Vraag 1

De Spread Operator

const obj1 = {a: 1, b: 2}

const obj2 = { ...obj1, c : 3}

console.log(obj2);

Antwoord: {a: 1, b: 2, c: 3 }

Uitleg:
Hier wordt de **Spread syntax (...)** gebruikt (Lecture 1.3, Slide 17). De drie puntjes "pakken de inhoud van obj1 uit" en strooien de eigenschappen (a en b) direct in het nieuwe object obj2. Daarna wordt c: 3 simpelweg toegevoegd aan de lijst. Het resultaat is één plat object.

Opgave 2: Object Destructuring (Namen vs. Volgorde)

const person = { name: "John", age: 30 };

const { one, two } = person;

console.log(one, two);

Antwoord: undefined undefined

Uitleg:
Dit is een test op **Object Destructuring** (Lecture 1.3, Slide 32). Bij objecten zoekt JavaScript naar variabelen met **dezelfde naam** als de eigenschappen in het object.

- JavaScript zoekt naar person.one en person.two.
- Omdat deze niet bestaan (er is enkel name en age), krijgen de variabelen de standaardwaarde **undefined**.
- *Let op:* Bij een Array [one, two] had dit wel gewerkt op basis van volgorde, maar bij een Object { } telt enkel de naam.

Opgave 3: Optional Chaining (Veilig navigeren)

const user = { profile: { name: "Alice" } };

console.log(user.profile?.name);

Antwoord: Alice

Uitleg:
Hier zien we **Optional Chaining (?.)** (Lecture 1.3, Slide 19).

- De computer kijkt eerst of user.profile bestaat.

- Omdat dit object bestaat, gaat hij verder naar .name en vindt de waarde "Alice".
- Was profile bijvoorbeeld null geweest, dan had de code niet gecrasht maar undefined teruggegeven.

Opgave 4: Iteratie over Objecten (De blokkade)

```
const obj = { a: 1, b: 2, c: 3 };

for (const key of obj) {
  console.log(key);
}
```

Antwoord: error

Uitleg:
Dit is een cruciale valstrik over **Iterables** (Lecture 1.3, Slide 46 & 49).

- De **for...of** loop werkt enkel op dingen die "doorloopbaar" (iterable) zijn, zoals een Array of een String.
- Een gewoon **Object** is in JavaScript **niet iterable**.
- Deze code veroorzaakt een TypeError: obj is not iterable. Om over een object te loopen moet je eerst Object.keys(obj) gebruiken.

Opgave 5: forEach en Type Coercion (Dwang)

```
let total = 10;

var arr = ["3", "4"];

arr.forEach(value => {
   total -= value;
});

console.log(total);
```

Antwoord: 3

Uitleg:
Deze vraag combineert de **forEach** methode (Lecture 1.3, Slide 50) met **Type Coercion** (Lecture 1.2, Slide 3).

- De array bevat strings ("3" en "4").
- Je gebruikt de min-operator (-). Omdat je niet kunt aftrekken van tekst, dwingt JavaScript de strings om getallen te worden.

- Stap 1: 10 - 3 = 7.
- Stap 2: 7 - 4 = 3.
- *Belangrijk:* Als er een + had gestaan, had JavaScript de tekst aan elkaar geplakt en was het resultaat een tekst (string) geweest. Bij -, * en / kiest hij altijd voor getallen.

Vraag 2

Wat is een correcte oproep van Object.fromEntries om het object {a: 1, b: 2, c: 3} te verkrijgen?

Optie 1: Object.fromEntries(["a", "b", "c"], [1, 2, 3]);

- **Status: ❌ FOUT**
- **Uitleg:** Je geeft hier **twee aparte argumenten** (twee losse arrays) mee aan de functie, gescheiden door een komma. Object.fromEntries verwacht echter maar **één** argument: een lijst met paren. Hij zal de tweede array ([1, 2, 3]) simpelweg negeren en proberen van de eerste array een object te maken, wat zal mislukken.

Optie 2: Object.fromEntries([["a", "b", "c"], [1, 2, 3]]);

- **Status: ❌ FOUT**
- **Uitleg:** Je geeft wel één grote array mee, maar de inhoud binnenin klopt niet. Object.fromEntries zoekt naar "paren" (lijstjes van exact 2 elementen). Hier geef je lijstjes van **3 elementen**. De computer weet nu niet wat de "key" is en wat de "value".

Optie 3: Object.fromEntries(['a', 1], ['b', 2], ['c', 3]);

- **Status: ❌ FOUT**
- **Uitleg:** Net als bij optie 1 geef je hier **meerdere losse argumenten** mee. De functie pakt alleen het eerste paar ['a', 1] en negeert de rest. Je zou dus hooguit een object {a: 1} krijgen, maar niet de volledige lijst. Bovendien ontbreken de buitenste haken die er een lijst van maken.

Optie 4: Object.fromEntries([['a', 1], ['b', 2], ['c', 3]]);

- **Status: ✅ JUIST**
- **Uitleg:** Dit is de correcte syntax (zie **Lecture 1.3, Slide 35-36**).
    1. Je geeft **één grote array** mee (de buitenste haken [ ]).

2. Binnen die array zitten **sub-arrays**, die elk fungeren als een **key-value paar**.
3. Elk sub-array heeft exact **2 elementen**: het eerste is de naam ('a'), het tweede is de waarde (1). Dit is exact de structuur die de functie nodig heeft om een object te bouwen.

Het Juiste Antwoord:

Object.fromEntries([['a', 1], ['b', 2], ['c', 3]]);

Examen-tip:

Denk aan de samenhang met Object.entries().

- Als je Object.entries({a: 1, b: 2}) doet, krijg je [['a', 1], ['b', 2]].
- Omdat fromEntries het **omgekeerde** doet, moet je hem dus ook exact die "array-in-array" structuur teruggeven.

Onthoud: **fromEntries = 1 Array met daarin paren van 2.**

Vraag 3

Vraag: Wat is het resultaat van volgende code en leg uit hoe je tot dit antwoord komt?

```
function someBigFunction() {
 let aRandomVariable = 5;


 function someSmallerFunction() {
  aRandomVariable += 1;
  return aRandomVariable;
 }


 someSmallerFunction(); // Eerste aanroep


 return someSmallerFunction; // Functie wordt teruggegeven
}


function anotherFunction() {
```

```
  const result = someBigFunction();
  return result(); // Tweede aanroep
}

console.log(anotherFunction());
```

Resultaat:

7

Uitleg (De Trace):

Om tot dit resultaat te komen, volgen we de uitvoering stap voor stap:

1. **Start:** De code begint onderaan bij console.log(anotherFunction()). We springen naar de functie anotherFunction.
2. **Lijn 16 (const result = someBigFunction()):** We roepen eerst de "moederfunctie" aan.
    - Binnen someBigFunction wordt aRandomVariable aangemaakt met de waarde **5**.
    - **Lijn 10 (someSmallerFunction()):** De binnenste functie wordt voor de **eerste keer** uitgevoerd.
        - Hij doet aRandomVariable += 1. De waarde in het geheugen wordt dus **6**.
    - **Lijn 12 (return someSmallerFunction):** De moederfunctie stopt en geeft de *definitie* van de kleine functie terug.
3. **Terug in anotherFunction:** De variabele result bevat nu de functie someSmallerFunction. Dankzij de **Closure** onthoudt deze functie de variabele aRandomVariable (die op dit moment **6** is).
4. **Lijn 17 (return result()):** Nu wordt de functie die in result zit voor de **tweede keer** uitgevoerd.
    - De code gaat opnieuw naar de regel aRandomVariable += 1.
    - Omdat de closure de waarde **6** had onthouden, wordt het nu: 6 + 1 = 7.
    - De functie geeft **7** terug.
5. **Output:** De waarde 7 wordt doorgegeven aan de console.log.

Waarom dacht je aan 6?

De meeste studenten vergeten dat de binnenste functie **twee keer** wordt uitgevoerd:

1. Eén keer "stiekem" binnen de moederfunctie zelf (lijn 10).
2. Eén keer expliciet wanneer je het resultaat aanroept (lijn 17).

Cruciaal concept: Een **Closure** (Slide 24, Lecture 2.1) zorgt ervoor dat aRandomVariable niet gewist wordt als de moederfunctie stopt. De variabele blijft "leven" in een rugzakje bij de kleine functie, waardoor de teller kan blijven doorlopen van 6 naar 7.

Tip voor het examen: Kijk altijd heel goed of een functie binnenin een andere functie al een keer wordt aangeroepen (()) of alleen wordt gedefinieerd!

Vraag 4

Welke naam moet je aan de constructor geven in een JavaScript klasse Person?

Optie 1: constructor

- **Status:** ✅ **JUIST**
- **Uitleg:** In JavaScript is constructor een **gereserveerd trefwoord** (reserved keyword). Elke klasse mag exact één methode hebben met deze naam. Deze methode wordt automatisch opgeroepen op het moment dat je een nieuw object aanmaakt met het woord new (zie **Lecture 2.2, Slide 29**).

Optie 2: person

- **Status:** ❌ **FOUT**
- **Uitleg:** JavaScript kijkt niet naar de naam van de klasse om de constructor te vinden. Als je een methode person() noemt, ziet de browser dit als een gewone, extra functie en niet als de opstart-functie van je object.

Optie 3: init

- **Status:** ❌ **FOUT**
- **Uitleg:** Hoewel init (afkorting voor initialization) in veel andere talen of oude JavaScript-patronen gebruikt wordt, is het binnen de moderne class syntax geen geldige naam voor de constructor.

Optie 4: Person

- **Status:** ❌ **FOUT**

- **Uitleg:** Dit is een veelgemaakte fout door studenten die ook in **Java of C++** programmeren. In die talen heeft de constructor dezelfde naam als de klasse. **In JavaScript is dit niet het geval.** De klasse heet Person, maar de machine binnenin heet altijd constructor.

Het Juiste Antwoord:

Constructor

Vraag 5

Op welke elementen heeft de volgende CSS-selector invloed?

HTML:

```
<div>
 <p>First</p>
 <p>Second</p>
 <p>Third</p>
 <img src="image.png" alt="Img">
 <p>Fourth</p>
 <p>Fifth</p>
</div>
```

CSS Selector:
div:nth-child(2), div :nth-of-type(2n-1)

Het Juiste Antwoord (Aanvinken):

- ✅ **First paragraph** (p1)
- ✅ **Third paragraph** (p3)
- ✅ **Fifth paragraph** (p5)
- ✅ **Img element** (img1)

Stapsgewijze Uitleg:

De selector bestaat uit twee delen, gescheiden door een komma. Elk deel wordt apart geëvalueerd:

Deel 1: div:nth-child(2)

Dit deel selecteert de div zelf, maar alleen als de div het tweede kindje is binnen de body (of de container waarin hij staat). In de meeste examenvragen gaan we ervan uit dat dit waar is. Als de div geselecteerd wordt, krijgt hij bijvoorbeeld een achtergrondkleur, maar de vraag peilt specifiek naar de elementen *binnen* de HTML-structuur die direct "getroffen" worden door de tweede, specifiekere selector.

Deel 2: div :nth-of-type(2n-1) (De kern van de vraag)

Let op de **spatie** tussen div en de dubbele punt. Die spatie betekent: "zoek alle afstammelingen **binnen** de div".
De formule **2n-1** staat voor alle **oneven nummers** (1, 3, 5, ...).

HET GEHEIM: nth-of-type telt per HTML-tag apart. JavaScript/CSS kijkt naar de lijst en begint voor de tags te tellen, maar begint voor de <img> tag **opnieuw bij 1**.

1. ** First **: Dit is de **1e** paragraaf. 1 is oneven. ✅ **Beïnvloed.**
2. ** Second **: Dit is de **2e** paragraaf. 2 is even. ❌ Niet beïnvloed.
3. ** Third **: Dit is de **3e** paragraaf. 3 is oneven. ✅ **Beïnvloed.**
4. **<img>**: Dit is de **1e** afbeelding in de div. De teller voor afbeeldingen begint hier op 1. 1 is oneven. ✅ **Beïnvloed.**
5. ** Fourth **: Dit is de **4e** paragraaf (de afbeelding wordt overgeslagen bij het tellen van p's!). 4 is even. ❌ Niet beïnvloed.
6. ** Fifth **: Dit is de **5e** paragraaf. 5 is oneven. ✅ **Beïnvloed.**

Waarom was jouw gedachte "Second paragraph" fout?

Jij dacht waarschijnlijk aan **nth-child(2)**.

- **nth-child** telt blindelings alles: 1, 2, 3, 4... ongeacht wat het is. Dan zou de "Second paragraph" inderdaad nummer 2 zijn.
- **nth-of-type** is slimmer: het telt per groepje. Het telt alle p's als één groep en alle img's als een andere groep.

Examen-tip:
Zie je op het examen een mix van verschillende tags (zoals p en img) in een lijst?

- Bij **nth-child**: telt iedereen mee.
- Bij **nth-of-type**: springt de teller op 1 bij elk nieuw soort tag!

Vraag 6

Welke statements over de onderstaande CSS-code zijn correct?

Code:

```
div {
 text-decoration: underline;
}

div::after {
 content: "hallo";
 width: 0;
 height: 1rem;
 display: block;
 background-color: limegreen;
}
```

Het Juiste Antwoord:

- ✅ **De string "hallo" zal onderlijnd zijn.**

Gedetailleerde uitleg per statement:

1. Dit voegt een nieuwe node aan de DOM tree toe op het einde van iedere div

- **Status:** ❌ **FOUT**
- **Uitleg:** Een pseudo-element (::after) heet "pseudo" omdat het **niet** echt in de HTML-structuur (de DOM) komt te staan. Je kunt het wel zien in de browser-inspector, maar het is geen echte node die je met JavaScript kunt opvragen via bijvoorbeeld document.getElementById. Het is puur een visuele toevoeging van de browser.

2. De tekst hallo heeft een groene achtergrondkleur

- **Status:** ❌ **FOUT (Instinker!)**
- **Uitleg:** Kijk goed naar de eigenschappen: er staat **width: 0**.
    - De achtergrondkleur wordt alleen getekend *binnen* de box van het element.

    - Omdat de box 0 pixels breed is, is er geen ruimte om de kleur limegreen te tonen.
    - De letters "hallo" zullen wel zichtbaar zijn (omdat tekst standaard "overstroomt" uit een te kleine box), maar de groene kleur zal je niet zien achter de letters.

3. De string hallo zal onderlijnd zijn

- **Status:** ✅ **JUIST**
- **Uitleg:** De div heeft de eigenschap text-decoration: underline. Een pseudo-element zoals ::after gedraagt zich als een "kind" van de div. Hierdoor neemt het de tekststijl van de ouder over. De tekst "hallo" wordt dus netjes onderlijnd, net als de rest van de tekst in de div.

4. De tekst hallo staat naast de div

- **Status:** ❌ **FOUT**
- **Uitleg:** Er staat **display: block** (zie Slide 24 van Lecture 3).
    - Normaal zijn pseudo-elementen inline (ze staan achter de tekst op dezelfde regel).
    - Door er een block van te maken, dwing je het element naar een **nieuwe regel** *binnen* de div. Het staat dus niet naast de div, maar eronder (op een eigen regel).

Samenvatting voor je examen:

- **::after / ::before** hebben altijd content: "" nodig om te bestaan.
- Ze staan **in** het element, niet ernaast.
- Ze erven tekststijlen (zoals kleur en onderlijning) over van hun ouder-element.
- Let op de **width** en **height**: bij width: 0 zie je geen achtergrondkleur, zelfs niet als de tekst er wel staat!

Vraag 7

Vraag: Welke van de volgende statements over TypeScript zijn correct? (Meerdere antwoorden mogelijk)

De Juiste Antwoorden:

- ✅ **Je kan in een interface aanduiden dat een property optioneel is door een vraagteken te gebruiken.**
- ✅ **Je kan een array met getallen op deze manier aanmaken: const numbers: number[] = [1, 2, 3, 4, 5];**

Gedetailleerde uitleg per statement:

1. Je kan in een interface aanduiden dat een property optioneel is door een vraagteken te gebruiken

- **Status:** ✅ **JUIST**
- **Uitleg:** Dit is een kernconcept van TypeScript. Als je een object beschrijft (bijv. een Mario-karakter), maar je weet niet zeker of elk karakter een "leeftijd" heeft, schrijf je:

```
interface Character {
  name: string;
  age?: number; // Het vraagteken maakt het optioneel
}
```

Dit voorkomt foutmeldingen als de data van de server (Taylor Swift of Mario) onvolledig is.

2. Je kan geen homogene array met een vaste lengte aanmaken

- **Status:** ❌ **FOUT**
- **Uitleg:** In TypeScript kan dit wél via **Tuples**. Als je bijvoorbeeld een coördinaat van een speler in je Labyrinth-spel wilt vastleggen op exact twee getallen, kun je dit doen:
  let position: [number, number] = [0, 0];
  Dit is een array met een vaste lengte van 2.

3. Moderne browsers kunnen TypeScript uitvoeren

- **Status:** ❌ **FOUT (Héél belangrijk!)**
- **Uitleg:** Geen enkele browser (Chrome, Firefox, Safari) begrijpt TypeScript. TypeScript is een "programmeertaal voor de developer". Voordat de website werkt, moet de code worden **gecompileerd** (omgezet) naar gewoon JavaScript.
- **Examen-link (Pagina 6):** De opgave zegt letterlijk: *"tsconfig.json zorgt er voor dat de TypeScript omgezet wordt naar assets/js/index.js"*. De browser leest dus de .js file, niet de .ts file.

4. Je kan een array met getallen op deze manier aanmaken: const numbers: number[] = [1, 2, 3, 4, 5];

- **Status:** ✅ **JUIST**
- **Uitleg:** Dit is de standaard manier in TypeScript om een array te "typeren". Je zegt hiermee tegen de computer: "In deze lijst mogen **alleen** getallen staan." Probeer je er een string in te zetten? Dan krijg je direct een foutmelding tijdens het programmeren.

5. Met TypeScript kun je geen DOM manipulaties uitvoeren

- **Status:** ❌ **FOUT**
- **Uitleg:** Juist wel! TypeScript heeft zelfs speciale types voor de DOM, zoals HTMLElement of HTMLButtonElement.
- **Examen-link (Pagina 7):** Je moet de Mario-characters dynamisch toevoegen aan de webpagina via TypeScript. Dat is de definitie van DOM-manipulatie.

Samenvatting voor jouw examenvoorbereiding:

1. **TypeScript is voor jou, niet voor de browser:** Het helpt je fouten te vinden *voordat* je de site opent.
2. **Interfaces zijn je vriend:** Gebruik ze om de structuur van de API-data (Taylor Swift/Mario) vast te leggen. Gebruik de ? voor velden die soms leeg zijn.
3. **Strict typing:** Gebruik altijd de syntax variabele: type. Dus niet let name = "Mario", maar const name: string = "Mario". Dat is wat Dieter Mourisse bedoelt met "Voeg overal waar kan types toe".

Vraag 8

Beschrijf zo exact mogelijk de verschillende return values van onderstaande functie in welke gevallen die optreden. *text* kan zowel een getal als een string zijn.

```
1 async function myFunction(text) {
2   return text.toUpperCase();
3 }
```

Antwoord:

Scenario 1: text is een string

- **Geval:** De input is een tekstwaarde (bijv. "hello").
- **Resultaat:** De methode .toUpperCase() werkt correct op een string.

- **Return value:** Een **Fulfilled Promise** (ingeloste belofte) met de tekst in hoofdletters als resultaatwaarde (bijv. "HELLO").

Scenario 2: text is een number

- **Geval:** De input is een getal (bijv. 123).
- **Resultaat:** Een getal in JavaScript heeft geen methode .toUpperCase(). Dit veroorzaakt een TypeError tijdens het uitvoeren van de code.
- **Return value:** Een **Rejected Promise** (afgewezen belofte) met de TypeError als reden (reason).