Arduboy

Starters Guide

Stijn Caerts

Jeugd, Cultuur en Wetenschap vzw

JEUGD, CULTUUR EN WETENSCHAP VZW

STIJN.CAERTS.BE – WWW.JEUGDCULTUURENWETENSCHAP.BE

Eerste versie, augustus 2019

Inhoudsopgave

Introductie tot C++

1. Inleiding

1.1 Wat is C++?

Programma's voor **Arduino** en **Arduboy** worden geschreven in de programmeertaal C++. Het is niet nodig om de hele programmeertaal te kennen en begrijpen voor je aan de slag kan gaan met programmeren. Daarom geven we hier een beknopt overzicht van de belangrijkste concepten die je nodig hebt om van start te gaan.

In de volgende hoofdstukken komen variabelen en types, controlestructuren (if-then-else, for, while) en functies en procedures aan bod. Tot slot zijn er nog twee hoofdstukken die dieper ingaan op de mogelijkheden van C++, namelijk arrays en lijsten, en klassen en objecten.

Voor een interactieve en uitgebreidere introductie tot C++, kan je terecht bij W3Schools (`https://www.w3schools.com/cpp/`).

1.2 Syntax

1.2.1 Puntkomma

Achter elke instructie wordt in C++ een puntkomma ; geplaatst. Deze puntkomma vertelt de compiler dat op die plaats een instructie eindigt. Als je een puntkomma vergeet te plaatsen, is het programma niet correct en zal je het programma niet kunnen compileren. De compiler zal dan een foutmelding geven.

1.2.2 Accolades

Accolades {} worden gebruikt om een blok code aan te duiden. De accolades moeten gebalanceerd voorkomen in de code, wat betekent dat voor elke openende accolade { er een bijhorende sluitende accolade } moet zijn. Het gebruik van accolades wordt later geïllustreerd bij de controlestructuren (if-then-else, for, ..), functies en klassen.

1.2.3 Commentaar

Om de leesbaarheid van je code te verhogen, is het nuttig om commentaar toe te voegen. In deze commentaar beschrijf je wat dit deel van de code juist doet. Hierdoor is het duidelijk wat je juist hebt geprogrammeerd, ook als je later opnieuw je code bekijkt.

In C++ zijn er twee verschillende manieren om commentaar toe te voegen.
Commentaar op één lijn wordt aangeduid met `//`. Alle tekst na `//` tot het einde van de lijn wordt beschouwd als commentaar en zal bijgevolg niet uitgevoerd worden.

■ Voorbeeld 1.1 — Commentaarlijn.

```
// Dit is een lijn commentaar
int a = 42;
```

◂

Commentaar over meerdere lijnen start met /* en eindigt met */. Alle tekst tussen /* en */ wordt door de compiler genegeerd.

■ Voorbeeld 1.2 — Commentaar over meerdere lijnen.

```
/*
  Deze commentaar neemt
  meerdere lijnen in beslag.
*/
int a = 42;
```

◂

2. Variabelen en types

Variabelen worden gebruikt om informatie of gegevens bij te houden in het programma. Een variabele is een plaats in het geheugen, met een zelfgekozen naam, waar gegevens in opgeslagen kunnen worden. In C++ heeft elke variabele een type. Een type komt overeen met het soort van gegevens dat opgeslagen kunnen worden in zo'n variabele.

2.1 Datatypes

Op basis van het datatype van een variabele wordt er geheugen gereserveerd om informatie in op te slaan. In C++ zijn de volgende primitieve datatypes gedefinieerd:

Definitie 2.1.1 — Primitieve datatypes.

bool	Booleaanse waarde: `true` of `false`
char	Karakter
int	Geheel getal
float	Kommagetal (met enkele precisie)
double	Kommagetal (met dubbele precisie)

Deze types worden gebruikt bij variabelen, maar ook bij functies. Deze primitieve datatypes kunnen ook gebruikt worden om complexere types samen te stellen, zoals bijvoorbeeld bij arrays en klassen.

Het type van een variabele is belangrijk voor de compiler omdat het type bepaald hoeveel geheugen er gereserveerd moet worden voor de variabele. Voor het opslaan van een `float`-variabele zijn er bijvoorbeeld 32 bits nodig, terwijl er voor de opslag van een kommagetal met dubbele precisie (`double`) 64 bits nodig zijn.

2.2 Variabelen

Een variabele kan gezien worden als een opslagplaats waaraan een naam verbonden is. In deze opslagplaats kan een waarde geplaatst worden.

2.2.1 Declareren

Alvorens informatie kan opgeslagen worden in een variabele, moet de variabele eerst *gedefinieerd* of *gedeclareerd* worden. De declaratie van een variabele vertelt de compiler dat er plaats in het geheugen gereserveerd moet worden om gegevens van het gegeven datatype in op te slaan.

Een nieuwe variabele declareren doe je met een instructie van deze vorm:

```
type naam_variabele;
```

■ Voorbeeld 2.1 — Declareren van een *integer* variabele.

```
// Declareer een nieuwe variabele van het type int met naam a
int a;
```

◂

Je kan bij het definiëren van een nieuwe variabele er ook meteen een waarde aan toekennen.

■ Voorbeeld 2.2 — Declareren van een *integer* variabele en een waarde toekennen.

```
/*
  Declareer een nieuwe variabele van het type int met naam a,
  en ken de waarde 42 toe
*/
int a = 42;
```

◂

Bij het kiezen van een variabelenaam moet je met de volgende regels rekening houden:

- namen moeten beginnen met een letter of een laag streepje (_);
- namen kunnen letters, cijfers en lage streepjes bevatten;
- namen zijn hoofdlettergevoelig;
- spaties of speciale karakters zijn niet toegelaten;
- C++-keywords (zoals `int`, `if`, ...) kunnen niet gebruikt worden als variabelenaam.

■ Voorbeeld 2.3 — Declareren en initialisatie van verschillende datatypes.

```
bool b = true;
char c = 'q';
int i = 42;
float f = 3.14;
double d = 3.14159265;
```

◂

2.2.2 Scope

De *scope* van een variabele is de plaats in het programma waar je een gedefinieerde variabele kan gebruiken. Je kan dit ook beschouwen als de levensduur van de variabele.

De scope van een variabele wordt bepaald door waar de variabele werd gedeclareerd. Dit kunnen we opdelen in 3 verschillende soorten plaatsen.

1. in een functie of blok: **lokale variabelen**,
2. in de definitie van een functie: **parameters**,
3. buiten alle functies: **globale variabelen**.

In het algemeen kan gesteld worden dat de scope van de variabelen beperkt is tot het blok, aangeduid met accolades {}, waarin de variabele gedefinieerd is.

■ **Voorbeeld 2.4 — Scope van variabelen.**

```
// Globale variabele g
int g = 12;

void setup() {
  // Lokale variabelen a en b
  int a = 42;

  /*
    Globale variabelen kunnen na declaratie
    overal gebruikt worden
  */
  int b = a + g;

  // Na declaratie moet het type van variabelen niet gespecifieerd worden
  g = b;
}

// Variabelen a en b kunnen buiten de functie setup() niet gebruikt worden
```

◂

2.2.3 Operatoren

Om de waarde in de variabelen aan te passen, maken we gebruik van *operatoren*. Hieronder geven we een overzicht van de belangrijkste operatoren in C++.

Rekenkundige operatoren

Rekenkundige operatoren worden gebruikt om veelgebruikte wiskundige operaties uit te voeren. In deze voorbeelden gebruiken we de variabelen $A = 10$ en $B = 3$.

Operator	Beschrijving	Voorbeeld
+	Optelling	`A + B` $\rightarrow 13$
-	Vermindering	`A - B` $\rightarrow 7$
*	Vermenigvuldiging	`A * B` $\rightarrow 30$
/	Deling	`A / B` $\rightarrow 2$
%	Modulus (rest na deling)	`A % B` $\rightarrow 1$
++	Verhoogt de waarde van de variabele met 1	`A++` $\Rightarrow$ `A` $\rightarrow 11$
--	Verlaagt de waarde van de variabele met 1	`A--` $\Rightarrow$ `A` $\rightarrow 9$

Toewijzende operatoren

De operator voor de toewijzing van een waarde aan een variabele is het gelijkheidsteken (=). Naast deze toewijzingsoperator bestaan er nog andere varianten, die kortere notaties zijn voor een rekenkundige operatie gevolgd door een toewijzing.

Operator	Voorbeeld	Equivalent
=	`A = 12`	`A = 12`
+=	`A += 2`	`A = A + 2`
-=	`A -= 3`	`A = A - 3`
*=	`A *= 4`	`A = A * 4`

Vergelijkende operatoren

Vergelijkende operatoren worden gebruikt om twee waardes met elkaar te vergelijken. Het resultaat van deze operatie is een Booleaanse waarde, `true` of `false`.

Operator	Beschrijving	Voorbeeld
==	Gelijk aan	`10 == 3` → `false`
!=	Niet gelijk aan	`10 != 3` → `true`
>	Groter dan	`10 > 10` → `false`
<	Kleiner dan	`5 < 10` → `true`
>=	Groter dan of gelijk aan	`10 >= 10` → `true`
<=	Kleiner dan of gelijk aan	`5 <= 10` → `true`

Logische operatoren

Logische operatoren worden gebruikt om meerdere Booleaanse waarden te combineren in een formule tot één waarheidswaarde.

Operator	Beschrijving	Voorbeeld
&&	Logische conjunctie (**AND**)	`(true && false)` → `false`
||	Logische disjunctie (**OR**)	`(true || false)` → `true`
!	Logische negatie (**NOT**)	`! true` → `false`

Voor meer voorbeelden van het gebruik van logische operatoren, kan je een kijkje nemen naar waarheidstabellen [3].

3. Controlestructuren

Controlestructuren bepalen de loop van het programma. Zo is het mogelijk om stukken code enkel in specifieke gevallen uit te voeren, of om een blok code meerdere keren te herhalen.

3.1 If-then-else

De eenvoudigste controlestructuur is het *if*-statement. Dit kan verder uitgebreid worden met een *else*-blok voor verdere controle over de uitvoeringsvolgorde.

3.1.1 If-then

Het if-statement wordt gebruikt om code uit te voeren die bedoeld is voor specifieke gevallen. Enkel wanneer aan de opgegeven conditie voldaan is, zullen de instructies in de body van het if-statement uitgevoerd worden. Als er niet aan de conditie voldaan wordt, zullen de instructies in de body overgeslagen worden.

Definitie 3.1.1 — If.

```
if (conditie) {
  /*
    instructies tussen deze accolades worden enkel
    uitgevoerd als de conditie tot true evalueert
  */
}
```

3.1.2 Else

Na een if-statement kan optioneel een else-blok geplaatst worden. De instructies in dit blok zullen enkel uitgevoerd worden als aan de conditie van het if-statement **niet** voldaan is. De code ziet er dan uit als volgt.

Definitie 3.1.2 — If-then-else.

```
if (conditie) {
  /*
    instructies in dit blok worden uitgevoerd
    als de conditie tot evalueert tot true
  */
} else {
  /*
  instructies in dit blok worden uitgevoerd
  als de conditie tot evalueert tot false
  */
}
```

3.1.3 Geneste if-statements

Als er meerdere condities zijn waarop getest moet worden, kan je gebruik maken van een genest if-statement. Hierbij plaats je in de else-blok een nieuw if-statement waarbij je de volgende conditie test. Dit ziet er uit als volgt.

■ **Voorbeeld 3.1 — Geneste if-statements.**

```
int x = 42;

if (x < 10) {
  x = x * 2;
} else {
  if (x > 50) {
    x = x - 25;
  } else {
    x++;
  }
}
```

◂

Omdat deze geneste if-statements snel onoverzichtelijk worden, kan deze code vereenvoudigd worden als volgt.

■ **Voorbeeld 3.2 — Else-if.**

```
int x = 42;

if (x < 10) {
  x = x * 2;
} else if (x > 50) {
  x = x - 25;
} else {
  x++;
}
```

◂

3.2 Switch

Als je een expressie wil testen op meerdere waardes, kan je in plaats van geneste if-statements ook gebruik maken van een *switch*-statement.

Definitie 3.2.1 — Switch.

```
switch (expressie) {
  case x:
    // instructies
    break;
  case y:
    // instructies
    break;
  default:
    // instructies
}
```

Hierbij wordt de expressie in de `switch` één keer geëvalueerd. Vervolgens wordt de waarde ervan vergeleken met de waarde in elke `case`. Als de waarden overeenkomen, dan wordt het bijhorende blok van instructies uitgevoerd.

Het `break`-statement zorgt ervoor dat, na een overeenkomstige `case` is gevonden, er niet nog op matches met andere `cases` gezocht moet worden. Het `default` keyword kan dan gebruikt worden om instructies uit te voeren in het geval dat er geen overeenkomstige `case` wordt gevonden.

■ **Voorbeeld 3.3 — Switch – dagen van de week.**

```
int dag = 5;
char c;

switch (dag) {
  case 1:
    c = 'M';
    break;
  case 2:
    c = 'D';
    break;
  case 3:
    c = 'W';
    break;
  case 4:
    c = 'd';
    break;
  case 5:
    c = 'V';
    break;
  case 6:
    c = 'Z';
    break;
```

```
  case 7:
    c = 'z';
    break;
  default:
    c = 'X';
}
```

◂

Oefening 3.1 — Switch – dagen van de week. Welk karakter is er na het uitvoeren van bovenstaande code (Voorbeeld 3.3) opgeslagen in variabele c? ◂

3.3 While-lus

Als je een stuk code meerdere keren achter elkaar wil laten uitvoeren, kan je gebruik maken van een lus. De `while`-lus bevat een conditie en een body met instructies. Zolang als de conditie waar is, worden de instructies in de body herhaald uitgevoerd.

Definitie 3.3.1 — While.

```
while (conditie) {
  // body van de lus
  instructie;
}
```

■ **Voorbeeld 3.4 — While.** Deze code berekent het kleinste veelvoud `vv` van het getal `x` dat groter is dan 24.

```
int x = 3;
int vv = 0;

while (vv <= 24) {
  vv += x;
}
```

◂

3.4 For-lus

Vaak zullen lussen voorkomen waarbij het aantal herhalingen vooraf vastligt. Deze lussen hebben dan volgende vorm.

■ **Voorbeeld 3.5**

```
int i = 0;

while (i < 10) {
  // instructies

  i++;
}
```

◂

Hierbij wordt voor het begin van de lus dan eerst de variabele `i` gedeclareerd, die in de lus gebruikt wordt om het aantal iteraties (herhalingen) te tellen. Hiervoor wordt op het einde van elke iteratie de waarde van `i` opgehoogd (`i++`). Dit soort lussen kan eenvoudiger worden geschreven met een `for`-lus.

Definitie 3.4.1 — For.

```
for (statement1; statement2; statement3) {
  // body, bevat een blok code dat telkens herhaald wordt
}
```

Waarbij:

statement1 één maal uitgevoerd wordt voor het begin van de lus. Vaak is dit de declaratie van een *loop-variabele*, bijvoorbeeld `int i = 0`.

statement2 de conditie van de lus voorstelt.

statement3 telkens na een iteratie van de lus wordt uitgevoerd. Vaak is dit een manipulatie van de loop-variabele, zoals `i++`.

De code uit Voorbeeld 3.5 kan dus als volgt geschreven worden met een `for`-lus.

■ **Voorbeeld 3.6**

```
for (int i = 0; i < 10; i++) {
  // instructies
}
```

◂

4. Functies en procedures

Functies en procedures zijn blokken code die je op verschillende plaatsen in je code kan gebruiken. In plaats van telkens dezelfde code neer te schrijven, kan je beter de gemeenschappelijke code in een functie plaatsen. Dit verhoogt ook de leesbaarheid van de code.

4.1 Functies definiëren

De definitie van een functie is opgebouwd zoals hieronder beschreven.

Definitie 4.1.1 — Functie.

```
type functienaam(argumenten) {
  // hier komt de gemeenschappelijke code

  return resultaat;
}
```

De definitie van een functie bestaat uit de hoofding en de body. In de hoofding wordt eerst het type van de functie genoteerd. Hierna volgt de naam van de functie, en tot slot worden de argumenten van de functie tussen de haakjes geplaatst. Daarna volgt de body van de functie binnen de accolades.

4.1.1 Type

Het type van de functie, is het type van de waarde die de functie teruggeeft. Een functie wordt vanuit de code aangeroepen, waarna de code in de body van de functie wordt uitgevoerd. Het resultaat van deze berekeningen wordt vervolgens teruggegeven aan de plaats waarop deze functie-aanroep plaatsvond. Daar kan het resultaat dan verder gebruikt worden.

De primitieve datatypes uit Definitie 2.1.1 kunnen gebruikt worden als type van een functie. Het is ook mogelijk dat een functie geen resultaat heeft. Zo'n functie noemen we een procedure. Als *return type* gebruiken we dan `void`, het lege type.

4.1.2 Argumenten

Via de *argumenten* of *parameters* kan informatie doorgegeven aan de functie. Deze parameters gedragen zich als variabelen binnen in de body van de functie. De verschillende parameters worden in de hoofding van elkaar gescheiden met een komma.

■ **Voorbeeld 4.1 — Functie – optelling.**

```
int optelling(int a, int b, int c) {
  int som = a + b + c;
  return som;
}
```

◂

■ **Voorbeeld 4.2 — Functie – minimum.**

```
int minimum(int a, int b) {
  if (a < b) {
    return a;
  } else {
    return b;
  }
}
```

◂

4.2 Functies aanroepen

Nadat een functie is gedefinieerd, kan deze functie aangeroepen worden op andere plaatsen in de code. Het aanroepen van een functie doe je als volgt:

```
functienaam(argumenten);
```

De functie `minimum()` uit Voorbeeld 4.2 kan dus op de volgende manier aangeroepen worden.

■ **Voorbeeld 4.3 — Aanroepen van de functie *minimum()*.**

```
int x = 42;
int y = 13;

int resultaat = minimum(x, y);
```

◂

5. Arrays en lijsten (ADVANCED)

6. Klassen en objecten (Advanced)

II Arduboy

7. Arduino

7.1 Programmastructuur

Elk Arduino-programma heeft dezelfde structuur. De belangrijkste kenmerken die steeds terugkeren zijn de globale variabelen, de `setup()` procedure en de `loop()` procedure. In de volgende secties bespreken we deze in meer detail.

Definitie 7.1.1 — Arduino programmastructuur.

```
// Declaratie van globale variabelen

void setup() {

}

void loop() {

}
```

7.1.1 Globale variabelen

Bovenaan het Arduino-programma worden de globale variabelen gedeclareerd. Deze globale variabelen kunnen daarna overal in het programma gebruikt worden, zowel in de `setup()` en `loop()` functie.

7.1.2 De `setup()` procedure

De `setup()` procedure wordt éénmaal uitgevoerd bij het opstarten van het programma.

In de `setup()` procedure ga je waarschijnlijk de waarde van verscheidene variabelen initialiseren. Het is belangrijk dat je deze variabelen declareert **buiten** de `setup()` functie. Anders zijn deze variabelen lokaal, en kunnen ze niet buiten de scope van de `setup()` functie gebruikt worden.

7.1.3 De `loop()` procedure

De `loop()` procedure doet precies wat de naam doet vermoeden. De code in deze procedure wordt steeds opnieuw uitgevoerd. Hier wordt de eigenlijk code van het Arduino-programma geschreven. De herhaling blijft duren tot de stroom van de Arduino wordt losgekoppeld of op de reset-knop wordt geduwd.

7.2 Importeren van libraries

Bij het programmeren voor Arduino is het veel voorkomend om gebruik te maken van *libraries*. Libraries bevatten functies die door andere ontwikkelaars zijn geschreven. Ze hebben als doel om de complexiteit van jouw code te verlagen door operaties te implementeren op een lager niveau. Hierdoor moet je bijvoorbeeld niet handmatig alle communicatie met een display programmeren, maar gebruik je hiervoor de functionaliteit van de library.

Het importeren van een library in je code doe je op de volgende manier:

```
#include <Library.h>
```

8. Arduboy

Part One

9. Text Chapter

9.1 Paragraphs of Text

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi.

Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

9.2 Citation

This statement requires citation [1]; this one is more specific [2, pagina 162].

9.3 Lists

Lists are useful to present information in a concise and/or ordered way[1].

9.3.1 Numbered List

1. The first item
2. The second item
3. The third item

9.3.2 Bullet Points

- The first item
- The second item
- The third item

9.3.3 Descriptions and Definitions

Name Description
Word Definition
Comment Elaboration

[1] Footnote example...

10. In-text Elements

10.1 Theorems

This is an example of theorems.

10.1.1 Several equations

This is a theorem consisting of several equations.

Theorem 10.1.1 — Name of the theorem. In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$\big|||\mathbf{x}|| - ||\mathbf{y}|||\big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{10.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{10.2}$$

10.1.2 Single Line

This is a theorem consisting of just one line.

Theorem 10.1.2 A set $\mathcal{D}(G)$ in dense in $L^2(G)$, $|\cdot|_0$.

10.2 Definitions

This is an example of a definition. A definition could be mathematical or it could define a concept.

Definitie 10.2.1 — Definition name. Given a vector space E, a norm on E is an application,

denoted $||\cdot||$, E in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \;\Rightarrow\; \mathbf{x} = \mathbf{0} \tag{10.3}$$
$$||\lambda\mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \tag{10.4}$$
$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \tag{10.5}$$

10.3 Notations

Notation 10.1. *Given an open subset G of $\mathbb{R}^n$, the set of functions φ are:*

1. *Bounded support G;*
2. *Infinitely differentiable;*

a vector space is denoted by $\mathcal{D}(G)$.

10.4 Remarks

This is an example of a remark.

The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

10.5 Corollaries

This is an example of a corollary.

Gevolg 10.5.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

10.6 Propositions

This is an example of propositions.

10.6.1 Several equations

Propositie 10.6.1 — Proposition name. It has the properties:

$$\big|\,||\mathbf{x}|| - ||\mathbf{y}||\,\big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{10.6}$$
$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{10.7}$$

10.6.2 Single Line

Propositie 10.6.2 Let $f, g \in L^2(G)$; if $\forall\varphi \in \mathcal{D}(G)$, $(f, \varphi)_0 = (g, \varphi)_0$ then $f = g$.

10.7 Examples

This is an example of examples.

10.7.1 Equation and Text

■ **Voorbeeld 10.1** Let $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ and denoted by: $x^0 = (1,1)$; consider the function:

$$f(x) = \begin{cases} \mathrm{e}^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \tag{10.8}$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \epsilon\}$ for all $\epsilon \in]0\,;5/2 - \sqrt{2}[$. ◂

10.7.2 Paragraph of Text

■ **Voorbeeld 10.2 — Example name.** Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. ◂

10.8 Exercises

This is an example of an exercise.

Oefening 10.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ◂

10.9 Problems

Probleem 10.1 What is the average airspeed velocity of an unladen swallow?

10.10 Vocabulary

Define a word to improve a students' vocabulary.
Vocabulaire 10.1 — Word. Definition of word.

IV

Part Two

11. Presenting Information

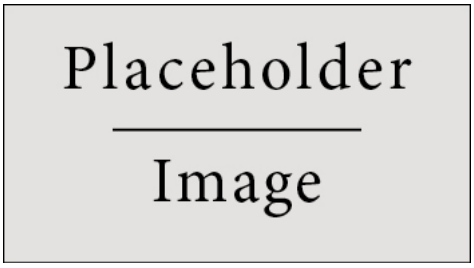
11.1 Table

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Tabel 11.1: Table caption

Referencing Table 11.1 in-text automatically.

11.2 Figure



Figuur 11.1: Figure caption

Referencing Figure 11.1 in-text automatically.

Bibliografie

Artikels

[1] James Smith. “Article title”. In: 14.6 (mrt 2013), pagina’s 1–8 (zie pagina 36).

Boeken

[2] John Smith. *Book title*. 1ste editie. Deel 3. 2. City: Publisher, jan 2012, pagina’s 123–200 (zie pagina 36).

Websites

[3] Wikipedia. *Waarheidstabel*. `https://nl.wikipedia.org/wiki/Waarheidstabel` (zie pagina 14).

Index

R

T

V