Programs

- what is a process?
- what is a program?

“process.excalidraw” could not be found.

View Of Memory

value: a piece of information you can move around and operate on

will you deal with all chunks/values the same?

chunks have different types.

Languages are free to to interpret as it likes!

JavaScript Types

Number :- It is a type to represent numeric values (all numeric values) no distinction between floating point numbers and integers.

Memory View: 64 bit

Examples:

```
100001;

20.2;

2.998e8; //scientific notation;

infinity;

-infinity;
```

```
-0;

Nan;  // Invalid Number
```

Strings :- Represents any sequence of characters

- different between javascript and c++

Examples:

```
"Double Quotes"

'Signle Quotes'

`Back Ticks Or String Literals`
```

back-ticks are special

```
` hello kitty
  hello kitty
`;

` ${100/2} = 50 `;
```

Boolean: Represent the truth value of a proposition. It can hold two values

- Examples

```
true
false
```

Empty Values: Denote absence of value.

```
null // absense of value

undefined // not given a value yet
```

- note: it is advisable to write your code as if they were interchangeable

---

More

These are what we call primitive values. and they are some more introduced recently in the language

- Big Integer

```
0n
5000n
```

- Symbol

Non-primitive

- Object and its sub-types

> An object is a collection of related data and/or functionality. These usually consist of several variables and functions (which are called properties and methods when they are inside objects). Let's work through an example to understand what they look like.

sub-types: Arrays, Functions...

> function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a

Operators

- **Definition**: Operators are special functions that are syntactically written differently. These functions are used to perform different types of basic mathematical, logical and other types of computations.
- **Examples**:
- Arithmetic

```
3 + 5; // addition

3 - 5; // subtraction

3 * 5 // multiplication

3 / 5 // division

3 % 5 // modulus
```

- Comparison

```
3 == 3; // equal to

3 != 3; // not equal

3 > 2; // greater

2 < 3; // smaller

3 >= 3; // greater than or equal

3 <= 3; // smaller than or equal
```

- Logical

```
true && true;
false && false;
```

```
!true
```

- Concatenation operator

```
"shehab" + " khaled";
```

- Bit-wise operators
- Assignment operators

Expressions

Operators allow us to combine values in interesting ways like

- (2 * 3) + 4;
- "shehab" + "khaled" + "mohamed"

There are what we call expressions!

Definition: A valid unit of code that resolves to a value

Variables

Definition: reference to a location in memory which has a useful name.

Syntax:

```
var a; // define a variable named a
a = 5; // assign 5 to variable a

---

var a = 5; // define a variable named a and assign 5 to it
```

Is var `a = 5;` an expression?

Statements

Definition: an instruction that performs an action. Make up the actual steps in your code.

- Different between statements and expressions

Programs

So using expressions and statements you can write meaningful programs.

```
let x = 5 + (5 * 2);
x = 4; // assignment operator
console.log(x * 2); // print statement
```

Not Enough!

Control Flow Statements

If Statement

```
if (condition) {
  // code to be executed if condition is true
}
```

```
if (condition) {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}
```

```
if (condition1) {
  // code to be executed if condition1 is true
} else if (condition2) {
  // code to be executed if condition1 is false and condition2 is true
} else {
  // code to be executed if both conditions are false
}
```

Switch Statement

```
switch (expression) {
  case value1:
    // code to be executed if expression equals value1
    break;
  case value2:
    // code to be executed if expression equals value2
    break;
  default:
    // code to be executed if expression doesn't match any case
}
```

Looping Statements

- **for Loop**: Repeats a block of code a specified number of times.

```
for (initialization; condition; increment) {
  // code to be executed repeatedly
}
```

- **while Loop**: Repeats a block of code as long as a specified condition is true.

```
while (condition) {
  // code to be executed repeatedly
}
```

- **do...while Loop**: Like a while loop, but the code block is executed once before the condition is tested.

```
do {
  // code to be executed at least once
} while (condition);
```

Jump Statements

- **break Statement**: Exits a loop or a switch statement.
- **continue Statement**: Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.

```
break;
continue;
```

Functions

- function declaration/statement

```
console.log(greet("mohamed")); // interesting

function greet(name){
        return `hello ${name}`
}


console.log(greet("shehab"));
console.log(greet());
```

- Default parameters

```
function add(a, b = 5){
        return a + b;
```

```
}
```

Types Again

```
var x = 5;
var y;

if(x == 5){
        y = "shehab";
}else{
        y = 2;
}
```

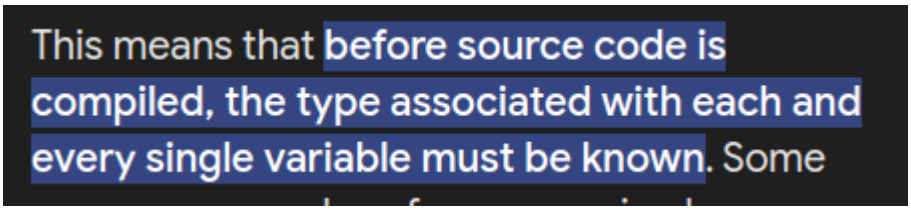
- How is this actually Exciting

C++ 😃

```
int x = 5;
string y = "hello";

x = "fofo"; // ?
y = 3 // ?
```

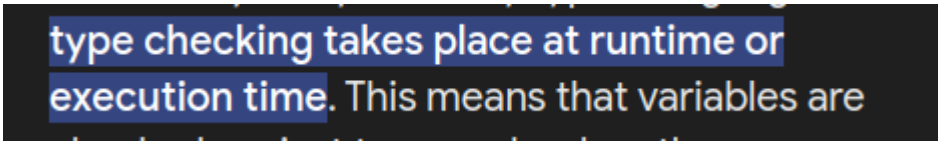
Statically Typed Languages



- types are associated with variables
- What about the auto keyword

what are javascript then?

Dynamically Types Languages



As a result of that variables aren't associated with specific types the values are what are associated with types.

Coercion (implicit Conversion)

Implicit conversion happens when the compiler automatically converts one data type to another without explicitly instructing it to.

c++

```
long a = 1;
int b = 2;

int c = a + b; (implicity converted )
```

```
double a = 1.2;
int b = 1;

double c = a + b; (implicity converted)
```

```
char a = 'a';
char b = 'b';
```

```
int c = a - b; // -1 (converted to an integer)
```

- what about?

```
int x = "3" + 3;
```

C++ is strict about its types

JavaScript

```
var result = '5' + 2; // 52

var result = '5' - 2; // 3

var check = (5 == '5'); // true

if(0) // false
        console.log("happy")
```

- Truthy and falsy values

Falsy values

```
if (!false) {
    console.log("false is falsy");
}
if (!0) {
    console.log("0 is falsy");
}
if (!"") {
    console.log("Empty string is falsy");
}
```

```
if (!null) {
    console.log("null is falsy");
}
if (!undefined) {
    console.log("undefined is falsy");
}
if (!NaN) {
    console.log("NaN is falsy");
}
```

Truthy Values

```
if (true) {
    console.log("true is truthy");
}
if (1) {
    console.log("1 is truthy");
}
if ("hello") {
    console.log("Non-empty string is truthy");
}
if ({}) {
    console.log("Empty object is truthy");
}
if ([]) {
    console.log("Empty array is truthy");
}
if (42n) {
    console.log("Non-zero BigInt is truthy");
}
if (Symbol()) {
    console.log("Symbol is truthy");
}
```

- So is everything is allowed in JavaScript

```
var x = "shehab";

x();
```

What will happen?

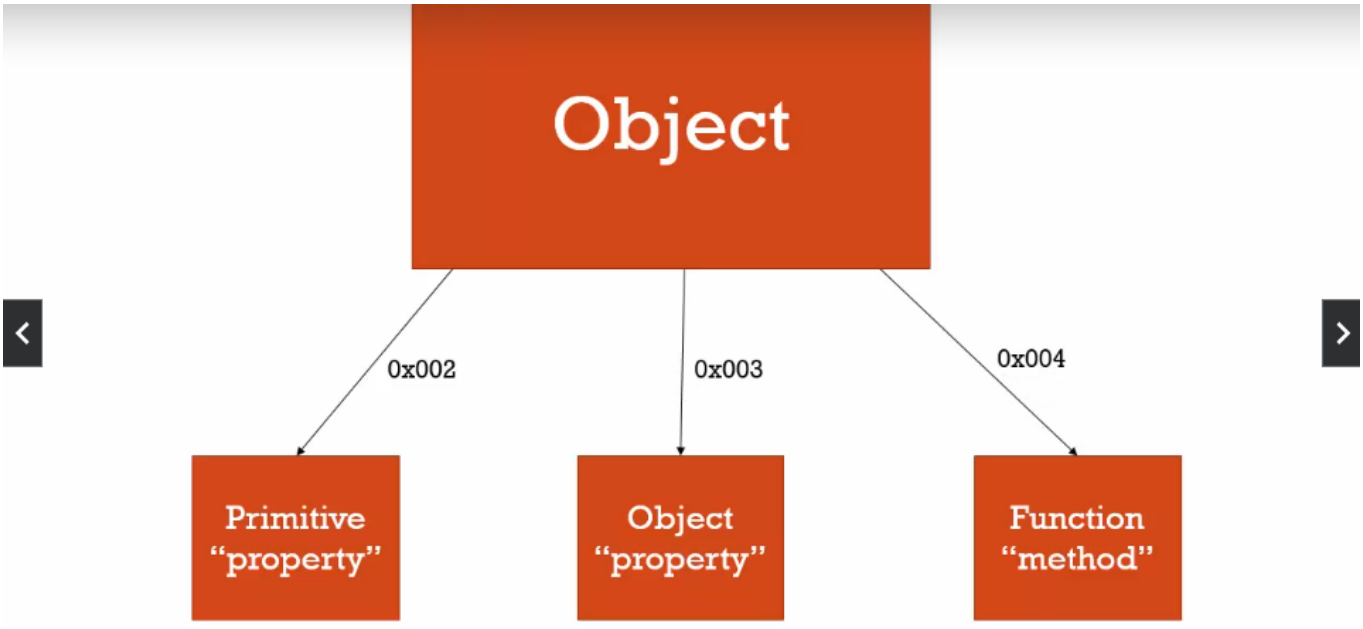
Type-strictness spectrum

Objects

- Don't make assumptions.
- Prototype Based not class based

Definition: A collection of values that are given names.

Objects and primitives: properties
Functions: Methods



- How can we create objects

```
const obj = {
        property1: value1, // property names may be an identifier
        2: value2, // names
        "property": value3, //strings,
        [expression] : value4 // expression but need to be in [ ]
}

const obj2 = {
        property1: value1, // property names may be an identifier
        2: value2, // names
        "property": value3, //strings,
        [expression] : value4 // expression but need to be in [ ]
}
```

- same object?

- **By Value and By Reference**
- **Prototypes and Inheritance**
- **Function as first class citizens**
- **Programming paradigms**