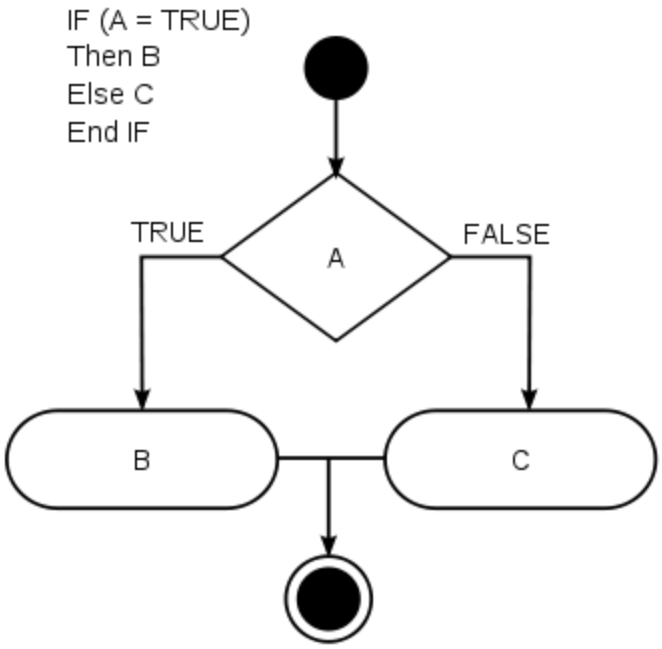
Control Flow

De **control flow** is de volgorde waarin de computer instructies uitvoert in een computerscript. Code wordt uitgevoerd van de eerste regel code in een bestand tot de laatste regel, tenzij het bepaalde structuren tegenkomt die deze control flow beïnvloeden, zoals condities en lussen. Wanneer bijvoorbeeld een gebruiker een formulier moet invullen om een niewsbrief te kunnen ontvangen en een vereist veld, in dit geval het e-mailadres niet invult, dan krijgt de bezoeker een melding dat het veld moet ingevuld worden met geldige data, in dit geval een geldig e-mailadres. Om deze functionaliteit te realiseren via JavaScript-code, moeten we gebruik maken van een **conditionele instructie** of statement (`if…else`).



If-Then-Else-diagram. Bron: https://commons.wikimedia.org/wiki/File:If-Then-Else-diagram.svg

Control flow kan onderverdeeld worden in 3 categorieën:

- **Condities**
  - `if…else`
  - `switch` en `case`
- **Uitzonderingen** *(Eng. Exceptions)*
  - `try…catch…finally`
- **Lussen**
  - `for`

- `for…in`
- `for…of`
- `while`
- `do…while`

Block statement

Een block statement wordt gebruikt om statements of instructies te groeperen die in sequentie worden uitgevoerd. Een block wordt **afgebakend** ***(Eng. delimited)*** door een paar **accolades** ***(Eng. curly brackets)*** en bevat een eigen scope.

```js
{
  statement_1;
  statement_2;
  .
  .
  .
  statement_n;
}
```

Voorbeeld:

```js
if (i < 100) {
  i++;
}
```

In dit voorbeeld is `{ i++; }` het block statement.

Block bereik

Sinds ECMAScript 2015 bevat JavaScript blok **bereik** ***(Eng. scope)*** via de `const` en `let` variabele. De `var` variabele heeft geen blok bereik.

```js
var y = 3;
{
  var y = 2;
}
console.log(y);
// Output: 2
```

De variabele `y` bevat in de `console.log` de waarde `2`. In Java de variabele `y` de waarde `3` bevatten.

```js
let y = 3;
{
  let y = 2;
}
console.log(y);
// Output: 3
```

Gebruiken bij de declaratie het `let` keyword, dan bevat `y` in de `console.log` de waarde `3`.

Conditionele statements

Een **conditionele statement** ***(Eng. conditional statement)*** is een verzameling van commando's die pas uitgevoerd worden indien een bepaalde conditie **waar** ***(Eng. true)*** is. JavaScript ondersteunt twee conditionele statements, namelijk `if...else` en `switch`.

`if…else` statement

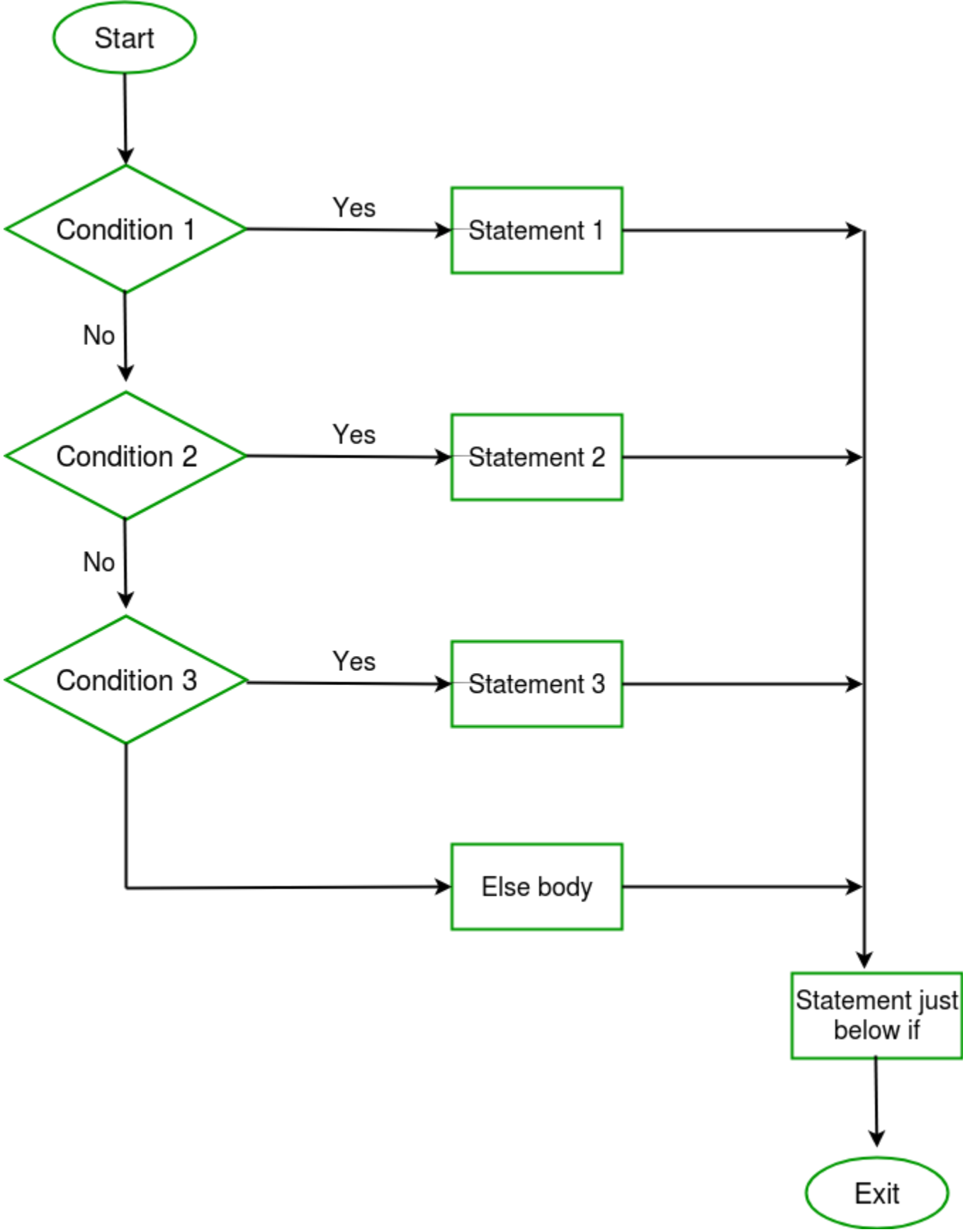
We gebruiken een `if` statement om commando's uit te voeren wanneer een bepaalde conditie waar is. De optionele `else` voert commando's uit indien de conditie **onwaar** ***(Eng. false)*** is. De anatomie van een `if...else` statement ziet er als volgt uit:

```js
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

Indien de conditie `true` is, dan zal `statement_1` **uitgevoerd worden** ***(Eng. execute)***. Indien de conditie `false` is, dan zal `statement_2` uitgevoerd worden.

Wanneer **meerdere** ***(Eng. multiple)*** condities in sequentie gestest moeten worden, dan ziet de code er als volgt uit:

```js
if (condition_1) {
  statement_1;
} else if (condition_2) {
  statement_2;
} else if (condition_n) {
  statement_n;
} else {
  statement_last;
}
```



Decision Making in C / C++ (if , if..else, Nested if, if-else-if). Bron: https://www.geeksforgeeks.org/decision-making-c-c-else-nested-else/

Indien `condition_1` `true` is, wordt `statement_1` uitgevoerd. Is `condition_1` `false` , dan wordt de `condition_2` geëvalueerd. Is deze conditie `true` , dan wordt `statement_2` uitgevoerd. Wanneer

`condition_2` de waarde `false` bevat, dan wordt de volgende conditie geëvalueerd. Indien alle condities de waarde `false` bevatten, dan zal `statement_last` uitgevoerd worden onder het `else` statement.

De waarden `null` , `0` , `NaN` , `""` en `undefined` vermeld als conditie, worden geëvalueerd als zijnde `false` .

```js
if (null) {
  statement_1;
}
```

In dit voorbeeld wordt `statement_1` niet uitgevoerd omdat de conditie `false` is.

```js
var state = new Boolean(false);
if (state) // conditie is true, state is een Boolean object waardoor de conditie waar i
if (state == true) // conditie is false
```

In het volgende voorbeeld definiëren we een functie `completeMessage` met twee argumenten, namelijk `action` (omschrijven van een actie) en `isError` (boolean waarde om aan te duiden dat de actie al dan niet fouten verroorzaakte). Indien het geen fouten veroorzaakte, dan geeft de functie de tekst `De actie ...` terug. Indien `isError` de waarde true bevat, dan zal de functie de tekst `Er is een fout ...` teruggeven.

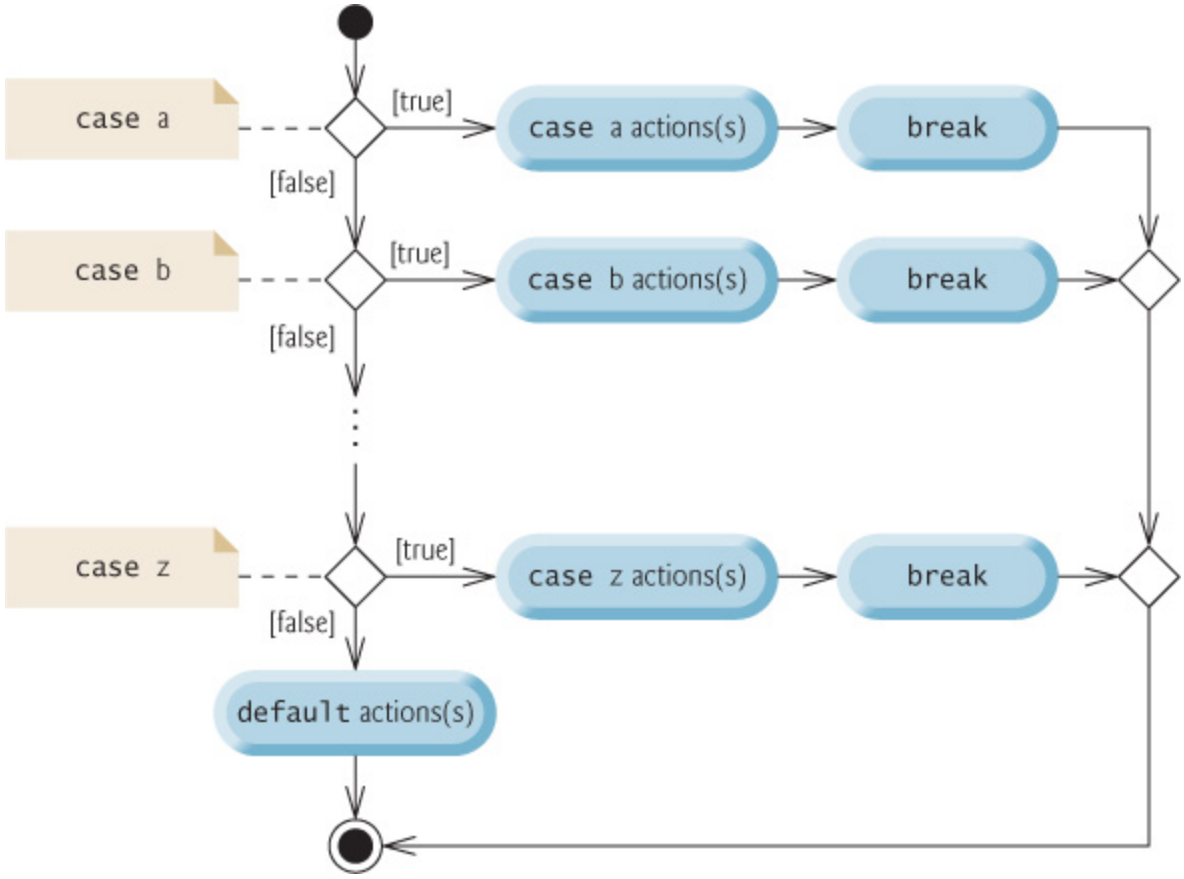
```js
function completeMessage(action, isError) {
  if (!isError) {
    return 'De actie ' + action + ' is met succes uitgevoerd!';
  } else {
    return 'Er is een fout opgetreden tijdens het uitvoeren van de actie ' + action + '
  }
}

var m1 = completeMessage('consumeren van data', false); // m1 bevat de waarde 'De actie
var m2 = completeMessage('consumeren van data', true); // m2 bevat de waarde 'Er is een
```

switch statement

Een `switch` statement evalueert een expressie om de waarde ervan te matchen aan een bepaalde label, via het `case` keyword. Wanneer een match gevonden wordt, dan zal het programma de instructies uitvoeren uit deze `case` . Een `switch` statement ziet er als volgt uit:

```js
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    ...
  default:
    statements_def
    [break;]
}
```



switch multiple-selection statement UML activity diagram with break statements. Bron: https://www.oreilly.com/library/view/javatm-how-to/9780133813036/ch05lev2sec23.html

Het programma gaat op zoek naar een `case` clausule waarvan het label overeenkomt met de waarde van de expressie. Indien een clausule gevonden, dan zullen de statements onder deze clausule uitgevoerd worden. Indien geen match gevonden wordt, dan zal de optionele `default` clausule uitgevoerd worden. Indien geen match gevonden wordt of na afhandelen van acties, zullen de instructies na het `switch` statement uitgevoerd worden. Swicth cases gebruikt strikte vergelijkingen (`===`). De waarden van cases moeten van hetzelfde type zijn als de waarde van de expressie.

```js
let mode = 4, dx = 0, dy = 0;
switch(mode) {
  case 1:
    dx++;dy=0;break;
  case 2:
    dx--;dy=0;break;
  case 3:
    dy++;dx=0;break;
  case 4:
    dy--;dx=0;break;
  default:
    dx = 0; dy = 0; break;
}
console.log(`The values are dx:${dx} and dy:${dy}`);
// Ouput: The values are dx:0 and dy:-1
```

In het voorbeeld bevat de `mode` variabele de waarde 4. Via het `switch` statement gaan we op zoek naar een match. Vervolgens worden de acties onder de match uitgevoerd. Het `break` statement zorgt ervoor dat het `switch` statement verlaten zal worden. Verwijderen we het `break` stetement uit de gematchte case, dan zullen de volgende cases uitgevoerd worden (`dy` zal dan de waarde `0` bevatten).

```js
let dayOfWeek = new Date().getDay();
let day = '';
switch(dayOfWeek) {
  case 0:
    day = 'Sunday'; break;
  case 1:
    day = 'Monday'; break;
  case 2:
    day = 'Tuesday'; break;
  case 3:
    day = 'Wednesday'; break;
  case 4:
    day = 'Thursday'; break;
  case 5:
    day = 'Friday'; break;
  case 6:
    day = 'Saturday'; break;
}
console.log(`Today it's ${day}.`);
```

In het voorbeeld initialiseren we een variabele `dayOfWeek` met als waarde de dag van de week uitgedrukt als een positief geheel getal. De waarde `0` komt overeen vmet zondag, waarde `1` is maandag … In het geval (`case`) de waarde `0` de waarde van de expressie is, kennen we de string `'Sunday'` toe aan de variabele `day` , vervolgens verlaten we het `switch` statement via het `break` keyword.

Weet dat je de bovenstaande code beter kan schrijven door gebruik te maken van een array:

```js
let dayOfWeek = new Date().getDay();
const daysOfWeek = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "
let day = days[dayOfWeek];
console.log(`Today it's ${day}.`);
```

← Operatoren

Lussen →

Laatst bijgewerkt: 28/8/2022 19:19:46