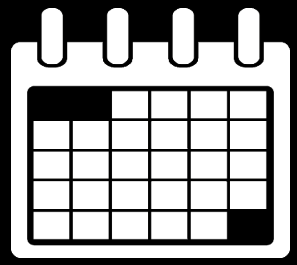
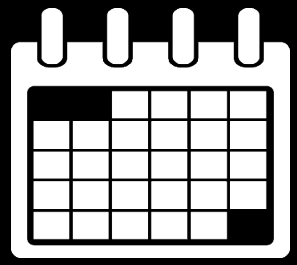
User Interfaces 1
JavaScript

Basis

KdG Karel de Grote Hogeschool



1. Variabelen
    1. Declaratie en scopes
    2. Types
    3. Converties
2. Expressies en statements
    1. Operatoren
    2. Toekenning
    3. Condities
    4. Lussen
3. Objecten



1. Variabelen
    1. Declaratie en scopes
    2. Types
    3. Converties
2. Expressies en statements
    1. Operatoren
    2. Toekenning
    3. Condities
    4. Lussen
3. Objecten

Variabelen

- Variabelen: waarden opslaan
  - Declaratie van variabele
  - Scope van variabelen: globaal – functie – block
- Variabelen hebben een type:
  - Primitief type: `Number`, `Boolean`, `String`
  - Object type: al de rest
  - Speciale gevallen: `null` en `undefined`
- Conversie van het ene type naar het andere
  - impliciet(/automatisch) – expliciet

Declaratie

- JavaScript is 'dynamic typed', d.w.z. dat bij het declareren van een variabele geen type wordt vastgelgd voor de soort data/waarde die de variabele kan bijhouden/verwijzen

- Bij declaratie van een variabele leg je wel de wijzigbaarheid en scope vast waarbinnen de variabele bereikbaar is, a.d.h.v. `var`, `let` of `const`

```
let myCounter;   //declaratie
myCounter = 1;   //toekenning of initialisatie
```

Het gebruik/aanspreken van niet gedeclareerde variabelen zorgt voor een impliciete declaratie via `var`!
Dus ALTIJD eerst expliciet declareren!

Scopes

1. Global scope
   - Overal bereikbaar!!
2. Function scope
   - Enkel bereikbaar binnen de functie
3. Block scope
   - Enkel bereikbaar binnen het omsluitend block (bv. statement)

=> scopes zijn genest, waarbij child-scopes ook toegang hebben tot hun parent-scope!

Er is ook een de scope 'Module', maar dit behandelen we niet in deze cursus!

`var`, `let` en `cons`

	Scope	Wijzigbaar	Impliciet
`const`	Block	Nee	Nee
`let`	Block	Ja	Nee
`var`	Function	Ja	Ja

Afspraken:

- Gebruik altijd een zo klein mogelijke scope
- Declareer altijd een variabele

=> maak **géén gebruik van `var`!**

var, let en const

```
var x = 1;
if (x === 1) {
    var x = 2;
    console.log(x);
}
console.log(x);
```

Output:
2
2

```
let x = 1;
if (x === 1) {
    let x = 2;
    console.log(x);
}
console.log(x);
```

Output:
2
1

```
const x = 1;
if (x === 1) {
    const x = 2;
    console.log(x);
}
console.log(x);
```

Output:
2
1

Primitieve types: `Number`

```
let myCounter;
myCounter = 23.173E23;
console.log(myCounter / 0);
```

```
let myText;
myText = "A";
console.log(-myText);
```

- Geen onderscheid tussen gehele getallen en kommagetallen: altijd floating point
- Tussen $\pm 1.79 \times 10^{308}$ en $\pm 5 \times 10^{-324}$
- Exact tot $\pm 2^{53}$ (=9.007.199.254.740.992)
- Rekenen: + - * / % en via `Math` functies
- Speciale getallen: `Infinity` en `NaN`

Primitieve types: `String`

```
let myVariable;
myVariable = "hello world\n";
console.log(myVariable.length);
```

- Immutable stuk tekst
  - D.w.z. niet overschrijfbaar op originele geheugenplaats
- Eigenschap `length` geeft de lengte
- Single óf double quotes: `"Hello"` of `'Hello'`
- Escapen van speciale karakters met \
- Samenvoegen met +
  - Template string literals adhv back ticks: `` `Hello ${name}` ``
- Je kan de afzonderlijke tekens ook met [] aanspreken

String methodes

```
myString = "Hello World!";
console.log(myString.charAt(1));
console.log(myString.slice(2,4));
console.log(myString.toUpperCase());
```

Veel methodes: (zie reference)

- `charAt()` → haal karakter op zero-based positie
- `indexOf()` → zero-based positie eerste match
- `substring()`
- `slice()`
- `split()`
- `toUpperCase()`
- …

Primitieve types: `Boolean`

```
let myVariable;
myVariable = "hello".length < 3;
console.log(myVariable);
```

- Twee waarden: `true` of `false`
- Een vergelijking resulteert in een `Boolean`
- Volgende waarden converteren naar `false`:
  (zie ook 'Implciete type convertie')

  `undefined` en `null`
  `0` en `-0`
  `NaN` en `Infinity`
  `""` of `''` `(lege string)`

  =>Al de rest converteert naar `true`

`null` en `undefined`

- `null`
  - Om aan te geven dat iets géén waarde heeft

```
let variableX = null;
console.log(variableX);
```

- `undefined`

```
let variableY;
console.log(variableY);
```

  - Variabelen die niet geïnitialiseerd (of zelfs niet gedeclareerd) zijn
  - Functies die niets retourneren geven `undefined` terug
  - Niet ingevulde parameters zijn `undefined`
  - Duidt meestal op een fout!

Impliciete type conversie (automatisch)

Indien nodig tracht JavaScript het type te converteren:

```
let myCounter = "1";
myCounter++;
console.log(myCounter);
```

=> myCounter wordt geconverteerd naar een Number

- Impliciete conversie (automatisch)
  - naar `String` → geen probleem (opgelet: `NaN` converteert naar "NaN")
  - naar `Number` → als het niet lukt: `NaN`
  - naar `Boolean` → zie type `Boolean`

Impliciete type conversie (automatisch)

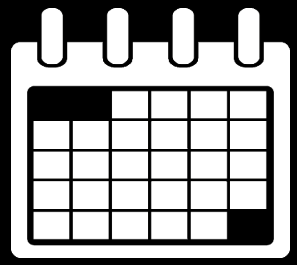
Waarde	Converteer naar `String`	Converteer naar `Number`	Converteer naar `Boolean`
undefined	"undefined"	NaN	false
null	"null"	0	false
true	"true"	1	true
false	"false"	0	false
""	""	0	false
"1.2"	"1.2"	1.2	true
"appel"	"appel"	NaN	true
0	"0"	0	false
NaN	"NaN"	NaN	false
Infinity	"Infinity"	Infinity	false
1	"1"	1	true

Expliciete type conversie

```
let number = "10";
console.log(10 + parseInt(number));
```

- Naar `Number`:
  - Globale functies: `parseInt(x)` en `parseFloat(x)`
- Naar `String`:
  - `toString()` methode
- Of gebruik wrapper constructors:
  `Number(x)`, `String(x)` of `Boolean(x)`

- Shortcuts:
  `x + ""` → zelfde als `toString()` of `String(x)`
  `+x` → zelfde als `Number(x)`
  `!!x` → zelfde als `Boolean(x)`



1. Variabelen
    1. Declaratie en scopes
    2. Types
    3. Converties
2. Expressies en statements
    1. Operatoren
    2. Toekenning
    3. Condities
    4. Lussen
3. Objecten

Expressies en statements

- Operatoren
  - Toekenning
  - Unaire – binaire – ternaire
  - (On)gelijkheid: `==` en `===`, `!=` en `!==`
- Statements:
  - Afsluiten met `;`
  - Blokken: `{ }`
- Toekenning: `=`
- Controle structuren
  - Conditioneel: `if` en `switch`
  - Lussen: `while` en `do..while`
    `for`, `for..of` en `for..in`

Toekenning

```
let variable = 100;
```

- = operator
- Lees als "wordt"
- Links van de = staat een variabele
- Rechts van de = staat een expressie
- Resultaat van de expressie wordt toegekend aan de variabele

Belangrijkste unaire operatoren

Operator	Betekenis	Type
++	Increment	Number
--	Decrement	Number
-	Negatie	
+	Conversie naar Number type	
!	Ontkenning	

Belangrijkste binaire operatoren

Operator	Betekenis	Type
+, -	Som of verschil	Number
+	Concatenatie	String
*, /, %	Rekenen	Number
<,<=,>,>=	Vergelijken op grootte	Number
<,<=,>,>=	Alfabetisch vergelijken	String
instanceof	Type controle	
in	Bevat	
==,!=	(on)gelijkheid	
===, !==	Strikte (on)gelijkheid	
&&, ||	Logische AND en OR	Boolean

Ternaire operator

- Eén ternaire operator: a ? b : c
  → "Als a dan b anders c"

```
console.log(input == "J" ? "ok" : "Niet ok");
```

- Ook samengestelde operatoren bestaan:

  *=, +=, -=, /=, …

Indien nodig wordt eerst een automatische conversie gedaan

(On)gelijkheid

```
let variable = "1";
console.log(variable == true);
```

== → gelijkheid eventueel na conversie

=== → strikte gelijkheid: zelfde waarde **en type**!

!= en !== → analoog

Statements

- (Line-)statements

; ontbreekt!

```
let variable = "1"
console.log(variable == true);
```

  - worden afgesloten met een ;
  - ; is optioneel => indien er geen staat wordt een newline als ; gezien.

Dit kan tot verwarring leiden, gebruik ALTIJD ; om zeker te zijn!

- Blok-statements: gebruik { } om een aantal statements te groeperen in een blok.
  - Voorbeelden zijn onder andere controle structuren (zoals `if`, `switch`, `for`, `while`…) maar ook functies

Conditions: `if` en `switch`

- `if/else/else if`

```
if (variable == 100) {
    //…
} else {
    //…
}
```

- `switch/case`

```
switch (expression) {
  case x:
    //…
  case y:
    //…
  default:
    //…
}
```

- Gelijkheid via `===` (op type dus)
- `break` niet vergeten!

Loops: `while`

- `while` of
  - Block-statement wordt enkel uitgevoerd zolang de conditie geldig is/blijft

```
while (i < 100) {
    //…
}
```

- `do/while`
  - Block-statement wordt minstens één maal doorlopen en zolang de conditie geldig is/blijft

```
do {
    //…
} while (i < 100)
```

Loops: `for`

- `for`

```
for (let i = 1; i <= 100; i++) {
    //…
}
```

- `for..of`
  - Iteratie van een 'iterable', zoals bv. Arrays!

```
for (const variable of iterable) {
    //…
}
```

Gebruik van `const` is niet noodzakelijk maar meestal is er geen nieuwe toekenning nodig. Indien dit wel noodzakelijk is kan ook `let` gebruikt worden.

- `for..in`
  - Iteratie van properties van een Object!
  - Geeft dus de key (= naam van de property)

```
for (const key in object) {
    //…
}
```

Oefening expressions 1

Wat is het verschil tussen de volgende uitdrukkingen?
Voorspel de uitvoer en test ze beiden uit in de console

```
let input = 'n';
console.log(input == "j" ? "ok" : "Niet ok");
```

```
let response = "n";
console.log(response = 'j' ? "ok" : "Niet ok");
```

Oefening expressions 2

Voorspel de uitvoer van de volgende uitdrukkingen en test ze uit in de console:

```
let invoer = 'Jos';
console.log(invoer == "Jos");
```

```
let waarde = 10;
console.log(waarde == "10");
```

```
let value = 100;
console.log("Dit geeft " + (value === "100"));
```

Oefening expressions 3

Vul het volgende script in de console aan. De bedoeling is om een aantal graden celsius om te zetten naar graden fahrenheit (zoek zelf de formule op).

```
let celsius = prompt("Geef de temperatuur in °C");
```

Gewenste uitvoer (bijvoorbeeld): 20°C = 68°F

Uitbreiding: Zorg ervoor dat je vooraf kunt kiezen of je van °C naar °F of van °F naar °C gaat omzetten!

opgelet: géén 'prompt' (of alert) mogelijk via node.js = server-side !!!

Oefening expressions 4

Deel 1:

Toon in de uitvoer van de console:
alle positieve **oneven** getallen onder de 10
Doe dit 3 keer:
met een for-lus,
met een while-lus en
met een do-while-lus.
Toon ook telkens de eindwaarde van de lusteller.

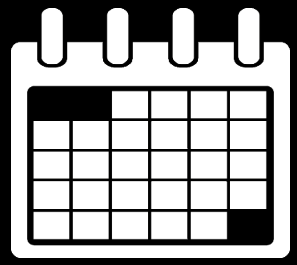
Deel 2 (maak gebruik van **`switch`**):

Overloop met een for-lus alle getallen van 2 tot en met 12 en druk ze onder elkaar af met console.log(…).

Voor de volgende waarden druk je niet het getal af maar een speciale naam:

2 – SnakeEyes, 3 – Trey, 7 – Seven,

11 – YoLeven, 12 - BoxCars



1. Variabelen
    1. Declaratie en scopes
    2. Types
    3. Converties
2. Expressies en statements
    1. Operatoren
    2. Toekenning
    3. Condities
    4. Lussen
3. Objecten

Objecten

- Standaard objecten
  - `String`, `Array`, `Date`, …
- Objecten maken
  - Object literals
  - `new Object()` gebruiken
- Nesting
  - Geneste array's
  - Objecten in array's

Standaard objecten - String

```
let string = "De Kiekeboes";
let stringObject = new String("De Kiekeboes");
let stringObject2 = new String("De Kiekeboes");
let stringObject3 = stringObject2;

console.log(typeof string);                          // string
console.log(typeof stringObject);                    // object

console.log(string == stringObject);                 // true, (opm. 2b)
console.log(string === stringObject);                // false, (opm. 1)
console.log(stringObject == stringObject2);          // false, (opm. 3a)
console.log(stringObject2 == stringObject3);         // true, (opm. 3b)
console.log(stringObject.valueOf() == stringObject2);  // true, (opm. 2b+2c)
console.log(stringObject.valueOf() === stringObject2); // false, (opm. 1+2c)
```

string (→ Call by Value)
object (→ Call by Reference)

(1)
het testen op een strikte gelijkheid === is enkel 'waar' wanneer beide types identiek zijn

(2)
a) enkel primitieve strings kunnen de inhoud van een string vergelijken/testen = stringwaarde
b) als minstens één van de operands een primitieve string is, zullen alle operands worden geconverteerd naar een primitieve string en wordt bijgevolg de stringwaarde vergeleken
c) de methode valueOf() zet een string om naar zijn primitieve string

(3)
a) het string-object vergelijkt objectvariabelen en niet de inhoud ervan,
b) beschouw het als een vergelijking van de adressen van het object die wanneer ze naar een ander object verwijzen, steeds verschillend zullen zijn

Een string kan zowel een primitief datatype als een object zijn. Indien nodig zal JavaScript automatisch een string naar een **`String`**-object omzetten. Dit gebeurt bv. (tijdelijk) bij alle stringfuncties.

Standaard objecten - String

```
let stringDoubleQuote = "De";
let stringSingleQuote = 'Kiekeboes';
let stringTemplateBackTick = `${stringDoubleQuote} ${stringSingleQuote}`;
```

Standaard objecten - Array

```
let array = []; // of array = new Array();
let dagen = new Array(7);
let kleuren = ["rood", "groen", "blauw"];

console.log(array.length);
console.log(dagen.length);
console.log(kleuren.length);
console.log(kleuren instanceof Array);
```

Uitvoer:

```
0
7
3
true
```

Standaard objecten - Array

```
let dagen = new Array(7);
dagen[0] = "maandag";
dagen[1] = "dinsdag";
dagen[6] = "zondag";
dagen[8] = "rustdag";
dagen[9] = null;
dagen[10] = undefined;
console.log(11, dagen[11]);

for (let i = 0; i < dagen.length; i++) {
    console.log(i, dagen[i]);
}
```

```
11 undefined
0 maandag
1 dinsdag
2 undefined
3 undefined
4 undefined
5 undefined
6 zondag
7 undefined
8 rustdag
9 null
10 undefined
```

```
let dagen = [];
dagen.push("maandag");
dagen.push("dinsdag");
dagen.push("woensdag");

for (let i = 0; i < dagen.length; i++) {
    console.log(i, dagen[i]);
}
```

```
0 maandag
1 dinsdag
2 woensdag
```

Standaard objecten - Date

```
let date = new Date();
console.log(date);
console.log(date.getFullYear());
console.log(date.toLocaleDateString());
console.log(date.toLocaleTimeString());

let birthday = new Date(1996, 11, 25);
console.log(birthday.toLocaleDateString());
```

note that (only) the month is 0-based

Uitvoer:

```
2020-04-10T08:24:32.559Z
2020
4/10/2020*
10:24:32
12/25/1996*
```

*format afhankelijk van je instellingen...

ISO dates: (YYYY-MM-DDTHH:MM:SS.SSSZ):
Date and time is separated with a capital T.

UTC time is defined with a capital letter Z.

```
Fri Apr 10 2020 10:24:32 GMT+0200 (Central European Summer Time)
2020
4/10/2020
10:24:32 AM
12/25/1996
```

Een object maken met `new Object()`

```
let person = new Object(); // let person = {};
person.name = "Jos";
person.age = 45;

console.log(person.name + " is " + person.age);
```

- Werkwijze
  - Maak eerst een algemeen object → `new Object();`
  - Voeg dan de gewenste attributen toe (geef ze bij voorkeur meteen een waarde, anders `undefined`)
- Opmerking
  - Je kunt nu de variabele 'person' ook als parameter aan een functie meegeven

Een object maken met `new Object()`

```
let person = new Object();
person["name"] = "Jos";
person["age"] = 45;

console.log(person.name + " is " + person.age);
```

Wat is het verschil met de voorgaande slide?

Attributen kan je ook definiëren met de array-notatie
(zie 'associatieve array'-oefening van deel 1)

```
let postcodes = new Array(3);
postcodes["Brussel"] = 1000;
postcodes["Antwerpen"] = 2000;
postcodes["Gent"] = 9000;

console.log(postcodes.Brussel);
```

Een object maken met `new Object()`

```
let person = new Object();
person["name"] = "Jos";
person["age"] = 45;

function showPerson(person) {
    console.log(person["name"]
        + " is " + person["age"]);
}

showPerson(person);
```

Hier is een functie **`showPerson`** toegevoegd die volledig los staat van het object

Een object maken met een literal

```
let person = {
    name: "Jos",
    age: 45
};

showPerson(person);
```

Wat valt er op?

- Geen **`new`** meer
- Attribuutnaam gevolgd door dubbelpunt
- Komma's tussen de attributen

(we maken gebruik van dezelfde **`showPerson`** functie)

Nesting: array in array

```
let personen = [["Jos", 45], ["Ann", 23], ["Tim", 39]];
// -- of --
let personen = [];
personen.push(["Jos", 45]);
personen.push(["Ann", 23]);
personen.push(["Tim", 39]);

for (let i = 0; i < personen.length; i++) {
    console.log(i, `${personen[i][0]} ${personen[i][1]}`);
}
```

```
0 Jos 45
1 Ann 23
2 Tim 39
```

Nesting: array met objecten

```
let personen = [
    {name: "Jos", age: 45},
    {name: "Ann", age: 23},
    {name: "Tim", age: 39},
];
// -- of --
let personen = [];
personen.push({name: "Jos", age: 45});
personen.push({name: "Ann", age: 23});
personen.push({name: "Tim", age: 39});

for (let i = 0; i < personen.length; i++) {
    console.log(i, `${personen[i].name} ${personen[i].age}`);
}
```

```
0 Jos 45
1 Ann 23
2 Tim 39
```