...arguments)
```

#### .bind( object, arg?)

- returns a new function with this value bound to the provided object
  - arguments can be passed optionally to preset arguments
- ```
ex. const newFoo = foo.bind( bar Object)
```

IIFE

- immediately invoked function expression
 - useful if the function is only to be used once
- ```
ex. (function() {console.log('hello')}())
```

#### closure

- a function that remembers all the variables that existed at the function's birth place

