MODERN JAVASCRIPT

JAVASCRIPT

STRINGS

```
let email = 'mario@thenetninja.co.uk';
console.log(email[5]);                                    @


console.log(email.length);                                23


console.log(email.toUpperCase());                         MARIO@THENETNINJA.CO.UK
console.log(email.toLowerCase());                         mario@thenetninja.co.uk


console.log(email.lastIndexOf('n'))                       14


// SLICE
console.log(email.slice(0, 10));                          mario@then


// Substr (start from, get n characters)
Console.log(email.substr(4, 10))                          o@thenetni


// Replace (replace the first, with character)
Console.log(email.replace('n', 'w'))                      mario@thenetninja.co.uk
```

.push & .pop : is a destructive method, meaning it is a method that changes the original value of a data

ARRAY

let ninja = ['shaun', 'ryu', 'chun-li'];

//Methods

Console.log(ninja.join('-')); shaun-ryu-chun-li

//indexOf

console.log(ninja.indexOf('chun-li')); 2

//concat

console.log(ninja.concat(['ken', 'crystal'])); ['shaun', 'ryu', 'chun-li', 'ken', 'crystal']

//push

console.log(ninja.push('ken')); 4

console.log(ninja); ['shaun', 'ryu', 'chun-li', 'ken']

//pop

console.log(ninja.pop()) Ken

console.log(ninja) ['shaun', 'ryu', 'chun-li']

Difference Between == and ===	
==	===
In this Comparison, Javascript does a type Conversion behind the scene. Therefore 25 == "25" = true Implicit Type Conversion	This Comparison is strict and in Javascript doesn't do a behind the scene type Conversion 25 === "25" = false

TYPE CONVERSION

* NUMBERS

```
let score = '100';

score = Number( score );
console.log( score + 1 );                    101
console.log( typeof score );                 Number
```

STRINGS

```
let age = 23;

age = String( age );
console.log ( age + 1 );                     '231'
console.log ( typeof age );                  String
```

BOOLEAN

```
console.log ( Boolean(100) )                 true
console.log ( Boolean(0) )                   false
console.log ( Boolean('0') )                 true
console.log ( Boolean('') )                  false
```

Explicit Type Conversion

CHAPTER 3

LOOPS & CONDITIONALS

```
// for loops

for (let i=0; i<5; i++){
  Console.log('in loop:', i);
}
console.log('loop finished!');
```

CONSOLE

```
0
1
2
3
4
loop finished!
```

```
// while loops
let i = 0;

while (i<5){
  Console.log('in loop:',i);
  i++;
}
```

CONSOLE

```
0
1
2
3
4
```

```
// do while loops (Executes atleast Once)
let i=5;

do {
  Console.log(i);
  i++;
} while (i<5);
```

CONSOLE

```
5
```

// If Statement

```
Const age = 17;

if (age < 14) {
  console.log("You are a kid");
} else if (age >= 14 && age < 18){
  Console.log("You are a teenager");
} else {
  Console.log("You are an Adult Sir");
}
```

CONSOLE

You are a teenager

// break and Continue

```
Const scores = [50, 25, 0, 30, 100, 20, 10];

for (let i = 0; i< scores.length; i++){
  if (scores[i] === 0){
    continue;
  }
  Console.log('Your score:', scores[i]);

  if (Scores[i] === 100){
    console.log('Congrats, you got the top score!');
    break;
  }
}
```

CONSOLE

Your score: 50
Your score: 25
Your score: 30
Your score: 100
Congrats, you got the top score!

// switch statements

Note: switch statements use strict equality "===".

```
Const grade = 'B';

switch (grade){
  Case 'A':
    console.log('you got an A!');
    break;
  case 'B';
    console.log('you got a B!');
    breakr
  case 'C';
    console.log('you got a C!');
    break;
  Case 'D';
    Console.log('you got a D!');
    break;
  default:
    console.log('not a valid grade');
}
```

CONSOLE

you got a B!

// Variables & block scope

```
let age = 30;
```

```
let age = 30;
```

✗ Fails: This won't work. We can't re-declare a global scope variable in a global scope.

```
let age = 30;

if(true){
  let age = 50;
}
```

✓ WORKS: This won't fail. We can re-declare a global scope variable in a local scope (inside our block scope)

Chapter 4

Functions & Methods

```
// function declaration
function greet(){
    console.log('hello there');
}

// function expression
const speak = function(){
    console.log('good day!');
};

// Calling functions
greet();
speak();
```

CONSOLE

```
hello there
good day!
```

Difference Between Function Declaration & Function Expression

Function Declaration	Function Expression
-This can be declared anywhere in our javascript and can also be called anywhere. Because when our javascript first runs they are hoisted at the top of the document	This can be declared anywhere but can't be called anywhere since they follow the flow of the document

// arguments & parameters

```
const speak = function (name = 'luigi', time = 'night') {
    console.log(`good ${time} ${name}`);
};

speak();
speak('shaun');
speak('shaun', 'morning');
```

default default

CONSOLE

good night luigi
good night shaun
good morning shaun

// returning functions

```
const area = function (radius) {
    return 3.14 * radius ** 2;
}

area(4);
```

CONSOLE

50.24

// returning in Arrow function =>

```
const area = (radius) => 3.14 * radius ** 2;

area(4)
```

CONSOLE

50.24

// method

methods are functions that are called using the . notation

e.g

```
Const name = 'shaun';
```

CONSOLE

```
let result = name.toUpperCase();
console.log(result)
```

SHAUN

// callbacks & forEach

What is a callback function?

A callback function is a function passed into another function as an argument.

e.g

```
const myFunc = (callbackFunc) => {
    let value = 50;
    callbackFunc(value);
};

myFunct(value => {
    console.log(value);
});
```

CONSOLE

50

// forEach

This is a method that takes in a callback function

e.g

let people = ['mario', 'luigi', 'ryu', 'shaun', 'chun-li'];

```
const logPerson = (person, index) => {
  console.log(`${index}. ${person}`);
};
```

CONSOLE

0. mario
1. luigi
2. ryu
3. shaun
4. chun-li

people.forEach((person, index) =>

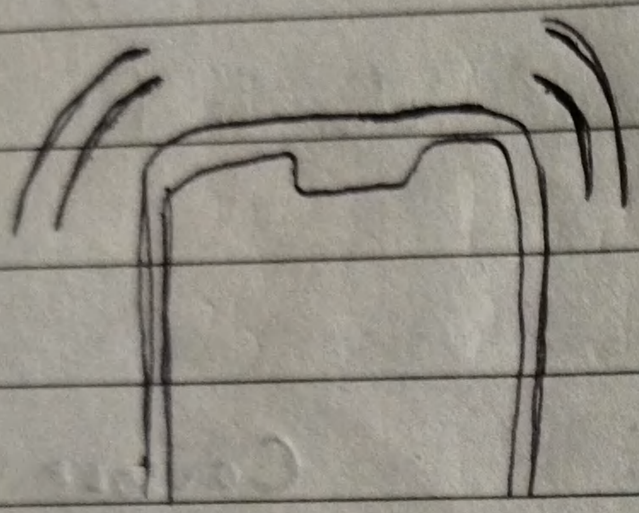
```
people.forEach(logPerson);
```

✦ OR we can write the callback function directly inside forEach Method - which is much preferable except when the call back function has several lines of code

Chapter 5

Objects

Objects in real life have properties & things they can do...

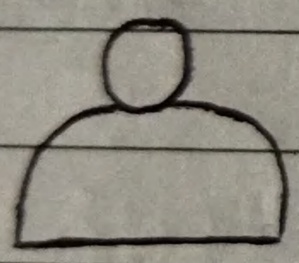


properties
- colour
- size
- model

things it can do (methods)
- ring
- bake a picture
- play music

Objects in Javascript have properties & things they can do (methods)...

user Object



properties
- email
- username
- gender

methods
- login
- logout

Javascript has some built in objects, such as a Day object, a maths Object and many more.

But more importantly Javascript also allows us to create our own object using what's known as "object literal notation".