Reader IoT Software*

Chris van Uffelen IoT Team

Voorjaar 2024

1 Inleiding

1.1 Introductie IoT-Software

In deze cursus leer je oplossingen te bouwen en testen waarbij meerdere componenten met elkaar communiceren.

In meerdere stappen wordt je voorbereid op het maken van een geïntegreerde oplossing waarin je meerdere stations gegevens laat verzamelen. De informatie in deze stations wordt verzameld op een centrale gateway. Vanaf je laptop kan je deze gateway benaderen om actuele gegevens te tonen.

Om deze geïntegreerde oplossing te maken moet je tot het volgende in staat zijn:

- Je gaat leren applicaties in C te schrijven. In C kan je applicaties schrijven die draaien op de Raspberry Pi en op de Arduino.
- Je gaat leren applicaties in Python te schrijven. Hiermee kan je servers bouwen die op de Raspberry draaien.
- Je gaat veel leren over protocollen, waar we met name ingaan op HTTP.
- Je gaat leren Rest servers te bouwen die met behulp van HTTP communicatie tussen machines mogelijk maken.

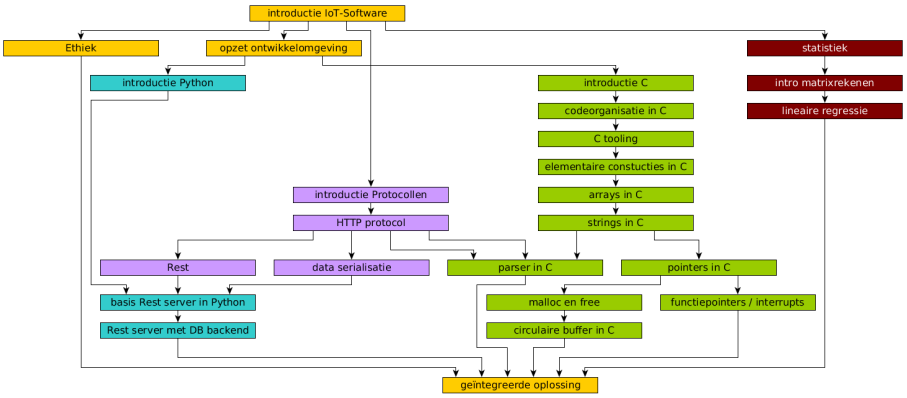
Bij dit alles hebben we aandacht voor kwaliteit van werken.

- Je gaat leren om de kwaliteit van je applicatie te waarborgen.
- Je gaat nadenken over de vraag wanneer een oplossing mogelijk is of het ook *verstandig* is.

Binnen het profiel ESD kom je in ieder semester in aanraking met een andere vorm van projectorganisatie. In het semester IoT is dit *prototypen*. In het kort is het doel van *prototyping* het vinden van de beste oplossing voor een probleem binnen gegeven kaders. Prototyping krijgt volop de aandacht in de projectfase.

In deze cursus komen veel verschillende onderwerpen aan de orde. Fig. 1 geeft de *work break-down structure* met daarin de onderlinge relatie tussen deze onderwerpen weer.

*ver. r1.6.0



Figuur 1: Onderwerpen in deze cursus.

Ieder blokje in dit schema komt overeen met een hoofdstuk in deze reader. Zoals je ziet is de titel van dit hoofdstuk gelijk aan het bovenste blokje. Wanneer je begrijpt wat het doel is van deze course en hoe we dit doel gaan bereiken ben je klaar voor ieder van de volgende onderwerpen in het schema.

Je ziet, er is niet één lijn die je van deze introductie naar het einddoel brengt, het is een netwerk van paden die je zal moeten afleggen. De intentie is dat je in ieder onderwerp werkt aan een opdracht waarmee je voor jezelf kan testen of je het onderwerp voldoende beheerst.

Voor ieder onderwerp kan je het volgende terugvinden:

- Doel van het onderwerp.
- Opdrachten voor het onderwerp.
- Toelichting bij het onderwerp om je verder te helpen met het uitwerken van de opdracht.

  De toelichting maakt deel uit van de stof die je ook voor de toets moet kennen, ook als er niet expliciet aandacht voor is geweest tijdens de les.
- In een aantal gevallen is ook achtergrondsinformatie toegevoegd.

  Deze informatie is toegevoegd in de hoop dat deze nuttig is, maar het maakt geen deel uit van de toets.

1.2 Opzet ontwikkelomgeving

1.2.1 Doel

Je hebt je Raspberry Pi[1] zo ingericht dat je in staat bent:

- C code te compileren

[1]Deze reader gaat ervan uit dat je een Raspberry Pi hebt. Door beperkte beschikbaarheid in hoge prijzen proberen we sinds semester 2 van het schooljaar 2022, 2023 het zonder de Raspberry Pi te doen. In het algemeen gebruik je dat je laptop (Linux, MacOSX of WSL2) in plaats van de Raspberry Pi.

- Python code uit te voeren
- je Raspberry Pi kan communiceren met een Arduino
- je code in te checken op de subversion server

Om dit doel te kunnen halen is het volgende nodig:

- Je kan vanaf je laptop je Raspberry Pi benaderen.
- Je kan vanaf je Raspberry Pi verbinding maken met de Ethernet Shield op je Arduino.
- Je kan vanaf je Raspberry Pi het internet bereiken.

1.2.2 Opdracht

In deze opdracht laat je je Arduino en Raspberry Pi met elkaar communiceren. Het gaat erom de communicatie werkend te krijgen, je hoeft code nog niet tot in detail te snappen.

Zorg dat je in je *working copy* van je subversion repository voor de volgende bestandsstructuur:

```
.
+- branches
+- tags
+- trunk
   +- techniek
   +- software
      +- <voornaam>                  # je eigen voornaam
        +- 1_2_opzetontwikkelomgeving # naam opdracht
           +- c_client
           |  +- client.c             # C client code
           +- py_client
           |  +- client.py            # Python client code
           +- server
              +- server.ino           # Arduino C code
```

In bestand `server.ino` komt een minimale webserver te draaien. Gebruik hiervoor de volgende code (Mellis z.d.):

```
#include <Ethernet.h>

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
byte IP[] = {192, 168, 1, 51};

EthernetServer server(80);

void setup() {
  Serial.begin(9600);

  Ethernet.begin(mac, IP);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}
```

```
void loop() {
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println(
              "Connection: close"); // the connection will
                                   // be closed after
                                   // completion of the
                                   // response
          client.println(
              "Refresh: 5"); // refresh the page
                             // automatically every 5 sec
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          // output the value of each analog input pin
          for (int analogChannel = 0; analogChannel < 6;
               analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        } else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }

    delay(1);
    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}
```

Zorg dat deze code op je Arduino draait. Test de werking in je browser en met `curl`[2]:

```
$ curl -i 192.168.1.11
HTTP/1.1 200 OK
Content-Type: text/html
Connection: close
Refresh: 5

<!DOCTYPE HTML>
<html>
analog input 0 is 264<br />
analog input 1 is 234<br />
analog input 2 is 220<br />
analog input 3 is 232<br />
analog input 4 is 257<br />
analog input 5 is 357<br />
</html>
```

De applicatie `curl` is een programma waarmee je op de *command line* een `URL` kan benaderen. De default is dat `curl` de *server* op poort 80 (HTTP) probeert te benaderen.

In bestand `client.py` staat een stukje Pythoncode dat een verzoek doet en het resultaat ervan print (Foord z.d.).

```
import urllib.request


def run():
    url = 'http://192.168.1.51/'
    with urllib.request.urlopen(url) as response:
        print(response.getcode())
        print(response.geturl())
        print(response.info())
        print(response.read().decode('UTF-8'))


if __name__ == "__main__":
    run()
```

Deze code in `client.py` kan je op de volgende manier uitvoeren:

```
$ python client.py
```

Wanneer dit niet werkt, kan het zijn dat je expliciet moet aangeven dat het gaat om versie 3 van python:

```
$ python3 client.py
```

In bestand `client.c` wordt hetzelfde verzoek in C uitgeprogrammeerd. Je ziet dat dit een behoorlijke lap code is. Deze code is *geen examenstof* (Deze code is zwaar gebaseerd op Jeremiah z.d.).

```
#include <netdb.h>      // struct hostent, gethostbyname
#include <netinet/in.h> // struct sockaddr_in, struct sockaddr
```

[2]De `$` moet je *niet* intypen! Het is de *prompt* waarachter je een opdracht kan typen.

```
#include <stdio.h>      // printf, sprintf
#include <stdlib.h>     // exit
#include <string.h>     // memcpy, memset
#include <sys/socket.h> // socket, connect
#include <unistd.h>     // read, write, close

// https://github.com/Caltech-IPAC/Montage/issues/5
#define h_addr h_addr_list[0]

void error(const char* msg) {
  perror(msg);
  exit(0);
}

int main(int argc, char* argv[]) {
  /* first what are we going to send and where are we going
   * to send it? */
  int portno = 80;
  char* host = "192.168.1.51";
  char* message_fmt = "%s %s HTTP/1.0\r\nHost: %s\r\n\r\n";

  struct hostent* server;
  struct sockaddr_in serv_addr;
  int sockfd, bytes, sent, received, total;
  char message[1024], response[4096];

  if (argc < 3) {
    puts("Parameters: <method> <request-URI>, e.g.: GET /");
    exit(0);
  }

  /* fill in the parameters */
  sprintf(message, message_fmt, argv[1], argv[2], host);
  printf("Request:\n%s\n", message);

  /* create the socket */
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
    error("ERROR opening socket");

  /* lookup the ip address */
  server = gethostbyname(host);
  if (server == NULL)
    error("ERROR, no such host");

  /* fill in the structure */
  memset(&serv_addr, 0, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(portno);
  memcpy(&serv_addr.sin_addr.s_addr, server->h_addr,
         server->h_length);

  /* connect the socket */
  if (connect(sockfd, (struct sockaddr*)&serv_addr,
```

```
            sizeof(serv_addr)) < 0)
    error("ERROR connecting");

  /* send the request */
  total = strlen(message);
  sent = 0;
  do {
    bytes = write(sockfd, message + sent, total - sent);
    if (bytes < 0)
      error("ERROR writing message to socket");
    if (bytes == 0)
      break;
    sent += bytes;
  } while (sent < total);

  /* receive the response */
  memset(response, 0, sizeof(response));
  total = sizeof(response) - 1;
  received = 0;
  do {
    bytes =
        read(sockfd, response + received, total - received);
    if (bytes < 0)
      error("ERROR reading response from socket");
    if (bytes == 0)
      break;
    received += bytes;
  } while (received < total);

  if (received == total)
    error("ERROR storing complete response from socket");

  /* close the socket */
  close(sockfd);

  /* process response */
  printf("Response:\n%s\n", response);

  return 0;
}
```

Kopieer deze code en compileer deze:

```
$ gcc -Wall -std=c99 -o client client.c
$ ./client GET /
```

Commit je code wanneer je de communicatie werkend hebt.

Demonstreer je applicatie aan de docent.

1.2.3 Toelichting

- Dit hoofdstuk gaat ervan uit dat je opzet iot netwerk met succes hebt doorlopen.
- De code voor de Arduino compileer je op je laptop met de Arduino IDE.

- Je Raspberri Pi kan je vanuit je laptop benaderen met ssh.

1.2.3.1 Subversion

- Uitgebreide informatie over Subversion vind je in het Svn Book.
- Er zijn veel *svn cheat sheets* beschikbaar.

1.2.3.2 Installatie van software op je Raspberry Pi Controleer of je op je Raspberry Pi de volgende applicaties hebt draaien: `python3` en `svn`. Een manier om te kijken of een applicatie beschikbaar is is de volgende:[3]

```
$ which python3
```

Wanneer je een antwoord krijgt, dan weet je meteen waar het programma staat welke je uitvoert. Krijg je geen antwoord, dan heb je òf te maken met een typefout, òf met een programma dat je nog installeren moet.

Applicaties op je Raspberry Pi installeer je in het normaal gesproken met het volgende commando:

```
$ sudo apt-get install <applicatie>
```

Op de plaats van `<applicatie>` komt in de naam van het te installeren pakket te staan. Vaak is deze naam gelijk aan de naam van de applicatie. Voor `svn` heb je dan minder succes, deze instatlleer je met:

```
$ sudo apt-get install subversion
```

Een zoekopdracht naar de juiste term helpt je meestal ook.

2 Protocollen

2.1 Introductie Protocollen

2.1.1 Lesdoelen

- Beschrijf wat bedoeld wordt met een protocol
- Noem de belangrijkste eigenschappen van een protocol
- Beschrijf een grammatica in *augmented Backus-Naur format*

2.1.2 Opdrachten

2.1.2.1 Geen protocol Beschrijf een manier waarop twee computers met elkaar kunnen communiceren wanneer er geen onderling protocol is afgesproken.

2.1.2.2 Formeel en volledig Wij implementeren in deze cursus het HTTP protocol.

- Waar staat dit protocol beschreven?

[3] In de tijd dat versie 2 en 3 veel naast elkaar gebruikt worden was `python3` de manier om expliciet versie 3 te gebruiken. Waarschijnlijk is `python` op je systeem ondertussen versie 3 *by default*.

- Wat kan er mis gaan wanneer de beschrijving niet *formeel* is?
- Wat kan er mis gaan wanneer deze beschrijving niet *volledig* is?

2.1.2.3 Gebaseerd op mijn favoriete spel van vroeger, Monkey Island ('Monkey Island (series)' z.d.), wordt het spel bestuurd door instructies in te typen, bijvoorbeeld

```
pickup stone
goto bar
look-at sword
look-at map with glass
put map into bag
give bag to friend
look-at bar with stone
```

En object is dan iets als `sword`, `stone`, `map`, `bag`, en person is dan iets als `pirate`, `friend`.

Schrijf een Backus-Naur grammatica voor `command` op basis waarvan gecontroleerd kan worden of een commando geldig is (los van de vraag of een `stone` wel in een `sword` gestopt kan worden).

Je mag het volgende als uitgangspunt nemen:

```
SP = " " [*SP]
ENTER = "\r\n" | SP "\r\n"

command = pickup_command / goto_command / lookat_command /
          put_command / give_command
```

2.1.3 Toelichting

2.1.3.1 Augmented Backus–Naur form (ABNF) Voordat we in detail gaan kijken naar het HTTP-protocol, doen we eerst een uitstapje naar ABNF ('Augmented BNF for Syntax Specifications: ABNF' 2008, RFC 5234). ABNF (een afkorting van Augmented Backus-Naur form) is een taal waarmee je weer andere talen kunt beschrijven. Zo'n taal, waarmee je een andere taal kunt beschrijven, heet ook wel een *meta-taal*.

ABNF is vooral bedoeld om communicatieprotocollen te beschrijven en wordt veel voor internetprotocollen gebruikt. Daarnaast wordt ABNF gebruikt om de grammatica van programmeertalen te beschrijven, vervolgens kunnen parsers, interpreters of compilers gebouwd worden kunnen checken of jouw code voldoet aan de standaardregels.

Wil je het HTTP-protocol in detail kunnen begrijpen, dan zul je ABNF moeten kunnen lezen.

We geven hier voorbeelden van een aantal veelgebruikte onderdelen van ABNF.

De basis is de *rule*, deze koppelt een naam aan een definitie.

```
BIT: "0" / "1"
```

Deze *rule* definieert de term `bit` welke links van de `:` staat. Aan de rechterkant staat dan eis waaraan een `bit` moet voldoen. In dit geval staat er dat een `bit`

een "0" of een "1" is. De rechterkant is overigens een voorbeeld van *alternatives* waarbij meerdere opties gegeven kunnen worden.

Met *bits* kunnen we *bytes* bouwen, bijvoorbeeld:[4]

```
BYTE: "B" BIT BIT BIT BIT BIT BIT BIT BIT
```

Hierboven vind je een voorbeeld van een *concatenation* waarin meerdere onderdelen achter elkaar gezet worden. Op grond van bovenstaande regel representeert `B11001100` een *byte* maar `11000011` niet.

Bovenstaande is natuurlijk onhandig. We kunnen handiger gebruik maken van een van de vormen van *repitition* (zie de RFC voor alle details), bijvoorbeeld:

```
BYTE: "B" 8BIT
```

Stel dat we nu ook een *binary number* willen definiëren met een variabel aantal `BIT`s, dan kan dat met

```
BINNUMBER: 1*BIT
```

Volgens bovenstaande definitie heeft een *binary number* ten minste één `BIT`.

Voor de repitions "`*1`" (dat is, ten hoogste één, en daarmee, nul of één), is een speciale vorm. Stel dat we de `B` in `BYTE` optioneel willen maken, dan kunnen we schrijven met behulp van een *optional sequence*:[5]

```
BYTE: ["B"] 8BIT
```

We kunnen ABNF-definities van commentaar voorzien:

```
; een byte is een reeks van 8 bits
BYTE: 8BIT
```

In bovenstaande hebben we veel voorkomende elementen uit de ABNF-notatie gehad. Er zijn nog een aantal constructies. Dus, als je op zoek bent naar oplossingen voor problemen, sla er dan de eerdergenoemde referentie nog eens op na.

ABNF-notatie wordt gebruikt in meerdere RFC's om op een formele manier de syntax van een protocol vast te leggen. Wij zullen het tegenkomen bij de definitie van de syntax van HTTP, Sec. 2.2.

2.2 HTTP Protocol

2.2.1 Doel

Na dit hoofdstuk moet je het volgende kunnen:

- Je kunt een valide HTTP *request* schrijven
- Je kunt een valide HTTP *response* schrijven
- Je kent de betekenis van de verschillende HTTP methods.
- Je weet de betekenis van veel voorkomende *status codes* in de *status line* van een *response message*.

[4] De voorbeelden in dit gedeelte zijn slechts ter illustratie. De wijze waarop hier een *byte* wordt gedefinieerd is helemaal niet bedoeld als "de" definitie van een *byte*!

[5] En je snapt nu waarschijnlijk ook dat `4*8` een aantal tussen vier en acht representeert.

- Je weet waar je informatie over het HTTP protocol kan vinden.
- Je kent relevante verschillen tussen `HTTP/1.0` en `HTTP/1.1`.

2.2.2 Opdracht

2.2.2.1 HTTP messages Er zijn twee soorten HTTP (*HypertText Transfer Protocol*) berichten, een *request* en een *response*. Zoek aan de hand van RFC 7230 ('Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing' 2014) uit hoe beide soorten er syntactisch correct uitzien.

Maak hiervan aantekeningen in je *logboek*.

2.2.2.2 Stroomschema Maak een beslisboom over welke `status-code` je terug moet krijgen in veelvoorkomende gevallen. De volgende statuscodes moeten worden afgedekt: 200, 201, 400, 401, 404, 405, 411, 500, en 501.

De eerste regel in deze beslisboom zou er als volgt uit kunnen zien:

- Het *request* heeft een fout in de syntax:

  `400 Bad Request`

2.2.2.3 Test je stroomschema Test je stroomschema met `telnet` door een bescheiden aantal requests en uit te proberen op http://ffel.github.io:[6]

Hier zie je een voorbeeld. Direct erna worden de verschillende stappen uitgelegd.[7]

```
$ telnet ffel.github.io 80
Trying 185.199.110.153...
Connected to ffel.github.io.
Escape character is '^]'.
GET / HTTP/1.1
Connection: close
Host: ffel.github.io

HTTP/1.1 200 OK
Server: GitHub.com
Content-Type: text/html; charset=utf-8

<!doctype html>
<html lang="en">
<!-- loads of HTML -->
</html>
Connection closed by foreign host.
```

Na de opdracht

```
$ telnet ffel.github.io 80
```

laat `telnet` zien dat er een verbinding gemaakt is:

[6] Het wordt steeds moeilijker een geschikte HTTP server te vinden, meer en meer wordt alleen HTTPS aangeboden. HTTPS is geen onderdeel van deze cursus.

[7] Er is heel wat uit het *response* geknipt om het overzicht te bewaren.

```
Trying 185.199.110.153...
Connected to ffel.github.io.
Escape character is '^]'.
```

Daarna type je het volgende *http request* met de naam van de website waar je verbinding wil maken.[8]

```
GET / HTTP/1.1<crlf>
Connection: close<crlf>
Host: ffel.github.io<crlf>
<crlf>
```

Je sluit af met tweemaal `<enter>`[9]. Waarom? Vervolgens zie je de reactie.

Kijk of de server van http://ffel.github.io ook de andere statuscodes teruggeeft wanneer je wijzigingen aan het request geeft. Ik ga ervan uit dat je je netjes gedraagt!

Leg je bevindingen vast in je *logboek*.

Ook is het handig om gebruik te maken van `netcat` in plaats van telnet: `netcat` leest het request uit een bestand, en je kan nu eenvoudig het bestand aanpassen. Wanneer je je request opslaat in een bestand met de naam "`get`", dan kan je `netcat` als volgt uitvoeren:

```
$ netcat ffel.github.io 80 < get
```

Het "`<`"-teken in de opdracht laat de inhoud van bestand `get` lezen en geeft deze aan de opdracht mee alsof je de betreffende tekst zelf in had getypt.

2.2.3 Toelichting

In dit hoofdstuk wordt ingegaan op hoe HTTP-messages (request/response) in detail zijn opgebouwd. Dit is geen uitputtende beschrijving, maar een inleiding in het lezen van de echte definitie RFC7230 . Uiteindelijk wordt er van je als HBO-ICT-student verwacht dat je dergelijke documentie kunt lezen en begrijpt hoe je de inhoud moet toepassen in software. Alle informatie in deze paragraaf komt uit de standaard ('Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing' 2014) en bijbehorende standaarden.

Een HTTP-message heeft altijd de volgende structuur (dus zowel een request- als responsebericht):

```
HTTP-message   = start-line
                 *( header-field CRLF )
                 CRLF
                 [ message-body ]
```

Elke regel wordt afgesloten met een `CRLF` (carriage return en line-feed, dit zijn twee karakters achter elkaar, namelijk eerst `0x0D` en dan `0x0A`.

[8] Github serveert vele websites vanaf dat IP adres, dit wordt ondersteunt in `HTTP/1.0` en nog niet in `HTTP/1.0`.

[9] aangegeven als *carriage return*, *line feed* (`<crlf>`).

2.2.3.1 start-line De eerste regel, de `start-line` is eigenlijk het enige echte verschil tussen een request- en responsebericht, de rest van het bericht is syntactisch gelijk. Er zijn twee voorkomens van een `start-line`, namelijk een `request-line` (voor requests) en een `status-line` (voor responses). Een start-line is als volgt gedefinieerd:

```
start-line     = request-line / status-line
```

2.2.3.2 request-line Een request-line begint altijd met een `method` token, gevolgd door een enkele spatie, de `request-target`, weer een spatie, de `HTTP-version` en aan het einde altijd een `CRLF`. De request-line is als volgt gedefinieerd:

```
request-line = method SP request-target SP HTTP-version CRLF
```

Zoals je ziet worden de onderdelen van de request-line gescheiden door spaties. Software die een request verwerken splitsen dan ook vaak de eerste regel op de spaties (en dat mag ook, omdat de drie delen van de request-line geen spaties mogen bevatten). In de definitie van de request-line staat ook beschreven wat er moet gebeuren mochten er fouten staan in het bericht (als het bericht niet voldoet aan de specificatie).

2.2.3.2.1 method Het eerste onderdeel van de request-line is de `method`. Een method is gedefinieerd als `token` (een token bestaat uit een samenstelling van 1 of meerdere karakters, maar geen spatie).

In het HTTP-protocol zijn een aantal standaard methods gedefinieerd. Het is mogelijk dat deze lijst in de toekomst uitgebreid wordt. Voor een aantal protocollen die een uitbreiding zijn op het HTTP-protocol zijn er al andere methods gedefinieerd ('Hypertext Transfer Protocol (HTTP) Method Registry' 2017). Hieronder de lijst van methods (bron: 'Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing' 2014, 22):

GET: *Transfer a current representation of the target resource.*
HEAD: *Same as GET, but only transfer the status line and header section.*
POST: *Perform resource-specific processing on the request payload.*
PUT: *Replace all current representations of the targetresource with the request payload.*
DELETE: *Remove all current representations of the target resource.*
CONNECT: *Establish a tunnel to the server identified by the target resource.*
OPTIONS: *Describe the communication options for the target resource.*
TRACE: *Perform a message loop-back test along the path to the target resource.*

Een server moet eigenlijk altijd de GET en HEAD methods ondersteunen.

2.2.3.2.2 request-target De `request-target` identificeert de *resource* waarop de request wordt uitgevoerd. Hiermee wordt bedoeld dat de request-target een string is waarmee aangegeven wordt op welk onderdeel van bijv. een website de request wordt uitgevoerd. Bij een GET-request is dat bijvoorbeeld de webpagina die opgehaald moet worden of bij een POST-request de PHP-pagina die de opgestuurde gegevens moet verwerken.

Een request-target is als volgt gedefinieerd:

```
request-target = origin-form
               / absolute-form
               / authority-form
               / asterisk-form
```

De opbouw van een request-target kan best complex zijn, voor deze course is het voldoende als je er van uitgaat dat de request-target er uitziet als een url zonder (HTTP-)protocol en domeinnaam, dus zoiets als `/web/pagina.html`.

2.2.3.2.3 HTTP-version Met de `HTTP-version` wordt aangegeven aan welke versie van het protocol wordt voldaan. In deze course gaan we niet in op HTTP/2, maar alleen op HTTP/1.0 en HTTP/1.1.

In een eerder hoofdstuk is al kort iets toegelicht over een paar verschillen tussen HTTP/1.0 en HTTP/1.1. Voor deze course zijn de belangrijkste verschillen:

- HTTP/1.0 sluit standaard de connectie na het versturen van een response.

  Voor elke request-responsecyclus wordt dus steeds opnieuw een tcp-connectie geopend. Als je dat niet wilt (bijv. uit performance-overwegingen), moet je expliciet de header `Connection: keep-alive` meesturen. Bij een HTTP/1.1 request blijft connectie standaard open en als je dat niet wilt, moet je de header `Connection: close` meesturen.

- HTTP/1.0 ondersteunt maar één host per IP-nummer.

  In het HTTP-protocol was het in eerste instantie niet mogelijk om meerdere websites aan één IP-adres te koppelen, het zogenaamde *shared hosting*. Vanaf HTTP/1.1 is dat wel mogelijk, maar dan moet je wel de header `Host: <website>` meegeven.

Op een Arduino is het aan te bevelen om de connectie altijd te sluiten nadat de response (op een request) is verwerkt of als je een response terugstuurt. Dat maakt het programma veel eenvoudiger te implementeren (en ook een stuk robuster). Daarnaast kun je het beste ook aangeven welke host je bedoeld, ook al is er maar één website aan dat IP-nummer gekoppeld (om latere problemen met shared hosting te voorkomen).

Dus altijd de volgende headers meesturen:

```
Host: <the host>
Connection: close
```

2.2.3.3 status-line De eerste regel van een response-bericht (van een server) is de `status-line`. Deze status-line begint met de `protocolversie`, gevolgd door een spatie, dan een driecijferige `status code`, weer een spatie, daarna een stukje `tekst` en de status-line wordt altijd afgesloten met `CRLF`.

```
status-line = HTTP-version SP status-code SP reason-phrase CRLF
```

De status-code is een driecijferige code die aangeeft wat het resultaat is van de request. Hieronder gaan we er wat verder op in. De `reason-phrase` is een stukje tekst dat beschrijft wat er bedoeld is met de status-code. Dit stukje tekst moet door cliënts genegeerd worden, in de HTTP/2-versie is deze tekst zelfs komen te vervallen ('Hypertext Transfer Protocol Version 2 (HTTP/2)' 2015).

2.2.3.3.1 status-code De status-code is een belangrijk onderdeel van de communicatie bij HTTP, daarmee wordt namelijk aangegeven wat het resultaat is na het verwerken van de request. Er zijn een aantal standaard ranges gedefinieerd ('Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content' 2014):

- `1xx`, Informational: *The request was received, continuing process*
- `2xx`, Successful: *The request was successfully received, understood, and accepted*
- `3xx`, Redirection: *Further action needs to be taken in order to complete the request*
- `4xx`, Client Error: *The request contains bad syntax or cannot be fulfilled*
- `5xx`, Server Error: *The server failed to fulfill an apparently valid request*

Een aantal response codes moet elke ontwikkelaar kennen ('HTTP Status Codes' 2017). De genoenmde hoofstukken zijn te vinden in de bron.

- `200 OK`: sectie 6.3.1
- `201 Created`: sectie 6.3.2
- `204 No Content`: sectie 6.3.5
- `304 Not Modified`: sectie 4.1
- `400 Bad Request`: sectie 6.5.1
- `401 Unauthorized`: sectie 3.1
- `403 Forbidden`: sectie 6.5.3
- `404 Not Found`: sectie 6.5.4
- `409 Conflict`: sectie 6.5.8
- `500 Internal Server Error`: sectie 6.6.1

Een meer complete lijst van de meeste codes vind je deze bron ('Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content' 2014, 49). Deze lijst bevat niet alle als standaard gedefinieerde status-codes, want een aantal van die codes zijn bedoeld voor andere protocollen dan HTTP, op deze pagina vind je de complete lijst ('Hypertext Transfer Protocol (HTTP) Status Code Registry' 2017).

Het kiezen van de goede status-code, om terug te geven bij een response, kan veel onnodige overhead schelen. Bijvoorbeeld, als er data wordt opgestuurd naar een online datastore (denk aan een webservice) en er hoeft geen data teruggestuurd te worden, kun je veel beter de code `204 No Content` teruggeven, dan de code `200 OK`. Bij de code `204` weet je namelijk direct dat er geen content is in het HTTP-bericht, terwijl bij een response `200` je eerst de headers en eventueel de content (die leeg is) moet parseren.

2.2.3.4 headers Na de `start-line` van een HTTP-bericht, komen de `header-fields`. Deze headers geven extra informatie over de inhoud (de `message-body`) van het bericht.

Een header bestaat uit een naam en waarde, gescheiden door "`:`". De `field-name` is een zogenaamd token, de `field-value` is een tekst, eventueel met optionele whitespaces (`OWS`) ervoor of erachter. Elke header wordt afgesloten met een CRLF

```
header-field   = field-name ":" OWS field-value OWS
```

In principe ben je vrij om headers al dan niet mee te sturen, je mag zelfs je eigen headers verzinnen. Er zijn wel een aantal standaard headers gedefinieerd, zodat servers en clients correct op elkaar kunnen reageren. Op de wikipedia-pagina List of HTTP header fields staat een mooie opsomming van header fields ('List of HTTP header fields' 2018).

2.2.3.4.1 Standaard request fields Een aantal belangrijke request fields zijn (zie ook hier):

- `Host`

  Met het header field `Host` geef je aan welk domein van de server je wil benaderen. Door deze header is het mogelijk om meerdere websites te hosten op één server. Zie ook hier.

  Voorbeeld:

  ```
  Host: www.han.nl
  ```

- `Content-Type`

  Met `Content-Type` geef je aan hoe de content van het bericht geïnterpreteerd moet worden. Bijvoorbeeld als je een POST-request doet, kun je met deze header aangeven dat de content bestaat uit json-text, xml-text etc. Dit hoeven niet alleen tekstformaten te zijn, het is ook mogelijk dat je een bestand opstuurt om te verwerken. Zie voor de definitie hier.

  Ook hier kun je je eigen types definiëren. Er is een standaardlijst van mogelijk media-types. De officiële lijst vind je hier ('Media Types' 2018), maar een mooie compacte lijst vind je hier ('Media type' 2018). Een aantal van deze types dien je te kennen:

  ```
  application/json
  application/x-www-form-urlencoded
  application/xml
  text/css
  text/html
  text/csv
  text/plain
  image/png
  image/jpeg
  image/gif
  ```

  Voorbeelden:

  ```
  Content-Type: application/xml
  Content-Type: image/jpeg
  ```

- `Content-Length`

  Met `Content-Length` geef je aan hoe groot de content is die opgestuurd wordt, zie hier. Met de waarde van dit field kan de ontvanger van het bericht bepalen of de volledige content al ontvangen is of niet. De lengte wordt uitgedrukt in `octet`, een datatype bestaande uit 8 bits (we noemen dat ook vaak een byte).

  Voorbeeld:

```
Content-Type: 76
```

- `Connection`

  Met de header `Connection` kun je aangeven hoe er omgegaan moet worden met de de verbinding, zie hier.

  Voorbeeld:

  ```
  Connection: close
  ```

2.2.3.4.2 Standaard response fields Een aantal standaard response fields zijn (zie ook hier):

- `Content-Type`, `Content-Length`, `Connection`

  Deze headers hebben dezelfde betekenis als bij een request.

- `Expires`

  Met deze header geeft de server aan wanneer de response als verlopen kan worden beschouwd. Dit is handig als de ontvanger een cache-mechanisme gebruikt. Zie ook hier. De waarde van `Expires` is een volledige `HTTP-date timestamp`, zie ook hier

  Voorbeeld:

  ```
  Expires: Thu, 01 Dec 1994 16:00:00 GMT
  ```

- `Location`

  Wordt gebruikt bij een redirection, de header bevat de URI waarnaar de client verwezen wordt

  Voorbeeld:

  ```
  Location: http://www.example.net/index.html
  ```

2.2.3.5 message-body De `message-body` is het laatste deel van een HTTP-bericht waarin de concrete inhoud van het bericht staat. De message-body is een reeks van `octet`-waardes (byte-waardes). Het is afhankelijk van wat in de header `Content-Type` staat hoe de ontvanger de data moet interpreteren. Als er in de header is aangeven dat het type bijv. `text/plain` is, zal een browser de content onbewerkt tonen (als "platte tekst"). Maar als het type `text/html` is, zal een browser de data zo verwerken dat er een web-pagina getoond wordt. Bij het type `image/gif` wordt de data weer geïnterpreteerd als een afbeelding.

```
message-body = *OCTET
```

Zie deze link voor de documentatie.

2.3 Data Serialisatie

2.3.1 Doel

- Je kan ten minste vijf vormen van dataserialisatie beschrijven.
- Je kan data in valide JSON vertalen.

- Je kan van een stuk JSON zeggen of dit valide JSON is of niet.

2.3.2 Opdracht

2.3.2.1 Todo-items in JSON In deze oefening breiden we een Rest server voor *todo items* voor.

Todo-items hebben:

- een ID
- een titel
- een beschrijving
- een datum waarop de item moet zijn afgerond
- een *done* status

Schrijf een lijstje van drie todo-items in een JSON formaat. Bedenk zelf titels en dergelijke. Gebruik https://json.org/ om ervoor te zorgen dat je formaat valide is.

2.3.2.2 En nu in XML Converteer het formaat uit de vorige opgave in XML. Wat zijn voor- en nadelen van XML ten opzichte van JSON.

2.3.3 Toelichting

2.3.3.1 Tekstgebaseerde formaten

- Extensible Markup Language (XML)

  XML is een standaard van het World Wide Web Consortium (W3C). Het is een *opmaaktaal* waarmee gestructureerde gegevens in platte tekst vormgegeven kan worden ('Extensible Markup Language' 2018). De syntax lijkt veel op HTML, waar ook gebruik gemaakt wordt van opmaak-elementen (tags).

  Voorbeeld van XML (het inspringen en nieuwe regels is alleen opgenomen voor de leesbaarheid in document):

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <playlist name="mylist" xml:lang="en">
      <song>
          <title>Little Fluffy Clouds</title>
          <artist>the Orb</artist>
      </song>
      <song>
          <title>Goodbye mother Earth</title>
          <artist>Underworld</artist>
      </song>
  </playlist>
  ```

- Atom

  Atom is een soort van webfeed ('Atom (bestandsformaat)' 2017). Webfeeds worden gebruikt om content van webpagina's automatisch te laten updaten,

zonder dat een gebruiker zelf de pagina moet verversen. Atom zelf is weer gebaseerd op XML, de standaard is vastgelegd in RFC4287 ('The Atom Syndication Format' 2005).

- JSON

  JSON, of JavaScript Object Notation, is een gegevensformaat dat gebruik maakt van tekst. De tekst is gestructureerd in de vorm van data-objecten die weer bestaan uit één of meerdere attributen. JSON wordt ook wel gezien als een alternatief voor XML, waarbij veel overhead achterwege gelaten wordt ('JSON' 2005).

  Voorbeeld:

```
{
"playlist": {
    "name": "mylist",
    "songs": [
        {
            "title": "Little Fluffy Clouds",
            "artist": "the Orb"
        },
        {
            "title": "Goodbye mother Earth",
            "artist": "Underworld"
        }
        ]
    }
}
```

- YAML

  YAML is net als JSON een formaat om data-objecten gestructureerd weer te geven. YAML wordt veel gebruikt voor configuratiebestanden ('YAML' 2018). Ook dit formaat is goed te lezen door een mens. YAML is gedefinieerd als een standaard ('YAML Ain't Markup Language (YAML™) Version 1.2' 2009).

  voorbeeld:

```
---
playlist:
  name: mylist
  songs:
  - title: Little Fluffy Clouds
    artist: the Orb
  - title: Goodbye mother Earth
    artist: Underworld
```

2.3.3.2 Binaire formaten

- MessagePack

  MessagePack is een binair uitwisselingsformaat voor eenvoudige datastructuren ('MessagePack' 2018). Omdat het formaat bytes gebruikt om de data

weer te geven, zijn de berichten veel compacter dan als er een tekstformaat gebruikt wordt ('MessagePack home page' z.d.). Op de homepage https://msgpack.org/ wordt een (eenvoudige) vergelijking gemaakt met JSON. Daarnaast zijn er veel implementaties beschikbaar voor verschillende programmeertalen.

- Apache Thrift

  Apache Thrift is een framework waarmee *scalable cross-language services* gebouwd kunnen worden ('Apache Thrift - home' z.d.). Dit framework beschrijft een aantal uitwisselingsformaten (binair) waar gebruik van gemaakt kan worden. Het formaat was in eerste instantie beschreven door Facebook ('Apache Thrift' 2018).

- BSON

  BSON staat voor Binary JSON, het is een *binary-encoded serialization of JSON-like documents* ('BSON (Binary JSON) Serialization' z.d.). Bij het ontwerpen is vooral rekening gehouden dat het een lichtgewicht formaat is, makkelijk 'doorheen te lopen' (*traversable*) en efficiënt is om de decoderen. ('BSON (Binary JSON) Serialization' z.d.)

- Protocol Buffers

  Protocol Buffers is door Google gedefinieerd als een platform-onafhankelijk en makkelijk uit te breiden formaat. Naast de definitie is er ook een framework (compiler) beschikbaar als je gebruik wil maken van dit formaat ('Protocol Buffers' z.d.).

2.4 Representational State Transfer (Rest)

Voordat we met het hoofdstuk "Rest" beginnen als onderdeel van het grotere blok "Protocollen" toch een opmerking vooraf. Rest is géén protocol. Het is een architecturele stijl: in het kader van de relevante leerdoelen moet je dat aan het einde van dit gedeelte kunnen uitleggen.

2.4.1 Doel

- Je kan de zes theoretische voorwaarden van een Rest sever benoemen en beschrijven.
- Je kan de praktische regels van een Rest API beschrijven.
- Je kan Rest request categoriseren in termen van *Safety* en *Idempotency*.
- Je kan uitleggen wat er wordt bedoeld met "Rest is een architecturele stijl".
- Je kan uitleggen welke response status code er moet worden teruggegeven bij een Rest request.
- Je kan een Rest API formeel beschrijven.

2.4.2 Opdracht

We willen voor een bedrijf een Rest service bouwen met *todo items* voor de verschillende gebruikers. Doel van deze opgave is om de API van deze service

formeel te beschrijven. We gaan dus de documentatie van deze service maken en dus (nog) geen software schrijven.

Ter illustratie volgen hier enkele voorbeelden van wat ondersteund moet worden:

```
GET /users/12/todos
```

Dit request geeft alle todo items voor gebruiker met ID `12`. Je snapt dat wanneer je een andere geldige user ID gebruikt, je dan voor die persoon de todo items krijgt.

```
POST /users/12/todos
```

Met dit `POST` request maak je een nieuw item aan voor gebruiker 12.

```
PUT /todos/433
```

Met dit `PUT` request kan je de status van van een todo item wijzigen.

```
DELETE /todos/19
```

Met dit `DELETE` request kan je een item verwijderen.

Bovenstaande geeft een eerste aardige indruk van de service, maar het voldoet absoluut niet als bruikbare API documentatie. Allerlei vragen blijven onbeantwoord:

- Welke gegevens moet ik meesturen om een todo item aan te maken?
- Mag ik iedere item verwijderen?
- Welke code krijg ik terug als een gebruiker niet bestaat?
- Mag ik alle todo items in één keer verwijderen met `DELETE /todos`?

Het is de bedoeling dat je nu bruikbare API documentatie gaat schrijven. Dit mag je met je subversion partner samen doen. Je loopt hierbij alle geldige *request targets* af (in Rest ook vaak *resource URI* genoemd, of kort *resource*) en je geeft voor ieder *method* (in Rest ook vaak *verb* genoemd) wat het effect is. Ook geef je een voorbeeld.

Hieronder volgt een voorbeeld:

Resource: `/users/<id>`

- `GET /users/<id>`

  Geeft overzicht van de gebruiker met ID `id`.

  Request:

  ```
  GET /users/12 HTTP/1.1
  Host: api.example.com
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Connection: close
  Content-Length: 54

  {"id":"http://api.example.com/users/12","name":"Anna"}
  ```

Opmerkingen:

  - Geeft `404 Not Found` wanneer `<id>` onbekend of ongeldig is.

- `POST /users/<id>`

  Geeft `404 Not Found`.

- `PUT /users/<id>`

  Wijzigt de gegevens van de gebruiker met ID `id`.

  Request:

  ```
  PUT /users/12 HTTP/1.1
  Host: api.example.com
  Content-Type: application/json
  Content-Length: 14

  {"name":"Jan"}
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Connection: close
  Content-Length: 53

  {"id":"http://api.example.com/users/12","name":"Jan"}
  ```

  Opmerking:

  - Geeft `404 Not Found` wanneer `id` onbekend of ongeldig is.
  - Geeft `403 Forbidden` wanneer geprobeerd wordt de `id` te veranderen.

- `DELETE /users/<id>`

  Verwijdert gebruiker met ID `id` uit het systeem. Dit kan alleen wanneer er geen todo items meer aan deze gebruiker zijn gekoppeld.

  Request:

  ```
  DELETE /users/12 HTTP/1.1
  Host: 192.168.1.11
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  Connection: close
  ```

  Opmerkingen:

  - Geeft `404 Not Found` wanneer `id` onbekend of ongeldig is.
  - Geeft `403 Forbidden` wanneer er nog todo items aan gebruiker met ID `id` gekoppeld zijn.

Maak op dezelfde manier API documentatie voor de volgende resources:

```
/todos
/todos/<id>
/users
```

```
/users/<id>
/users/<id>/todos
/users/<id>/todos/<id>
```

Maak zonodig gebruik van RESTful Best Practices.

2.4.3 Toelichting

2.4.3.1 JSON in HTTP JSON is zeer geschikt om te versturen via HTTP, omdat het tekstgebaseerd is. Hieronder een eenvoudig voorbeeld van een request en een responst (met een JSON-text in de body)

Request (https://jsonplaceholder.typicode.com/posts/1)

```
GET /posts/1 HTTP/1.1
User-Agent: Fiddler
Host: jsonplaceholder.typicode.com
```

Response (een aantal headers is weggehaald voor de leesbaarheid)

```
HTTP/1.1 200 OK
Date: Wed, 13 Jun 2018 14:50:14 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 292
Connection: keep-alive
.... < nog veel meer headers > ....

{
"userId": 1,
"id": 1,
"title": "sunt aut facere repellat",
"body": "quia et suscipit\nsuscipit"
}
```

2.4.3.2 Theoretische Definitie van Rest REST is een software-architectuur vooral bedoeld voor gedistribueerde systemen op internet ('Representational state transfer' 2017). De term is door Roy Fielding geïntroduceerd (Taylor 2000).

De REST-architectuur beschrijft zes voorwaarden waaraan een systeem moet voldoen ('HTTP Status Codes' 2017).

- Uniform Interface

  Deze voorwaarde geeft aan hoe de interface tussen client en server is. Het is vooral de bedoeling dat deze 'los' van elkaar staan, m.a.w. een client heeft geen kennis van hoe de server functioneert, de client weet alleen hoe de server aangeroepen kan worden. De interface is gebasseerd op het manipuleren van resources. In deze course doen we REST via HTTP, dus het manipuleren van resources gaat via requests naar een webserver (uitleg)

  - `GET /books/978-3-16-148410-0`

    Levert een response met gegevens van het boek (met dat ISBN)

  - `POST /books/978-3-16-148410-0`

Voegt een boek toe aan het systeem

- `PUT /books/978-3-16-148410-0`

  Wijzigt of vervangt de gegevens van dat boek

- `DELETE /books/978-3-16-148410-0`

  Verwijdert het boek uit het systeem

Verder, de uniform interface schrijft ook voor dat de responseberichten 'zelfbeschrijvend' moeten zijn. Daarmee wordt bedoeld dat uit het bericht dat teruggestuurd wordt, afgeleid kan worden wat de inhoud betekent (dus geen cryptische afkortingen, maar zinvolle namen e.d.).

- Stateless

  Net als HTTP betekent dit dat een server geen status bijhoudt over een client en de requests. Dit verhoogt de schaalbaarheid van de applicatie.

- Cacheable

  De responses moeten aangeven of ze ge-cached kunnen worden of niet. Als een resource in de cache gezet kan worden, dan scheelt dat performance bij het herhaaldelijk aanroepen van dezelfde resource.

- Client-Server

  Een REST-architectuur heeft een strikte scheiding tussen een client en een server. Een REST-server is alleen verantwoordelijk voor het aanleveren van de gevraagde data, de client is verantwoordelijk voor het verwerken (bijv. in een interface)

- Layered System

  Een client die verbinding maakt met een REST-server weet niet of dat die server een intermediair is of een end server is. De aangeroepen REST-server zou bijvoorbeeld een intermediair systeem kunnen zijn die zelf weer andere systemen aanroept.

- Code on Demand (optioneel)

  Deze voorwaarde is geen verplichte voorwaarde. Het geeft aan dat je niet verplicht bent om een REST-server eerst helemaal uit te ontwikkelen, maar dat je gaandeweg functionaliteiten gaat toevoeg als clients daar behoeft aan hebben. Natuurlijk is het wel de bedoeling dat de bestaande functionaliteit ongewijzigd blijft.

In deze course wordt er dieper op REST ingegaan bij het onderwerp Python. Daar zul je een eigen REST-server en clients gaan implementeren.

2.4.3.3 Praktische Rest Voorwaarden Rest maakt gebruik van het HTTP protocol. Dit protocol voorziet in *methods* die in Rest als het werkwoord van een verzoek kunnen worden gezien. Zo vraag je met `GET` om informatie, wijzig je deze met `PUT`, verstuur je nieuwe met `POST` en verwijder je met `DELETE`.

Deze *Rest verbs* laat je los op een *resources*. Je gebruikt de *HTTP request target* om het resource te beschrijven. Je krijgt hiermee heel redelijke zinnen als:

```
GET /posts
POST /users/22/posts
```

Het is de conventie om de zelfstandige naamwoorden die je voor resources gebruikt in het *meervoud* te houden. Meestal loopt dit taalkundig goed, en je hoeft niet meer te twijfelen of je nu de enkelvoud of meervoud moet gebruiken.

Houd het ook simpel: dus, gebruik

```
GET /user/223/posts
```

in plaats van het veel minder leesbare

```
GET /api?user=223&type=posts
```

Het is mogelijk om een Rest service zo te bouwen dat meerdere formaten worden ondersteund. Wanneer er een keuze is tussen JSON en XML, kan je de volgende opties aanbieden:

```
GET /user/18.json
GET /user/18.xml
```

In beide gevallen krijg je dezelfde data, alleen verschilt de vorm waarin je deze data krijgt.

Rest is vooral ook bedoeld voor communicatie tussen machines. Een *client* doet een verzoek aan de *server* in de vorm van een HTTP request. Het is dan ook heel praktisch dat daar waar mogelijk een antwoord gegeven wordt in de vorm van een bruikbare link.

Stel, er wordt een nieuwe ticket aangemaakt voor een gebruiker

```
POST /user/2383/ticket HTTP/1.1
Host: 192.168.1.11
Connection: close
Content-Type: application/json
Content-Length: 46

{"Description":"Fix Issue","Due":"2018-12-01"}
```

De server kan nu een reactie geven waarin meteen ook een link naar dit ticket te vinden is:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json
Content-Length: 40

{"ID":"http://192.168.1.11/ticket/3221"}
```

Deze link kan meteen weer door de client gebruikt worden.

2.4.3.4 Safety en Idempotency De ontwikkelaar van een Rest service moet ervoor zorgen dat een service, waar mogelijk, *safe* en *idempotent* is.

Een actie is *safe* wanneer de actie de *state* van de server niet wijzigt.

Een actie is *idempotent* wanneer het niet uitmaakt of dezelfde actie éénmaal of meerdere malen wordt uitgevoerd.

De onderstaande tabel geeft de gewenste situatie:

verb	safe	idempotent
`GET`	ja	ja
`PUT`	nee	ja
`DELETE`	nee	ja
`POST`	nee	nee

Je bent dus hardstikke fout bezig met het ondersteunen van een request als:

```
GET http://api.example.com/todo/233/completed/true
```

of

```
PUT http://api.example.com/user/20/incrementItem
```

Waarom eigenlijk?

2.4.3.5 Response Status Codes In tbl. 2 is wat discussie geweest over `403` bij `PUT`. Volgens docentenhandleiding is deze wel relevant.

Je kan het schema in tbl. 2 gebruiken als startpunt voor het bepalen van response codes.

Tabel 2: Startpunt bepalen van response codes.

	`/user`	`/user/<id>`
`GET`	`200 OK`	`200 OK` `404 Not Found`: id onbekend of ongeldig
`POST`	`201 Created`: geef link terug	`404 Not Found`

`PUT`	`404 Not Found`
	`405 Method Not Allowed`
		`200 OK`
		`204 No Content`
		`404 Not Found`: id onbekend of ongeldig
		`403 Forbidden`: veld niet herkend

`DELETE`	`404 Not Found` `405 Method Not Allowed`	`200 OK` `404 Not Found`: id onbekend of ongeldig

2.4.3.6 HATEOAS In een van de voorbeelden voor de opgaven bij REST geeft de server de volgende response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Connection: close
Content-Length: 54

{"id":"http://api.example.com/users/12","name":"Anna"}
```

Ik wil het hier even hebben over de vorm waarin het `id` wordt teruggegeven. Deze wordt als een link teruggegeven waarmee meteen weer de REST server kan worden bevraagd.

Dit is op meer plekken verstandig. Stel dat een nieuwe gebruiker gemaakt wordt (`POST /users` met bijbehorenden gegevens), dan is het prettig meteen als id een adres terug te krijgen waarmee de gegevens van deze gebruiker opgevraagd kan worden.

Deze vorm wordt *Hypermedia As The Engine Of Application State* genoemd, afgekort als "HATEOAS": daar waar mogelijk geeft de server informatie terug in de vorm van een link waar je als gebruiker meteen weer mee aan de slag kan.

Een ander voorbeeld waarin HATEOAS handig is, is in de reactie op `GET /users`. In plaats van een antwoord met duizenden gebruiker te maken, krijg je een hanteerbare subset terug én een `next` veld (en waar zinvol een `back` veld) in de vorm van een request die gebruikt kan worden om de volgende batch gebruikers op te halen.

3 De programmeertaal Python

3.1 Introductie Python

3.1.1 Doel

In één van de vorige hoofdstukken heb je al gezien hoeveel efficiënter Python code kan zijn in vergelijking met C code. In deze course ga je een *working knowledge* opdoen van deze taal.

Na dit hoofdstuk kan je het volgende.[10]

- Je kan een eenvoudige Python 3 applicatie schrijven met functies.
- Je kan een Python 3 applicatie uitvoeren.

3.1.2 Opdracht

3.1.2.1 Kennismaking met Python De Python documentatie ('Our Documentation: Python' z.d.) is werkelijk prima.

Blader door de Python Tutorial ('The Python Tutorial' z.d.) en besteed met name aandacht aan de volgende secties:

- 3. An informal introduction to Python.
- 4. More Control Flow Tools.

  Zorg dat je tot en met sectie 4.6 snapt.
- 5. Data Structures.

  Zorg dat je de volgende secties snapt:

[10] In deze *reader* wordt vaak `python3` gebruikt. Dit komt uit de tijd dat versies 2 en 3 van Python naast elkaar gebruikt werden.
Ondertussen is versie 2 *end-of-life*. In iedere *up-to-date* distributie kan je de opdracht `python` gebruiken in plaats van `python3`.

  - De basis van 5.1.3 *list comprehensions*, want, de typische python-constructie

    ```
    values = [x**2 for x in range(20) if x**2 % 2==0]
    ```

    komt je steeds meer in code tegen, waardoor je deze constructie in ieder geval moet kunnen lezen.

  - 5.5 Dictionaries

  - 5.6 Looping Techniques

- 6. Modules

  Zorg dat je het gedeelte *voor* sectie 6.1 snapt zodat je je code over meerdere bestanden kan verdelen.

Met aandacht bedoel ik: lees door, kijk of je het snapt, probeer zo nu en dan een stukje code en experimenteer.

3.1.2.2 Python oefeningen Maak voor ieder van de onderstaande problemen een oplossing in Python.

Je kan hierbij gebruik maken van het volgende template:

```
def run(val):
    print("value: {}".format(val))


if __name__ == "__main__":
    v = input("give me a value: ")
    run(v)
```

Dan nu de opdrachten:

1. Schrijf een applicatie waarin temperatuur in Fahrenheit wordt verwacht.

   De temperatuur moet worden geconverteerd naar graden Celcius en vervolgens moet deze worden ingedeeld in een van de volgende categorieën: “te koud”, “koel”, “lekker”, “warm”, en “te heet”.

2. Schrijf een applicatie waarin een veilig wachtwoord kan worden gegenereerd. Maak deze zo goed mogelijk configureerbaar.

   Natuurlijk, er zullen allerlei generators te vinden zijn op internet. Gebruik deze niet maar probeer zelf eens wat te schrijven.

3. Schrijf een applicatie waarin een wachtwoord gecontroleerd kan worden op veiligheid.

   Ook nu geldt dat je er veel meer van leert zelf iets te ontwikkelen.

4. Er wordt gezegd dat wanneer je een groep van 30 personen bij elkaar hebt, dat dan de kans dat twee mensen op dezelfde dag jarig zijn zo’n 50% is. Dit staat bekend als het *birthday paradox*.

   Laat met behulp van een applicatie zien dat dit klopt, of juist niet.

Formateer je code met `autopep8`. Commit je code volgens het schema uit sec. 1.2.

3.1.2.3 Extra Python oefeningen

Wanneer je nog op zoek bent naar een extra uitdaging.

1. Deze youtube video van Veritasium legt een interessant probleem voor waar uiteindelijk een heel efficiënte oplossing voor is. Doel van deze opgave is om een programma te schrijven dat die oplossing test.

   Het probleem is de volgende:

   Er is een ruimte met honderd genummerde kistjes met daarin papiertjes met willekeurig de nummers 1 tot en met 100, ieder nummer komt één keer voor. Er zijn 100 gevangenen (ook met nummers 1 tot en met 100) die vrij worden gelaten onder de volgende voorwaarde: zij worden één-voor-één in de ruimte gelaten en mogen maximaal 50 willekeurige kistjes openen. De gevangene moet binnen die 50 pogingen zijn nummer terugvinden.

   Wanneer het iedere gevangene lukt om zijn of haar nummer te vinden, dan worden alle honderd gevangenen vrij gelaten. Al wanneer een gevangene het niet lukt, dan blijft iedereen gevangen.

   Vals spelen is verboden. Wel mogen de gevangene voordat de eerste de ruimte in gaat overleggen over een strategie.

   Wanneer iedere gevangene lukraak kistjes gaat openen is de kans dat de gevangenen vrij komen minimaal. Maar, met de juiste strategie (en hier zit het interessante) stijgt de kans naar ongeveel een derde dat het alle gevangenen lukt om binnen (voor iedere gevangene) 50 pogingen zijn of haar nummer te vinden.

   De strategie is als volgt: Iedere gevangene loopt als eerste naar het kistje met zijn of haar nummer. De kans dat daarin ook het juiste nummer zit is minimaal. De gevangene leest het nummer op het papiertje uit het kistje (legt deze weer terug en sluit het kistje) en gaat naar het kistje met de nummer op het papiertje: De gevangene gaat net zolang door tot hij of zij zijn nummer heeft gevonden of tot de 50 pogingen voorbij zijn.

   Schrijf een programma dat 100 keer deze strategie uitprobeert en houd bij hoe vaak de gevangenen erin slagen om vrij gelaten te worden. Als de bewering klopt en als je programma correct is, dan zou dat aantal rond de 30 moeten schommelen.

3.1.3 Toelichting

In dit hoofdstuk volgt een korte toelichting op een aantal typische Python-constructies. Deze zijn niet persé nodig op de opdrachten te maken, er wordt wel vanuit gegaan dat je deze constructies snapt.

3.1.3.1 main module

Bestand `foo.py` bevat een functie en een naïeve test:

```
def foo():
    print("this is foo")

foo()
```

We willen functie `foo` ook gebruiken in `main.py`:

```
import foo

def bar():
    print("this is bar")

if __name__ == "__main__":
    bar()
    print("bar calls foo:")
    foo.foo()
```

Met `python.py main.py` krijgen we de volgende uitvoer:

```
$ python bar.py
this is foo
this is bar
bar calls foo
this is foo
```

De eertste keer uitvoer `this is foo` komt door de test die in `foo.py` naïef wordt uitgeverd. Dit zou niet gebeuren wanneer de oplossing in `bar.py` ook in `foo.py` gebruikt zou worden.

3.1.3.2 imports Gelukkig kunnen we in python code verdelen over meerdere bestanden.

Zo hebben we `foo.py` met een aantal handige functies:

```
def foo():
    print("foo")


def qux():
    print("qux")


def foobar():
    print("foobar")
```

Deze willen we in `bar.py` gebruiken:

```
import foo


def bar():
    foo.qux()


if __name__ == "__main__":
    bar()
```

De import definieert een *name space* `foo` (identiek aan de naam van het bestand). Deze *name space* moet voorafgaan aan de functie `foo.qux()` die wordt uitgevoerd. Op deze manier voorkomen we dat `qux()` in `foo` verward wordt met een eventuele `qux()` in een ander pakket.

We kunnen ook "de *name space* `foo` importeren in die van `bar`":

```
from foo import *


def bar():
    foo()
    qux()
    foobar()


if __name__ == "__main__":
    bar()
```

De functies `foo()`, `qux()` en `foobar()` kunnen worden gebruikt alsof ze in dit bestand zijn gedefinieerd. Het voordeel is dat de code met minder *clutter* te lezen is, maar wanneer `foo.py` veel functies bevat (meer formeel: "symbolen"), dan is het risico groot dat onduidelijk is welke variant van, zeg, `foobar()` gebruikt wordt.

Er is een tussenweg met `bar.py` als

```
from foo import qux, foobar as my_func


def bar():
    qux()
    my_func()


if __name__ == "__main__":
    bar()
```

In deze laatste variant wordt heel expliciet een keuze gemaakt welke functies te importeren. Ter illustratie krijgt `foobar` een andere naam om te voorkomen dat deze clasht met een eventuele `foobar` in `bar.py`.

Er zijn meer varianten. Bijvoorbeeld:

```
import numpy as np
```

Dit is een variant van het eerste voorbeeld in deze sectie waarbij de *name space* in een andere (meestal kortere) variant wordt gebruikt.

3.1.3.3 string formats, inclusief getallen

Hier zijn twee varianten, de methode `format()` en de `f` string.

```
def format_values():
    i = 42
    f = 12.7
    s = "a few words"

    print("integer: {}, float: {}, string: '{}'".format(i, f, s))
    print(f"integer: {i}, float: {f}, string: '{s}'")

    # loads of info in https://pyformat.info/
    print(
        "integer: {:10d}, float: {:7.2f}, string: '{:15s}'".format(i, f, s))
```

```
    print(
        f"integer: {i:10d}, float: {f:7.2f}, string: '{s:15s}'")

    print(
        "integer: {:10d}, float: {:7.2f}, string: '{:>15s}'".format(i, f, s))
    print(
        f"integer: {i:10d}, float: {f:7.2f}, string: '{s:>15s}'")


if __name__ == "__main__":
    format_values()
```

De uitvoer is:

```
$ python code.py
integer: 42, float: 12.7, string: "a few words"
integer: 42, float: 12.7, string: "a few words"
integer:         42, float:   12.70, string: "a few words    "
integer:         42, float:   12.70, string: "a few words    "
integer:         42, float:   12.70, string: "    a few words"
integer:         42, float:   12.70, string: "    a few words"
```

3.1.3.4 boolean expressies, `None`

Onderscheid tussen wat booleaanse expressies en de combinatie ervan als conditie voor de `if`.

```
b1 = 3 > 1
b2 = 9 < 7
b3 = "ann" > "bob"
b4 = 9 != 3*3
b5 = True or False

if b1 and not b2:
    print("true")

if (b1 or b2) and (b2 or b3):
    print("true")

if not((b1 or b2) and not b4):
    print("true")
```

3.1.3.5 None

De waarde `None` geeft aan dat een variabele géén waarde heeft. De controle erop is met "`is`":

```
if a is None:
    # https://docs.python.org/3/library/constants.html?highlight=none#None
    print("variable a has no value")

if a is not None:
    print(f"variable a has value {a}")

if not a is None:
    print(f"variable a has value {a}")
```

3.1.3.6 Collecties en `in`

```
lst = ["aap", "noot", "mies", "wim", "vis", "vuur"]
dic = {"a": "aap", "b": "boom", "c": "citroen", "e": "ekster"}

for item in lst:
    print(item)

for i, val in enumerate(lst):
    print(i, val, lst[i])

for key, val in dic.items():
    print(key, val, dic[key])

print("mies" in lst)

vals_in_dic_as_lst = dic.values()

print("c" in dic, "ekster" in dic, "ekster" in vals_in_dic_as_lst)
```

Het keyword `in` wordt hier op twee verschillende manieren gebruikt, in de `for`-*loop* en om na te gaan of een waarde in een collectie voor komt.

Om na te gaan of een *value* in een collectie voor komt, moet van de *values* eerst een `list` worden gemaakt.

Hierbij het resultaat:

```
$ python code.py
aap
noot
mies
wim
vis
vuur
0 aap aap
1 noot noot
2 mies mies
3 wim wim
4 vis vis
5 vuur vuur
a aap aap
b boom boom
c citroen citroen
e ekster ekster
True
True False True
```

3.1.3.7 `with`

Meer info hier (link).

Voorziet in een compacte manier om code die een exceptie kan gooien te draaien. Objecten die werken met `with` definiëren een `__enter__()` en `__exit__()` methode die impliciet worden uitgevoerd.

Als er een fout gegooid wordt, dan wordt deze met `with` op dezelfde manier gegooid als in de `try catch`, dus, daar hebben we beperkt een voordeel. Wellicht

is hét voorbeeld dat het *vrijgeven van het resource* impliciet gebeurt en dus niet snel vergeten wordt.

Dus, wanneer de onderstaande code slaagt (geen exceptie gooit), dan is de *resource management* impliciet. Maar, een eventuele fout levert dezelfde stack dump op als het meer naïef openen en lezen van een niet-bestaande bestand.

```
lineno = 0

with open(fname) as file:
    while line := file.readline():
        lineno += 1
        print(f"{lineno:6d}  {line}", end="")
```

Wanneer we fouten willen afvangen, dan moet dit schijnbaar toch met `try`:

```
lineno = 0

try:
    with open(fname) as file:
        while line := file.readline():
            lineno += 1
            print(f"{lineno:6d}  {line}", end="")
except FileNotFoundError:
    print("sorry, file not found")
```

Resources worden impliciet vrijgegeven, fouten expliciet opgevangen.

3.1.3.8 optional arguments, named arguments

```
def function(a, b, c=12, d="text"):
    print(f"a: '{a}', b: '{b}', c: '{c}', d: '{d}'")
```

Geldige uitvoer van `function` is (er is geen expliciete reden voor eerst getallen en daarna strings):

```
function(1, 2)
function(1, 2, 3, 4)

function("I'm a", c="doei", b="wee")
function(d="hoi", c="doei", a="ach", b="wee")
```

Dus:

- Argumenten met default waarde hoeven niet opgevoerd te worden als *positional argument*,
- Argumenten kunnen zowel als *positional* en als *keyword argument* worden opgevoerd, *positional arguments* moeten voor *keyword arguments*.

In het algemeen:

- Een deel van je argumenten geeft extra configuratiemogelijkheden, deze krijgen een default waarde. In het algemeen voer je deze als *keyword argument* op terwijl de rest als *positional argument* wordt opgevoerd.

3.1.3.9 Virtuele omgeving

```
$ python -m venv flask
The virtual environment was not created successfully because ensurepip is not
available.  On Debian/Ubuntu systems, you need to install the python3-venv
package using the following command.

    apt install python3.11-venv

You may need to use sudo with that command.  After installing the python3-venv
package, recreate your virtual environment.

$ sudo apt install python3-venv

Setting up python3-venv (3.11.2-1+b1) ...

$ python -m venv flask

$ flask/bin/pip3 install flask

Successfully installed ...
```

3.1.3.10 list comprehension

Zie de tutorial (link).

```
lst = [x**2 for x in range(10)]
lst = [a//2 - 2 for a in lst]
lst = [b for b in lst if b > 0]
```

Je hebt de volgende onderdelen:

- Het gedeelte voor de `for`, dit is de operatie waarvan het resultaat in de nieuwe lijst wordt geplaatst
- Het gedeelte tussen `for` en `in`, dit is de naam van de variabelen waarmee de operatie kan worden uitgevoerd komend uit de verzameling na `in`.
- Het gedeelte na `in`: de verzameling waaruit wordt geput.
- Het optionele gedeelte vanaf `if`: waarmee de input-waarden kunnen worden gefilterd.

  Er staat een voorbeeld van een `yield` functie op https://mypy-lang.org/index.html.

3.1.3.11 decorators Een *decorator* waarmee een ogenschijnlijke eenvoudige functieaanroep op een complexe manier wordt uitgevoerd zonder dat je die complexiteit hoeft uit te schrijven.

Ik heb drie varianten van de code, hieronder toon ik de tweede variant.

```
def repeat_decorator(fn):
    def decorated_fn(name):
        fn(name + " a")
        fn(name + " b")
    return decorated_fn
```

```
@repeat_decorator
def hello_world(name):
    print("Hello {}!".format(name))

if __name__ == "__main__":
    hello_world("sjef")
```

De *decorator* zorgt ervoor dat het lijkt alsof de functie `hello_world()` wordt uitgevoerd. Maar, in plaats daarvan wordt de functie met de naam van de *decorator* uitgevoerd met `hello_world()` als argument.

Die `repeat_decorator()` geeft op zijn beurt weer een functie terug. Dus, waar het lijkt dat je de functie `hello_world` uitvoert, wordt er in `repeat_decorator()` ter plekke een nieuwe functie gemaakt (waar `hello_world` een onderdeel van is) en het is die nieuwe functie die wordt uitgevoerd.

Met andere woorden, de functie die lijkt te worden uitgevoerd wordt vervangen door een ter-plekke gebouwde functie, en het is die laatste functie die wordt uitgevoerd met de argumenten die je meegeeft. Het aantal argumenten dat je aan de eerste functie lijkt mee te geven zijn in feite de argumenten die je aan de tweede functie meegeeft!

3.1.3.12 property

3.1.3.13 class Zie voor het gemak `Sync/han/courses/esd-profiel/onderwijs_onderhoud/matrix_rep`

- `__repr__`, `__str__`

3.1.3.14 iterators, yield?

3.1.3.15 regular expressions

3.1.3.16 command line arguments (zie `bin/normalize_photos.py`)

3.1.3.17 unit tests

3.1.3.18 type hints en type checks

https://peps.python.org/pep-0484/, https://mypy-lang.org/index.html; mogelijk https://stackoverflow.com/a/734385 voor `instanceof`.

3.2 Basis Restserver in Python

3.2.1 Doel

Na dit hoofdstuk kan je het volgende.

- Je kan een virtuele omgeving opzetten als basis voor een server in Python.
- Je kan packages voor Python installeren.
- Je kan een eenvoudige server in Flask schrijven.

- Je kan de werking van je eigen server met `curl` testen.
- Je kan verschillende HTTP requests met Flask afhandelen.
- Je kan foutmeldingen op een Rest wijze afhandelen.
- Je kan HATEOAS principes toepassen in je response.

3.2.2 Opdracht

In de beschrijving die nu volgt wordt je stap-voor-stap meegenomen met het bouwen van een basisversie van een Rest service.

Deze beschrijving is gebaseerd op een oudere versie zodat je alert moet zijn op de volgende zaken:

- Er kunnen nog sporen zijn van een oudere versie van Python, `python2`. Wij gebruiken nu `python3`.
- De gebruikte *resources* komen niet overeen met je eigen ontwerp zoals je deze in de sectie Inleiding Rest gemaakt hebt.

  Pas je eigen implementatie aan aan je eigen ontwerp.

3.2.2.1 Hello Server In dit gedeelte gaan we een eenvoudige server draaiend krijgen.

We draaien servers in een zgn. Python Virtual Environment. Dat wil zeggen dat we packages alleen voor deze applicatie downloaden en beschikbaar maken. Op deze manier kunnen we een andere applicatie in een andere virtuele omgeving gebruik laten maken van andere versies van hetzelfde package.

Zorg dat je in je *subversion working copy* een nieuwe directory voor je opdracht hebt en ga daarheen.

```
$ cd 3_2_restBasis/hello
$ which python
/usr/bin/python
$ python --version
Python 3.10.6
```

Mooi, we hebben `python3`, maar voor een virtuele omgeving hebben we meer nodig.

```
$ sudo apt-get install python3-venv
```

Nu kunnen we in `hello` onze eerste virtuele omgeving maken.

```
$ python -m venv flask
```

Nu kunnen we namelijk zien wat er in de virtuele omgeving zit:

```
$ tree .
.
+-- flask
    +-- bin
    |   +-- activate
<whooa, there's lots more ...>
```

Er is een directory `flask` gekomen (daar hebben we om gevraagd), met daarin een complete bestandsstructuur met meer dan 300 bestanden. En dat allemaal voor alleen maar onze kleine server...

> Let even extra op. De ervaring leert dat het niet fijn is om de directory `flask` en alles wat eronder zit in subversion te zetten.
>
> Voeg `flask` dus niet toe aan `svn`.
>
> Zometeen geef ik instructies hoe je dit gemakkelijker kan voorkomen.

We gaan nu de software voor de server installeren en omdat we deze *in onze virtuele omgeving* willen hebben, gebruiken we *executables* in die virtuele omgeving. Let dus op dat je niet de "normale" `pip3` gebruikt, die zou immers voor het hele systeem installeren.

```
$ flask/bin/pip3 install flask
Collecting flask

... lots of output ...

Successfully installed
    Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7
    flask-1.0.2 itsdangerous-0.24
```

Eindelijk is het dan tijd voor onze eigen code. Eerst een controle of we in de juiste directory zitten:

```
    $ pwd
    .../3_2_restBasis/hello
```

Dat is goed, dan nu de code.

```
$ nano hello.py
```

Neem in `hello.py` het volgende over

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    return "Hello, World!"


if __name__ == '__main__':
    app.run(debug=True)
```

Hier zitten een paar nieuwe zaken in:

- `from flask import Flask`

  Uit de library `flask` wordt het "symbool" `Flask` geïmporteerd. Vanaf nu kunnen we variabelen van dit type maken alsof we de definitie zelf in dit bestand hebben getypt.

- `app = Flask(__name__)`

Dit is een manier om `app` te initialiseren als een variabele van het type `Flask`, onze server instantie.

- `@app.route('/')`

  Dit is een zgn. *decorator*.

  Het is een manier om aan `app` te zeggen dat wanneer een request binnenkomt op *request target* (of *resource* in Rest termen) "`/`", dat dan de functie `index()` moet worden uitgevoerd.

Laten we de server starten (let erop dat we weer de *executable* uit de virtuele omgeving opstarten):

```
$ flask/bin/python3 hello.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
<gives some more information>
```

De server draait op IP adres `127.0.0.1`, ook bekend als *localhost*. Dat is een chique term voor "deze computer zelf". Verder draait de server op *poort* 5000.

Om de server te testen moeten we op een tweede scherm inloggen op de raspberry om daar het volgende uit te voeren:

```
$ curl -v http://localhost:5000
<info>
```

Herken je tussen alle info het verzonder *request* en de ontvangen *response*? Ook in het scherm waar de server draait zie je een log regel:

```
127.0.0.1 - - [10/Sep/2018 16:48:29] "GET / HTTP/1.1" 200 -
```

Maar, we willen ook zo graag de server kunnen benaderen vanaf onze laptop.

Sluit de server af met `ctrl-c` (soms aangegeven als `^C`) zodat je de opdrachtprompt weer ziet.

Dan gaan we nu de server zo starten dat deze ook op de laptop bereikbaar wordt.

```
$ export FLASK_APP=hello.py
$ flask/bin/flask run --host=0.0.0.0
```

De tweede opdracht start de `flask` server. Deze opent het bestand dat je in de regel erboven gezet hebt. In die eerste regel heb je als het ware de *shell variabele* `FLASK_APP` gezet.

Verder geef je met `--host=0.0.0.0` aan dat je op requests van alle bronnen moet ingaan, dus ook van je laptop (of Arduino).

Nu kunnen we ook op de laptop de server bereiken:

```
$ curl -v 192.168.1.11:5000
```

Je had natuurlijk al gezien dat de `-v` (voor *verbose*) meer informatie geeft dan zonder. Deze gaan we van nu af aan door `-i` vervangen, dat geeft minder ruis.

En voer http://192.168.1.11:5000 nu eens in als adres in je browser.

Je eerste server draait, en op dit moment ga ik ervan uit dat je

- Python in een virtuele omgeving kan draaien,

- Flask kan starten waarbij alle devices binnen je netwerk contact kunnen maken.
- Weet dat je `flask` niet moet inchecken op `svn`.

3.2.2.2 Voeg flask toe aan de Ignore List Had ik al gezegd dat je de directory `flask` niet moet inchecken?

Je kan voorkomen dat je `flask` per ongeluk incheckt. Eerst weer even controleren of we in dezelfde directory zitten.

```
$ pwd
3_2_restBasis/hello
```

Dan voer je nu de volgende cryptische opdracht uit:

```
$ svn propset svn:ignore flask .
```

Nu je dat hebt gedaan, kan je je eigen code inchecken.

> Mocht je met `svn status` zien dat je toch `flask` hebt toegevoegd (maar nog niet gecommit), dan kan je dit probleem weer oplossen met de aanwijzing in https://stackoverflow.com/a/8197658:
>
> ```
> $ svn rm --keep-local flask
> ```
>
> Daarna kan je `flask` alsnog op de ignore list houden.

3.2.2.3 Eerste versie ToDo Rest server In deze toelichting beschrijft wordt een server beschreven die de *requests* uit tbl. 3 kan afhandelen.

Tabel 3: Af te handeren *requests* voor Rest server.

method	adres	actie
`GET`	`http://[hostname]/todo/tasks`	Toon lijst met taken
`GET`	`http://[hostname]/todo/tasks/[task_id]`	Toon één taak
`POST`	`http://[hostname]/todo/tasks`	Creëer een nieuwe taak
`PUT`	`http://[hostname]/todo/tasks/[task_id]`	Update een bestaande taak
`DELETE`	`http://[hostname]/todo/tasks/[task_id]`	Verwijder een taak

Zorg ervoor dat je een implementatie bouwt die de resources hanteert zoals jij in je eigen API beschrijving hebt gegeven.

3.2.2.3.1 Nieuwe Virtuele Omgeving In de vorige sectie werken we in directory `3_2_restBasis/hello`, we gaan nu in `3_2_restBasis/todo` aan het werk.

Maak deze directory en maak hierin een virtuele omgeving. Je weet ondertussen hoe dat moet en anders ben je te snel door de opdracht heengegaan.

3.2.2.3.2 Code Maak een bestand `todo.py` en zet hierin de volgende code. Kijk eerst of het werkt, en pas daarna de code aan aan je eigen resources.

```
from flask import Flask, jsonify

app = Flask(__name__)

tasks = [
    {
        'id': 1,
        'title': "Leer ReST API's schrijven",
        'description': "ReST API's schrijven is geweldig!",
        'done': False
    },
    {
        'id': 2,
        'title': "Leer een beetje Python",
        'description': "De wereld bestaat niet alleen uit C",
        'done': False
    }
]


@app.route('/todo/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})


if __name__ == '__main__':
    app.run(debug=True)
```

Bekijk de code en ga na of je alles snapt.

Is `tasks` nu als JSON gedefinieerd? Nee, dat is het niet, alleen de manier waarop je *lists* en *dictionaries* in Python definieert, lijkt wel heel sterk op JSON.

Is `task` met valide JSON gedefinieerd?

3.2.2.3.3 Start de server Start de server zodat de code uit `todo.py` wordt uitgevoerd. Zorg dat de server ook vanaf je laptop bereikbaar is. (Hoe moest dat ook al weer, heb je echt overal aan gedacht?)

```
$ curl -i 192.168.1.11:5000
```

Oei, als dit een HTTP server is, dan hebben we één probleem. Maar als we dit als een Rest server zien, dan hebben we twee problemen. Welke?

We proberen het nog een keer

```
$ curl -i 192.168.178.29:5000/todo/tasks
```

Dit levert resultaat. Is dit nu identiek gelijk aan de definitie van `tasks` in de code?

3.2.2.3.4 Eén Taak Tonen - Variable Rules Met variable rules kunnen we argumenten meegeven aan het request.

In de onderstaande code wordt het gebruikt om een taak id mee te geven:

```
from flask import Flask, jsonify, abort
```

We gaan `abort` gebruiken.

```
@app.route('/todo/tasks/<int:task_id>', methods=['GET'])
def get_task(task_id):
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
        abort(404)
    return jsonify({'task': task[0]})
```

De code om één taak te tonen. Laten we het proberen:

```
> curl -i http://192.168.1.11:5000/todo/tasks/1
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 157
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 13:27:42 GMT

{
  "task": {
    "description": "Rest API's schrijven is geweldig!",
    "done": false,
    "id": 1,
    "title": "Leer Rest API's schrijven"
  }
}
```

Wat gebeurt er als we een niet bestaande ID gebruiken:

```
> curl -i http://192.168.1.11:5000/todo/tasks/3
HTTP/1.0 404 NOT FOUND
Content-Type: text/html
Content-Length: 233
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 13:29:32 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server.
If you entered the URL manually please check your spelling
and try again.</p>
```

Terecht een foutmelding, maar is de vorm nu helemaal lekker?

3.2.2.3.5 Verbeterde Foutmeldingen We gaan wat aan die HTML foutmeldingen doen (eindelijk dus dat tweede probleem oplossen).

We voegen toe aan de code:

```
from flask import Flask, jsonify, abort, make_response
```

We gaan `make_response` gebruiken.

```
@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'error': 'Not found'}), 404)
```

Het knapt er een stuk van op wanneer je een fout krijgt:

```
> curl -i http://192.168.1.11:5000/todo/tasks/3
HTTP/1.0 404 NOT FOUND
Content-Type: application/json
Content-Length: 27
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 13:33:36 GMT

{
  "error": "Not found"
}
```

3.2.2.3.6 POST een nieuwe taak Kunnen we de applicatie zo maken dat we met een `GET` een nieuwe taak kunnen maken?

Vast wel. Maar is het ook verstandig?

Wij gebruiken `POST` om een nieuwe taak te maken. Deze keer de volledige code tot nu toe.[11]

```
from flask import Flask, jsonify, abort, make_response,
    request

app = Flask(__name__)

tasks = [
    {
        'id': 1,
        'title': "Leer Rest API's schrijven",
        'description': "Rest API's schrijven is geweldig!",
        'done': False
    },
    {
        'id': 2,
        'title': "Leer een beetje Python",
        'description': "De wereld bestaat niet alleen uit C",
        'done': False
    }
]


@app.route('/todo/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})


@app.route('/todo/tasks/<int:task_id>', methods=['GET'])
def get_task(task_id):
```

[11] Op een enkele plek is code over meerdere regels verspreid in verband met de beperkte breedte van de code in de *reader*. Mogelijk moet je dit ongedaan maken in je eigen code.

```
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
        abort(404)
    return jsonify({'task': task[0]})


@app.route('/todo/tasks', methods=['POST'])
def create_task():
    if not request.json or 'title' not in request.json:
        abort(400)
    task = {
        'id': tasks[-1]['id'] + 1,
        'title': request.json['title'],
        'description': request.json.get('description', ""),
        'done': False
    }
    tasks.append(task)
    return jsonify({'task': task}), 201


@app.errorhandler(400)
def bad_request(error):
    return make_response(jsonify({'error': 'Bad request'}),
        400)


@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'error': 'Not found'}),
        404)


if __name__ == '__main__':
    app.run(debug=True)
```

Wanneer we met `curl` onze service uitproberen, is het netjes om de server te vertellen dat we JSON gaan sturen.[12]

```
$ curl -i -H "Content-Type: application/json" -X POST \
 -d '{"title":"Read a book"}' \
 http://192.168.1.11:5000/todo/tasks

HTTP/1.0 201 CREATED
Content-Type: application/json
Content-Length: 105
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 13:48:44 GMT

{
  "task": {
    "description": "",
```

[12] De *backslash* in de opdracht wordt gebruikt om een opdracht over meerdere regels te verdelen. Wanneer je de opdracht op één regel zet, moet je de *backslashes* niet overtypen.

Verder een vriendelijke herinnering dat de bijgaande `curl` opdrachten niet meer op windows draaien (je moet er anders met de quotes omgaan.)

```
    "done": false,
    "id": 3,
    "title": "Read a book"
  }
}
```

Om het volledige request te zien, kan je `--trace-ascii -` toevoegen[13] in plaats van de `-v` vlag, in de complete dump kan je het volgende ontwaren:

```
POST /todo/tasks HTTP/1.1
Host: 192.168.1.11:5000
User-Agent: curl/7.43.0
Accept: */*
Content-Type: application/json
Content-Length: 23

{"title":"Read a book"}
```

Wanneer je wat aan het experimenteren slaat kan het gebeuren dat je alle toegevoegde items op enig moment kwijtraakt. Wat is daar de reden van?

3.2.2.3.7 Wijzigen en Verwijderen - `PUT` en `DELETE`

We kunnen de `PUT` en `DELETE` ook toevoegen:

```
@app.route('/todo/tasks/<int:task_id>', methods=['PUT'])
def update_task(task_id):
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
        abort(404)
    if not request.json:
        abort(400)
    if 'title' in request.json and not isinstance(
            request.json['title'], str):
        abort(400)
    if 'description' in request.json and not isinstance(
            request.json['description'], str):
        abort(400)
    if 'done' in request.json and
            not isinstance(request.json['done'], bool):
        abort(400)
    task[0]['title'] = request.json.get('title',
        task[0]['title'])
    task[0]['description'] = request.json.get(
        'description', task[0]['description'])
    task[0]['done'] = request.json.get('done',
        task[0]['done'])
    return jsonify({'task': task[0]})


@app.route('/todo/tasks/<int:task_id>', methods=['DELETE'])
def delete_task(task_id):
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
```

[13] Volgens `man curl` zorgt toevoeging van `-` dat de uitvoer van *trace* naar `stdout` wordt gestuurd.

```
        abort(404)
    tasks.remove(task[0])
    return jsonify({'result': True})
```

We kunnen bijvoorbeeld een taak op done zetten, we hoeven daarvoor alleen het `done` veld te zetten. Hoe wordt dit in de code afgehandeld?

```
> curl -i -H "Content-Type: application/json" -X PUT \
    -d '{"done":true}' http://192.168.1.11:5000/todo/tasks/2
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 150
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 14:14:28 GMT

{
  "task": {
    "description": "De wereld bestaat niet alleen uit C",
    "done": true,
    "id": 2,
    "title": "Leer een beetje Python"
  }
}
```

En zo verwijder je een item met `curl`:

```
curl -i -X DELETE http://192.168.1.11:5000/todo/tasks/2
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 21
Server: Werkzeug/0.12.2 Python/2.7.10
Date: Mon, 28 Aug 2017 13:15:35 GMT

{
  "result": true
}
```

3.2.2.3.8 Betere ID's Tot nu toe geeft de server data terug met `id`'s die niet heel gebruiksvriendelijk zijn. Het is een goede gewoonte om in plaats van kale id's bruikbare `uri`'s te geven.

We voegen hiervoor een functie `make_public_task` toe die een id omzet naar de bijbehorende `uri`.

Deze functie gebruiken we om de `return` waarden van de verschillende functies mee op te sieren, intern slaan we gewoon nog de ouderwetse `id` waarden op.

We hebben weer een toevoeging aan onze imports:

```
from flask import Flask, jsonify, abort, make_response,
    request, url_for
```

De functie `make_public_task()` en hoe deze te gebruiken in `get_tasks()`:

```
def make_public_task(task):
    new_task = {}
    for field in task:
```

```
        if field == 'id':
            new_task['uri'] = url_for(
                'get_task', task_id=task['id'],
                    _external=True)
        else:
            new_task[field] = task[field]
    return new_task


@app.route('/todo/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': [make_public_task(task) for
        task in tasks]})
```

Natuurlijk moeten ook de `return` van de andere handlers worden aangepast.

Dit is het resultaat:

```
> curl -i http://192.168.1.11:5000/todo/tasks/1
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 195
Server: Werkzeug/0.12.2 Python/2.7.9
Date: Fri, 25 Aug 2017 14:36:32 GMT

{
  "task": {
    "description": "Rest API's schrijven is geweldig!",
    "done": false,
    "title": "Leer Rest API's schrijven",
    "uri": "http://raspberrypi:5000/todo/tasks/1"
  }
}
```

3.2.2.4 Testscript Je hebt tot nu toe je API getest met verschillende `curl` opdrachten. Je kan aan de hand daarvan een script make door verschillende `curl` opdrachten in een bestand te zetten. Met dit script kan je op een basale manier je API testen:

```
$ sh testscript
```

Wanneer je tevreden bent over de resultaten, dan kan je een *referentiefile* maken:

```
$ sh testscript > result.ref
```

Iedere volgende test kan je naar een ander bestand schrijven:

```
$ sh testscript > result.out
```

Tot slot kan je dan je laatste uitvoer vergelijken met de referentie:

```
$ diff result.ref result.out
```

Bij voorkeur heb je identieke resultaten. Mogelijk zijn er timestamps die afwijken.

Nu kan je op ieder moment controleren of je API na een wijziging nog steeds dezelfde resultaten levert.

Commit al je code en scripts (maar niet de virtuele omgevingen).

3.3 Restserver met Sqlite database

Iedere keer wanneer we de server uit de vorige sectie opstarten zijn alle wijzigingen weg. In dit hoofdstuk voegen we een database toe.

3.3.1 Doel

- Je kan in je eindopdracht een database gebruiken voor je oplossing.

3.3.2 Opdracht

3.3.2.1 Database Maken We gaan gebruik maken van sqlite, een SQL database waarvoor je verder weinig software voor hoeft te installeren.

We hebben wat SQL nodig om onze initiële database te definiëren, we slaan deze op in `create.sql`:

```
create table items (
    id integer primary key autoincrement,
    title text,
    description text,
    done boolean not null check (done in (0,1))
);

insert into items (title, description, done) values (
    "Leer ReST API's schrijven",
    "Zelf ReST API's schrijven is geweldig!",
    0);

insert into items (title, description, done) values (
    "Leer een beetje Python",
    "De wereld bestaat niet alleen uit C",
    0);
```

Deze code gebruiken we om de database te maken. De database wordt opgeslagen in `todo.db`:

```
$ sqlite3 todo.db < create.sql
```

Dat is het. Laten we even testen voor we de database in combinatie met Flask gaan gebruiken, zie hiervoor de volgende code.

```
$ sqlite3 todo.db
SQLite version 3.8.10.2 2015-05-20 18:17:19

sqlite> .header on
sqlite> .mode column
sqlite> select * from items;

id  title                     description                           done
--  ------------------------  ------------------------------------  -----
1   Leer Rest API's schrijven  Zelf Rest API's schrijven is geweldig!  0
2   Leer een beetje Python    De wereld bestaat niet alleen uit C    0
```

```
sqlite> insert into items (title, description, done) values
    ("Leer wat SQLite", "Niet iedere DB heeft een server nodig", 0);

sqlite> select * from items;

id  title                      description                              done
--  -------------------------  ---------------------------------------  ----
1   Leer Rest API's schrijven  Zelf Rest API's schrijven is geweldig!  0
2   Leer een beetje Python     De wereld bestaat niet alleen uit C     0
3   Leer wat SQLite            Niet iedere DB heeft een server nodig   0

sqlite> .quit
```

3.3.2.2 Servercode Aanpassen

We wijzigen de servercode op basis van de flask tutorial ['Flask Tutorial' (z.d.a)][14].

Om te beginnen passen we de includes wat aan: `g` wordt toegevoegd als import vanuit `flask` en we importeren delen uit `sqlite3` die we voor het gemak als `sqlite3` beschikbaar maken:[15]

```
from flask import Flask, jsonify, abort, make_response, \
    request, url_for, g
from sqlite3 import dbapi2 as sqlite3
```

Laten we als voorbeeld de aangepaste versie voor `/todo/tasks` bekijken:

```
@app.route('/todo/tasks', methods=['GET'])
def get_tasks():
    db = get_db()
    cur = db.execute('select * from items order by id')
    rows = cur.fetchall() # list of Row objects, can't jsonify

    tasks = []
    for row in rows:
        tasks.append({'id': row[0], 'title': row[1],
              'description': row[2], 'done': row[3] == 1})

    return jsonify({'tasks': [make_public_task(task) for \
        task in tasks]})
```

In de eerste paar regels voeren we een SQL statement op de database uit, dit levert `rows` op. Dit is een reeks met `Row` objects die door `jsonify` niet worden geaccepteerd. Zo'n `Row` is vergelijkbaar met een array met gegevens. Deze moet ook in een zgn. *dictionary* worden omgezet en ook moet de waarde voor `done` naar een *boolean* worden omgezet.

We hebben net `get_db()` gebruikt. Dit is één van enkele functies die we zelf moeten definiëren om met databases om te kunnen gaan. Hieronder is de volledige lijst van extra functies:

[14] Ondertussen is de flask tutorial geupdate ('Flask Tutorial' z.d.b). We moeten nagaan in hoeverre dit consequenties heeft voor de inhoud van dit hoofdstuk.

[15] Ook nu geldt weer dat de oorspronkelijke Python-code vanwege korte regels moet worden opgesplit. Dit gebeurt in het algemeen door de af te breken lijn af te sluiten met "\" of na een *binary operator*.

```
def get_db():
    """
    Opens a new database connection if there is none yet
    for the current application context.
    """
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


def connect_db():
    """Connects to the specific database."""
    rv = sqlite3.connect('todo.db')
    rv.row_factory = sqlite3.Row
    return rv


@app.teardown_appcontext
def close_db(error):
    """
    Closes the database again at the end of the request.
    """
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()
```

Iedere keer wanneer we in een functie toegang tot de database nodig hebben kunnen we deze krijgen met `get_db()`. Flask zorgt er zelf voor dat `close_db()` wordt uitgevoerd.

Verder introduceren we nog een aantal handige functies. Denk bijvoorbeeld eens na hoe de code voor `POST` veranderd moet worden. Na een succesvolle `insert` van de nieuwe taak willen we deze nieuwe taak meteen meegeven in de response. We moeten hiervoor weten onder welke `id` de nieuwe taak is opgeslagen. Ook moeten we die taak dan weer uit de database halen. Omdat we vaker een specifieke taak uit de database moeten halen, maken we hiervoor een aparte functie:

```
def last_id():
    db = get_db()
    cur = db.execute('select last_insert_rowid() from items')
    rows = cur.fetchall()
    return rows[0][0]


# return list of tasks (i.e. in this case, one task)
def retrieve_task(task_id):
    db = get_db()
    # task_id is forced into a tuple:
    cur = db.execute("select * from items where id = ?",
        (task_id, ))

    rows = cur.fetchall() # list of Row objects, can't jsonify
    tasks = []
    for row in rows:
        tasks.append({'id': row[0], 'title': row[1],
              'description': row[2], 'done': row[3] == 1})
```

```
    return tasks
```

Ik geef nu nog de aangepaste handler voor de `POST`. Nu moet je voldoende informatie hebben om de `GET` voor een specifieke taak, de `PUT` en de `DELETE` zelf verder aan te passen.

```
@app.route('/todo/tasks', methods=['POST'])
def create_task():
    if not request.json or 'title' not in request.json:
        abort(400)
    db = get_db()
    cur = db.execute( \
    "insert into items (title, description, done) values (?,?,0)",
                    (  # arguments in a tuple
                        request.json['title'],
                        request.json.get('description',
                           "")))
    db.commit()
    tasks = retrieve_task(last_id())
    return jsonify({'tasks': [make_public_task(task) for \
        task in tasks]})
```

3.4 Unittests in Python

We gaan bij het gedeelte over C straks dieper in op het schrijven van *unittests* (zie hoofdstuk op pagina). Maar voor de eindopdracht komen we ook *unittests* in Python-code tegen.

Dit hoofdstuk geeft wat achtergronden.

3.4.1 Doel

- Je schrijft *unittests* om je Python-code te testen.

3.4.2 Toelichting

Stel, we hebben een briljante implementatie met code waarvan we de kwaliteit willen blijven garanderen, in bestand `my_code.py` vinden we:

```
def calc_sum(a, b):
    return a + b


def calc_diff(a, b):
    return a - b


def run():
    a = 12
    b = 9
    print("{} + {} = {}".format(a, b, calc_sum(a, b)))
    print("{} - {} = {}".format(a, b, calc_diff(a, b)))
```

```
if __name__ == "__main__":
    run()
```

De ongekende complexiteit zit natuurlijk in de functies `calc_sum` en `calc_diff`. Om die kwaliteit te garanderen, kunnen we een tweede Python-bestand maken, bijvoorbeeld `unittests.py`[16] met daarin code om de werking van deze twee functies te testen.

```
import my_code


def test_sum():
    assert my_code.calc_sum(12, 9) == 21


def test_diff():
    assert my_code.calc_diff(12, 9) == 3
```

Nadat `my_code` geïmporteerd is, worden twee functies uitgevoerd die een test doen op `calc_sum` en `calc_diff`.[17]

Om de code in `unittests.py` te kunnen testen moeten we (eenmalig) een *test framework* installeren. Dit kan met[18]

```
$ pip3 install --user pytest
```

Iedere keer dat je code wijzigt kan je bestaande code testen door de *unittests* te draaien:

```
$ pytest unittests.py
```

De uitvoer laat luid en duidelijk zien of je tests slagen of niet.

3.5 Objectgeörienteerd Programmeren in Python

Tot nu toe hebben we Python op een procedure manier gebruikt, dus, met nadruk op data en functies waarmee we die data kunnen behandelen. En dit is voor veel toepassingen prima. Object Oriëntatie is *niet* het antwoord op alles. . . .

Maar het is mogelijk om in Python op een object georiënteerde manier programmeren. Deze objecten modelleer je objecten zoals deze in de realiteit voor komen.

De manier waarop je OO programmeert is enigszins verschillend wanneer je python2 en python3 vergelijkt. Let dus goed op wanneer je informatie zoek. Voorkom dat je python2 practices zonder meer van toepassing verklaard op python3.

Een informatieve introductie vind je, zoals altijd, in de officiële Python documentatie ('Classes' z.d.). Een samenvatting hiervan vind je in dit hoofdstuk.

[16] Let hier op de `s` als laatste letter van `unittests.py`. Zonder deze `s` krijg je heel vreemde foutmeldingen omdat `python` elders `unittest.py` vindt.

[17] Dit gaat met behulp van `assert` functies. Voor nu voldoet het dat deze gelezen kunnen worden als "stel vast dat . . . ". Vergelijkbare functies in C worden weer behandeld en nader uitgelegd in hoofdstuk.

[18] Er zijn meerdere *test frameworks* voor Python beschikbaar. Wij gebruiken min of meer toevallig `pytest` ('Pytest helps you write better programs' z.d.).

In dit hoofdstuk vind je geen oefeningen. Maar wees ervan bewust dat dit wel toetsstof voor de theorietoets is.

3.5.1 Doel

- Je kan klassen toevoegen aan je Python code
- Je kan van klassen overerven.

3.5.2 Toelichting

3.5.2.1 Een Python klasse We blijven in de "todo" sfeer om een aantal Python concepten toe te lichten.

Bekijk de volgende code:

```
import time

class Item:

    """ base for todo and log items """

    ids = 0  # class member, shared over all Item instances

    def __init__(self, description):
        self.description = description
        self.created = self.now()
        Item.ids += 1
        self.id = Item.ids

    def now(self):
        return time.localtime()

    def __repr__(self):
        return "item {}".format(self.id)

    def __str__(self):
        return "item {} ({}): {}".format(self.id,
            time.asctime(self.created), self.description)


if __name__ == "__main__":
    i1 = Item("geef class voorbeeld")
    i2 = Item("geef overerving voorbeeld")
    todo = [i1, i2]  # list of todo items
    print(i1)       # uses __str__
    print(todo)     # uses __repr__ (represent) short form
    print(Item.__doc__)  # prints class doc string
```

In deze code vind je de definitie van `Item` en onderaan een test waarmee we deze code kunnen testen. Een aantal opmerkingen over deze code:

- De definitie van class `Item` begint met `class Item`.
- Hieronder vind je de zgn. *docstring*, dit is als string de documentatie van deze klasse.

De test onderaan laat zien dat je deze string met `Item.__doc__` kan opvragen.

- De variable `ids` is wat je in andere talen een *static variabele* zou noemen. Deze waarde wordt gedeeld over alle instanties van deze klasse.

  Hier gebruiken we deze *class variable* om de huidige ID bij te houden.

- De functie `__init__` is de *constructor*. Je ziet dat hier een aantal *instance variables* worden gedefiniëerd: `self.description`, `self.created` en `self.id`.

  De variabel `self` is wat in andere talen vaak `this` voor wordt gebruikt: een referentie naar de instantie zelf.

  De ondersteuning van klassen in Python is heel expliciet: je moet de waarde `self` in iedere methode als eerste waarde meegeven. Verder gebruik je deze ook als je naar een instance member verwijst: `self.created = self.now()`.

  Let erop dat in het laatste voorbeeld de `self` niet als argument wordt meegegeven. Het is dus `self.now()` en niet `now(self)`!

- De methoden `__str__` (van *string*) en `__repr__` (van *representation*) zijn twee van de vele van dergelijke `__<value>__` waarden in Python.

  Wanneer je dit bestand uitvoert, dan zal je zien dat standaard de `__str__` methode gebruikt wordt om een instantie af te drukken.

  Zosnel je een hele collectie afdrukt (zoals in `print(todo)` gebeurt), dan wordt gebruik gemaak van `__repr__`. In het algemeen geef je met `__repr__` een meer compacte weergave.

3.5.2.2 Overerving Python ondersteunt overerving (en net als C++ zelfs *multiple inheritance*). Bestudeer de onderstaande code met daarin een normale, enkelvoudige, overerving van `Item` in klasse `TodoItem`.

```
import time


class Item:

    """base for todo and log items"""

    ids = 0  # class member, shared over all Item instances

    def __init__(self, description):
        self.description = description
        self.created = self.now()
        Item.ids += 1
        self.id = Item.ids

    def now(self):
        return time.localtime()

    def __repr__(self):
```

```
        return "item {}".format(self.id)

    def __str__(self):
        return "item {} ({}): {}".format(self.id,
            time.asctime(self.created), self.description)


class TodoItem (Item):

    """todo item subclass"""

    def __init__(self, description):
        super().__init__(description)
        self._log = []
        self._done = False

    def addLog(self, description):
        self._log.append(Item(description))

    def done(self):
        self._done = True

    def __str__(self):
        val = super().__str__()
        if self._done:
            val += " [x]\n"
        else:
            val += " [ ]\n"
        for l in self._log:
            val += "- {}\n".format(l)
        return val


if __name__ == "__main__":
    i1 = TodoItem("kook maaltijd")
    i1.addLog("recept bedacht")
    i1.addLog("boodschappen gedaan")
    i1.done()
    print(i1)
```

In Python 3 is overerving en de uitvoering van methoden in de super klasse een stuk eenvoudiger geworden dan in Python 2.

Ter illustratie kan in bovenstaande code aan een `TodoItem` log items worden toegevoegd (die weer van het type `Item` zijn.

De functie `__str__` maakt gebruik van functionaliteit van `Item` om zelf tot een nette manier te komen om een todo item af te drukken.

Zorg dat je deze code snapt en zelf op vergelijkbare manier een klasse met subklassen kan schrijven.

4 Wiskunde

4.1 Regressie

4.1.1 Lesdoelen

- Je kan een verzameling waarden kenmerken aan de hand van gemiddelde en spreiding.
- Je kan gemiddelde en spreiding op een efficiënte manier implementeren op een controller met beperkte resources.
- Je kan een trend in een verzameling waarden kenmerken door een lineaire regressie te berekenen.
- Je kan berekeningen met matrices uitvoeren.
- Je kan de berekeningen in Python uitvoeren.

4.1.2 Opgaven

4.1.2.1 Gemiddelde en standaardeviatie in een populatie Op enig moment wordt van alle aanwezigen in een bioscoop de leeftijd gevraagd. Dit is de uitkomst.

```
41 34 27 31 26 19 53 53 42 50 25 15
```

(Je ziet het, het is niet zo'n heel populaire film).

a. Bereken het gemiddelde μ

b. Bereken de standaarddeviatie van de populatie σ met

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

Wat is het nadeel van deze methode wanneer je de spreiding in een heel grote verzameling waarden moet berekenen?

c. Bereken dezelfde standaarddeviatie σ op een meer efficiënte manier.

Wat is het grote voordeel van deze berekening ten opzichte van de vorige berekening?

Wat is het nadeel wanneer je in plaats van de leeftijd in jaren tijdsduren in milliseconden hebt?

d. Stel dat de bioscoop weet dat de gemiddelde leeftijd van alle bezoekers in het afgelopen jaar 28 jaar is. Gebruik deze informatie om het gemiddelde van de mensen in de zaal en de spreiding te berekenen zodat je berekening beter omgaat met grote getallen.

4.1.2.2 Steekproef Stel, we willen proberen om met de 12 waarnemingen uit de vorige opgave de gemiddelde leeftijd van alle bezoekers van de betreffende bioscoop te schatten.

a. Is dit zondermeer een goede methode, waarom wel / niet?

b. Bereken gemiddelde en standaarddeviatie van de steekproef. Zijn deze waarden gelijk aan de waarden van de vergelijkbare waarden van de populatie?

4.1.2.3 Schatting van een verband Tbl. 4 geeft de set van waarnemingen uit sec. 4.1.3.3. Beantwoord de volgende vragen.

Tabel 4: Gefingeerde waarnemingen.

i	x	y
1	47.52	56.17
2	84.38	90.46
3	31.91	48.80
4	36.10	55.84
5	36.94	46.17
6	70.64	69.13
7	16.14	37.22
8	68.98	81.17
9	54.15	59.57
10	56.31	72.49
11	13.57	25.58
12	69.17	71.45

a. Klopt het dat je met een rechte lijn $y = b_0 + b_1 x$ een goede beschrijving van deze set kan geven?

b. Bereken b_0 en b_1.

c. De werkelijke waarden zijn $\beta_0 = 20$ en $\beta_1 = 0.8$. Waarom wijken b_0 en b_1 af van de respectievelijke waarden β_0 en β_1?

d. Bereken R^2.

4.1.2.4 Regressie en kwaliteit De dataset uit sec. 4.1.3.5 is niet goed te fitten met een rechte lijn. Toch gaan we dat in deze opgave doen.

a. Bereken b_0, b_1 en R^2 wanneer we de data toch met een rechte lijn willen fitten.

b. Schat op basis van het model de responsetijd wanneer

- de CPU load 10% is
- de CPU load 90% is

Plot beide resultaten in de grafiek. Wat vind je van de uitkomst?

4.1.2.5 Regressie met een parabool In de vorige opgave hebben we met een dataset gewerkt die niet goed met een rechte lijn te beschrijven is. We kunnen deze beter beschrijven met een parabool (wat is dat ook al weer?). Het liefst zou ik je vragen dit met de hand uit te rekenen, maar dat vind ik zelfs teveel werk voor jullie...

We gaan de berekening met python uitvoeren.

a. Ga naar de folder waarin je de code voor deze opgave wilt schrijven en installeer daar een virtuele omgeving met het pakket numpy. Met dit pakket kan je allerlei geavanceerde berekeningen uitvoeren, ondermeer matrixberekeningen.[19]

```
$ python3 -m venv localenvironment
$ localenvironment/bin/pip3 install numpy
```

b. Draai en bestudeer de volgende code.

```
import numpy as np

a = np.array([[5, 5], [1, 5]])
print("2x2 matrix:")
print(a)

b = np.array([[1, 2, 3, 4]])
print("1x4 matrix:")
print(b)

c = np.array([[5, 6, 7, 8]])
print("another 1x4 matrix:")
print(c)

d = np.concatenate((b, c))
print("combine two 1x4 matrices into a 2x4 matrix:")
print(d)

e = np.concatenate((np.power(b, 0), np.power(b, 1), np.power(b, 2)))
print("combine variations of b into a 3x4 matrix:")
print(e)

# matrix operaties
print("add:")
f = b + c
print(f)

print("multiply (mind the operator):")
g = a @ d
print(g)

print("transpose:")
h = np.transpose(e)
print(h)

print("matrix inverse:")
i = np.linalg.inv(a)
print(i)
```

Na de lessen over matrixrekenen moet je snappen wat er in deze code

[19] Mogelijk werken de gegeven opdrachten niet op een Raspberry Pi. Mogelijk moet `sudo apt install libatlas-base-dev` worden uitgevoerd voordat `numpy` geïnstalleerd wordt. Op enig moment was deze opdracht de eerste van de reeks gegeven opdrachten.

gebeurt.

c. Schrijf code waarin je met matrices de berekening uit de vorige vraag overdoet, dus, waarin je de *slope* en het *intercept* berekend van de beste rechte lijn door de data.

   Het resultaat van deze berekening moet natuurlijk overeenkomen met het resultaat dat je in de vorige vraag hebt berekend.

d. Schrijf nu code waarin je met matrixberekening de regressie voor een parabool uitvoert.

4.1.2.6 Matrix-typen Geef een voorbeeld van een:

a. Kolommatrix

b. Vierkante matrix

c. Symmetrische matrix

d. Diagonaalmatrix

e. Identiteitsmatrix

4.1.2.7 Matrix-operaties (maar geen inverse) Bepaal van de volgende sommen of deze te berekenen zijn. Wanneer dat niet het geval is, leg uit waarom niet. Wanneer de berekening wel te maken is, maak deze.

a.
$$\begin{bmatrix} 8 & 3 \\ 16 & 10 \\ -2 & 4 \end{bmatrix}^{\mathrm{T}}$$

b.
$$\begin{bmatrix} 10 \end{bmatrix}^{\mathrm{T}}$$

c.
$$\begin{bmatrix} 8 & 3 \\ 16 & 10 \\ -2 & 4 \end{bmatrix} + \begin{bmatrix} 8 & 3 & 0 \\ 16 & 10 & 7 \end{bmatrix}$$

d.
$$\begin{bmatrix} 8 & 3 \\ 16 & 10 \\ -2 & 4 \end{bmatrix}^{\mathrm{T}} + \begin{bmatrix} 8 & 3 & 0 \\ 16 & 10 & 7 \end{bmatrix}$$

e.
$$\begin{bmatrix} 8 & 3 \\ 16 & 10 \\ -2 & 4 \end{bmatrix} + \begin{bmatrix} 8 & 3 & 0 \\ 16 & 10 & 7 \end{bmatrix}^{\mathrm{T}}$$

f.
$$\begin{bmatrix} 8 & 3 \\ 16 & 10 \\ -2 & 4 \end{bmatrix}^{\mathrm{T}} + \begin{bmatrix} 8 & 3 & 0 \\ 16 & 10 & 7 \end{bmatrix}^{\mathrm{T}}$$

g.

$$\begin{bmatrix} 8 & 3 & -1 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 1 & 0 & 7 \end{bmatrix}$$

h.

$$\begin{bmatrix} 9 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \end{bmatrix}$$

i.

$$\begin{bmatrix} 2 & 1 & 3 \\ 7 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 3 & 1 & 1 & 0 \\ 2 & 3 & 3 & 1 \\ 7 & 4 & 9 & -1 \end{bmatrix}$$

j.

$$\begin{bmatrix} 3 & 1 & 1 & 0 \\ 2 & 3 & 3 & 1 \\ 7 & 4 & 9 & -1 \end{bmatrix} \times \begin{bmatrix} 2 & 1 & 3 \\ 7 & 1 & 0 \end{bmatrix}$$

k.

$$\begin{bmatrix} -9 & 18 \\ 47 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

l.

$$\begin{bmatrix} -9 & 18 \\ 47 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

4.1.2.8 Inverse matrices Bepaal van de volgende matrices of deze een inverse matrix hebben, bereken deze indien mogelijk en controleer of de matrix vermenigvuldigd met zijn inverse inderdaad de identiteitsmatrix oplevert.

a.

$$A = \begin{bmatrix} 8 & 3 \\ 16 & 10 \end{bmatrix}$$

b.

$$B = \begin{bmatrix} 6 & 4 \\ 3 & -1 \end{bmatrix}$$

c.

$$C = \begin{bmatrix} 9 & 3 \\ 6 & 2 \end{bmatrix}$$

d.

$$D = \begin{bmatrix} 10 & 5 \\ 14 & 3 \end{bmatrix}$$

e.

$$E = \begin{bmatrix} 2 & 14 \\ 1 & 5 \end{bmatrix}$$

f.

$$F = \begin{bmatrix} 2 & 18 \\ 1 & 9 \end{bmatrix}$$

g.

$$G = \begin{bmatrix} 5 & 5 \\ 1 & 5 \end{bmatrix}$$

4.1.3 Toelichting

4.1.3.1 Populaties, gemiddelden en spreiding Stel, we willen informatie over de leeftijd van alle aanwezigen in de klas.

1. We kunnen een tabel maken van alle leeftijden, dat wil nog omdat we met relatief weinig zijn.
2. We kunnen een histogram maken met verschillende categorieën.
3. Maar hoe kunnen we nu iets compacter informatie geven over de leeftijdsverdeling?
   - Stel we willen deze klas vergelijken met een deeltijdklas en weten welke studenten ouder zijn.

     Je kan een gemiddelde van de populatie μ bepalen:

     $$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$
4. Naast een gemiddelde wil je soms ook iets zeggen over de spreiding van de leeftijd in de klas: bestaat een klas alleen uit HAVO scholieren die tot nu toe alles in één keer gehaald hebben, of zijn we een mooi samenraapsel van studenten met allerlei achtergronden.
   - Een poging voor een maat van spreiding ς:

     $$\varsigma = \frac{\sum_{i=1}^{n} (x_i - \mu)}{n}$$

     Dit levert precies de waarde nul op, omdat waarden onder en boven het gemiddelde elkaar precies opheffen.
   - Poging twee

     $$\varsigma = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

     Nu krijgen we wel een spreiding die bekend staat als *variantie*:

     $$\operatorname{var} x = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

     Het symbool wordt ook wel als σ^2 geschreven.
   - Poging twee verder verbeterd

     Nadeel van de vorige poging is de eenheid van $\operatorname{var} x$: wanneer x in cm, dan is $\operatorname{var} x$ in cm^2, het kwadraat dus. We kunnen de wortel nemen:

     $$\sigma = \sqrt{\operatorname{var} x} = \sqrt{\frac{\sum_{i=n}^{n} (x_i - \mu)^2}{n}}$$

Deze waarde σ staat bekend als de *standaarddeviatie van de populatie*. Die "populatie" slaat erop dat we de spreiding van lengte (of leeftijd) van iedereen in onze klas hebben berekend.

- Beter te berekenen.

  De vorige berekening klopt, maar is niet fijn om te berekenen (waarom niet).

  Er is een betere methode:

$$\sigma = \sqrt{\frac{\sum x^2 - \sum x \cdot \sum x/n}{n}}$$

- Nog beter te berekenen met $x' = x - m'$ (waarin x' veel dichter bij nul moet komen te liggen dan x en waarbij m' een heel eenvoudige schatting is van μ):

$$\sigma = \sqrt{\frac{\sum x'^2 - \sum x' \cdot \sum x'/n}{n}}$$

  Want, voor de spreiding maakt het niet uit wat het middelpunt is, wel geldt

$$\mu = \frac{\sum x'}{n} + m'$$

4.1.3.2 Steekproeven De tijd die het duurt voordat een server een reactie stuurt, de relatie tussen een sensorwaarde en de temperatuur in de kamer. Dit zijn twee voorbeelden van waarden waarover we wellicht ooit een uitspraak over moeten doen. Alleen, we kunnen nooit al die waarden meten. Hoe moeten we hier mee omgaan?

We kunnen een steekproef doen (in het Engels, een *sample*). Stel, we meten eenmaal de tijd die het duurt voordat een server een reactie stuurt. Kunnen we nu iets zinnigs zeggen? We weten in ieder geval meer dan wanneer we geen meting hebben. Maar, misschien hebben we te maken met een uitzondering (de server had het even heel druk).

Uiteindelijk zouden wij het gemiddelde μ en de standaarddeviatie σ willen kennen, maar dat gaat dus nooit lukken. Het enige wat wij nog kunnen doen is deze waarden te *schatten* aan de hand van de steekproef.

De *gemiddelde van de steekproef* bereken we nu als volgt:

$$m = \frac{\sum_{i=1}^{n} x_i}{n}$$

Voor het gemiddelde gebruiken we nu het symbool m in plaats van μ: een andere steekproef levert waarschijnlijk een andere waarde m op. De (hypothetische) gemiddelde van de populatie μ verandert niet.

De *standaarddeviatie van de steekproef* wordt iets anders berekend dan de standaarddeviatie van de populatie:

$$s = \sqrt{\frac{\sum_{i=n}^{n}(x_i - m)^2}{n-1}}$$

In de teller staat niet meer hét gemiddelde van de populatie μ, maar de schatting van dat gemiddelde m.

De waarde van de noemer is nu $n-1$ en niet meer n. Het gevolg hiervan is dat de waarde van s iets groter zal zijn dat de berekende waarde σ. Met andere woorden, wanneer we met s de waarde σ proberen te schatten, wordt in de de berekening gehouden dat we een schatting aan het maken zijn die iets naar boven wordt bijgesteld.

Alle besproken optimalisaties van de berekening van de standaarddeviatie blijven ook voor de standaarddeviatie van de steekproef gelden: houdt er alleen rekening mee dat de noemer steeds $n-1$ wordt in plaats van de gegeven noemer n.

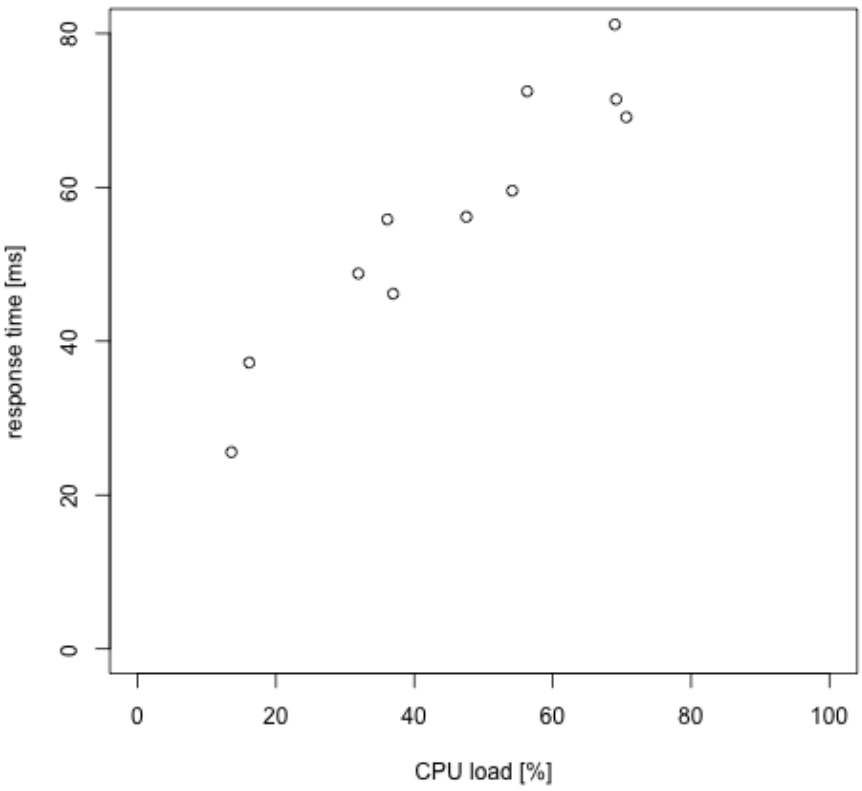
4.1.3.3 Lineaire regressie met expliciete functies In tbl. 5 vind je een (volkomen fictieve) relatie tussen x (CPU load in procent) en y (responsetijd request in ms) van een RPi. Je vindt deze ook terug in fig. 2.

Tabel 5: Kopie van tbl. 4.

i	x	y
1	47.52	56.17
2	84.38	90.46
3	31.91	48.80
4	36.10	55.84
5	36.94	46.17
6	70.64	69.13
7	16.14	37.22
8	68.98	81.17
9	54.15	59.57
10	56.31	72.49
11	13.57	25.58
12	69.17	71.45

Je ziet dat er een duidelijk verband is tussen de CPU load (de x-waarde) en de responsetijd (de y-waarde): wanneer de CPU load hoger wordt, dan duurt het in het algemeen langer voordat de je een response van deze server krijgt.

Net zoals het fijn is om een groep waarnemingen samen te vatten in termen van gemiddelde en spreiding, zou het fijn zijn wanneer we een relatie als deze tussen CPU load en responsetijd kunnen samenvatten. Met wat fantasie kan je een rechte lijn trekken door de puntenwolk die de beschrijving tussen CPU load en responsetijd samenvat:



Figuur 2: De waarnemingen uit tbl. 5 in een diagram.

$$\text{response tijd} = \text{CPU load} \times \text{helling} + \text{intercept}$$

Dit is niets anders dan de vergelijking van een rechte lijn $y = ax + b$ die je ook op de HAVO hebt gehad.

We gaan er vanuit dat er zoiets is als "de echte relatie tussen CPU load en responsetijd":

$$y = \beta_0 + \beta_1 \cdot x$$

Hierin zijn β_0 het werkelijke intercept en β_1 de werkelijke helling die de relatie voor onze server (of voor alle servers van dit type) beschrijven. Net zoals we eerder "het werkelijke gemiddelde" μ en "de werkelijke spreiding" σ niet kunnen kennen maar slechts kunnen schatten, geldt dit ook voor het verband tussen CPU load en responsetijd: we zullen β_0 en β_1 nooit kennen, maar we kunnen deze proberen te schatten als b_0 en b_1.

$$y = b_0 + b_1 \cdot x$$

In deze vergelijking zijn b_0 en b_1 schattingen van β_0 en β_1. Deze waarden zijn te berekenenen met de volgende vergelijkingen:

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}$$

en

$$b_0 = \frac{\sum_{i=1}^{n} y_i}{n} - b_1 \frac{\sum_{i=1}^{n} x_i}{n}$$

4.1.3.4 Coefficient of Determination Iedere poging om door een puntenwolk een rechte lijn te berekenen leidt wel tot een antwoord, maar hoe goed is dat antwoord? Dit is een hele wetenschap op zich, maar we beperken ons hier tot de *coefficient of determination*, beter bekend als R^2.

De waarde R^2 is goed uit te rekenen wanneer n, $\sum x$, $\sum y$, $\sum x^2$, $\sum y^2$, en $\sum xy$ bekend zijn:

$$R^2 = \frac{\left(\sum_i x_i y_i - \frac{\sum_i x_i \cdot \sum_i y_i}{n}\right)^2}{\left(\sum_i x_i^2 - \frac{\left(\sum_i x_i\right)^2}{n}\right) \cdot \left(\sum_i y_i^2 - \frac{\left(\sum_i y_i\right)^2}{n}\right)}$$

Stel dat je vindt dat $R^2 = 0.7$, dan betekent dit dat 70% van de variatie in de y waarden wordt verklaard door je model. Waar de overige 30% van de variatie vandaan komt blijft onduidelijk.

Pas op dat bovenstaande vergelijking alleen geldig is een rechte lijn tussen de x en y waarden. Voor kromme lijnen zoals we deze gaan behandelen vanaf sec. 4.1.3.5 moeten we terug naar de formele definitie van de *coefficient of determination* R^2.

De formele definitie van de *coefficient of determination* vergelijkt de mate waarin je model (bijvoorbeeld $\hat{y} = b_0 + b_1 \cdot x$, waarbij $\hat{y}$ de voorspelde waarde van y is op grond van het model) beter in staat is om y uit x te voorspellen dat de boute uitspraak: "wat de waarde van x ook is, mijn voorspelling voor de bijbehorende y is niet beter dan de gemiddelde waarde van alle y-waarden".

Met andere woorden, de vraag is steeds hoe goed kunnen we de meetwaarde y uit de bijbehorende meetwaarde x voorspellen. Hoeveel beter is $\hat{y}$ een goede voorspeller voor y waarin $\hat{y}$ uit je model volgt, bijvoorbeeld

$$\text{voorspel } y \text{ uit model: } \hat{y} = b_0 + b_1 \cdot x + \cdots$$

vergeleken met het idee dat we niet beter weten dan y uit de gemiddelde waarden $\bar{y}$ voorspellen:

$$\text{voorspel } y \text{ uit het gemiddelde: } \bar{y}$$

Dus, met de waarnemingen y, de voorspelling uit het model $\hat{y}$, en het gemiddelde van de waarnemingen $\bar{y}$ volgt nu:

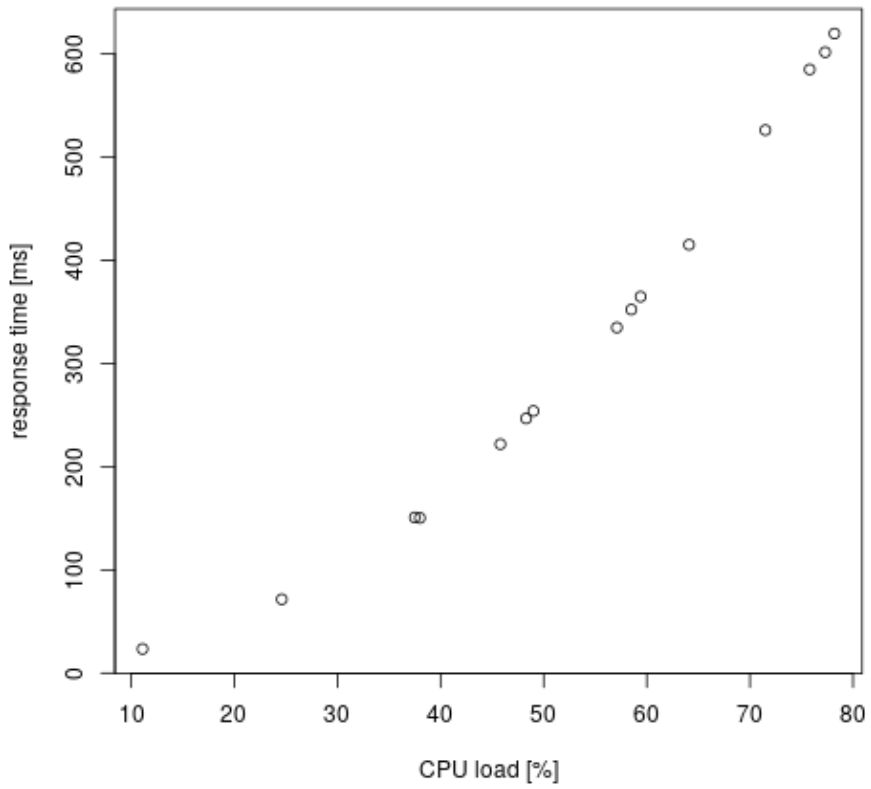
$$R^2 = 1 - \frac{\sum (y - \hat{y})^2}{\sum (y - \bar{y})^2}$$

Wanneer het model de waarnemingen y heel goed voorspellen, dan is de tweede term ongeveer nul waardoor $R^2 \approx 1$. Wanneer het model het niet beter doet dan het gemiddelde, dan loopt de tweede term richting 1, waardoor $R^2 \approx 1 - 1 \approx 0$.

In de eindopdracht moet je deze tweede benadering volgen. De enorme vergelijking voor R^2 aan het begin van deze sectie is een uitwerking hiervan voor de situatie dat je model $\bar{y} = b_0 + b_1 \cdot x$ is, oftewel, een rechte lijn. In dat geval is het dus mogelijk om R^2 op grond van de zes "variabelen" als n, $\sum x$, en $\sum x^2$ te berekenen.

4.1.3.5 Uitbreiding van de mogelijkheden Het blijkt dat er een fout heeft gezeten in het eerste experiment waarin gekeken is hoe de CPU load een effect heeft op de responsetijd van de server. Er is een nieuw experiment uitgevoerd. De resultaten vind je in fig. 3[20]. Voor alle volledigheid volgen in tbl. 6 de meetwaarden met x de CPU load in (ms), en y de responsetijd in (ms).

[20] de data zijn weer volledig *fake* ...



Figuur 3: Resultaten van verbeterd experiment.

Tabel 6: Data uit fig. 3.

i	x	y
1	58.5	352.2
2	37.5	150.8
3	77.3	601.6
4	64.1	415.0
5	78.2	619.7
6	45.8	221.8
7	24.6	71.5
8	49.0	253.9
9	11.1	23.4
10	48.3	246.7
11	59.4	364.7
12	38.0	150.3
13	75.8	584.9
14	57.1	334.8
15	71.5	526.1

Je ziet het, het is niet verstandig hier een rechte lijn doorheen te trekken. Dit leidt of tot een onderschatting bij lage CPU loads, of bij een onderschatting bij hoge CPU loads.

Echter, de formules voor b_0 en b_1 die we tot nu toe besproken hebben, helpen ons alleen voor het schatten van een rechte lijn door de data.

We zullen dus iets moeten. En daarvoor is rekenen met matrices noodzakelijk.

4.1.3.6 Waarom Matrices Wanneer je nog eens goed naar de afbeelding kijkt van de gemeten data, dan zou je wel eens tot de conclusie kunnen komen dat de data door een parabool worden beschreven.

$$y = \beta_0 + x\beta_1 + x^2\beta_2$$

En wij moeten de waarden β_0, β_1 en β_2 bepalen. Hoe doen we dat? Wat is eigenlijk het probleem dat we aan het oplossen zijn?

Eigenlijk hebben we in ons experiment te maken met het volgende stelsel van vergelijkingen waarin de b waarden *schatters* zijn voor de β waarden.

$$\begin{aligned}
y_1 &= b_0 + x_1 b_1 + x_1^2 b_2 + \epsilon_1 \\
y_2 &= b_0 + x_2 b_2 + x_2^2 b_2 + \epsilon_2 \\
y_3 &= b_0 + x_3 b_3 + x_3^2 b_2 + \epsilon_3 \\
&\vdots \\
y_{15} &= b_0 + x_{15} b_{15} + x_{15}^2 b_2 + \epsilon_{15}
\end{aligned}$$

De ϵ waarde staat bekend als de *error term* en geeft voor iedere meting i de afwijking tussen de gevonden waarde en de waarde volgens het model.

Een regressieberekening is niets anders dan die hele stelsel van vergelijkingen zo oplossen (dat wil zeggen waarden voor b_0, b_1 en b_2 vinden) dat de som van de fouten, $\sum_i \epsilon_i$ zo klein mogelijk is.

Maar dit gaat een enorme schrijfpartij opleveren. Of wacht, gelukkig kunnen we gebruik maken van matrices waarmee het hele stelsel veel compacter is op te schrijven:

$$Y = X\mathbf{b} + \epsilon$$

Is dit echt hetzelfde? Ja, dit is echt hetzelfde. De symbolen Y, X, $\mathbf{b}$ en ϵ[21] zijn *matrices* en het is gebruikelijk om matrices van "normale waarden" (meer officieel, van *skalars*) te onderscheiden door ze of in hoofdletters of in vet te schrijven. (Helaas is ϵ in het vet ook ϵ, je ziet dus niet altijd het verschil.)

De matrix Y staat voor de volgende matrix:[22]

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{14} \\ y_{15} \end{bmatrix}$$

Deze matrix heeft 15 rijen en 1 kolom, Y is een 15×1 matrix. En het heeft opvallend veel weg van een `array`. Een dergelijke matrix uitprinten gaat enorm veel papier kosten. Vandaar dat van een matrix ook wel eens zijn *transpose* gegeven wordt.

Hoe denk je dat matrix X eruit ziet? Ik zal de *transpose* van X geven, aangeduid als X^T:

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & \cdots & x_{14} & x_{15} \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & \cdots & x_{14}^2 & x_{15}^2 \end{bmatrix}$$

Of, we hadden ook kunnen schrijven

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & \cdots & x_{14} & x_{15} \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & \cdots & x_{14}^2 & x_{15}^2 \end{bmatrix}^T$$

[21] Er is helaas geen vette variant van het symbool ϵ waardoor het duidelijker wordt het het hier om een matrix gaat.

[22] Niet alle elementen passen in de matrix, dit wordt aangegeven met de puntjes. Je bent zelf wel in staat de ontbrekende elementen erin te bedenken!

Zoek de verschillen! Let dus op dat X 15 rijen en 3 kolommen heeft en dat de *transpose* van X, dus X^T 3 rijen en 15 kolommen heeft.

De matrix $\mathbf{b}$ is

$$\mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Een 3×1 matris dus.

Voordat we verder gaan met de mogelijke berekeningen met matrices kijken we nog even naar de dimensies van onze eerste matrixberekening:

$$\begin{array}{ccccccc} Y & = & X & \mathbf{b} & + & \epsilon \\ [15 \times 1] & & [15 \times 3] & [3 \times 1] & & [15 \times 1] \end{array}$$

Wanneer je matrix X vermenigvuldigt met matrix $\mathbf{b}$ dan vermenigvuldig je een 15×3 matrix met een 3×1 matrix. Dit levert een 15×1 matrix op. In een schema:

$$[15 \times 3] \times [3 \times 1] \quad \rightarrow \quad [15 \times 1]$$

Het is altijd zo dat in een matrixvermenigvuldiging het aantal kolommen in de linker matrix (3) gelijk moet zijn aan het aantal rijen in de rechter matrix (3). Als twee matrices vermenigvuldigd kunnen worden, dan levert dit altijd een nieuwe matrix op met het aantal rijen van de linker matrix (15) en het aantal kolommen in de rechter matrix (1).

4.1.3.7 Matricesberekeningen

4.1.3.7.1 Optellen en aftrekken van matrices Alleen twee matrices met gelijke dimensies kunnen bij elkaar worden opgeteld of van elkaar worden afgetrokken. Niks vreemds hier

$$\begin{bmatrix} 2 & 3 \\ 6 & 9 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 8 & 3 \\ 4 & 7 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 10 & 16 \\ 4 & 11 \end{bmatrix}$$

en

$$\begin{bmatrix} 2 & 3 \\ 6 & 9 \\ 4 & 8 \end{bmatrix} - \begin{bmatrix} 8 & 3 \\ 4 & 7 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} -6 & 0 \\ 2 & 2 \\ 4 & 5 \end{bmatrix}$$

4.1.3.7.2 Vermenigvuldigen van matrices Hier heb je een voorbeeld van een vermenigvuldiging:

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathit{4} & \mathit{9} \\ \mathit{3} & \mathit{2} \end{bmatrix} \times \begin{bmatrix} \mathbf{4} & \mathit{2} \\ \mathbf{1} & \mathit{4} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \cdot \mathbf{4} + \mathbf{2} \cdot \mathbf{1} & \mathbf{1} \cdot \mathit{2} + \mathbf{2} \cdot \mathit{4} \\ \mathit{4} \cdot \mathbf{4} + \mathit{9} \cdot \mathbf{1} & \mathit{4} \cdot \mathit{2} + \mathit{9} \cdot \mathit{4} \\ \mathit{3} \cdot \mathbf{4} + \mathit{2} \cdot \mathbf{1} & \mathit{3} \cdot \mathit{2} + \mathit{2} \cdot \mathit{4} \end{bmatrix} = \begin{bmatrix} 6 & 10 \\ 25 & 44 \\ 14 & 14 \end{bmatrix}$$

Door verschillende lettertypen te gebruiken hoop ik dat je ziet wat er gebeurt.

Op de eerste rij van de linker matrix staan net zoveel getallen als in de eerste kolom van de rechter matrix. Het element linksboven in antwoord van de vermenigvuldiging vind je door steeds de overeenkomstige waarden in de eerste rij van de linker matrix en de eerste kolom van de rechter matrix met elkaar te vermenigvuldigen:

$$\mathbf{1} \cdot \mathbf{4} + \mathbf{2} \cdot \mathbf{1} = 6$$

Deze procedure voer je voor iedere van de zes elementen in het antwoord uit.

Het schema in fig. 4 kan behulpzaam zijn (bron: Engo z.d.):

Let op dat het product $A \cdot B$ in het algemeen een andere waarde geeft dan $B \cdot A$! In officiele terminologie zeg je dat matrixvermenigvuldigingen *niet-commutatief* zijn.

Wanneer je drie matrices met elkaar vermenigvuldigd, als in $A \cdot B \cdot C$ dan maak het niet uit of je eerst $A \cdot B$ berekend (als in $(A \cdot B) \cdot C$) of eerst $B \cdot C$ (als in $A \cdot (B \cdot C)$). Men zegt dat matrixvermenigvuldigingen *associatief* zijn.

4.1.3.8 Vermenigvuldigen van een matrix met een skalar Zo nu en dan komt het voor dat een matrix met een enkele waarde (een *skalar*) wordt vermenigvuldigd.

Deze is gelukkig weer simpel:

$$2 \cdot \begin{bmatrix} 6 & 10 \\ 25 & 44 \\ 14 & 14 \end{bmatrix} = \begin{bmatrix} 2 \cdot 6 & 2 \cdot 10 \\ 2 \cdot 25 & 2 \cdot 44 \\ 2 \cdot 14 & 2 \cdot 14 \end{bmatrix} = \begin{bmatrix} 12 & 20 \\ 50 & 88 \\ 28 & 28 \end{bmatrix}$$

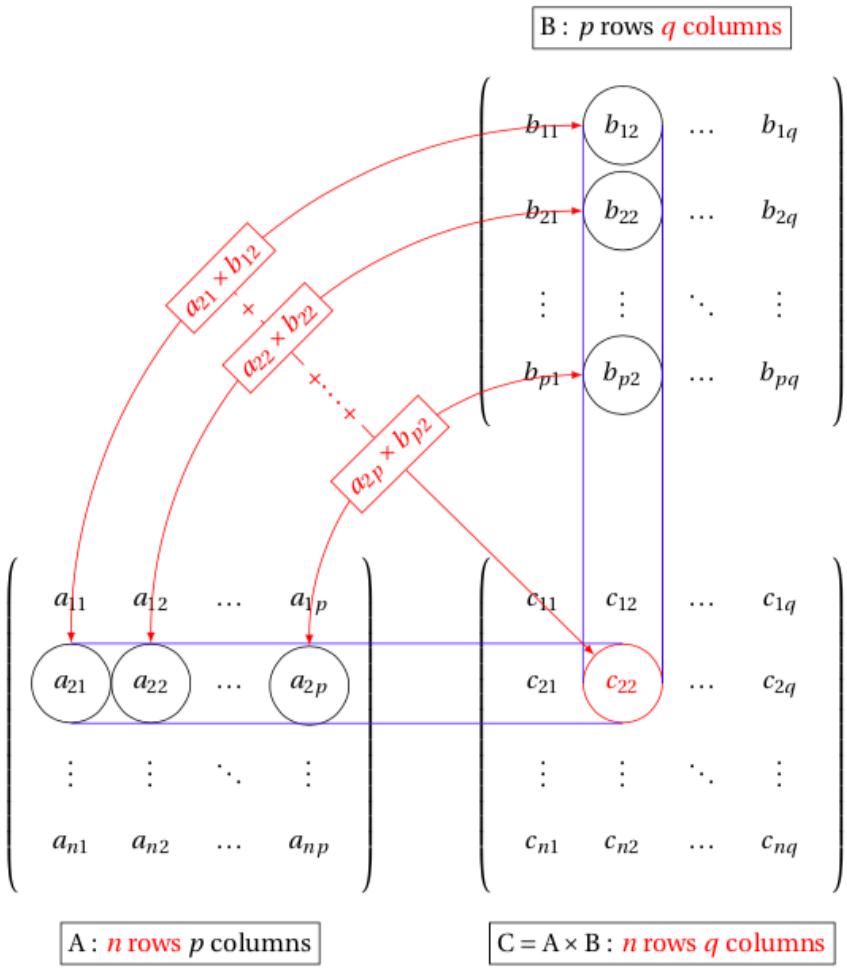
4.1.3.9 Delen van matrices Matrices kunnen niet op elkaar gedeeld worden. Een hele opluchting lijkt me.

Maar (er is altijd een maar), de deling 24/8 kunnen we ook berekenen als de vermenigvuldiging

$$24 \times \frac{1}{8} = 3$$

Bekend als “delen is hetzelfde als vermenigvuldigen met het omgekeerde”.

Matrices kunnen vermenigvuldigd worden. De vraag is, kunnen ze ook “omgekeerd” worden. Het antwoord op deze vraag is “ja” (soms).



Figuur 4: Schema voor matrixvermenigvuldigingen (Engo z.d.).

4.1.3.10 Inverse van een matrix De docent H. Hofstede legt het allemaal heel netjes uit (Hofstede z.d.):

Wanneer we zeggen dat we we een matrix A hebben waarvan de inverse matrix B is, dan betekent dat

$$A \cdot B = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1b_1 + a_2b_3 & a_1b_2 + a_2b_4 \\ a_3b_1 + a_4b_3 & a_3b_2 + a_4b_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

De vermenigvuldiging van A en B hebben we gezien. En net zoals geldt dat

$$8 \cdot \frac{1}{8} = 1$$

moet een matrix vermenigvuldigd met zijn inverse matrix de matrix variant van 1 opleveren. En dat staat hierboven nu precies: rechts staat de identiteitsmatrix I_2.

We hebben hier een stelsel van vier vergelijkingen:

$$\begin{aligned} a_1b_1 + a_2b_3 &= 1 \\ a_1b_2 + a_2b_4 &= 0 \\ a_3b_1 + a_4b_3 &= 0 \\ a_3b_2 + a_4b_4 &= 1 \end{aligned}$$

Wanneer je dit netjes uitwerkt dan vind je

$$b_1 = \frac{a_4}{a_1a_4 - a_2a_3}, \quad b_2 = \frac{-a_2}{a_1a_4 - a_2a_3}, \quad b_3 = \frac{-a_3}{a_1a_4 - a_2a_3}, \quad b_4 = \frac{a_1}{a_1a_4 - a_2a_3}$$

Je ziet dat de noemer $a_1a_4 - a_2a_3$ steeds weer terugkomt. Deze staat bekend onder de naam determinant: wanneer de determinant ongelijk is aan nul, dan is de inverse van de betreffende matrix te berekenen. Wanneer de determinant van een matrix nul is, dan is de matrix *singulier*.

In het algemeen is de inverse van een 2×2 matrix als volgt te berekenen:

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

4.1.3.11 Lineaire Regressie met matrixberekening In een van de vorige secties hebben we een lineair model beschreven met behulp van matrices:

$$Y = X\mathbf{b} + \epsilon$$

We hebben gezegd dat om $\mathbf{b}$ zo gaan berekenen zodat de som van de elementen in ϵ zo klein mogelijk is. We zijn op dit moment niet zo geïnteresseerd in de exacte waarde van die som. Om die reden laten we ϵ uit de berekening.

$$Y = X\mathbf{b}$$

Met de volgende uitdrukking berekenen we $\mathbf{b}$:

$$\mathbf{b} = (X^T X)^{-1} X^T Y$$

Wanneer je beseft dat b, X, en Y matrices zijn, dan zie je dat het een flinke rekenpartij is. Maar alles hebben we behandeld en we zouden de berekening moeten kunnen uitvoeren.

5 De programmeertaal C

5.1 Introductie C

In dit gedeelte hebben we een eerste kennismaking met de taal C.

5.1.1 Doel

- Je kan de taal C in een historische context plaatsen
- Je kan een eenvoudige C applicatie schrijven met functies
- Je kan een C applicatie compileren en uitvoeren

5.1.2 Opdracht

5.1.2.1 Tijdlijn Maak een tijdlijn met daarop de jaartallen voor de volgende gebeurtenissen

- introductie MS-DOS
- introductie Windows 3.11
- introductie C
- introductie C++
- introductie Java
- introductie Fortran
- introductie OO
- introductie Pascal

- introductie Linux
- introductie Python
- introductie Go
- introductie Unix

5.1.2.2 Weersgegevens Schrijf een applicatie waarin je weergegevens kan converteren. Voeg een aparte functie toe voor ieder van de volgende conversie:

- converteer graden Celsius naar graden Fahrenheit.
- converteer graden Fahrenheit naar graden Celsius.
- converteer windsnelheid in km/h naar windkracht.

Let op: er zijn meerdere manieren om de berekening van windsnelheid naar windkracht te maken. Een `if-else` structuur voldoet voor deze opdracht.[23]

Extra eisen:

- temperatuur en windsnelheid zijn floating point waarden.
- windkracht is een integer type.
- Je functies hebben de volgende declaraties:

  ```
  float temperatureToF(float c);
  float temperatureToC(float f);
  int windToBeaufort(float speed);
  ```

- functie `main` bevat een aantal voorbeelden voor ieder van de functies.

  Deze functie schrijft voor iedere functie de conversie netjes weg, als:

  ```
  windsnelheid 80.0 km/h is windkracht 9
  0.0 graden Celcius is 32.0 graden Fahrenheit
  100.0 graden Fahrenheit is 37.8 graden Celcius
  ```

- functie `main` komt boven de conversiefuncties.
- code is ingecheckt in *subversion*.

5.1.3 Toelichting

5.1.3.1 hello world Het meest eenvoudige C programma is

```
#include <stdio.h>

int main() {
    printf("hello world\n");
}
```

[23] Wil je toch gebruik maken de vergelijking die je gevonden hebt, dan zal je waarschijnlijk gebruik maken van de `pow`-functie. Dat kan als je in je code de juiste *header file* toevoegt en de *linker* vertelt dat deze ook de *math library* mee moet linken. Dit kan door `-lm` toe te voegen aan de `gcc`-opdracht. Wanneer je een `Makefile` gebruikt moet je expliciet aangeven dat je gebruik maakt van een *linker* optie: Voeg aan je `Makefile` de regel "`LDLIBS=-lm`" toe.

Deze applicatie maakt gebruik van functie `printf` uit de standaard bibliotheek. Deze functie is gedeclareerd in `stdio.h`.

Iedere applicatie in C heeft een functie `main()`. De applicatie start met het uitvoeren van deze functie en de applicatie stopt wanneer deze functie doorlopen is.

Laten we zeggen dat deze functie is opgeslagen in bestand `hello.c`.

5.1.3.2 compileren De code uit de vorige sectie is te compileren met het volgende commando:

```
$ gcc -std=c99 -Wall hello.c -o hello
```

Je geeft aan `gcc` een aantal opties mee:

`-std=c99`: Geeft aan welke versie van C je gebruikt, wij maken in het algemeen gebruik van C99.
`-Wall`: Klinkt als "geef alle waarschuwingen", in de praktijk krijg je de meeste waarschuwingen.
`-o hello`: De uitvoer wordt geschreven naar bestand `hello`.

Wanneer je deze opdracht uitvoert en er zitten geen fouten in je code, dan levert deze actie bestand `hello` op. Deze is uit te voeren met:

```
$ ./hello
```

Om te voorkomen dat je "per ongeluk" deze applicatie uitvoert, moet je aangeven waar de applicatie te vinden is, vandaar de "`./`".

Het bouwen van de applicatie verloopt in een aantal stappen:[24]

1. *Preprocess*

   De preprocessor lost alle `#` instructies op. In het geval van de bovenstaande code wordt de inhoud van de *header file* `stdio.h` ingevoegd.

2. *Compile*

   De inhoud van de *source code file* wordt omgezet naar een *object file*.

   Deze komen we later nog tegen als tussenproduct.

3. *Link*

   Alle benodigde binaire bestanden worden samengevoegd tot de executable.

5.1.3.3 Functiedeclaraties en -definities C ontstond lang voordat object geörienteerd programmeren onstond. C heeft dan ook geen klassen. C heeft wel functies. In C code moet een functie bekend zijn voor je deze gebruikt.

Je kan een functie *declareren* en je kan een functie *definiëren*. Een functiedeclaratie zegt zoveel als dat een functie bestaat, een functiedefinitie beschrijft de werking van een functie. Zie de volgende code:

[24] Volgens cpp reference bestaat het hele proces uit *zeven fasen*, maar het voldoet de bovenstaande drie te kennen. Maar, een belangrijk gevolg van deze zeven fasen is dat commentaar vervangen is door een spatie voordat de preprocessor aan het werk gaat.

```
#include <stdio.h>

// declaratie van bar, naam parameter is optioneel
int bar(int);

// definitie van foo (is meteen de impliciete declaratie)
float foo() {
    bar(18);
    return 42;
}

int main() {
    // main kan gebruik maken van foo en bar
    // omdat beide bekend zijn
    printf("foo is %.2f\n", foo());
    bar(109);
}

// nu pas de definitie van bar
int bar(int val) {
    printf("bar is %d\n", val);
    return 3*val;
}
```

Wat print deze applicatie trouwens uit?

5.2 Codeorganisatie in C

5.2.1 Doel

- Je kan code onderverdelen in code en header files
- Je kan meerdere bestanden met `gcc` compileren
- Je kan meerdere bestanden met een `Makefile` compileren

5.2.2 Opdracht

Deel de conversiecode die je in het voorgaande onderwerp hebt geschreven zinvol op in `main.c`, `conversion.h` en `conversion.c`. Compileer de code met `gcc` via een `Makefile`.

Check het resultaat in op *Subversion*.

5.2.3 Toelichting

5.2.3.1 header en source files We gaan onze code opdelen in meerdere bestanden. Redenen om dit te doen zijn de volgende:

- Dit geeft je de mogelijkheid om code in logische blokken op te delen.
- Het compileerproces zal uiteindelijk sneller verlopen omdat alleen bestanden die gewijzigd zijn opnieuw gecompileerd hoeven te worden.

We hebben onze main code in `main.c`:[25]

```
#include "hello.h"

int main() { hello(); }
```

Met `#include "hello.h"` geven we aan dat het *lokale bestand* `hello.h` moet worden ingevoegd.[26]

Eerder hebben we als `#include <stdio.h>` gezien, met vishaken in plaats van met quotes. Met de vishaken geef je aan dat een bestand in een geïnstalleerde bibliotheek ingevoegd moet worden (het bestand `stdio.h` staat niet bij je `main.c` in de buurt).

Terug naar `main.c`, deze maakt gebruik van de declaraties in `hello.h`:

```
#ifndef hello_h
#define hello_h

void hello();

#endif
```

O, in een keer een overdosis aan *preprocessor instructions*: als het "symbool" `hello_h` niet bestaat, defineer het dan.

Met deze instructies wordt voorkomen dat tijdens de preprocessor fase dit bestand meerdere keren wordt ingelezen. De eerste keer dat `hello.h` wordt ingelezen, wordt het hele betand gelezen en wordt symbool `hello_h` (inderdaad, de naam van de *header file*) gedefinieerd. De tweede keer negeert de preprocessor de inhoud tot deze `#endif` tegenkomt.

De declaraties in `hello.h` zijn gedefinieerd in `hello.c`:

```
#include "hello.h"
#include <stdio.h>

void hello() { printf("%s\n", "hello world!"); }
```

Allereerst voegen we `hello.h` in (dit is standaardprocedure). Omdat `hello()` gebruik maakt van `printf()` voegen we ook `stdio.h` in. We voegen `stdio.h` in `hello.c` en niet in `hello.h` in omdat de gebruiker van `hello.h` niet hoeft te weten dat we `printf()` gebruiken.

5.2.3.2 Compileren vanaf de command line Bovenstaande code is op verschillende manieren te formateren. Eén manier is:

```
$ gcc -Wall -std=c99 -o main main.c hello.c
```

We compileren met `gcc`, we willen alle *warnings* (`-Wall`), we compileren voor C99 (`-std=c99`), de uitvoer gaat naar `main` (`-o main`) en tot slot geven we de invoer aan (`main.c` en `hello.c`).

Je kan ook eerst `hello.c` naar een zgn. *object file* compileren:

[25] De code is wellicht wat opmerkelijk geformateerd, dit is het gevolg van de automatische formatering.

[26] Het zijn de quotes "`"`" die maken dat een lokaal bestand ingevoegd wordt.

```
$ gcc -Wall -std=c99 -c hello.c
```

Dit is mogelijk voor ieder code bestand zonder `main()` functie (deze laatste compileren naar een *executable*).

Wanneer we files afzonderlijk compileren kunnen we tijdwinst behalen omdat alleen gewijzigde bestanden gecompileerd hoeven worden. Nu we `hello.o` hebben, kunnen we de *executable* compileren:

```
$ gcc -Wall -std=c99 -o main main.c hello.o
```

We hebben nu op twee manieren de *executable* `main`. Deze kunnen we als volgt uitvoeren:

```
$ ./main
hello world!
```

Denk aan de `./`, zonder expliciet pad naar je *executable* wil je omgeving je opdracht niet uitvoeren.

5.2.3.3 Compileren met een eenvoudige Makefile Ben je al klaar met het steeds `-Wall -std=c99 -o ...` te typen? We kunnen gebruik maken van `make`, een al wat ouder systeem om software te bouwen. Het programma `make` weet hoe C code gecompileerd te worden. Wanneer al onze code in `helloworld.c` zou zitten, dan kunnen we `make` op de volgende manier gebruiken:

```
$ make helloworld
```

Onze `make` ziet dat er een bestand `helloworld.c` in de map staat en gaat er vanuit dat daarmee `helloworld` wel te maken zit.

In ons voorbeeld hebben we het onszelf iets ingewikkelder gemaakt. We moeten `make` vertellen dat `main` ook afhankelijk is van `hello.o`. Onze `make` ziet dat er wel een bestand `hello.c` is, en gaat er vanuit dat daarmee `hello.o` gebouwd kan worden.

Plaats de volgende inhoud over in het bestand `Makefile` (met een hoofdletter `M`) in dezelfde map als `main.c`, `hello.c` en `hello.h`:

```
CFLAGS=-std=c99 -Wall

main: hello.o
```

In dit bestand geven we aan dat `main` afhangt van `hello.o`, tegelijk geven we aan dat we met een aantal opties willen compileren (`-std=c99` en `-Wall`).

Nu we het bestand `Makefile` hebben kunnen we onze applicatie als volgt bouwen:

```
$ make
```

Dat is het. We gaan in Makefiles in op meer mogelijkheden met *make files*.

Je bent nu begonnen naar het schrijven van een volledige make file. Tot op het niveau waarop wij make files gaan schrijven is dat een leuke hobby. Pas op dat je je niet volledig gaat verliezen in het maken van make files, je kan er een *day time job* van maken.

Toch heel kort de essentie van hoe we in een make file aangeven hoe we iets bouwen:

```
<target>: <dependencies>
<tab><recipe>
```

In ons voorbeeld hebben we de *target* `all`, in dit geval gewoon een string waarmee we de lezer laten zien dat met deze regel alles gemaakt wordt. De *dependency* is `hello` die `make` herkent als gerelateerd aan `hello.c`. De *recipe* is in dit geval leeg omdat voor standaard problemen `make` weet hoe er gecompileerd moet worden. Mocht je ooit een keer een recipe moeten schrijven, let er dan op dat iedere regel met precies één `<tab>` teken begint.

5.3 C Tooling

C is niet altijd een even gemakkelijke taal om in te programmeren. Gelukkig zijn er meerder tools die helpen met het hoog houden van je codewaliteit. In dit onderdeel installeer je de tools en maak je een begin deze tools te gebruiken.

5.3.1 Doel

- Je leert je compileerproces te vereenvoudigen met behulp van `make`.
- Je leert je code te formateren met `clang-format`.
- Je leert unittests voor C te schrijven.
- Je leert een aantal tools om debuggen te vereenvoudigen:
  - `assert`
  - voorwaardelijk printen
- Je leert fouten op te sporen met `cppcheck`.
- Je leert dit alles te combineren in één `Makefile`.

5.3.2 Opdracht

5.3.2.1 Bestaande code testen We hebben in voorgaande secties code geschreven om temperatuur te converteren van de ene eenheid in de andere en om windkracht uit te drukken in Beaufort.

Vul deze code aan met unittests en een `Makefile`. Met de opdracht `make` moet:

- de code gecontroleerd worden met `cppcheck`,
- de unittest gebouwd en uitgevoerd worden,
- de code geformateerd worden met `clang-format`,
- de in dit proces gebouwde bestanden weer worden opgeruimd.

5.3.2.2 Verdiep je in debugmethoden In de toelichting in deze sectie worden een aantal voorbeelden uitgediept die niet aan de orde gekomen zijn bij de inleiding van dit gedeelte. Werk de onderdelen waar je nog moeite mee hebt een keer door. Vergeeft voorval gebruik van `printf` en de asserts niet. Deze zijn nog niet aan de orde gekomen en je moet ze wel beheersen: kijk of de voorbeelden ook op je Raspberry werken.

5.3.3 Toelichting

5.3.3.1 Code formatting Je moest eens weten hoeveel tijd verloren gaat aan zinloze discussies over hoe code geformateerd moet worden. Iedereen heeft zijn persoonlijke voorkeur, er zijn minstens tweemaal zoveel stijlen als er ontwikkelaars zijn.

Stop ermee. Gebruik een code formatter en accepteer de resultaten.

In deze course gebruiken we `clang-format`.

Zorg ervoor dat je code geformateerd is voor je deze commit. Hiermee voorkom je dat in git wijzigingen worden opgeslagen die alleen maar het gevolg zijn van herformateren.

Je moet de formatter eenmaal installeren, waarschijnlijk iets als:

```
$ sudo apt-get install clang-format
```

Eenmaal geïnstalleerd formateer je alle code in een map met:

```
$ clang-format -style='{PointerAlignment: Left}' -i *.c *.h
```

dat wil zeggen, zolang je je code alleen in `.c` en `.h` bestanden hebt staan.

Is alles geformateerd met `clang-format` geweldig? Neuh, ik ben zelf niet zo te spreken over de volgende code:

```
void hello() { printf("%s\n", "hello world!"); }
```

Maar zoals ik al zei: gewoon accepteren en verder gaan met software ontwikkelen. Echt, geloof me, het is het gewoon niet waard op zoek te gaan naar de volgende tool (die het ook niet perfect doet).

5.3.3.2 Cpp Check Er zijn allerlei tools waarmee je code kan controleren. Eén zo'n tool is cpp check. Mocht je deze nog niet hebben, dan installeer je deze als vanouds met:

```
$ sudo apt-get install cppcheck
```

In de *root* van je code kan je je code controleren met[27]:

```
$ cppcheck --enable=all --inconclusive --std=c99 .
```

Wanneer je het resultaat van cppcheck (of van wat voor een andere command line tool dan ook) in een bestand wil opslaan, doe je dat door de uitvoer naar een bestand te sturen.[28]

```
$ cppcheck --enable=all --inconclusive --std=c99 . \
    > check.out
```

Wanneer `check.out` al bestaat en je wilt uitvoer toevoegen, gebruik dan `>>` in plaats van `>`.

[27] Wanneer je klaar bent met opmerkingen over het niet vinden van de bestanden in de standaardbibliotheek, kan je deze onderdrukken met de extra optie `--suppress=missingIncludeSystem`.

[28] Wederom, een opdracht kan op meerdere regels getypt worden door deze te splitsen met "\". Wanneer je deze opdracht op één regel overtypt, laat dan de "\" achterwege.

Kijk kritisch naar de foutmeldingen. De meeste zijn de moeite waard om op te lossen, maar soms is er sprake van een *false positive*, deze kan je dan met goede reden negeren.

5.3.3.3 Unit tests Er is geen standaard ondersteuning voor *unit tests* in C. We moeten een keuze maken uit één van de vele opties. In navolging van Klemens (2015, 52) maken we gebruik van unit tests in glib.

5.3.3.3.1 Installeer GLib Welkom in de wondere wereld van C dependencies. We moeten een externe library installeren om te kunnen unit testen. Dit kan tot allerlei onverwachte en soms lastig op te lossen problemen leiden.

Laten we hopen dat we GLib kunnen installeren met:

```
$ sudo apt-get install glib2.0-dev
```

Met wat geluk kunnen we geholpen worden met `pkg-config`, installeer ook deze:

```
$ sudo apt-get install pkg-config
```

Wanneer we beide gevonden hebben, kunnen we het volgende uitvoeren:

```
$ pkg-config --cflags glib-2.0
```

(de uitvoer wordt hier niet getoond). Wat we als antwoord krijgen zijn precies de compileer opties die we nodig hebben om in onze code gebruik te maken van `glib`.

5.3.3.3.2 Unittest Voorbeeld Hier volgt weer een mal stukje functionaliteit om als voorbeeld te gebruiken voor je unit tests.

Stel we hebben de volgende, bijzonder complexe, code in `main.c`:

```
#include "adder.h"
#include <stdio.h>

int main() {
  int a = 2, b = 6;

  printf("%d + %d = %d\n", a, b, add(a, b));
}
```

Deze code maakt gebruik van de functie `add(a, b)` en we willen er graag voor zorgen dat deze code goed werkt. Dat betekent *unit tests*.

De functie `add()` wordt gedelareerd in `adder.h`:

```
#ifndef adder_h
#define adder_h

// add two int values
int add(int, int);

#endif
```

Ik ben benieuwd hoe deze functie is gedefinieerd in `adder.c`:

```
#include "adder.h"

// addTwoValues is not exposed in adder.h
// it is the function that does the difficult task of
// adding two integer functions.
int addTwoValues(int a, int b) { return a + b; }

// add is declared in adder.h.  Users of adder.h may think
// that add does the difficult addition, however, it merely
// delegates its task to addTwoValues().
int add(int a, int b) { return addTwoValues(a, b); }
```

Ahhh, de functie `add()`, beschikbaar gemaakt in `adder.h` maakt gebruik van `addTwoValues()` die als het ware als een *private* functie wordt beschouwd (of meer precies, die zomaar zou kunnen veranderen zonder dat je dit in `adder.h` merkt).

Ik ben eigenwijs en ik wil beide functies kunnen unit testen (of beter, ik wil ook functies die niet in de *header file* staan toch kunnen testen, en daar is dit een (slecht?) voorbeeld van).

Hier volgt het bestand met onze unit tests, voor de gelegenheid maar `unittest.c` genoemd.

```
#include "adder.h"
#include <glib.h>

// declare functions local to adder.h
int addTwoValues(int a, int b);

void test_add() {
  g_assert_true(add(1, 2) == 3);     // bool test
  g_assert_cmpint(add(3, 4), ==, 7); // int comparison test
}

void test_addTwoValues() {
  int i = addTwoValues(18, -18);
  g_assert_cmpint(i, <, 1);
}

int main(int argc, char** argv) {
  g_test_init(&argc, &argv, NULL);

  g_test_add_func("/add/1", test_add);
  g_test_add_func("/add/2", test_addTwoValues);

  return g_test_run();
}
```

Met functie `test_add)` wordt een functie toegevoegd waarin `add()` wordt getest. Over de test zometeen meer. Op regel 20 wordt deze functie `test_add` toegevoegd aan de de testen functies. Let er dus op dat wanneer je iets wilt testen, je én een functie met tests moet schrijven én deze functie moet toevoegen aan de set met te testen functies.

De eerste regel code in de test in functie `test_add()` ziet er vrij normaal uit. Met `g_assert` wordt getest of een waarde nul of niet-nul is. Feitelijk hebben we hier te maken met een test op *boolean values*. Je verwacht dat `add(0,0)` als resultaat `0` levert. Dit is in C een manier om `false` aan te geven. De test slaagt wanneer het argument *niet-nul* (ofwel `true`) is, vandaar de logische *not* `!`.

De tweede regel code in de test in functie `test_add()` ziet er mal uit. Dat komt omdat `g_assert_cmpint` (net als `g_assert` trouwens) geen normale functies zijn, maar macro's. Een uitgebreide toelichting vind je in de referentie. Op dezelfde pagina vind je meer manieren om tests uit te voeren.

Maar, `addTwoValues()` staat toch helemaal niet in `adder.h` dat op regel 1 wordt ingevoegd? De oplossing staat op regel 5: hier wordt `addTwoValues()` nogmaals gedeclareerd zodat de compiler bij het compileren van `unittest.c` niet overstuur raakt omdat het niet weet wat `addTwoValues()` voor een functie is.

5.3.3.3.3 Geholpen door een Makefile Hoe draaien we de unittest? Doordat de code van GLib afhankelijk is, is het intypen van het `gcc` opdracht op de *command prompt* geen pretje meer. Om dit te voorkomen maken we gebruik van een iets uitgebreidere variant van de `Makefile`. Deze staat in dezelfde map als `main.c`, `adder.c` en `adder.h`.

```
CFLAGS=-std=c99 -Wall `pkg-config --cflags glib-2.0`
LDLIBS=`pkg-config --libs glib-2.0`

all: unittest run

unittest: adder.o

main: adder.o

run:
    ./unittest
```

Op de vierde staat de eerste afhankelijkheid: `all` is afhankelijk van `unittest`. Dit is de regel die wordt uitgevoerd wanneer je `make` uitvoert. Deze sluit af met het uitvoeren van `./unittest`, maar voor die tijd wordt de afhankelijkheid `unittest` op regel 7 uitgevoerd. Op die regel wordt `unittest` gebouwd (nadat zonodig ook `adder.o` is gebouwd). En dat betekent compileren.

Compileren met de afhankelijkheid naar GLib is een feestje, en dit wordt geholpen door `CFLAGS` op regel 1 voor het compileren en `LDLIBS` op regel 2 voor het linken. In beide regels komt `pkg-config` voor, en *let op*, de gebruikte quotes zijn zgn. *back ticks* en niet de standaard *single quotes*. Het resultaat is dat `pkg-config` wordt uitgevoerd zoals we dat ook in het eerste deel van deze sectie hebben gezien.

Uiteindelijk is het resultaat:

```
$ make
make
cc -std=c99 -Wall `pkg-config --cflags glib-2.0` \
    -c -o adder.o adder.c
cc -std=c99 -Wall `pkg-config --cflags glib-2.0` \
```

```
    unittest.c adder.o  `pkg-config --libs glib-2.0` \
    -o unittest
./unittest
/add/1: OK
/add/2: OK
```

Op de eerste paar regels zie je de uitvoer van het compileer en link proces, op de laatste drie regels zie je dat de unittest wordt uitgevoerd en dat alle tests met het verwachte resultaat worden uitgevoerd.

5.3.3.3.4 Test setup en teardown Wanneer je wel eens vaker serieus unittests hebt uitgevoerd, dan weet je dat bovenstaande niet het hele verhaal is. Een beetje unittest heeft de mogelijkheid om een test op te zetten (*test set up*) en om de rommel weer op te ruimen (*test tear down*).

Dat is ook mogelijk in het hier gepresenteerde test-framework. In plaats van `g_test_add_func` moet je dan `g_test_add` gebruiken die ook de mogelijkheid geeft om je *set up* functie en *tear down* functie aan te geven. Zie bijvoorbeeld Klemens (2015, 52) voor meer informatie.

5.3.3.4 Debugging

5.3.3.4.1 Print waarden Nog gebruikt door vele ontwikkelaars: `printf`. En niet zonder reden. Hier en daar een `printf` doet wonderen om te snappen wat er gebeurt.

Stel, we willen 14! (dat is 14 faculteit, ofwel `1*2*3* .. *13*14` uitrekenen).

```
#include <stdio.h>

int main() {
  int ans = 1;
  int f = 14;

  for (int i = 1; i <= f; i++) {
    ans *= i;
  }

  printf("%d! = %d\n", f, ans);
}
```

Deze code, opgeslagen in `main.c`, is te bouwen en uit te voeren:

```
$ make main
$ ./main
14! = 1278945280
```

Zoals je kan verwachten, een groot getal. Gelukkig kan mijn HP rekenmachine ook faculteiten uitrekenen. Even kijken, $14! = 87\,178\,291\,200$, dat is ook een groot getal, maar wel een ander groot getal. We hebben een probleem.

Laten we tussenresultaten uitprinten:

```
#include <stdio.h>
```

```
int main() {
  int ans = 1;
  int f = 14;

  for (int i = 1; i <= f; i++) {
    ans *= i;
#ifndef NDEBUG
    printf("* %2d %15d\n", i, ans);
#endif
  }

  printf("%d! = %d\n", f, ans);
}
```

We hebben onze testcode netjes ingepakt tussen `#ifndef` en `#endif` zodat we de test naar believen kunnen meecompileren of niet. Het symbool `NDEBUG` wordt vaker gebruikt om aan te geven dat je *niet* wilt debuggen. We sluiten hier aan bij deze conventie, maar let er dus op dat je `#ifndef` en niet `#ifdef` gebruikt!

Laten we de code met tests compileren:

```
$ gcc -std=c99 -Wall -o main main.c
```

We draaien de test:

```
$ ./main
*  1               1
*  2               2
*  3               6
*  4              24
*  5             120
*  6             720
*  7            5040
*  8           40320
*  9          362880
* 10         3628800
* 11        39916800
* 12       479001600
* 13      1932053504
* 14      1278945280
14! = 1278945280
```

Dat ziet er helemaal niet gek uit, tot we bij `*14` komen: het getal `1278945280` is kleiner dan het getal bij `*13`. En wacht eens, het getal bij `*13` is helemaal niet 13 keer groter dan het getal bij `*12`.

Stom natuurlijk, we hebben `int` als type voor `ans` genomen, en dat is bij een berekening van een faculteit niet handig. Laten we kijken wat een `long int` doet in plaats van een `int`:

```
#include <stdio.h>

int main() {
  long int ans = 1;
  int f = 14;

  for (int i = 1; i <= f; i++) {
```

```
    ans *= i;
#ifndef NDEBUG
    printf("* %2d %15ld\n", i, ans);
#endif
  }

  printf("%d! = %ld\n", f, ans);
}
```

Laten we kijken zonder de debug code mee te nemen in de bouw:

```
$ gcc -Wall -std=c99 -D NDEBUG -o main main.c
$ ./main
14! = 87178291200
```

De optie `-D NDEBUG` wordt hier toegevoegd waarmee het symbool `NDEBUG` gedefinieerd wordt. Per saldo wordt het debuggen hiermee uitgeschakeld. Je snapt hoe je hetzelfde effect in je `Makefile` voor elkaar kan krijgen.

Het resultaat ziet er een stuk beter uit, maar ja, hoe ver kunnen we `f` laten stijgen voor we weer in de problemen zitten?

5.3.3.4.2 Asserts In dit gedeelte bekijken we de mogelijkheden die `assert` ons geeft.

We kijken weer naar een triviaal voorbeeld in de hoop dat de mogelijkheden van `assert` duidelijk worden.

```
#include <assert.h>
#include <stdio.h>

// precondition: b != 0
int div(int a, int b) {
  assert(b != 0);

  return a / b;
}

int main() {
  int a = 5, b = 2;

  printf("%d / %d = %d\n", a, b, div(a, b));

  b = 0; // even though we have a pre condition for b
  printf("%d / %d = %d\n", a, b, div(a, b));
}
```

In deze code hebben we een functie `div()` waarmee twee getallen op elkaar gedeeld worden. In het commentaar boven `div()` staat dat het getal waardoor je deelt niet nul mag zijn. Omdat dit een pre-conditie is, betekent dat de rest van de applicatie ervoor verantwoordelijk is dat dat dan ook niet gebeurt.

We compileren deze code in `main.c` en voeren deze uit:

```
$ gcc -std=c99 -Wall -o main main.c
$ ./main
```

```
5 / 2 = 2
Assertion failed: (b != 0), function div,
    file main.c, line 6.
Abort trap: 6
```

We worden hier op niet-mis-te-verstane manier op de hoogte gesteld dat onze code de pre-conditie van `div()` breekt.

Het is nu onze verantwoordelijkheid ervoor te zorgen dat de code zo verandert wordt dat de deling door nul niet meer kan voor komen. Met andere woorden: `assert` is *niet* bedoeld om problemen te voorkomen, het is bedoeld om pre- en post-condities te controleren.

Op het moment dat we de code hebben aangepast en we willen deze in productie nemen, dan kunnen we de code compileren zonder de `assert`:

```
$ gcc -std=c99 -Wall -D NDEBUG -o main main.c
$ ./main
5 / 2 = 2
Floating point exception: 8
```

Ok, nu hebben we dus gewoon te maken met een incompetente ontwikkelaar (of met een die een voorbeeld wil stellen). De `assert` in de code is buiten werking gesteld door het symbool `NDEBUG` te definieren (zie de documentatie), maar men heeft nagelaten de code zo te veranderen dat aan de pre-condities wordt voldaan.

5.3.3.4.3 Gnu Debugger De ultieme droom van iedere C ontwikkelaar is `gdb`, de Gnu Debugger te beheersen. Met deze debugger kan je een programma volgen wanneer deze draait, maar ook achteraf nagaan waarom deze gecrashed is.

De tijd ontbreekt om hier in de course aandacht aan te besteden. Wel wil ik je wijzen op deze tutorial, het is er een uit de serie van "Beej Guides" en ziet er zo op het eerste gezicht goed uit.

5.4 Elementaire constructies in C

5.4.1 Doel

- Je leert hoe C met boolean waarden om gaat.
- Je leert met `typedef` eigen definities te maken.
- Je leert met `sizeof` na te gaan hoeveel bytes een waarde inneemt.
- Je leert enumeraties te maken en te gebruiken.
- Je leert `struct` te maken en te gebruiken.
- Je leert zo efficiënt mogelijke variabelen te kiezen.

5.4.2 Opdracht

Schrijf een applicatie met de bestanden `Makefile`, `unittest.c`, `weer.h`, `weer.c` zo dat je bij je bij het uitvoeren van

```
$ make
```

de volgende uitvoer krijgt:

```
clang-format -i *.c *.h
cc -std=c99 -Wall `pkg-config --cflags glib-2.0` \
    -c -o weer.o weer.c
cc -std=c99 -Wall `pkg-config --cflags glib-2.0` \
    unittest.c weer.o  `pkg-config --libs glib-2.0` \
    -o unittest
./unittest
/test/1: OK
/test/2: OK
/test/3: OK
/test/4: OK
/test/5: OK
cppcheck -q --enable=all --inconclusive  --std=c99 .
(information) Cppcheck cannot find all the include files
(use --check-config for details)
rm *.o
rm unittest
```

Vooruit, je krijgt `unittest.c`:

```
#include "weer.h"
#include <glib.h>
#include <stdio.h>

void test_1() {
  Verwachting v = {
      .neerslag = 22, .temperatuur = 38.0, .windkracht = 6};
  Code c = waarschuwing(v);

  g_assert_cmpint(c, ==, ROOD);
}

void test_2() {
  Verwachting v = {
      .neerslag = 42, .temperatuur = 18.0, .windkracht = 2};
  Code c = waarschuwing(v);

  g_assert_cmpint(c, ==, GEEL);
}

void test_3() {
  Verwachting v = {
      .neerslag = 0, .temperatuur = 25.0, .windkracht = 2};
  Code c = waarschuwing(v);

  g_assert_cmpint(c, ==, GROEN);
}

void test_4() {
  Verwachting v = {
      .neerslag = 22, .temperatuur = 38.0, .windkracht = 6};
  Code c = waarschuwing(v);
```

```
  if (g_test_subprocess()) {
    bericht(c, v);
    return;
  }

  g_test_trap_subprocess(NULL, 0, 0);
  g_test_trap_assert_stdout(
      "!! Weeralarm: neerslag: 22 mm, temperatuur: 38.0 "
      "celsius, windkracht: 6\n");
}

void test_5() {
  Windkracht w = 8;
  Verwachting v = {
      .neerslag = 18, .temperatuur = 16.0, .windkracht = w};

  g_assert_cmpint(sizeof(v.windkracht), ==,
                  1); // altijd en overal!
}

int main(int argc, char** argv) {
  g_test_init(&argc, &argv, NULL);

  g_test_add_func("/test/1", test_1);
  g_test_add_func("/test/2", test_2);
  g_test_add_func("/test/3", test_3);
  g_test_add_func("/test/4", test_4);
  g_test_add_func("/test/5", test_5);

  return g_test_run();
}
```

En, omdat het vandaag een mooie dag is, ook de eerste regels van `Makefile`:

```
CFLAGS=-std=c99 -Wall `pkg-config --cflags glib-2.0` -save-temps
LDLIBS=`pkg-config --libs glib-2.0`

all: fmt check clean

unittest: weer.o
```

Verder zal je het zelf moeten doen. In de code zitten de nodige structs en enumeraties. Verder mag je er van uitgaan dat windkracht nooit groter is dan 12.

Tot slot, zo maar wat bij elkaar verzonnen grenzen. De code is rood wanneer tenminste één van het volgende geldt:

- neerslag meer dan 80 mm
- temperatuur hoger dan 35 graden of lager dan -15 graden
- windkracht hoger dan 8 beaufort

In het geval van code rood begint het weerbericht met `!! Weeralarm:`.

De code is geel wanneer de code niet rood is en ten minste één van het volgende

geldt:

- neerslag meer dan 30 mm
- temperatuur hoger dan 30 graden of lager dan -10 graden
- windkracht hoger dan 5 beaufort

In het geval van code geel begint het weerbericht met `Pas op vandaag:`.

In alle andere gevallen is de code groen. Het weerbericht begint met `Mooi weer vandaag:`.

Uiteraard check je je resultaat in volgens de conventies.

5.4.3 Toelichting

5.4.3.1 boolean waarden Booleaanse waarden is nu typisch iets waarbij je ziet dat C een wat oudere taal is. Het type bestaat simpelweg niet.

In het algemeen komt het hier op neer: een integer type met de waarde `0` wordt beschouwd als `false`, alle andere waarden worden beschouwd als `true`.

De waarden `nul` en `niet nul` komen maar heel zelden expliciet terug. Dit komt omdat je vaak uit komt met de *booleaanse expressie*:

```
int c = 12;

if (c < 10) {
    printf("%c is less than 10\n", c);
}
```

Van `false` weet je dat het altijd `nul` is. Maar een waarde die `true` is kan ieder ander getal zijn. Soms is het wel fijn dat je `true` waarde gewoon `1` is. je zou iets als het volgende willen:

```
int t = 42; // een van de vele true opties

if (t) {
    t = 1;
}
```

Dit is wat omslachtig. We kunnen gebruik maken van de *logical not*:

```
int t = 18; // true
int f = 0;  // false

printf("%d, %d\n", !!t, !!f);
```

Wat wordt er nu geprint, en waarom?

5.4.3.2 sizeof en grootte van variabelen Van iedere waarde en van ieder type kan je vragen hoeveel bytes het in het geheugen verbruikt.

Bestudeer de volgende code:

```
#include <stdio.h>

int main() {
```

```
  int i = -3;

  size_t s1 = sizeof(i);
  size_t s2 = sizeof(int);

  printf("%lu %lu\n", s1, s2);
}
```

Op mijn machine is de uitvoer `4 4`. Maar deze waarde is afhankelijk van het platform.

Wat is het grote probleem hier mee?

Verder vallen er een aantal zaken op. Het *type* van `s1` en `s2` is `size_t`. Aan het eind van dit onderwerp moet je snappen wat er aan de hand is.

De *conversion specification* in de *format string* van `printf` bevat *length modifier* `l` (voor `long`) en *specifier* `u` (voor `unsigned decimal`). Zie verder ook

```
$ man 3 printf
```

5.4.3.3 Integer Types Wanneer je de standaard integer types op cpp reference bestudeert, dan vallen een aantal zaken op:

- Er zijn nogal wat verschillende types voor integer waarden, en dan is `char` nog niet eens vermeld.
- De grootte van de types is afhankelijk van het systeem.

Met name dat laatste is vervelend. We werken bij *embedded software development* nogal eens met systemen die anders zijn dan onze laptop. In deze course komen we bijvoorbeeld de Arduino tegen die in heel veel opzichten heel beperkt is in vergelijking met wat we gewend zijn.

Het probleem is dat we er dus niet van kunnen uitgaan of onze code op de laptop ook goed om een embedded systeem draait. Zo kan een `int` op ons systeem vrolijk doorgroeien terwijl het tot een overflow op een embedded systeem leidt.

Daarom gaan we vanaf nu gebruik maken van de integer types zoals gedefinieerd in `stdint.h`, en dan met name `int8_t`, `int16_t`, `int32_t` en `int64_t`. De unsigned int varianten als `uint16_t` kunnen we beter reserveren voor waarden waarop we *bit operaties* doen.

5.4.3.4 Floating point waarden Her en der in deze *reader* vind je variabelen van het type `float` (voor *floating point values*). Maar, gebruik van `float`s is meestal geen goed idee.

Begin jaren negentig maakte je de blits met een 386 processor. Of eigenlijk niet, je maakte echt de blits wanneer er ook een 387 *math co-processor* in je machine zat. De 387 processor gaf hardware-ondersteuning om berekeningen uit te voeren met *floating point* waarden. Tot die tijd moesten voor "normale" computers dergelijke berekeningen in software worden uitgevoerd wat relatief traag was: De *integer* berekening `18 x 27` is veel sneller uit te voeren dan de vergelijkbare *floating point* berekening `18.0 x 27.0`. De eerste berekening levert

ook een exact resultaat op, de tweede een, soms verrassende, benadering. Zo is de berekening `0.1 + 0.2` niet exact uit te voeren!

Waarom is gebruik van het type `float` niet verstandig?

Voor je *laptop* (sinds de 486 worden *floating point* berekeningen zondermeer hardwarematig ondersteund) kan je beter gebruik maken van het type `double` dan van het type `float`. Waarden van het type `double` (van "*double precision*") zijn tweemaal zo groot in het geheugen en de afrondingsfouten zijn daardoor veel kleiner. Nu jullie meer dan `640 kB` geheugen ter beschikking hebben is de extra benodigde geheugen zelden meer een probleem.

Ook is het gebruik van *floating point* waarden op *micro controllers* niet handig: de kans is aanzienlijk dat deze technologisch niet verder zijn dan de oude 386 machines: controleer of er hardwareondersteuning is voor *floating point* getallen. De Atmega op onze Arduino's hebben dergelijke ondersteuning niet. Wanneer je toch *floating point* berekeningen uitvoert, dan wordt er extra software op de controller gezet om deze berekeningen mee te maken.

Uiteindelijk, zie gebruik van `int` en `float` als de sporen van een oudere man die heeft leren programmeren toen *resources* nog beperkt waren. Als je ze in de voorbeelden leest, dan weet jij ondertussen beter.

5.4.3.5 Gebruik van `static`

Er zijn twee manieren waarop *specifier* `static` zinvol gebruikt kan worden. Eerst behandelen we `static` in combinatie met variabelen, daarna in combinatie met functies.

Waar mogelijk, probeer het gebruik van "globale variabelen" te voorkomen. Dat zal niet altijd lukken, maar bekijk in ieder geval of een van de volgende twee opties voorkomt dat je in een functie een globale variabele direct benadert.

In de onderstaande code blijft de waarde van `counter` behouden als de functie verlaten wordt. Toch heeft geen andere functie toegang tot `counter`.

```
int fun() {
    static int counter = 0; // scope van counter blijft beperkt
    count++;
    return count;
}
```

Hebben we dan toch te maken met globale variabelen, bekijk dan of deze minder magisch gebruikt kunnen worden (wat een zinvolle bijdrage is, maar overigens weinig met `static` te maken heeft).

```
int my_global;

int fun(int val) {
    int a = val;
}

int main(void) {
    my_global = 12;
    fun(my_global);
}
```

In bovenstaande code heeft `fun` er geen weet van dat er een relatie ligt met `my_global`. De functie is daardoor leesbaarder (geen plots opduikende `my_global`), en wellicht meer nog, `fun` is nu geschikt voor *unit tests*.

De tweede context waarbinnen `static` een rol speelt:

```
static int my_local_add_fun(int a, int b) {
    return 2*a + b - a; // silly addition
}

int add(int a, int b) {
    return my_local_add_fun(a, b)
}
```

De functie `my_local_add_fun` wordt gelinkt met *internal linkage* (zie *cpp reference*). Het gevolg hiervan is dat de functie buiten het bestand waarin deze is opgeslagen (meer formeel, buiten de *compilation unit*) niet beschikbaar is. In C++ termen zou je kunnen zeggen dat de functie `private` is.

5.4.3.6 enumeraties Enumeraties zijn bij elkaar-horende lijsten van constante integerwaarden. Zo kunnen we bijvoorbeeld de toestanden van een verkeerslicht opsommen:

```
enum verkeerslicht { ROOD, GEEL, GROEN, KNIPPEREND, UIT };
```

Vanaf nu is onze code een stuk leesbaarder:

```
enum verkeerslicht noord = ROOD;
enum verkeerslicht zuid = GROEN;
```

In dit geval heeft `ROOD` impliciet de waarde `0` gekregen, `GEEL` `1`, enzovoorts.

We mogen ze ook expliciet een waarde geven:

```
enum spaces { SPACE = ' ', TAB = '\t',  CR = '\r',
    LF = '\n' };
```

Let wel op, het blijven integer waarden, je mag er niet `CRLF = "\r\n"` aan toevoegen.

Het voordeel van een enumeratie boven een `#define RED 0` is meerledig:

- In ons enumeratievoorbeeld geven we `ROOD` geen expliciete waarde.
- Compilers *kunnen* testen voor waarden buiten de enumeratie, bijvoorbeeld `enum verkeerslicht = 10` kan door een compiler herkend worden.

  Let op: er wordt nadrukkelijk een voorbehoud gemaakt, een compiler *hoeft dergelijke checks niet uit te voeren*. Maar gelukkig, `cppcheck` bijvoorbeeld is weer streng en helpt ons.
- Het voordeel van een opsomming is dat er een natuurlijke groepering ontstaat: `ROOD` en `GROEN` horen bij elkaar, `SPACE` en `TAB` ook.

Let wel, we hebben ook weer te maken met dat C een oude taal is: de volgende code is problematisch:

```
enum schaakstuk { TOREN, PION }; // en alle andere
enum gebouw { HUIS, TOREN }; // villa's, hutten ...
```

Er is intern geen scheiding tussen de waarden van `schaakstuk` en `gebouw`. Om die reden mogen we `TOREN` niet tweemaal definieren.

Verder helpen enumeraties ons niet tegen allerlei soorten fouten:

```
enum kleur { ROOD, GROEN };
enum vorm { VIERKANT, DRIEHOEK };

enum vorm figuur = ROOD;
```

wordt vrolijk gecompileerd en uitgevoerd. Voor de compiler zijn het uiteindelijk allemaal gewoon integer waarden. Desalnietemin, enumeraties kunnen goed helpen je code veel leesbaarder te maken.

5.4.3.7 structs We kunnen ook variabelen groeperen in een `struct`:

```
struct meting {
    int timestamp;
    float temperature;
};

struct meting m = { .timestamp = 72361873,
    .temperature = 25.8 };
```

Bovenstaande initialisatie van meting `m` werkt alleen bij initialisaties, dus in combinatie met de *declaratie van `m`*. Het werkt ook alleen maar vanaf `C99`.

Anders moet je terugvallen op:

```
struct meting m;
m.timestamp = 72361873;
m.temperature = 25.8;
```

Het voordeel van een `struct` is dat je deze, bijvoorbeeld, kan gebruiken als return waarde van een functie:

```
struct meting geefVolgendeMeting();
```

Houd in een dergelijk geval de grootte van een `struct` beperkt (de `sizeof(m)`), omdat er op enig moment tweemaal de grootte van de struct gereserveerd moet zijn.[29]

5.4.3.8 typedef In C kunnen met de *specifier* `typedef` nieuwe namen voor gegevenstypen worden gemaakt (Kernighan en Ritchie 1990, sec 6.7). Net als bij enumeraties komt dit vooral ten goede aan de leesbaarheid van de code en minder aan de mogelijkheden voor de compiler om je code te checken. Maar soms kan het handig zijn, je moet in ieder geval de constructie kunnen lezen.

```
typedef int Temperatuur;

Temperatuur kookpunt = 100;
```

In ieder geval ben je in voorgaande secties al heel wat zinnige typedefs tegengekomen.

[29] Let op, bij *pointers* komt nog een belangrijke aanvulling op de syntax met structs. Wanneer in een functie een *pointer* naar een struct meegeeft, dan kan je de verschillende *members* benaderen als `m->timestamp` en `m->temperature`.

Typedefinities kunnen ook gebruikt worden in combinatie met `enum` en `struct`. Ook hier geldt dat je dit vooral moet weten om code van anderen te kunnen lezen. De discussie of het gebruik van typedefs nu wel of geen goed idee is, is hevig. Dat je minder hoeft te typen met typedefs is een argument dat je maar heel stilletjes voor jezelf moet houden ...

We kunnen de `enum verkeerslicht` die we eerder hebben gezien met een typedef korter declareerbaar maken:

```
typedef enum verkeerslicht { ROOD, GROEN } Stoplicht;

Stoplicht west = ROOD;
```

De grijze code geeft de enumeratie, de rode code geeft de typedefinitie van van de enumeratie tot `Stoplicht`. Met andere woorden, type `Stoplicht` is nu een alias voor de `enum verkeerslicht`.

Dit kan worden ingekort door een *anonieme enumeratie* met typedef een naam te geven:

```
typedef enum { ROOD, GROEN } Stoplicht;

Stoplicht oost = GROEN;
```

In deze laatste oplossing mag `enum verkeerslicht zuidoost;` niet meer.

Nogmaals, moet je dit in je eigen code willen? Alleen met een goede reden, zou ik zeggen. Moet je het herkennen en kunnen lezen? Zeker.

Ook de `struct meting` kunnen we met een `typedef` onderhanden nemen.[30]

```
typedef struct {
    int timestamp;
    float temperatuur;
} Datapunt;

Datapunt d  = { .timestamp= 982739187, .temperatuur= -14.3 };
```

Aan de lage temperatuur zie je al dat je je moet afvragen of je dit in je eigen code wel wil. Maar ja, je moet het wel kunnen herkennen, begrijpen. En er zijn toetsmomenten waarop je het ook moet kunnen schrijven...

Laat ik eindigen met een relativerend woord na alle kritische uitingen. Wij zijn in voorgaande secties typedefs als `size_t` en `int8_t` tegengekomen. En dit zijn echt goede voorbeelden van typedefinities die ons helpen om onze software over verschillende systemen portable te houden.

Overigens, in veel kringen is de *suffix* "`_t`" bedoeld om te benadrukken dat het hier om een typedefinitie gaat.

5.5 Arrays in C

We hebben allemaal al lang geleerd gebruik te maken van arrays in de talen die je tot nu toe bent tegengekomen. De taal C kent ook arrays. Maar ook nu zullen

[30] De gegeven waarde voor `int`, `982739187` past in een 32-bits `int`, maar mogelijk niet op de `int` op je Arduino!

we merken dat C een oude taal is: een heleboel kenmerken die we verwachten van arrays blijken in C nog niet ondersteund te worden.

5.5.1 Doel

- Je kan arrays gebruiken in je code.
- Je weet waar arrays in C afwijken van arrays in meer modernere talen.

5.5.2 Opdracht

Schrijf een demo-applicatie (met een `Makefile` met unittests, cppcheck, formatering, etc) met daarin de volgende functionaliteit.

Op enig moment hebben we een array met de volgende meetwaarden:

```
float temp[] = {
19.00, 19.37, 19.05, 18.03, 17.18, 16.18, 16.95, 16.24,
15.98, 15.57, 14.97, 15.50, 15.65, 15.19, 15.35, 15.35,
15.99, 16.55, 16.45, 17.23, 17.12, 17.70, 17.70, 16.74,
16.76, 16.02, 16.96, 17.01, 17.88, 18.65, 18.51, 18.21,
18.27, 19.19, 18.77, 18.85, 19.39, 19.62, 18.85, 17.53,
18.32, 19.03, 19.78, 20.63, 20.80, 20.27, 20.40, 20.00,
20.74, 21.20, 20.78, 20.01, 20.63, 20.30, 19.75, 19.80,
18.69, 18.72, 17.96, 18.27, 18.27, 16.50, 16.30, 15.55,
15.62, 16.10, 16.36, 17.43, 17.29, 16.35, 16.37, 16.07,
16.44, 16.56, 16.73, 18.28, 19.93, 20.74, 22.45, 21.80,
22.15, 21.02, 20.16, 18.55, 17.24, 17.08, 17.40, 17.20,
18.58, 17.98, 18.43, 18.17, 18.48, 18.97, 19.57, 19.52,
19.35, 19.67, 20.55, 21.26, 20.95, 19.39, 17.64, 17.30,
16.44, 15.88, 16.77, 15.88, 14.59, 14.49, 14.33, 15.65,
15.67, 15.05, 14.41, 14.73, 16.22, 16.72, 15.29, 15.68,
16.14, 16.26, 16.35, 17.15, 17.77, 19.79, 20.09, 19.82,
19.79, 19.56, 19.41, 19.35, 19.73, 21.21, 21.09, 21.42,
21.79, 20.74, 20.96, 20.15, 20.22, 19.36, 19.03, 18.02,
17.44, 16.92, 16.98, 16.14, 15.90, 15.66, 15.89, 16.18,
16.84, 17.49, 18.03, 18.86, 19.87, 20.64, 20.88, 21.59,
21.14, 19.75, 19.11, 18.62, 18.17, 18.11, 17.87, 17.84,
17.45, 16.81, 17.46, 17.46, 18.76, 19.64, 21.08, 20.60,
21.04, 20.85, 20.31, 19.23, 20.33, 20.78, 20.48, 19.04,
19.06, 18.51, 19.18, 18.70, 18.92, 18.50, 19.32, 18.80,
18.91, 19.35, 19.20, 18.20, 17.76, 16.86, 16.93, 17.72};
```

(voor de geïnteresseerden, deze reeks is gegenereerd met Perlin Noise met daaroverheen wat extra ruis).

5.5.2.1 moving average We kunnen deze reeks "meer *smooth*" maken door er de *rolling mean* van te berekenen. Dit is het handigst met een voorbeeld uit te leggen.

```
positie:         0    1    2    3    4
reeks :         24   56   82   12   89
rolling mean :       54   50   61
```

We nemen steeds drie waarden en berekenen daarvan het gemiddelde. We kunnen het eerste gemiddelde berekenen van de getallen op posities 0, 1, en 2: het

gemiddelde van 26, 56, en 82 is (afgerond) 51. Dit getal komt in het midden van die drie getallen, dat is positie 1. Het effect is dat we een veel mooiere curve krijgen.

Maak twee varianten van een *smooth* functie, een variant `smooth_in_place` die de waarden in de oorspronkelijke *array* aanpast en een tweede variant `smooth` die het resultaat in een tweede array schrijft. Let er bij `smooth` op, dat het niet (nog niet) mogelijk is om een functie een nieuwe *array* terug te laten geven.

In het geval van `smooth_in_place` gebruik je een deel van de oorspronkelijke *array* niet meer en in het geval van `smooth` is de tweede *array* korter dan de oorspronkelijke reeks.

Wil je extra uitdaging, accepteer dan een extra parameter waarmee het aantal elementen dat wordt gebruikt instelbaar is. Dit getal moet dus altijd oneven zijn (en positief). In bovenstaande voorbeeld hebben we 3 gebruikt, maar ook 5 en 7 geven bruikbare resultaten.

5.5.2.2 Maximale reeks Schrijf een functie waarmee je de periode met de hoogste temperatuur kan vinden. Dus, in welke periode van 3 dagen is de temperatuur het hoogst?

Schrijf een tweede variant die de maximale reeks beschikbaar maakt voor de uitvoerder van de functie.

Ook nu kan je jezelf extra uitdagen door de periode instelbaar te maken.

5.5.3 Toelichting

5.5.3.1 Arrays declareren en initialiseren Hieronder zie je een aantal voorbeelden van array declaraties:

```
int arr1[12];
int arr2[] = {1, 2, 4, 8, 16};
int arr3[15] = {0};
```

Array `arr1` is niet geïnitialiseerd en je moet ervan uitgaan dat de inhoud *garbage* is. Array `arr2` is expliciet geïnitialiseerd. Hiermee is ook de grootte van de array bepaald. Array `arr3` is een combinatie. De grootte is expliciet vastgelegd en een deel is geïnitialiseerd. Daarmee wordt automatisch de overige elementen op `0` geïnitialiseerd.

5.5.3.2 C Arrays kennen niet hun grootte Ga er van uit dat een array in C niet zijn grootte kent. Ja, er zijn truucs. Gebruik ze niet om te voorkomen dat je jezelf in de voeten schiet.

Dit is precies de reden dat de grootte van een array als functieargument wordt meegegeven:

```
void print_array(int arr[], int sz) {
    for (int i=0; i<sz; i++)
        printf("arr[%d] = %d\n", i, arr[i]);
}

int main() {
```

```
    int array[12] = {0};
    print_array(array, 12);
}
```

Het is een goede gewoonte de grootte van een array in een `#define` vast te leggen:

```
#define SIZE 12

int arr1[SIZE];
```

C staat niet toe om dit met `const int SIZE = 12` te doen, omdat een `const int` pas gegarandeerd in run-time bepaald kan worden.

Het gebruik van een variabele `int`, dus als

```
int size = 22;
int arr3[size];
```

is niet gegarandeerd ondersteund in C na C99. Zeker omdat wij bij ESD mogelijk te maken hebben met meer obscure devices kunnen we tegen dit probleem aanlopen. Wij maken daarom in dit vak geen gebruik van zgn. *variable length arrays*.

5.5.3.3 Arrays en functies

We kunnen in C geen functie maken die een array teruggeeft, dus, doe het volgende niet:

```
int[] makeIntArray(int size); // niet doen!!
```

We kunnen de waarden van een array dat als argument wordt meegegeven veranderen, een degelijke functie is dus te maken:

```
void inplaceSort(int arr[], int arrc);
```

Wil je duidelijk maken dat er niet "gerommeld" wordt met de elementen in een array, geef dit dan aan met de *`const` qualifier*:

```
void printElements(const int arr[], int arrc);
```

De meeste compilers zullen je een foutmelding geven wanneer er toch een poging gedaan wordt de arraywaarden aan te passen.

5.6 Strings in C

De taal C kent geen type `string`. Om een regel tekst op te slaan wordt een array van characters gebruikt. Zoals je je herinnert, moet je bij een array altijd expliciet de lengte bijhouden. Hier wordt bij strings in C vaak een "truuc" uitgehaald om dit te voorkomen: we gebruiken een bijzonder teken, de `null` character of `\0`, om aan te geven wat het einde van de string is. Dit lijkt een geweldig idee, maar blijkt allerlei risico's met zich mee te brengen.

5.6.1 Doel

- Je kan de verschillen en overeenkomsten met andere arrays in C verwoorden.
- Je kent de risico's van verkeerd gebruik van strings in C.
- Je kan applicaties schrijven met strings in C.

5.6.2 Opgave

5.6.2.1 Controleer een string voor Rest verbs Stel dat we de volgende enumeratie hebben:

```
enum verb { UNKNOWN, GET, POST, PUT, DELETE };
```

Schrijf een functie `isVerb` waarmee van een string gezegd kan worden of het een verb is. Wanneer de string niet als verb wordt herkend, dan geeft de functie `UNKNOWN` terug. Maak gebruik van de functie `strncmp` om te controleren of een string een bepaalde waarde heeft.

5.6.2.2 Tel witruimte Schrijf een functie `countSpaceChars` waarmee in een string het aantal spaties, tabs en newlines getelt wordt. Zo moet de string "`\thi there!\r\n`" de waarde 4 opleveren.

Schrijf een tweede functie `countSpace` waarmee in een string het aantal *reeksen* witruimte geteld wordt. In het eerdere voorbeeld "`\thi there!\r\n`" moet deze functie 3 opleveren, vanwege de drie stukken witruimte "`\t`", " " (spatie), en "`\r\n`" (die als één wordt geteld).

5.6.3 Toelichting

Strings in C zijn arrays van characters. Maar let op. De array `woord` in

```
char woord[] = "hoi";
```

bestaat niet uit 3, maar uit 4 elementen, nl. `h`, `o`, `i`, en de *null character* `\0`.

We moeten niet vergeten deze *null character* toe te voegen wanneer een string helemaal als array initiëren:

```
char woord[] = {'d', 'o', 'e', 'i', '\0'};
```

Met deze "truuc" kunnen we functies maken waaraan we de string als argument meegeven, maar waar we niet de lengte van de array hoeven mee te geven:

```
int len(char str[]) {
   int c = 0;

   while (str[c] != '\0')
      c++;

   return c;
}
```

Met bovenstaande (wat naïeve) implementatie bereken je de lengte van een string (normaal zou je daar `strlen` voor gebruiken).

Hoe slim dit idee ook lijkt, in de praktijk blijkt dit allerlei problemen op te leveren. Wat doet een dergelijke functies wanneer er geen `\0` in de string zit? Of hoe gemakkelijk is het een functie te schrijven die alleen het eerste gedeelte van een correcte string kopieert en dus niet de `\0` meekopieert?

We zijn al beter af (dat is, we schrijven betere code) wanneer we gebruik maken van functies als `strncpy` en `strncmp`. Dat zijn dus functies met zo'n `n` in het

midden om aan te geven dat er een maximaal aantal tekens gekopieerd of vergeleken wordt.

Maar, lees de documentatie van `strncpy` maar eens door: je bent dan nog niet zeker. Wanneer er geen `\0` wordt meegekopieerd, dan is het resultaat zonder `\0`. Je doet er dus altijd verstandig aan om in zo'n geval zelf een `\0` op de laatste plek in de array te zetten.

5.7 Pointers in C

Dit gedeelte bevat ook het blok "malloc en free".

5.7.1 Doel

- Je kan een geheugenmodel tekenen die de werking van pointers toelicht.
- Je kan elementaire operaties met pointers uitvoeren.
- Je kan pointers toepassen als functie-parameters.
- Je kan extra geheugen voor je applicatie beschikbaar maken met `malloc` en deze weer vrijgeven met `free`.

  Je kent de verschillen tussen `malloc`, `calloc` en `realloc`.
- Je kan arrayoperaties omschrijven naar functioneel identieke pointeroperaties en andersom.
- Je kan meerdimensionale geheugenstructuren bouwen en deze weer vrijgeven.

5.7.2 Opgaven

5.7.2.1 swap functie Schrijf een functie `swap` die twee pointers naar integers krijgt.

Het voorbeeld hieronder moet je een beetje op weg helpen:

```
int i = 10, j = 20;
swap(&i, &j); // i wordt 20, j wordt 10
```

Test deze functie!

5.7.2.2 bubble sort De bubble-sort is een relatief eenvoudige (maar niet heel efficiënte) manier om een array met waarden te sorteren.

Deze is te schrijven met door gebruik te maken van je functie `swap`.

Schrijf een functie `bubble` zodat de volgende code werkt:

```
int numbers[10] = {82, 98, 62, 63, 17, 65, 19, 59, 40, 56};
bubble(numbers);

// dit levert je een gesorteerde array op
```

Voor een array met `n` elementen kan je de bubble sort volgens de volgende pseudo-code implementeren:

```
for i = 0 .. n:
    for j = 0 .. n-1:
        if number[j] > number[j+1]:
            swap(j, j+1)
```

Zie je niet meteen voor je wat er gebeurt, print dan na iedere ronde een tussenresultaat.

Schrijf deze functie eerst zoveel mogelijk in *array notatie*. Als die functie eenmaal werkt, schrijf er dan een in *pointer notatie*.

Test je functie! (Dat kan je doen door in het resultaat te testen dat element `i` kleiner of gelijk moet zijn dan element `i+1`.

5.7.2.3 valgrind Een fout met pointers is snel gemaakt, zeker wanneer je gebruik maakt van `malloc` en `free`.

Er bestaat een applicatie met de naam valgrind die je kan helpen bij het vinden van fouten.[31]

Installeer `valgrind` en doorloop de tutorial: http://valgrind.org/docs/manual/quick-start.html.

Voeg `valgrind` toe aan je `Makefile` waarmee je je bubblesort hebt gemaakt.

5.7.2.4 gemiddelde van een vooraf aangegeven aantal cijfers Schrijf een programma waarbij de gebruiker (via `scanf`) het aantal cijfers ingeeft dat hij wil invoeren, en daarna cijfers kan invoeren (je mag er van uitgaan dat de gebruiker alleen de cijfers 0, 1, 2, 3, 4, 5, 6, 7, 8 of 9 zal invoeren).

Je kan de volgende functie gebruiken om cijfers in te lezen:

```
int read(const char *prompt) {
  printf("%s > ", prompt);
  char c;
  scanf(" %c", &c); // spatie zorgt ervoor dat je alleen
                    // "echte" tekens leest

  return c - '0';
}
```

Ga zelf na welk *header file* je moet *includen* voor `scanf`.

Print het gemiddelde, gevolgd door een herhaling van de lijst van alle ingevoerde cijfers. Realiseer dit door gebruik te maken van een pointer met `malloc` (dus niet met een “heel grote array”).

Zorg er wel voor dat al het geheugen weer wordt vrijgegeven aan het einde van je programma.

5.7.2.5 Som van een vooraf onbekend aantal cijfers Schrijf een programma waarmee je cijfers kan lezen totdat de gebruiker een `s` geeft in plaats van een getal.

Je kan weer de functie `read()` gebruiken, maar denk even na hoe je de `s` afvangt.

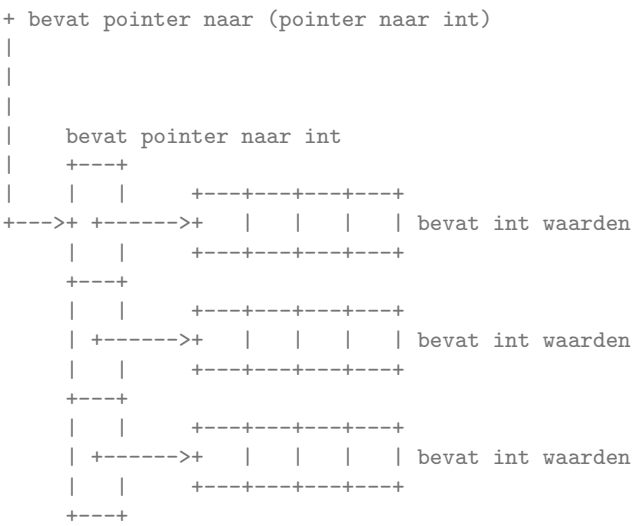
[31] Het gebruik van `valgrind` valt vooralsnog buiten de toetsstof.

Reserveer voor tien getallen geheugen met `malloc` en breid uit met `realloc` wanneer dat nodig is. Controleer steeds of je het geheugen ook krijgt waar je om vraagt.

Print nadat de gebruiker een `s` gegeven heeft alle getallen en de som.

Geef het gealloceerde geheugen terug.

5.7.2.6 2D pointers Maak een programma dat met behulp van `malloc` de volgende structuur opbouwt:



Geef de integers door jou gegeven waarden (bijvoorbeeld, 0, 1, 2, 3, ...).

Zoek per rij de hoogste waarde en print deze.

Houdt er rekening mee dat `malloc` op ieder moment kan aangeven dat de gevraagde geheugen niet beschikbaar is. Op dat moment moet het "geheugen-bouwwerk" dat je al hebt weer netjes worden afgebroken.

Als je programma wel succesvol loopt, dan moet aan het einde al het geheugen weer worden vrijgegeven. Controleer met `valgrind` of er nog problemen zijn en los die op.

5.7.3 Toelichting

Pointers zijn variabelen waarvan de waarde verwijst naar een plek in het geheugen. Vaak verwijst een pointer naar een andere variabele.

Hiermee kunnen we een aantal beperkingen in C oplossen waar we tot nu toe tegenaan zijn gelopen, bijvoorbeeld:

- We hebben tot nu toe alleen arrays gezien waarvan de grootte tijdens het compileren vast staat.

  Met pointers kunnen we tijdens het uitvoeren van het programma om extra geheugen vragen.

- Tot nu toe hebben we nog geen mogelijkheid om een functie een array te laten teruggeven.

  Met pointers kunnen we geheugen "alloceren". Dergelijke pointers gedragen zich op vergelijkbare manier als arrays.

- Met pointers kunnen we er voor zorgen dat functieargumenten door de uitvoerende functie verandert kunnen worden.

  We hebben al gezien dat een functie de elementen van een array kan wijzigen en met pointers kunnen we dat nu ook voor elkaar krijgen met andere typen variabelen.

- Met pointers kunnen we supercompacte en alleen voor echte ontwikkelaars leesbare code schrijven. Alle anderen zullen niet in staat zijn je code te kunnen lezen en begrijpen.

En zo is het dus precies. Pointers zijn zinvol. Je kan er ook onleesbare code mee schrijven. Dit was "vroeger" nodig om code ook heel efficiënt te laten zijn. Zo was code in pointer-notatie efficiënter dan code in arraynotatie. Echter, de meeste compilers genereren nu met beide soorten code dezelfde efficiënte uitvoer. Laten we dus vooral leesbare code schrijven!

5.7.3.1 Declaraties We declareren een "pointer naar een integer waarde", `ip` als volgt:

```
int *ip = NULL;
```

Zeker wanneer je net (weer) met pointers werkt, kan dit verwarrend zijn, omdat de `*` niet alleen in de declaratie wordt gebruikt.

De volgende notatie wordt ook door de compiler gecompileerd:

```
int* ip = NULL;
```

Deze code lijkt iets meer te benadrukken dat het om een "pointer naar een integer" gaat. Maar laat je niet verrassen. Het is namelijk mogelijk om declaraties te combineren:

```
int* ip, x, arr[12]
```

Hoort de `*` nu bij `int` of bij `ip`? De `*` staat bij `int` maar heeft niets van doen met de `x` of met de `arr[12]`. Kortom, verwarring alom.

Mocht je kiezen voor `int*`, let erop dat `clang-format` je weer terugzet naar de conventie `int *ip`. Wen er maar aan. Of toch niet. We kunnen `clang-format` aangeven dat we meer C++-stijl opmaak van pointer declaraties willen:

```
$ clang-format -style='{PointerAlignment: Left, ColumnLimit: 75}' -i *.c
```

In het bovenstaande geven we aan dat `*` (en `&`) *operators* in declaraties naar links moeten. Als bonus geven we ook een bovengrens aan de breedte van de code.

5.7.3.2 Address-of operator Wat hebben we aan pointers als we er geen waarde aan kunnen geven. Omdat wij niet weten welk adresruimte aan je

applicatie gegeven wordt, kunnen we geen absolute waarden aan de pointer geven.

Voor ons is het alleen zinvol om het adres van een andere variabele te geven:

```
int i = 18;
int *ip = &i;
```

In bovenstaande krijgt onze pointer `ip` het adres van variabele `i`. Let er dus op dat we de "address-of operator `&`" meestal in combinatie met "normale" variabelen gebruiken.

Altijd? Nee, niet persé. Ook het volgende is mogelijk:

```
int i = 3292;
int *ip = &i;
int **ipp = &ip;
int ***ippp = &ipp;
```

Geen idee wat je er in dit verband mee moet, maar het kan wel.

5.7.3.3 Dereferentie operator. Geweldig, een variabele die naar een plek in het geheugen verwijst. Maar we willen soms ook wel weten wat er op dat adres leeft.

```
int i=2387;
int *ip = &i;

// wat leeft er dan op dat adres ip?
printf("op adres %p van ip leeft het getal %d\n", ip, *ip);
```

Dus, met de "dereferentie operator `*`" kunnen we van een *pointer* vragen waarheen deze verwijst. Dit mag ook alleen maar wanneer het zinvol is. We kunnen dus eerst de pointer vragen of het adres niet "leeg" is:

```
if (ip)
    printf("%d\n", *ip);
else
    printf("pointer ip is NULL\n");
```

5.7.3.4 Pointers en arrays Pointers en arrays gedragen zich op vergelijkbare wijze. Ze zijn net niet helemaal identiek.

```
int arr[30] = {0};
int *ip;

// we kunnen met ip door de array heenlopen:
ip = arr;
for (int i=0; i<30; i++) {
    printf("%d\t%d\t%d\n", arr[i], ip[i], *(ip+i));
}
```

Je ziet het, de notatie `ip[i]` lijkt wel erg veel op `arr[i]`. Maar we kunnen op `ip` ook de pointernotatie `*(ip+i)` loslaten.

En ja, je kan zelfs met `*(arr+i)` door de array heenlopen, maar waarvoor zou je dat willen?

Wat dus relevant is, is dat je

```
int *ip
```

kan lezen als `ip` gedraagt zich als een array met integer waarden. Maar als `ip` in `int *ip` zich kan gedragen als een array, dan kan `ipp` in `int **ipp` zich gedragen als een "array met pointer naar integer waarden". En we hebben net gezien dat "een pointer naar integer waarden" zich kan gedragen als een array met integer waarden.

Kortom, `ipp` in `int **ipp` kan zich gedragen als een array met arrays met integer waarden, oftewel, als een twee-dimensionale array. Dit is nu nog een grappige *gimmick*, maar het wordt zometeen relevant met `malloc`.

5.7.3.5 Malloc en Free Met `malloc` kunnen we aan het besturingssyteem vragen om geheugen voor het programma te reserveren (of *alloceren*, wat de naam *memory allocation* of malloc in het kort verklaart).

```
int *arr = malloc(10*sizeof(int));
```

We vragen om ruimte voor tien integer waarden aan het operating systeem. Dit is ruimte op de zgn. *heap*. Als deze ruimte beschikbaar is, dan krijgen we een pointer naar die ruimte terug. Als die ruimte er niet is, dan krijgen we `NULL` terug.

We moeten altijd controleren of we deze ruimte krijgen:

```
#include <stdlib.h>

int *arr = malloc(10*sizeof(int));

if (!arr) {
    printf("could not allocate memory");
    return;
}
```

De code `!arr` geeft true terug wanneer `arr` gelijk is aan `NULL`. Het meer expliciete `if (arr == NULL)` vind je wellicht leesbaarder.

Geheugen dat je krijgt, moet je ook weer teruggeven wanneer je het niet meer nodig hebt.

```
free(arr);
```

Let erop dat het geheugen dat je met `malloc` krijgt niet geïnitialiseerd is. Wil je graag geïnitialiseerde geheugen, dus met alles op nul gezet, gebruik dan `calloc`. Om mij niet duidelijke reden heeft `calloc` niet één argument (zoals `malloc`), maar twee:

```
int *arr = calloc(100, sizeof(int));

if (arr)
    printf("arr[66] = %d\n", arr[66]);
```

Net als bij `malloc` moet geheugen verkregen via `calloc` met `free` worden teruggegeven.

Het gebeurt dat je geheugen verkregen met `malloc` wilt vergroten of verkleinen. Hiervoor gebruik je `realloc`.

```
int *arr = malloc(100*sizeof(int));

arr = realloc(arr, 200*sizeof(int));
```

Er zijn drie scenario's mogelijk:

1. Het bestaande blok geheugen wordt aangepast naar de nieuwe omvang.
2. Het bestaande blok wordt met inhoud en al gekopieerd naar een andere locatie in het geheugen waar ook ruimte is voor de uitbreiding.
3. Het is niet mogelijk de gevraagde aanpassing uit te voeren.

In de eerste twee gevallen krijg je een bruikbare pointer terug, in het derde geval krijg je een `NULL` pointer terug.

Normaal gesproken moet je dus controleren wat er gebeurt:

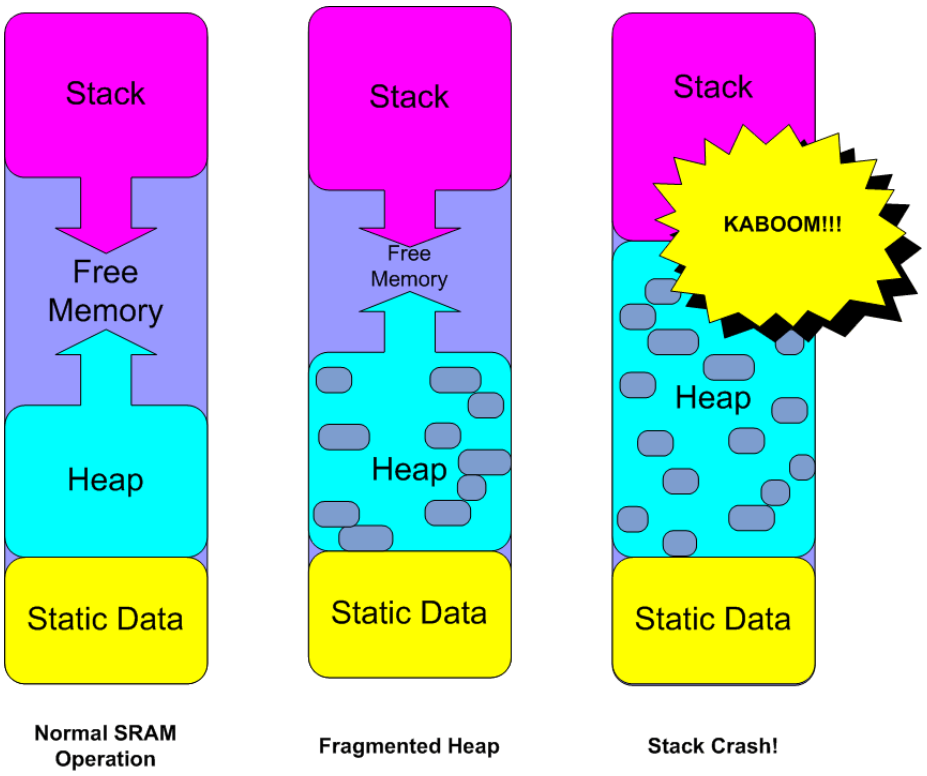
```
int *arr = malloc(100, sizeof(int));

int *tmp = realloc(arr, 1000 * sizeof(int));

if (tmp == NULL) {
    // geen extra geheugen, je kan verder met arr of stoppen
} else {
    // extra geheugen gekregen, maar arr is niet meer valide!

    arr = tmp; // we kunnen weer verder met arr
}
```

5.7.3.6 Problemen met `malloc` en `free` Fig. 5, (bron: Earl z.d.) is een schema van het geheugenmanagement op een microcontroller als de Arduino. Er zijn vier blokken met geheugengebruik zichtbaar:

1. De *static data*, hierin staat informatie waarvan de omvang gedurende de looptijd van de applicatie niet toeneemt.
2. De *stack*, hierin is informatie over de functies die in uitvoering zijn opgeslagen, iedere keer wanneer een nieuwe functie wordt uitgevoerd groeit de stack. Wanneer een functie klaar is met de uitvoering krimpt de stack.
3. De *heap*, hierin is de geheugenruimte te vinden die met `malloc` is gealloceerd.
4. De vrije ruimte. Dit is de ruimte die voor de stack en de heap nodig zijn om de applicatie te kunnen laten functioneren.

   Wanneer er onvoldoende vrije ruimte is, geeft `malloc` `NULL` terug. Echter wanneer de stack ruimte nodig heeft en die is er niet meer, dan komt de applicatie in een ongedefiniëerde state waarin de applicatie onberekenbaar gedrag gaat vertonen.

Het probleem met `malloc` is dat de organisatie van de *heap* niet geoptimaliseerd is op ruimtegebruik. Zo kunnen er stukken worden vrijgegeven met `free` die vervolgens te klein zijn voor de volgende `malloc`. Als gevolg hiervan ontstaat er



Figuur 5: Het ontstaan van een *stack crash*.

een *fragmented heap* die in omvang groter is dan de som van gealloceerde geheugen. Als gevolg hiervan kan een *stack crash* voor komen wanneer er rekenkundig nog ruim voldoende geheugen beschikbaar moet zijn.

5.7.3.7 Pass by reference We hebben al gezien dat een functie de waarden in een array kan aanpassen. Dit werkt tot nu toe niet voor andere argumenten. We kunnen dit voor elkaar krijgen met pointers.

```
#include <stdio.h>

struct student {
  int id;
  int credit;
};

void mod(int* ip, int* arr, struct student* sp) {
  *ip = 42;
  arr[0] = 42;
  sp->credit = 42; // beter leesbaar dan (*sp).credit
}

int main() {
  int i = 10;
  int arr[] = {1, 2, 3};
  struct student s = {.id = 123, .credit = 30};

  mod(&i, arr, &s);

  printf("%d, %d, %d\n", i, arr[0], s.credit);
}
```

In deze code vind je `sp->credit`. Deze constructie is ondertussen meerdere keren langsgekomen, maar nooit verder toegelicht.

De *operator* "`->`" is de *member access through pointer operator* (zie *cpp reference* voor details).

In bovenstaande code is `sp` een verwijzing naar een `struct`. Om over `sp` toegang te krijgen tot de *members* van de struct kunnen we gebruik maken van de *dereference operator*: `*sp.id`. Helaas geeft `*sp.id` niet het resultaat wat je ervan hoopt, we hebben van doen met een "Meneer Van Dalen Wacht Op Antwoord"-probleem, oftewel, we hebben te maken met *operator precedence* (zie wederom *cpp reference* voor details).

Zoals in de som `6 + 2 * 3` als gevolg van "Meneer Van Dalen", euh, ik bedoel als gevolg van *operator precedence* eerst `2 * 3` wordt uitgevoerd en daarna pas de optelling, zo wordt in `*sp.id` eerst de *member access operator* "`.`" uitgevoerd en pas daarna de *dereference operator* "`*`", en dat levert een fout op, want je kan geen *dereference operator* loslaten op de integer waarde `sp.id`. Haakjes helpen ons, waardoor we wel zinvol `(*sp).id` kunnen uitvoeren, want nu wordt eerst de *dereference operator* losgelaten op `sp` en daarna de *member access* op de struct.

Veelvuldig schrijven van constructies als "`(*sp).id`" is nadelig voor de leesbaarheid van je code, en ook foutgevoelig. In plaats van het omslachtige "`(*sp).id`"

mogen we gebruik maken van de *member access through pointer operator*, "`sp->id`", om exact hetzelfde effect te bereiken.

5.7.3.8 Notatie Tbl. 7 geeft een korte samenvatting van de verschillende notaties. In dit schema ben ik ervan uitgegaan dat `ip` bedoeld is als een *pointer naar een* `int` en `vec` een *pointer naar een lijst van* `int`.

Tabel 7: Samenvatting *pointer* notatie.

Declaratie	Waarde	Verwijzing
`int i`	`i`	`&i`
`int *ip`	`*ip`	`ip`
`int arr[10]`	`arr[0]`	`&arr[2]`
	`*(arr+1)`	`arr`
`int *vec`	`*(vec+2)`	`vec+2`
	`vec[2]`	

5.7.3.9 Gedrag Het is ook handig scherp voor ogen te hebben hoe variabelen in combinatie met een operatie zich gedragen. Zie hiervoor tbl. 8.

Tabel 8: Gedrag van `int i`, `int* p`, en `int arr[6]`.

Operatie	Gedraagt zich als	Voorbeeld
`i`	integer waarde	`i = 0;`
`&i`	verwijzing	`p = &i;`
`p`	verwijzing	`p = NULL; p = &i;`
`*p`	integer waarde	`i = *p * 18; *p += 7;`
`arr[2]`	integer waarde	`arr[2] = 4;`
`arr`	verwijzing	`p = arr`

5.8 Const Correctness

Const Correctness speelt een grote rol in C++ (OSM), en terecht: het draagt bij aan het voorkomen van *bugs*. Ook C kent `const` waarmee je de *compiler* aangeeft dat een variabel niet aan de linker kant van een expressie mag staat, dus, dat er niet geschreven wordt. Ik moet bekennen dat op moment van schrijven we in de overige C hoofdstukken hier te weinig mee doen.

5.8.1 Doel

- Je kan de *compiler* hints geven welke variabelen op enig moment niet aan de linker kant van een `=`-teken in een expressie mogen staan.
- Voor *pointers* ben je in staat onderscheid te maken tussen schrijven van "waarde waarheen verwezen wordt" en schrijven van de "verwijzing".

5.8.2 Opdrachten

5.8.2.1 Constant pointers

Gegeven de volgende code

```
int val1 = 417;
int val2 = 912;

const int* ptr1 = &val1;
int const* ptr2 = &val1; // const after notation

int const* const ptr3 = &val1; // const after notation
const int* const ptr4 = &val1;

int* const ptr5 = &val1; // const after notation
```

Bepaal van de volgende expressie welke wel en niet toegestaan worden(hint, laat de *compiler* helpen).

a. `*ptr1 = val2;`

b. `ptr1 = &val2;`

c. `*ptr2 = val2;`

d. `ptr2 = &val2;`

e. `*ptr3 = val2;`

f. `ptr3 = &val2;`

g. `*ptr4 = val2;`

h. `ptr4 = &val2;`

i. `*ptr5 = val2;`

j. `ptr5 = &val2;`

5.8.2.2 Functieargumenten

Gegeven de volgende C functies:

```
void f1_arr(int arr[], int sz) {
  if (sz > 0) {
    arr[0] += 17;
  }
}

void f2_arr(const int arr[], int sz) {
  if (sz > 0) {
    arr[0] += 17;
  }
}

void f1(int* arr) {
    arr[0] += 1;
}

void f2(const int* arr) {
  arr[0] += 2;
```

```
  arr = &arr[1];
}
```

Commentarieer in bovenstaande functies niet-toegestane expressie uit zodat je compileerbare en werkende functies overhoudt.

5.8.2.3 Gebruik van functies Gegeven de volgende code die gebruik maakt van (gecorrigeerde) code uit de vorige opdracht:

```
int arr[] = {1, 2, 3, 4, 5};

const int c_arr[] = {10, 20, 30};
```

a. Bepaal welke van iedere regel in de volgende code in welk van de drie categorieën deze valt:

   1. Toegestaan
   2. Toegestaan, maar zou niet mogen
   3. Niet toegestaan

   De code

   ```
   f1_arr(arr, 5);
   f1_arr(c_arr, 5);

   f2_arr(arr, 5);
   f2_arr(c_arr, 5);
   ```

b. Doe hetzelfde als bij (a) voor de volgende code

   ```
   f1(arr);
   f1(c_arr);

   f2(arr);
   f2(c_arr);

   f3(arr);
   f3(arr);
   ```

   Doe dit met enige zorgvuldigheid, een aantal zaken kan je wellicht enigszins verbazed. Denk na of je het toch kan verklaren.

5.8.3 Toelichting

Op enig moment volgt hier wellicht een meer uitgebreide toelichting. Nu denk ik dat met behulp van de *compiler* bovenstaande opdrachten goed uit te voeren zijn.

5.9 Functiepointers en Interrupts

5.9.1 Doel

- Je kan pointers naar functies definiëren en gebruiken.
- Je kan een *interrupt service routine* schrijven.

- Je kent de regels rondom het gebruik van *interrupt service routines*.
- Je kent de voordelen van interrupts boven de gebruikelijke vorm van "check in loop" (*polling*).

5.9.2 Opdrachten

5.9.2.1 Flexibele code met functiepointers. Stel dat we een functie hebben waarmee we klinkers kunnen herkennen.

```
int isKlinker(char c) {
    if (c == 'a' || c == 'e' /* en de rest */) {
        return 1;
    }
    return 0;
}
```

Op dezelfde manier kunnen we ook herkenners schrijven van andere karakters (getallen, bijzondere tekens, letters na de `k` in het alfabet, hoofdletters, etc).

- Schrijf nog zo'n herkennerfunctie.
- Schrijf een functie die een *null terminated character array*[32] accepteert én een herkennerfunctie, dus iets als:

  ```
  int counter (const char* txt, /* functie pointer */) {
      // implementatie
  }
  ```

- Schrijf tot slot een `main` functie waarin je deze `counter` meerdere keren op een mooie tekst loslaat met telkens een andere "herkennerfunctie".

5.9.2.2 Interrupts

1. Zoek in de Arduino documentatie ('AttachInterrupt, Arduino Reference' z.d.) het volgende op:
   - welke pinnen zijn geschikt voor interrupts?
   - wat is de functie van `digitalPinToInterrupt()`?
   - wat is het effect van `CHANGE`, `RISING`, `FALLING`?
2. Bekijk het codevoorbeeld in de Arduino documentatie ('AttachInterrupt, Arduino Reference' z.d.):

   Waarom wordt `byte state` met `volatile` gemarkeerd?
3. Er wordt gezegd dat je vanuit een zgn. *interrupt service routine* niet naar `Serial` kan schrijven.

   Onderzoek met je Arduinokit of dit ook voor jouw Arduino het geval is.

   Wat is de redenering achter deze uitspraak?
4. Waar wordt in bovenstaande code gebruik gemaakt van zgn. functiepointers?

[32] In moderne talen bekend onder de naam `string`.

Waarom.

Wat is de syntax waarmee je een functiepointer definieert?

Wat is de syntax waarmee je een typedefinitie van een functiepointer definieert?

5.9.3 Toelichting

5.9.3.1 Functiepointers Zoals het mogelijk is om een *pointer* naar een variabele te maken, is het ook mogelijk om *pointers naar functies* te maken. Een *functiepointer* is een verwijzing naar een *functie van een bepaald type*, dus, onderdeel van de verwijzing zijn de argumenten en de return-waarde van de functie.

Stel de volgende functies voor (de exacte nietszeggende inhoud van de functie doet nu niet ter zake):

```
int foo(char c, float f) { return 2; }

int bar(char d, float g) { return 3; }

void baz(char e) { printf("run baz with %c\n", e); }
```

De functies `foo` en `bar` zijn van *hetzelfde type*, namelijk een `char` en een `float` als parameters (in die volgorde) en een `int` als returnwaarde.

Functie `baz` is van een ander type omdat deze functie een andere combinatie van argumenttypen en returnwaarde heeft.

Laten we nu een *pointer* naar `foo` maken. De definitie van deze *pointer* ziet er complex uit omdat returnwaarden en argumenten onderdeel zijn van de definitie van de *pointer*[33]

```
int (*fp)(char, float) = foo;
```

Aan de linkerkant van het `=`-teken staat bijna een normale functie behalve het in rood aangegeven gedeelte. Het `(*)` gedeelte leest de compiler als dat het hier om een *functie-pointer* gaat, de `fp` in `(*fp)` geeft aan dat de variabele `fp` de pointer naar de functie is.

Nu kunnen we de *pointer* gebruiken alsof het een functie is:

```
printf("%d\n", fp('a', 1.0)); // print 2
```

Het aardige van een functie-pointer is is dat we deze naar een andere functie kunnen laten verwijzen:

```
fp = bar;
printf("%d\n", fp('a', 1.0)); // print 3
```

Omdat functie `baz` van een andere type is, mag het volgende niet[34]

```
fp = baz; // warning: wrong function type
```

[33] Net als dat bij "gewone" *pointers* het type waarheen verwezen wordt onderdeel is van die *pointer*, zo is het bijvoorbeeld: *pointer* naar een `int`.

[34] Mijn *compiler* geeft hier slechts een waarschuwing, het was wellicht beter geweest wanneer deze code niet compileert.

Wel kunnen we een tweede type functiepointer maken waar die verwijst naar `baz`. Ter illustratie laten we deze eerst naar `NULL` verwijzen.

```
void (*fp2)(char e) = NULL;
fp2 = baz;
fp2('a');
```

5.9.3.2 Interrupts Een applicatie in C doorloopt de structuur die in de code is aangegeven door middel van functies: bij uitvoeren begint de applicatie bij `main`, doorloopt *functies*, wandelt door *loops* en *condities* worden gecontroleerd.

Echter, is bestaat een constructie die hier loodrecht op staat. De normale loop der dingen in een applicatie kan worden *onderbroken*. Meestal is dit door een signaal van hoge urgentie waar actie op ondernomen moet worden. In het engels heet een dergelijke constructie een *interrupt* (Voor uigebreide informatie zie Gammon z.d.).

Op de Arduino is een aantal pinnen aanwezig dat in staat is te luisteren naar een dergelijk signaal. Op moment dat de spanning op zo'n pin plotseling verandert, wordt de zogenaamde *interrupt handler*, ook bekend als *interrupt service routine* uitgevoerd.

Op moment dat een dergelijk signaal binnenkomt wordt de *interrupt service routine* dan ook op het allereerste moment dat mogelijk is uitgevoerd. Dit is letterlijk zo. Zelfs in het geval dat en een integer[35] geschreven wordt kan het proces na het schrijven van de eerste byte onderbroken worden, dan wordt de *interrupt service routine* uitgevoerd om pas daarna de tweede byte van de weg te schrijven integer waarde weg te schrijven.

Dit heeft een aantal consequenties:

1. De uit te voeren *interrupt service routine* (ISR) moet zo klein mogelijk van omvang zijn.
2. Er is functionaliteit die in een *interrupt service routine* niet beschikbaar is.[36]
3. Wanneer je in een *interrupt service routine* naar een variabel schrijft die elders in je applicatie gebruikt wordt, dan moet je dat gebruik beschermen tegen onverhoeds gedeeltelijk overschijven in de *interrupt service routine*.
4. Compilers optimaliseren je code. Een van de controles die in het algemeen worden uitgevoerd is dat code die vanuit de `main` niet wordt geraakt weggeoptimaliseerd worden.

   Variabelen die alleen in de *interrupt service routine* veranderen worden als gevolg van deze optimalisatie gezien als constante waarde. De *compiler* is namelijk niet slim genoeg om code in de ISR te herkennen.

   Dergelijke variabelen moeten beschermd worden tegen wegoptimaliseren.

[35] 2 bytes op de Arduino.

[36] In eerdere uitvoeringen van de Arduino kon je in een ISR niet schrijven naar de seriële monitor omdat dat op zijn beurt interrupt-gebaseerd was.

Laten we door een stukje Arduino-code met een ISR heenlopen[37]. De code is afkomstig van wederom de Arduino Reference ('AttachInterrupt, Arduino Reference' z.d.).

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin),
    blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

Met `attachInterrupt` wordt een de controle op een *interrupt* in de code ingeschakeld. Deze functie kent 3 argumenten.

- Het eerste argument is het *interrupt nummer*. Deze is in een tabel op te zoeken, maar de functie `digitalPinToInterrupt` vertaalt voor jou pinnummers naar interruptnummers.
- Het tweede argument is de *functiepointer* die verwijst naar de *interrupt service routine*, dat is de functie `blink`.

  Als bij functiepointers is dit een verwijzing naar een type functie, namelijk:

  ```
  void (*isr) ();
  ```

  Dus, geen argumenten, geen returnwaarde.
- Het laatste argument geeft aan wanneer het *interrupt* af gaat. In het voorbeeld is dat wanneer de spanning op de lijn verandert. Tbl. 9 geeft de verschillende mogelijkheden.

  Bij `LOW` zal het interrupt heel vaak afgaan. Dit wordt het minst gebruikt.

Tabel 9: Mogelijke *triggers* voor een interrupt op een Arduino.

waarde	betekenis
`CHANGE`	de spanning op de lijn verandert
`FALLING`	de spanning gaat naar beneden
`RISING`	de spanning gaat omhoog
`LOW`	de spanning is laag

[37] Deze code is *niet* van toepassing in onze C-code, het is specifiek Arduino C++-code.

- In de functie `blink` wordt de waarden van `state` veranderd. De compiler zal `state` niet als variabel herkennen en deze bij optimalisatie als een constante beschouwen.

  Door de variabel `state` als `volatile` te kwalificeren, blijft de compiler `state` als mogelijk te veranderen zien (zie ook 'Volatile Type Qualifier' z.d.).

Tot slot, delen van je code zijn te markeren als dat deze niet onderbroken mogen worden door interrupts. Redenen kunnen zijn dat de code tijdskritisch is en de uitvoering niet langer dan de code zelf duurt. Een andere reden is dat hier variabelen gelezen of geschreven kunnen worden die ook in de ISR van waarde veranderen.[38]

Wederom uit de Arduino Reference ('interrupts() Arduino Reference' z.d.):

```
void setup() {}

void loop() {
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Interrupts die in het kritische gedeelte binnenkomen worden gebufferd en nadat ze weer mogen worden afgehandeld (aangegeven met `interrupts()`) alsnog uitgevoerd.

Houd dus ook de kritische code zo beperkt mogelijk. Je wil uiteindelijk niet dat een interrupt als gevolg van een alarm op hartbewaking of het afgaan van een airbag wordt uitgesteld doordat we in een `noInterrupts()` gedeelte zitten...

5.10 Circulaire Buffers

5.10.1 Doel

- Je kent de werking van een circulaire buffer.

  Je kan deze toevoegen aan je project.

- Je kan een circulaire buffer die zich in de *free store* bevindt schrijven.
- Je kan C code in een Arduino-project gebruiken en deze ook met je eigen `Makefile` lokaal testen en onderhouden.

5.10.2 Opdracht

In deze opgave implementeer je een circulaire buffer waar je ook in je eindopdracht plezier van gaat hebben. Nu is dit natuurlijk waar voor alle opdrachten waar je aan werkt, maar vooruit, je snapt de hint.

Je gaat dit project volgens de volgende strucuur opzetten:

[38] Deze tweede reden is in het algemeen geen goed idee. Voorkom dat in de ISR variabelen geschreven worden die meer dan een *byte* lang zijn en dit probleem doet zich niet voor.

```
16_circBuffer/
+-- circBuffer
    +-- Makefile
    +-- cbuff
    |   +-- circularbuffer.c
    |   +-- circularbuffer.h
    +-- unittests.c
```

Ik vind het geen enkel probleem wanneer `16_circBuffer` bij jou iets anders heet.

Je krijgt `circularbuffer.h` en `unittests.c`. Aan jou de opdracht om `circularbuffer.c` en `Makefile` te schrijven zodat alle tests in `unittests.c` met succes doorlopen worden.

Deze opdracht is in meerdere stappen opgedeeld.

5.10.2.1 header file

Begrijp je allemaal wat hier staat?

```
#ifndef circularbuffer_h
#define circularbuffer_h

#include <stdint.h>

/**
 * define the variable type to store in the buffer
 */
typedef unsigned long cbtype;
/**
 * buffers come in two variants
 */
enum cbmode { OVERWRITE_IF_FULL, IGNORE_IF_FULL };
/**
 * buffer data, do not modify in client!
 */
typedef struct {
  cbtype* data;
  enum cbmode mode;
  int8_t size;
  int8_t start;
  int8_t count;
} cbuffer;

/**
 * initialize a new buffer, in case NULL is returned,
 * buffer init failed
 */
cbuffer* cbInit(int8_t size, enum cbmode mode);
/**
 * free the buffer, a new buffer can be created
 * instead.
 *
 * cbFree returns NULL to allow for b = cbFree(b);
 */
cbuffer* cbFree(cbuffer* buffer);
```

```
/**
 * check whether data can be read from the buffer
 */
int cbAvailable(cbuffer* buffer);
/**
 * peek the oldest value in the buffer, value
 * remains available for read.
 */
cbtype cbPeek(cbuffer* buffer);
/**
 * read and remove the oldest value in the buffer.
 */
cbtype cbRead(cbuffer* buffer);
/**
 * add a new value to the buffer, adding may
 * fail depending on the buffer mode.
 */
int8_t cbAdd(cbuffer* buffer, cbtype value);

#endif
```

5.10.2.2 unittests

```
#include "cbuff/circularbuffer.h"
#include <glib.h>

void buffer_init() {
  cbuffer* b1 = cbInit(100, OVERWRITE_IF_FULL);
  g_assert_true(b1);
  g_assert_true(b1->data);
  g_assert_cmpint(b1->size, ==, 100);
  g_assert_cmpint(b1->start, ==, 0);
  g_assert_cmpint(b1->count, ==, 0);

  b1 = cbFree(b1);

  g_assert_false(b1);
}

void buffer_add_overwrite() {
  cbuffer* b = cbInit(3, OVERWRITE_IF_FULL);

  int succes = cbAdd(b, 100);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);

  succes = cbAdd(b, 200);
  g_assert_true(succes);
  g_assert_cmpint(b->data[1], ==, 200);

  succes = cbAdd(b, 300);
  g_assert_true(succes);
  g_assert_cmpint(b->data[2], ==, 300);
```

```
  succes = cbAdd(b, 400);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 400);

  b = cbFree(b);
}

void buffer_add_ignore() {
  cbuffer* b = cbInit(3, IGNORE_IF_FULL);

  int succes = cbAdd(b, 100);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);

  succes = cbAdd(b, 200);
  g_assert_true(succes);
  g_assert_cmpint(b->data[1], ==, 200);

  succes = cbAdd(b, 300);
  g_assert_true(succes);
  g_assert_cmpint(b->data[2], ==, 300);

  g_assert_cmpint(b->mode, ==, IGNORE_IF_FULL);
  g_assert_cmpint(b->size, ==, 3);
  g_assert_cmpint(b->count, ==, 3);

  succes = cbAdd(b, 400);
  g_assert_false(succes);
  g_assert_cmpint(b->data[0], ==, 100);

  b = cbFree(b);
}

void buffer_read_peek() {
  cbuffer* b = cbInit(4, OVERWRITE_IF_FULL);
  int succes;
  cbtype val;

  succes = cbAdd(b, 100);
  g_assert_true(succes);
  succes = cbAdd(b, 200);
  g_assert_true(succes);
  succes = cbAdd(b, 300);
  g_assert_true(succes);
  succes = cbAdd(b, 400);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);
  g_assert_cmpint(b->data[1], ==, 200);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->data[3], ==, 400);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 4);
```

```
  val = cbPeek(b);
  g_assert_cmpint(val, ==, 100);
  val = cbPeek(b);
  g_assert_cmpint(val, ==, 100);
  val = cbRead(b);
  g_assert_cmpint(val, ==, 100);
  val = cbRead(b);
  g_assert_cmpint(val, ==, 200);

  g_assert_cmpint(b->start, ==, 2);
  g_assert_cmpint(b->count, ==, 2);

  val = cbPeek(b);
  g_assert_cmpint(val, ==, 300);
  val = cbRead(b);
  g_assert_cmpint(val, ==, 300);

  g_assert_true(cbAvailable(b));

  val = cbPeek(b);
  g_assert_cmpint(val, ==, 400);
  val = cbRead(b);
  g_assert_cmpint(val, ==, 400);

  g_assert_false(cbAvailable(b));

  b = cbFree(b);
}

void buffer_add_overwrite_2() {
  cbuffer* b = cbInit(3, OVERWRITE_IF_FULL);
  int succes;

  succes = cbAdd(b, 100);
  g_assert_true(succes);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 1);

  succes = cbAdd(b, 200);
  g_assert_true(succes);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 2);

  succes = cbAdd(b, 300);
  g_assert_true(succes);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 3);

  succes = cbAdd(b, 400);
  g_assert_true(succes);
  g_assert_cmpint(b->start, ==, 1);
  g_assert_cmpint(b->count, ==, 3);

  succes = cbAdd(b, 500);
```

```
  g_assert_true(succes);
  g_assert_cmpint(b->start, ==, 2);
  g_assert_cmpint(b->count, ==, 3);

  g_assert_cmpint(b->data[0], ==, 400);
  g_assert_cmpint(b->data[1], ==, 500);
  g_assert_cmpint(b->data[2], ==, 300);

  b = cbFree(b);
}

void buffer_add_read() {
  cbuffer* b = cbInit(5, OVERWRITE_IF_FULL);
  int succes;
  cbtype val;

  succes = cbAdd(b, 100);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 1);

  succes = cbAdd(b, 200);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);
  g_assert_cmpint(b->data[1], ==, 200);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 2);

  succes = cbAdd(b, 300);
  g_assert_true(succes);
  g_assert_cmpint(b->data[0], ==, 100);
  g_assert_cmpint(b->data[1], ==, 200);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 3);

  val = cbRead(b);
  g_assert_cmpint(val, ==, 100);
  g_assert_cmpint(b->data[1], ==, 200);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->start, ==, 1);
  g_assert_cmpint(b->count, ==, 2);

  val = cbRead(b);
  g_assert_cmpint(val, ==, 200);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->start, ==, 2);
  g_assert_cmpint(b->count, ==, 1);

  succes = cbAdd(b, 400);
  g_assert_true(succes);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->data[3], ==, 400);
```

```
  g_assert_cmpint(b->start, ==, 2);
  g_assert_cmpint(b->count, ==, 2);

  succes = cbAdd(b, 500);
  g_assert_true(succes);
  g_assert_cmpint(b->data[2], ==, 300);
  g_assert_cmpint(b->data[3], ==, 400);
  g_assert_cmpint(b->data[4], ==, 500);
  g_assert_cmpint(b->start, ==, 2);
  g_assert_cmpint(b->count, ==, 3);

  val = cbRead(b);
  g_assert_cmpint(val, ==, 300);
  g_assert_cmpint(b->data[3], ==, 400);
  g_assert_cmpint(b->data[4], ==, 500);
  g_assert_cmpint(b->start, ==, 3);
  g_assert_cmpint(b->count, ==, 2);

  succes = cbAdd(b, 600);
  g_assert_true(succes);
  g_assert_cmpint(b->data[3], ==, 400);
  g_assert_cmpint(b->data[4], ==, 500);
  g_assert_cmpint(b->data[0], ==, 600);
  g_assert_cmpint(b->start, ==, 3);
  g_assert_cmpint(b->count, ==, 3);

  val = cbRead(b);
  g_assert_cmpint(val, ==, 400);
  g_assert_cmpint(b->data[4], ==, 500);
  g_assert_cmpint(b->data[0], ==, 600);
  g_assert_cmpint(b->start, ==, 4);
  g_assert_cmpint(b->count, ==, 2);

  val = cbRead(b);
  g_assert_cmpint(val, ==, 500);
  g_assert_cmpint(b->data[0], ==, 600);
  g_assert_cmpint(b->start, ==, 0);
  g_assert_cmpint(b->count, ==, 1);

  g_assert_true(cbAvailable(b));

  val = cbRead(b);
  g_assert_cmpint(val, ==, 600);
  g_assert_cmpint(b->start, ==, 1);
  g_assert_cmpint(b->count, ==, 0);

  g_assert_false(cbAvailable(b));

  b = cbFree(b);

  g_assert_false(b);
}

int main(int argc, char** argv) {
```

```
  g_test_init(&argc, &argv, NULL);

  g_test_add_func("/buffer/init", buffer_init);
  g_test_add_func("/buffer/add/overwrite",
                  buffer_add_overwrite);
  g_test_add_func("/buffer/add/overwrite/2",
                  buffer_add_overwrite_2);
  g_test_add_func("/buffer/add/ignore", buffer_add_ignore);
  g_test_add_func("/buffer/read/peek", buffer_read_peek);
  g_test_add_func("/buffer/add/read", buffer_add_read);

  return g_test_run();
}
```

5.10.3 Toelichting

Circulaire buffers spelen een rol in situaties waarin in korte tijd meer data binnenkomt dan verwerkt of opgeslagen kan worden. In een *controller* met beperkt geheugen kunnen we ervoor kiezen om in geval er meer data binnenkomt dan verwerkt kan worden alleen de meest recente data op te slaan. Hiervoor kan een circulaire buffer gebruikt worden.

5.10.3.1 Essentie Circulaire Buffer Een circulaire buffer is niet veel anders dan een array-achtige verzameling waarin binnenkomende waarden achter elkaar worden ingeschreven. Op moment dat de laatste positie is beschreven wordt de eerstvolgende waarde aan het begin van de array geschreven en wordt vandaar oude waarden overschreven[39]

Laten we aan de hand van een getallenvoorbeeld in tbl. 10 kijken. We hebben hiervoor een buffer van heel beperkte omvang: vier waarden. Zo op het oog is dit een heel logisch proces.

Tabel 10: Acties op een circulaire buffer en de gevolgen ervan.

actie	0	1	2	3
Initialiseer een lege buffer				
Sla 10 op	10			
Sla 20 op	10	20		
Sla 30 op	10	20	30	
Lees oudste waarde (10)		20	30	
Sla 40 op		20	30	40
Sla 50 op	50	20	30	40
Sla 60 op	50	60	30	40
Lees oudste waarde (30)	50	60		40

Intern wordt een huishouding bijgehouden met `size`, `count`, en `start`.

`size`: De grootte van de circulaire buffer, deze waarde verandert niet.
`count`: Het aantal waarden dat in de circulaire buffer is opgeslagen.

[39] Dat is het circulaire idee.

`start:` De locatie van de oudst geschreven waarde in de buffer.[40]

Voor de reeks acties hierboven is de administratie weergegeven in tbl. 11. De administratie geeft steeds de situatie weer nadat de actie is uitgevoerd.

Tabel 11: De interne administratie in de circulaire buffer op de acties uit tbl. 10

na de actie	size	count	start
Initialiseer een lege buffer	4	0	0
Sla 10 op	4	1	0
Sla 20 op	4	2	0
Sla 30 op	4	3	0
Lees oudste waarde (10)	4	2	1
Sla 40 op	4	3	1
Sla 50 op	4	4	1
Sla 60 op	4	4	2
Lees oudste waarde (30)	4	3	3

5.10.3.2 Functies Het is niet de bedoeling dat iedere gebruiker van een circulaire buffer zelf de administratie gaat aanpassen. Sterker, geen enkele gebruiker zou weet moeten hebben van de administratie[41].

De circulaire buffer die wij gaan bouwen wordt aangesproken via een aantal functies.

`int available():` Deze functie geeft aan of volgens de administratie waarden zijn opgeslagen in de circulaire buffer.

`add(value):` Met deze functie kan je een waarde aan de circulaire buffer toevoegen.[42]

`value read():` Met deze functie wordt de oudste waarde gelezen. Dit kan alleen als er waarden beschikbaar zijn (`available()`). Eenmaal gelezen is de waarde verdwenen.

`value peek():` Met deze functie kijk je wat de eerstvolgende uitvoering van `read()` zou opleveren. Dit kan alleen als er meer waarden beschikbaar zijn (`available()`). Je kan meerdere keren `peek()` achter elkaar uitvoeren omdat `peek()` de administratie niet wijzigt.

5.11 Parser in C

5.11.1 Doel

- Je kan beschrijven wat een tokenizer en een parser doen.
- Je kan C code lezen en deze door middel van concrete vragen verhelderen.
- Je kan een gedeeltelijke implementatie in C lezen, *refactoren* en uitbreiden.

[40] Of, in geval van een lege buffer, de locatie waar de eerste waarde komst te staan.

[41] Er bestaan ook andere administraties, bijvoorbeeld `size`, `start`, `stop`.

[42] We geven de buffer één van twee modi, de versie die hier wordt uitgelegd waarin oude waarden zonodig worden overschreven, of een veel eenvoudiger waarin zonodig nieuwe waarden worden genegeerd.

5.11.2 Opdracht

5.11.2.1 Krijg voorbeeldimplementatie draaiend Om deze opdracht uit te voeren heb je de voorbeeldimplementatie nodig die de docent je zal uitreiken. Deze implementatie omvat ondermeer:

- Een basisimplementatie van een *parser* en een *tokenizer* in C.

  Deze implementatie voldoet niet aan alle codestandaarden die we in deze cursus verwachten. Bijvoorbeeld, de code moet over meerdere bestanden verdeeld worden en veel functies zijn te lang.

- Een `ino`-bestand dat gebruik maakt van de C-implementatie van *tokenizer* en *parser*.

- Een C unittest waarin enkele complete requests door de C implementatie wordt afgehandeld.

  Dit is geen typisch gebruik van unittest. Een unittest test in het algemeen één functie. Juist die tests ontbreken nog.

- Een `Makefile` om de C-code te controleren en de unittests uit te voeren.

- Een testscript in Python en een bijbehorende `Makefile` om een testomgeving op te zetten en de test uit te voeren die de HTTP requests zelf uitvoert en het resultaat controleert.

  Een deel van die tests faalt nog.

De opdracht is:

- Zorg dat je de unittests van de voorbeeldimplementatie kan uitvoeren (deze zouden allen moeten slagen).

- Zet het Arduino project (de map met ondermeer de het `ino`-bestand) op je Arduino met Ethernet-shield.

  Zorg dat de *shield* verbonden is met je laptop. Je zou nu de de python-test moeten kunnen bouwen en uitvoeren (`make` in de juiste map, kijk of je de betreffende `Makefile` ook snapt).

5.11.2.2 Bestudeer de code De meeste C code moet op dit moment leesbaar zijn, maar er zitten hier en daar nieuwe of niet vaak gebruikte constructies in.

- Kijk de C-code in het mapje `server` globaal door.

- Bestudeer de C-code in de *header file* en formuleer de vragen die je beantwoord moet hebben om die code te snappen.

  Kijk of je een antwoord kan vinden (met elkaar overleggen) en stel de onbeantwoorde vragen aan de docent.

5.11.2.3 Slagende Python acceptatietests

- Wat is het verschil tussen de unittests in C en de acceptatietests in Python?

- Een van Python-tests slaagt nog niet. Om deze te laten slagen heb je een idee van de werking van de code in C nodig. De oplossing beperkt zich tot een aanpassing van het `ino`-bestand.
- Zorg dat je van het `ino`-bestand snapt:
  a. Hoe wordt C-code geïnclude?
  b. Waar wordt de `F`-macro voor gebruikt?
  c. Hoe wordt geregeld dat vanuit de C-code berichten over de seriële verbinding kunnen worden gestuurd terwijl de C-code geen enkele weet heeft van het bestaan daarvan?

5.11.2.4 Laat de C code POST request afhandelen Voor deze opdracht is goed begrip van de *tokenizer* en *parser* nodig. Serieus werk aan de vorige opdrachten moet je een heel eind gebracht hebben. Het laatste beetje doe je in deze opdracht op.

Een POST opdracht heeft data in de *message body* en als gevolg daarvan ook een `Content-Length` *header*. Vooralsnog mag je de `Content-Length` negeren en het *request* opbouwen uit alleen een *request line* en de *message body*, dus

```
POST /data HTTP/1.0

63
```

In dit *request* wordt de waarde 63 opgestuurd.

- Schrijf een unittest in C die bovenstaande POST *request* test (zorg voor de correctie *white space* en *end-of-line* karakters). Deze test faalt uiteraard.
- Denk na wat de *tokenizer* moet doen. Schrijf een *unit test* in C die dat gedrag test.

  Je kan hiervoor meerdere keren de *tokenizer* functie op de inhoud van je *request* loslaten en controleren of je de juiste resultaat krijgt. Zonodig moet je de declaratie van de functie in je *header file* zetten om de unittest te kunnen compileren.
- Denk na wat de *parser* moet doen. Voer de vorige opdracht uit voor de *parser* functie.

  In je unittest moet je nu steeds de *tokenizer* functie uitvoeren gevolgd door de *parser* functie.
- De combinatie van beide opgaven zou moeten leiden tot een takenlijstje van taken die je moet doen om deze opdracht voor elkaar te krijgen.
- Denk na over hoe je de *response* struct moet uitbreiden. Je zal een *response code* moeten toevoegen en het is ook wel fijn dat de opgestuurde waarde ('63') met die *response* mee de Arduino code inkomt zodat deze beschikbaar komt voor gebruik.

  Dit gebruik is nu niet meer dan een uitvoer naar de seriële monitor.
- Wanneer je dit werkend hebt, breid dan de `ino`-code verder uit en voeg een acceptatietest toe.

- Wanneer je mogelijkheden ziet om ook ondersteuning voor de `Content-Length` toe te voegen, doe dat dan.

5.11.2.5 Refactor de C code De code die je hebt gekregen is van onvoldoende kwaliteit. In ieder geval zijn de functies te lang en wellicht zit er teveel code in één bestand.

- Refactor de C code.
- Doe een *regressietest* op de implementatie.

  (Dat wil zeggen dat je je bestaande tests uitvoert om na te gaan of de code zich nog identiek gedraagt).

5.11.3 Toelichting

5.11.3.1 Voor degenen die `platform.io` gebruiken Bij IoT techniek heb je mogelijk leren werken met PlatformIO. In plaats van de `ino`-bestanden die je bij EPD hebt gebruikt schrijf je je code in `cpp` bestanden.

Waar in `ino`-bestanden automatisch `Arduino.h` wordt ge-include, moet je dat in de `cpp`-bestanden expliciet doen. In combinatie met includen van `ethernet.h` is de volgorde van balang: eerst `Arduino.h`, daarna `ethernet.h`. Wanneer je in de omgekeerde volgorde include, dan werkt je ethernetverbinding niet.

6 Ethiek

Als echte techneuten in een snel ontwikkelende omgeving kunnen we voor steeds meer problemen een oplossing bedenken. Maar, we moeten ons ook de vraag stellen of we dat altijd zouden moeten willen. In dit kader proberen we op grond van argumenten verschillende gezichtspunten uit te wisselen.

6.1 Trolley Problem

Zie https://www.youtube.com/watch?v=1sl5KJ69qiA ("*The Trolley Problem in Real Life*", VSauce, 35 min).

- Het klassieke voorbeeld van de ethiek-discussie.
- Centraal staat "wat zou je doen, wanneer niets doen de dood van 5 personen betekent en de *handel*The Trolley Problem in Real Life overhalen actief de dood van 1 andere persoon".
- Uit filmpje van `vsauce` blijkt dat veel mensen verstarren en helemaal niets kunnen doen. Daar houdt ethiek helemaal geen rekening mee.
- Volgens Steen (2023) heeft de Engelse filosofe en bedenker van dit gedachtenexperiment, Philippa Foot, het helemaal niet bedacht als een praktisch probleem ten opzichte van zelf-rijdende auto's.

  Of meer, het gaat om de complexiteit van dergelijke beslissingen, niet om het idee dat een dergelijk probleem softwarematig is op te lossen.

  Het doel was om menselijke voorkeuren vast te stellen, heel algemeen blijkt:

  - mensen boven dieren,
  - grote aantallen boven kleine,

  maar ook culturele verschillen

  - sommige culturen kiezen jong boven oud, andere culturen andersom.

- Dit probleem wordt gepresenteerd als een dilemma (de keuze uit twee opties.)

  Dit is een klassiek stijlfiguur, en er is ook altijd een “derde” uitweg.

  Vragen die ook gesteld kunnen worden (Steen 2023):

  - Waarom zit er geen claxon op de trein?
  - Waarom doen de remmen het niet?
  - Zijn er soortgelijke ongelukken geweest, wat is er gedaan met de analyse ervan?
  - Is onderhoud gebrekkig?
  - Is er bezuinigd op onderhoudsbudget?
  - Is er op onderhoudspersoneel bezuinigd?
  - Krijgt personeel fatsoenlijk betaald?
  - Is er een cultuur waarin problemen kunnen worden aangekaart?
  - Zijn werkomstandigheden behoorlijk?
  - Zijn er goede procedures voor personeel dat aan het spoor werkt?
  - Is treintransport geprivatiseerd? Met welk doel?
  - Staat door concurrentie op het spoor de veiligheid onder druk?
  - Is er centrale verkeerscontrole? Waarom greep deze niet in?

- Concusies:
  - Er is meer dan alleen rationeel gedrag
  - Juist ethische discussie (de vraag “wat is goed?”) kan vaak zinvol breder getrokken worden.

6.2 Post Office schandaal

https://news.ycombinator.com/item?id=38937705, https://www.postofficehorizoninquiry.org.uk/

- Toon begin van de vier delige serie: deze heeft schandaal weer onder de aandacht gebracht, 50 slachtoffers hebben zich alsnog gemeld.
- Post office is een oud instituut dat communicatie in Brits koninkrijk gaande te houden met sporen terug naar 1516.

- Vanaf 1999 werd een nieuw systeem uitgerold, Horizon.

  Er zat een fout in de berekening van de balans.

  Sub-postmasters zijn vaak kleine zelfstandigen met een contract met de Post Office, een van de zaken waarvoor ze tekenen is dat tekorten persoonlijk aangevuld moeten worden.

- *Sub-postmasters* kregen bij problemen te horen dat zij de enige waren.
- Post Office heeft het recht zelf justitieel onderzoek te doen, hoeft niet door politie gedaan te worden.

  Ik begrijp ook dat Post Office al vrij snel intern op de hoogte was dat er wat speelde.

- Er is rumour geweest en onderzoeken, Post Office heeft tegengewerkt.
- Tussen 1999 en 2015 meer dan 4000 beschuldigd en 900 *subpostmaters* veroordeeld voor diefstal en fraude, 236 tot gevangenisstraf.

Naar aanleiding van deze casus:[43]

- De vraag die je je kan stellen is "waar ligt jouw verantwoordelijkheid wanneer je ergens in deze keten werkt?"

  Vanuit ethiek wordt gezien dat je verantwoordelijkheid hebt daar waar je kennis hebt en waar je verantwoordelijkheid krijgt (*knowlegde and agency*, Steen 2023, hfdstk 7).

- Ons handelen / gedrag zit op een schaal van zwart (fout) over vele gradaties grijs naar wit. Over zwart hoeven we het niet veel te hebben. Alleen maar wit is heel vaak geen optie. Van de voormalige Amerikaanse minister van BZ, Henry Kissinger wordt gezegd deze quote te gebruiken: "Mij probleem is niet dat ik tussen 'goed' en 'slecht' moet kiezen, maar heel vaak tussen twee soorten 'slecht' " (helaas geen bron).
- Dit nadenken over "wat is goed te doen" wordt ondergebracht in de tak van filosofie die "ethiek" heet.

  Dit is een vakgebied dat door de tijd ontwikkeld, meestal doordat een vorm van ethiek gaten laat vallen die weer gedicht moeten worden.

- Een anders (sterkere) drijvende kracht achter ontwikkeling van ethiek zijn maatschappelijke problemen die potentieel een bedreiging vormen en die dus moeten worden aangepakt.

  Dit was al zo bij wat we beschouwen als begin van de ethiek, de drie griekse filosofen Sokrates, Plato, en Aristoteles. Hedonisme ("persoonlijk genot boven alles") was een bedreiging omdat de maatschappij aan decadentie ten onder dreigde te gaan (klinkt dit ietwat bekend?).

  > Ik wil iets zeggen over sofisten naar filosofen, van retoriek naar didaktiek, van waarde naar waarheid.

[43] Er is een kans dat ik omwille van de korte introductie in de ogen van sommigen van de ontwikkeling van ethiek een karikatuur maak.. Lees een goed boek wanneer je beter geïnformeerd wil worden.

"Retoriek" heeft nog steeds een grote plaats: kunst.

6.3 Stromingen in de ethiek

6.3.1 Aristoteles

"Het is het waard om voor te leven"

risico's:

- decadentie
- kan heel erg een innerlijke waarde zijn die moeilijk te delen is, of zelfs in conflict.

verder:

- beide zijn aanleiding om "waarheid" boven "waarde" te zetten, "waarheid" is iets wat tussen mensen gedeeld wordt.

6.3.2 Kant

"Er zijn logische principes"

Dit is een verabsolutering van een waarheid die buiten ons staat, maar waar we via logica onszelf en anderen kunnen overtuigen.

risico's:

- vergeet het logische en leef vol overtuiging uit niet-logische principes
- verlies de menselijke realiteit uit het oog

6.3.3 Mill

"Je moet een logische afweging maken"

Is dit een manier om iets aan Kant's risico's te doen?

risico's

- Het is lastig appels met fietspompen vergelijken
- Neem je de *externalities* mee?

6.4 Context

Wat is de situatie waarin we ons wagen aan ethische uitspraken? Ethische problemen lijken misschien gemakkelijk op te lossen wanneer we op een zaterdag onder vrienden wat ethische thema's de revue laten paseren. Maar wat als dezelfde vragen in een officiële *meeting* aan de orde komen en er hangt bijvoorbeeld een investering van vele miljoenen vanaf?

In een voetbalkleedkamer laat je je wellicht andere uitspraken ontlokken dan erbuiten. Is dat erg?

Een gast aan tafel bij een van de vele praatprogramma's roept wat. De kijker die zich na een dag hard werken wat probeert te ontspannen kan zich daardoor

laten amuseren. Maar heeft die gast aan tafel dezelfde autoriteit wanneer jij je mening moet vormen over een thema en je mening is weer van doorslaggevend belang in een besluit dat moet worden genomen?

Een theatergroep speelt op tv (weer aan een praattafel) een stuk over wat er mis is in de bankensector. Maar, wat willen ze? Willen ze iets aan de kaak stellen, of willen ze kaartjes verkopen. Dat antwoord is vast een beetje van beide, maar hoe groot is de druk om ten gunste van die kaartje hier en daar de waarheid zwarter voor te stellen dan deze werkelijk is?

Een collega vraagt bij de koffiemachine welk *framework* je suggereert. Dezelfde collega vraagt dezelfde vraag in een *meeting* waar besloten wordt welk *framework* te gebruiken. Beantwoord je beide vragen met dezelfde zorgvuldigheid? Het is helemaal niet vreemd wanneer dat niet zo is: bij de koffiemachine kan je je standaard prikkelende antwoord geven die je hobby reflecteert, in de *meeting* zal je waarschijnlijk een veel zorgvuldiger afweging maken.

Kortom,

- Wat zijn je belangen van spreker én van aangesprokene?
- Hoe groot zijn je belangen van spreker én van aangesprokene?
- Is er een derde toehoorder?

  Wat zijn haar belangen.

7 Eindopdracht

Het uitgangspunt van deze opdracht is om een opstelling te ontwikkelen waarin de relatie tussen de metingen van twee sensoren bepaald wordt.

De opdracht wordt individueel uitgevoerd. Zorg dat je regelmatig je voortgang onder je eigen account in Subversion incheckt.

7.1 Voorbereiding

1. Maak een keuze voor twee sensoren waartussen je de relatie wil bepalen. Overleg met de docent van dienst over je voorstel.

   De voorkeur gaat uit naar twee sensoren die beide dezelfde grootheid (licht, temperatuur) meten maar ieder volgens een andere techniek.

2. Maak een stappenplan voor ontwikkeling van de opstelling. Houdt hierbij het uitgangspunt van de opdracht voor ogen. De functionele requirements moeten herkenbaar zijn in de te zetten stappen.

   Je hebt de vrijheid om extra stappen toe te voegen of om requirements die helpen bij de testbaarheid van je opstelling naar voren te halen ondanks dat dergelijke requirements niet direct bijdragen aan het uitgangspunt van deze opdracht.

   Bespreek het stappenplan met je docent voordat je begint.

3. Let bij het uivoeren van de tests op de faciliteiten die in de `Makefile` worden aangeboden. Bijvoorbeeld, de volgende opdrachten voeren alles uit wat nodig is om afhankelijkheden te installeren en test uit te voeren:

```
$ make          # draai integratietests op je C-code op je laptop
$ make arduino  # draai acceptatietests op je Arduino server
$ make rpi      # draai acceptatietests op je RPi server
```

7.2 Schriftelijk gedeelte van de eindopdracht

Let op: dit gedeelte is aangepast ten opzicht van de eindopdracht in *reader* versie 1.2.

Dit gedeelte van de opdracht kan je pas zinvol maken nadat je je een werkende oplossing hebt gebouwd die wordt beschreven in sec. 7.3. Je wordt in dit gedeelte gevraagd je oplossing te vergelijken met code-stijlvoorschriften. Het is op dit moment geen probleem dat je oplossing niet in overeenstemming is met de code-stijlvoorschriften, het gaat er alleen om dat je herkent waar je code niet in overeenstemming is met de code-stijlvoorschriften.

Zoals je wellicht al begrepen wordt C nog steeds veel gebruikt voor het schrijven van *embedded software*. Ook begrijp je ondertussen dat een fout gemakkelijk gemaakt is. Om de kans op fouten te verkleinen zijn er standaarden. Eén voorbeeld is een heel compacte set van de NASA en *jet propulsion laboratory* met 10 *rules for developing safety-critical code* (Holzmann 2006), een kopie is bijgevoegd in Onderwijs Online, of, vraag anders de docent.

1. Lees het artikel (dus, Holzmann (2006)).
2. Loop de 10 regels langs en geef aan waar je code zich niet houdt aan de voorschriften in Holzmann (2006).

   Geef kort (een of enkele alinea's) aan of het haalbaar is om alsnog aan de voorschrift te voldoen. Geen aan waarom niet, danwel hoe dan.

Dit gedeelte van de opdracht resulteert in (naar ik verwacht) twee tot drie A4-tjes. Zorg dat dit document voldoet aan de AIM-controlekaart en voeg een pdf-versie toe aan je upload.

In het assessment is deze tekst onderdeel van het gesprek over je oplossing.

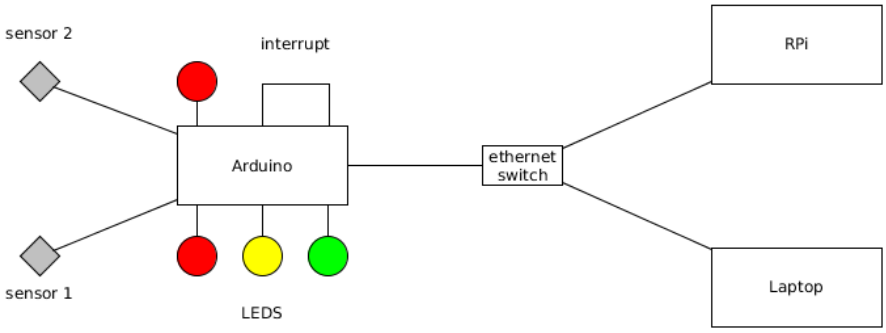
Nogmaals, het is geen probleem wanneer je oplossing niet voldoet aan de 10 regels, wel wordt van je gevraagd zinvol na te denken over je code en over deze regels.

7.3 Functionele Requirements

Bouw een oplossing volgens het schema in fig. 6 en ontwikkel hiervoor de software volgens de gegeven requirements.

1. De Arduino bechikt over een HTTP server die de API zoals verderop is behandeld implementeert.

   Deze server implementeert het HTTP protocol strict en geeft verschillende status codes terug:



Figuur 6: Schets van de te maken opstelling voor de eindopdracht.

code	toelichting
400	fout in de syntax
404	onbekende *request target*
200	honoreer verzoek; zonder wijziging
201	honoreer verzoek; met wijziging

Bij wijziging van interne variabelen als gevolg van een verzoek wordt een 201 teruggegeven.

2. De Arduino heeft twee sensoren die dezelfde grootheid op verschillende manieren bepaalt.

3. De Arduino leest tien keer per seconde beide sensoren uit.

   De meetwaarden worden in twee circulaire buffers opgslagen (voor beide sensoren één). De grootte van de circulaire buffers is gelijk aan elkaar en kan via een request worden aangepast (zie hieronder).

   Zorg ervoor dat je bij opstarten van de Arduino werkende circulaire buffers hebt met 12 posities in iedere buffer. Op deze manier ben je niet meteen afhankelijk van het zetten van de grootte via een HTTP *request*.

4. De Arduino houdt een lopend gemiddelde en standaardeviatie van beide sensormetingen individueel bij. Deze waarden gaan dus over meer waarden dan die in de circulaire buffer zijn opgeslagen.

5. Met een request op de Arduino worden de waarden in de aangegeven circulaire buffer uitgelezen en wordt het gemiddelde berekend en teruggegeven.

   De gemiddelden worden als *floating point* waarden berekend en op één cijfer achter de decimale punt afgerond.

   Als gevolg van deze actie wordt de circulaire buffer geleegd.[44]

[44] De `compleet` test in de *unit test suite* dwingt deze regel af. Let erop, de kans is groot dat je ten onrechte de indruk hebt dat er een fout in deze test zit.

6. In analogie met het vorige punt kan ook om de langdurige statistieken van een van beide sensoren gevraagd worden.[45]

   De statistieken worden als *floating point* waarden berekend en op één cijfer achter de decimale punt afgerond.

7. De Arduino beschikt over een interrupt die de berekening van de langdurige gemiddelden reset en de initialisatie van de circulaire buffers in de *default*-grootte uitvoert.

   Deze interrupt kan eenvoudigweg worden getriggert door de interrupt-pin met een draad te verbinden met bijvoorbeeld de 5V pin.

8. De Arduino heeft vier LED's die informatie geven over de status van de Arduino.

   - Er zijn twee rode LEDS. De rode LED gaat branden op moment dat de bijbehorende circulaire buffel vol is.

     (De buffers zelf zijn in `overwrite_if_full` modus).

   - De groene LED is voor tenminste een halve seconde aan op moment dat de buffers en langdurige gemiddeld door middel van een *interrupt* gereset worden.

   - De gele LED is voor tenminste een halve seconde aan op moment dat de Arduino een HTTP request ontvangt en verwerkt.

   - Alle vier de LEDs gaan aan op moment dat een herinitialisatie van een van beide circulaire buffers niet goed gaat.

     Vanaf dat moment hoef je niet meer te garanderen dat je Arduino goed werkt.

9. Je schrijft een python-script dat op je laptop draait en dat iedere 12 seconde informatie bij de Arduino ophaalt en doorstuurt naar de Raspberry Pi.

10. De Raspberry Pi heeft een Rest server (geschreven in Python) die setjes van twee gemiddelde sensorwaarden kan ontvangen en in een SQLite database opslaat.

11. Op moment dat de Raspberry Pi meer dan drie setjes ontvangen en opgeslagen heeft kan de Raspberry de regressielijn berekenen van een tweedegraadsfunctie die de lijn zo goed mogelijk beschrijft.

    Ook wordt de kwaliteit van de regressielijn berekend met R^2.

12. De Arduino implementeert de API uit tbl. 13.

Tabel 13: Te implementeren API voor de Arduino.

route	toelichting
`PUT /config/mode`	`active` of `passive` in msg body
`PUT /config/cbuffsize`	`int` waarde in msg body
`DELETE /sensors/1`	verwijder alle waarden voor `1|2`

[45] Zie de tabel: een apart request voor gemiddelde en standaarddeviatie.

route	toelichting
`POST /sensors/1`	ontvang `int` meetwaarde voor `1|2`
`GET /sensors/1/avg`	geef μ voor `1|2` als in `38.2`
`GET /sensors/2/stdev`	geef σ voor `1|2` als in `5.3`
`GET /sensors/1/actual`	geef m uit buffer als in `37.3`

Opmerkingen:

- De modes zijn `passive` en `active`. Deze waarde wordt in de *message body* van het request verstuurd.

  In *passive mode* worden de sensoren niet uitgelezen, in *active mode* wel. In *passive mode* kunnen dus alleen metingen via HTTP binnenkomen.

- De buffergrootte is een integer waarde die in de *message body* van het request wordt verstuurd.

- De `POST /sensors/1` geeft een integer waarde tussen `0` en `1023` in de *message body*.

- De `DELETE` reset de berekening van de langlopende gemiddelden μ en σ en leegt de circulaire buffer. De circulaire buffer blijft verder intact.

- De `GET avg` en `GET stdev` geven *floating point* waarden in de *message body* afgerond op precies 1 cijfer achter de decimale punt.

  De waarden komen uit de berekening van het langlopende gemiddelden μ en σ.

- De `GET actual` geeft een waarde met een getal achter de decimale punt in de *message body*.

  De waarde m uit tbl. 13 is te berekenen als de gemiddelde waarde van de integer-waarden die in de betreffende circulaire buffer zitten. De waarde m wordt met één cijfer achter de decimale punt verstuurd.

13. De Raspberry Pi implementeert de API uit tbl. 14.

Tabel 14: Te implementeren API voor de Raspberry Pi.

route	toelichting
`POST /data`	ontvang set van twee meetwaarden
`GET /statistics`	geef b_0, b_1, b_2 en R^2
`DELETE /statistics`	verwijder alle opgeslagen waarden

Opmerkingen:

- In de `POST` stuur je twee gemiddelde waarden op door middel van een spatie gescheiden.

  (in de vorige versie stond hier “twee gemiddelde integer waarden”, maar deze waarden komen met een decimale punt van de Arduino.

Kijk dus of je hier zonder verdere problemen werken kan met waarden met een getal achter de decimale punt).

- Met `GET /statistics/` kan een statistische rapportage van de opgeslagen meetparen verkregen worden. Deze bestaat uit:
  - de regressieparameters b_0, b_1, b_2.
  - de *coefficient of determination* R^2.

  De vier cijfers worden door spaties gescheiden en hebben precies 1 cijfer achter de decimale punt.

  Let op dat je bij de berekening van R^2 de juiste benadering uit sec. 4.1.3.4 volgt.

7.4 Beoordeling

1. Deze opdracht is individueel. Je bent de auteur van je eigen code.

   Je mag als uitgangspunt de code gebruiken die bij de behandeling van *parser* en *tokenizer* is uitgegeven.

   Tot in zeer beperkte mate mag externe code gebruikt worden: de bron is duidelijk vermeld.

2. Je ontwikkelt je oplossing aan de hand van de unittest die je hebt gekregen. Je oplossing moet in staat zijn andere, vergelijkbare unittests met succes uit te voeren.

3. Je demonstreert je oplossing en voert de dan verkregen unittest uit.

   Vragen over je oplossing kan je beantwoorden.

4. Gesprek over je bevindingen uit sec. 7.2.

5. Bij goedkeuring heb je in principe een voldoende. Je levert je werk in in isas.

knock-out: geen of onvoldoende verslag, slechte code, twijfel aan eigen code (bijvoorbeeld op grond van de subversion-geschiedenis) of onvoldoende begrip van de oplossing.

8 Achtergrondinformatie

Dit is in hoofdzaak lesmateriaal uit eerdere uitvoeringen. Het is *geen* examenstof, maar wellicht wel op enig moment handig om achter de hand te hebben.

8.1 Authenticatie in HTTP

Let op: met ingang van het eerste semester van het schooljaar 2019-2020 is dit hoofdstuk geen onderdeel meer van de toetsstof van dit vak. Mogelijk komt dit materiaal van pas bij het IoT-project.

8.1.1 Doel

- Je kan op hoofdlijnen beschrijven hoe Digest Access Authenticatie werkt.
- Je kan beschrijven wat de rol is van de verschillende velden in de `WWW-Authenticate` header en de `Authorization` header in het proces van authenticatie.

8.1.2 Toelichting

Documentatie is te vinden in RFC 7616 waar ook het volgende voorbeeld uit komt.

Mufasa stuurt een request naar http://www.example.org/dir/index.html:

```
GET /dir/index.html HTTP/1.1
Host: www.example.org
// tweemaal een crlf ...
```

De server wil dat je je hiervoor *authenticeert*, dus, de server wil in dit geval dat Mufasa het bewijs levert dat Mufasa Mufasa is. Of meer precies, dat de gebruiker in het bezit is van gebruikersnaam en wachtwoord van Mufasa.

De server laat dit met de volgende response weten.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
   realm="http-auth@example.org",
   qop="auth, auth-int",
   algorithm=SHA-256,
   nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
   opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
WWW-Authenticate: Digest
   realm="http-auth@example.org",
   qop="auth, auth-int",
   algorithm=MD5,
   nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
   opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

Er wordt hier gebruik gemaakt van een `WWW-Authenticate` header field, of meer precies, twee van deze velden. Deze geven Mufasa de keuze op welke manier hij zichzelf wil authenticeren. De *header field value* is bijzonder lang en voor de leesbaarheid over meerdere regels afgedrukt.

Wanneer Mufasa gebruik maakt van een browser, dan zal deze browser Mufasa om zijn gebruikersnaam en wachtwoord vragen en vervolgens het volgende request versturen.

```
GET /dir/index.html HTTP/1.1
Host: www.example.org
Authorization: Digest username="Mufasa",
   realm="http-auth@example.org",
   uri="/dir/index.html",
   algorithm=MD5,
   nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
```

```
nc=00000001,
cnonce="f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ",
qop=auth,
response="8ca523f5e9506fed4657c9700eebdbec",
opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

In dit gedeelte is aan het vernieuwde verzoek om `/dir/index.html` in te zien de `Authorization` header toegevoegd op basis waarvan de server kan vaststellen dat het inderdaad Mufasa is die het verzoek doet.

De vraag is nu of we snappen wat er gebeurt. In de tweede plaats gaan we kijken wat allemaal facultatief is, want onze Arduino gaat het zo wel erg zwaar krijgen...

8.1.2.1 WWW-Authenticate response header

- `Digest`

  Dit geeft aan dat we gebruik maken van *digest authentication* en niet van een van de andere opties.

- `realm`

  Dit is informatie op basis waarvan de gebruiker weet welk gebruikersnaam-en-wachtwoord-paar hij moet gebruiken.

  Het is niet gek dat hier een syntax gebruikt wordt die je ook van `ssh` sessies kent.

- `qop` - quality of protection

  De opties zijn `auth` en `auth-int`. Ofwel, de gebruiker mag kiezen of hij alleen authenticatie wil (een berekening op basis van zijn naam en wachtwoord) of dat hij ook een integriteitscheck wil. Voor dit laatste is een berekening nodig waarin ook de inhoud van zijn request wordt meegenomen. De server is dan in staat of het volledige bericht goed is aangekomen of niet.

- `algorithm`

  De server geeft aan welke *hash functies* deze ondersteunt.

  Van oudsher is dit alleen MD5, omdat deze niet meer veilig is kunnen ook andere hashes als `SHA-256` worden gebruikt.

  Hier raken we een interessante discussie. Het is al de vraag of het ons gaat lukken om MD5 op de Arduino draaiend te krijgen. Ik acht de kans dat we SHA-256 draaiend krijgen heel klein.

- `nonce`

  Een nonce is een eenmaal uitgegeven en dus unieke waarde. Deze moet voorkomen dat meerdere maken versturen van hetzelfde verzoek een identieke hash oplevert.

- `opaque`

Dit is een waarde die een client van de server krijgt, de client moet deze mee terugsturen naar de server. De server kan deze waarde gebruiken om informatie over de sessie op te slaan.

8.1.2.2 Authorization request header Deze header wordt dus met het volgende *request* meegestuurd.

- `Digest`

  Net als bij de response header moet worden aangegeven om welke vorm van authenticatie het gaat.

- `response`

  Dit is het resultaat van een berekening waarbij gebruik wordt gemaakt van ondermeer het wachtwoord van de gebruiker. Deze berekening kan niet worden omgedraaid: het is in principe onmogelijk om vanuit de `response` terug te rekenen naar het wachtwoord.

  Met andere woorden, een correcte response is het bewijs dat de verzender in het bezit is van het juiste wachtwoord.

  Zie verderop voor een uitleg over hoe de berekening van het request in elkaar zit.

- `realm`, `nonce`, `opaque`

  Deze waarden zijn directe kopieën van de waarde die de client in het voorgaande response heeft ontvangen.

- `algorithm`, `qop`

  In deze waarden geeft de client aan van welk van de gegeven opties de client gebruik maakt.

- `username`

  De naam van de gebruiker zoals bekend bij de server.

- `uri`

  Normaal gesproken identiek aan de request target. Dit is nodig omdat een *proxy* tussen de server en de client de originele request target mag wijzigen.

- `nc` - nonce count

  Dit is een hexadecimale teller waarmee de client aangeeft hoe vaak de ontvangen nonce gebuikt is.

  Omdat de `nc` wordt meegenomen in de berekening van het bewijs kan de `nc` niet halverwege worden gewijzigd. Nu dat niet meer kan, is de server in staat om na te gaan of niet iemand anders probeert hetzelfde request te versturen.

- `cnonce` - client nonce

  Dit is een willekeurige en unieke waarde die de client gebruikt bij het berekenen van het bewijs en die door de server gebruikt kan worden om na te gaan of het berekende bewijs klopt.

Mocht iets je nog niet duidelijk zijn, of wil je alle details, bestudeer dan RFC 7616 verder.

8.1.2.3 Berekening van het bewijs De berekening van de `response` waarde waarmee bewijs wordt gelevert dat je over het juiste wachtwoord beschikt vindt op de volgende manier plaats:

Eerst wordt de waarde `secret` berekend:

```
secret = MD5(Mufasa:http-auth@example.org:Circle of Life)
```

Hier zijn `username` (`Musafa`), `realm` (`http-auth@example.com`), en `wachtwoord` (`Circle of Life`)[46], aan elkaar geplakt met een dubbele punt ":". Ik heb hier bewust alle quotes weggelaten om duidelijk te maken welke tekens precies worden meegenomen.

Verder hebben we de waarde van `data` nodig. We plakken hier de waarden van `nonce`, `nc`, `cnonce`, `qop`, en nog een laatste waarde met `:` aan elkaar:

```
data =
    "7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v" + ":" +
    "00000001" + ":" +
    "f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ" + ":" +
    "auth" + ":" +
    "39aff3a2bab6126f332b942af96d3366"
```

Omdat dit allemaal zo lang is, moet ik wel gebruik maken van quotes om aan te geven uit welke waarden `data` is opgebouwd. Maar voor alle duidelijkheid, in dit geval bevat `data` geen enkele quote en ook geen enkele spatie, maar dus wel enkel dubbele punten.

De laatste waarde uit `data` is het resultaat van

```
MD5(GET:/dir/index.html) = 39aff3a2bab6126f332b942af96d3366
```

We zijn bijna bij de berekening van de `response`. We plakken de `secret` en `data` weer aan elkaar met een dubbele punt en we berekenen hiervan de MD5 hash:

```
response = MD5(
    "3d78807defe7de2157e2b0b6573a855f" + ":" +
    "7ypf/xl ... d3366") =
    8ca523f5e9506fed4657c9700eebdbec
```

En dit is precies de waarde die als `response` is meegegeven. (Je snapt dat bovenstaande code een ingekorte versie is van de waarde waarvan de hash is berekend.)

8.2 Broadcasts

Let op: met ingang van het eerste semester van het schooljaar 2019-2020 is dit hoofdstuk geen onderdeel meer van de toetsstof van dit vak. Mogelijk komt dit

[46] Er zit volgens https://www.rfc-editor.org/errata/eid4495 een typefout in het voorbeeld-wachtwoord in de RFC! Het tweede woord moet `of` zijn en niet `Of` zoals in de RFC wordt gegeven. Alleen met `of` is de juiste `response` te berekenen.
secret = MD5(Mufasa:http-auth@example.org:Circle of Life) = 3d78807defe7de2157e2b0b6573a855f

materiaal van pas bij het IoT-project.

Stel, je wil een weerstation toevoegen aan je opstelling, en je wilt dat het weerstation en de gateway elkaar automatisch kunnen vinden. Nu kan je het IP adres van de gateway meecompileren met de Arduino-code, maar dit is geen robuuste oplossing.

Het Internet Protocol heeft een optie om een bericht te "broadcasten". Dit wil zeggen dat alle devices op het netwerk het bericht ontvangen.

We kunnen van dit mechanisme gebruik maken door de Arduino een bericht te laten broadcasten. De gateway luistert naar dit soort berichten, en zelfs wanneer het weerstation het IP-adres van de gateway niet kent, kan een weerstation de gateway bereiken.

Het weerstation kan een dergelijk bericht ontvangen en het IP protocol voegt het IP adres van de afzender toe aan het bericht. Daarmee kent het weerstation het IP adres van het nieuwe weerstation!

Er is een klein probleem wat onze aandacht verdient. Onze verbindingen die we tot nu toe zijn tegengekomen, lopen over TCP (een laag bovenop IP). In TCP wordt veel moeite gedaan om betrouwbare verbindingen tussen twee partijen op te zetten. Dit concept botst met het idee van een IP broadcast is, wat veel meer het concept is van "een pakket over de schutting gooien, en degene die deze leest, mag er iets goeds mee doen".

We kunnen de broadcast dus niet gebruiken in combinatie met TCP. Gelukkig kunnen we het wel gebruiken met het broertje (zusje?) van TCP, nl. UDP. UDP staat veel dichter bij IP dan TCP. UDP pakketten worden verstuurd en "vergeten": er is geen enkele garantie dat een UDP pakket aankomt, en wanneer je er meerdere stuurt, is het geen enkele garantie dat deze in de juiste volgorde aankomen. Het voordeel van UDP boven TCP is, is dat het simpeler is dan TCP en omdat allen moeite die moet worden gestoken in de betrouwbaarheid van TCP niet nodig is, is UDP, als het werkt, veel efficiënter dan TCP. UDP kan dus worden gebruikt in combinatie met de broadcasts in IP. En net als TCP kent UDP poorten, hetgeen UDP onderscheid van IP.

Wij gaan gebruik maken van het broadcastadres `255.255.255.255` welke in IP de *broadcast address van het zero network*, hetgeen binnen IP het *lokale netwerk* is. De methode die we hier gaan toepassen kan dus enkel binnen één en hetzelfde netwerk gebruikt worden bron.

8.2.1 Doel

- Je kan het verschil tussen TCP en UDP beschrijven.
- Je kan gebruik maken van UDP broadcasts in je opstelling

8.2.2 Opgave

Maak een *proof of concept* voor het verbinden van weerstations en de gateway zonder dat deze op voorhand elkaars IP adres kennen.

Je zou dit in de volgende fasen kunnen doen:

1. Zorg dat het voorbeeld in de toelichting werkt, en zorg dat je snapt wat er in de code gebeurt.
2. Maak een beschrijving hoe weerstation en gateway elkaar vinden en wanneer wordt overgegaan van UDP communicatie naar reguliere TCP communicatie.
3. Bouw je eerste implementatie om zodat deze je beschrijving implementeerd.

8.2.3 Toelichting

In het geval van TCP is er sprake van een *server-client* configuratie waarin twee partijen een wat verschillende taak hebben. Dit idee bestaat in UDP eigenlijk niet. Iedere partij kan luisteren en zenden maar voor de coördinatie tussen luisteren en zenden ben je als ontwikkelaar zelf verantwoordelijk.

8.2.3.1 Voorbeeld Arduinocode De volgende code laat een Arduino iedere 2 seconden een bericht broadcasten. Verder luistert deze continu naar UDP berichten op poort `8888`.

```
#include <Ethernet.h>
#include <EthernetUdp.h>

const int BROADCAST_INTERVAL = 2000; // ms between broadcast
const unsigned int UDP_PORT = 8888;

unsigned long previousMillis = 0;

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
IPAddress ip(192, 168, 1, 21);

IPAddress broadcast(255, 255, 255, 255);

EthernetUDP Udp;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; // buffer to hold
                                           // incoming
                                           // packet,

void setup() {
  Serial.begin(9600);

  Ethernet.begin(mac, ip);

  Serial.print(
      "size of UDP_TX_PACKET_MAX_SIZE is currently ");
  Serial.println(UDP_TX_PACKET_MAX_SIZE); // 24

  Udp.begin(UDP_PORT);
}

void loop() {
  // broadcast UDP message after BROADCAST_INTERVAL ms
  unsigned long currentMillis = millis();
```

```
  if (currentMillis - previousMillis > BROADCAST_INTERVAL) {
    previousMillis = currentMillis;

    Serial.println("broadcast UDP");
    broadcastUdp();
  }

  // listen for responses
  listenUdp();
}

void broadcastUdp() {
  Udp.beginPacket(broadcast, UDP_PORT);
  Udp.write("hello gateway");
  Udp.endPacket();
}

void listenUdp() {
  int packetSize = Udp.parsePacket();
  if (packetSize) {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From ");
    IPAddress remote = Udp.remoteIP();
    for (int i = 0; i < 4; i++) {
      Serial.print(remote[i], DEC);
      if (i < 3) {
        Serial.print(".");
      }
    }
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
    Serial.println(packetBuffer);
  }
}
```

8.2.3.2 Voorbeeld Pythoncode De volgende code laat een python-programma luisteren op poort 8888. Wanneer er een bericht binnenkomt stuurt deze een bericht terug met daarin het IP adres van de verzender om te laten zien dat deze code uit een ontvangen (broadcast) bericht het IP adres kan achterhalen.

```
# http://www.dabeaz.com/python/PythonNetBinder.pdf, slide 1-40 en verder
from socket import *

s = socket(AF_INET, SOCK_DGRAM)
s.bind(("", 8888))

maxsize = 32  # not in example
```

```
while True:
    data, addr = s.recvfrom(maxsize)
    print("server received {}: {}.".format(data, addr))
    ip, port = addr
    resp = "Ack {}\0".format(ip)
    s.sendto(resp.encode('ascii'), addr)
```

8.3 Logging

Let op: met ingang van het eerste semester van het schooljaar 2019-2020 is dit hoofdstuk geen onderdeel meer van de toetsstof van de schriftelijke toets van dit vak. Wel komt dit onderdeel aan de orde in het beroepsproduct.

8.3.1 Doel

- Je bent in staat om bruikbare logging aan je applicatie toe te voegen.
- Met deze logging ben je in staat om problemen met één van de temperatuursensoren vast te stellen.
- Met deze logging ben je in staat om problemen met één van je weerstations vast te stellen.

8.3.2 Opdracht

Bedenk op basis van de handreiking in de toelichting een stategie voor de volgende aspecten:

- In een situatie van minimaal drie weerstations wil je in je log terug kunnen vinden dat één van de temperatuursensoren afwijkende waarden geeft.
- Zorg ervoor dat je in je log op de hoogte gesteld wordt van het verkeer tussen de Gateway en het Weerstation.

  Wanneer de communicatie prima verloopt, moet deze op een heel laag urgentie-niveau in je log komen. Echter, wanneer de communicatie problematisch is, moet deze op een hoog urgentie-niveau gelogd worden.
- Zorg ervoor dat je in je log terug kan vinden dat één van je weerstations niet meer reageert op requests vanuit de Gateway.

8.3.3 Toelichting

Logging wordt beschreven als “a means of tracking events that happen when some software runs” (Sajip z.d.b).

Een aantal goede algemene tips komen uit Xebia (z.d.):

- Log op meerdere niveaus.

  Door aan ieder bericht een niveau mee te geven, kan je schakelen tussen heel gedetailleerde logs (wanneer je een probleem aan het oplossen bent) of tussen compacte logs die alleen een melding geven wanneer er iets grondig fout gaat.

- Zorg ervoor dat iedere gebeurtenis eenmaal voorkomt in een log.

  Dit lijkt triviaal, maar kan heel lastig zijn: wanneer meerdere onderdelen in je applicatie een probleem constateren, is het niet raar dat iedere onderdeel daar melding van maakt. Een dergelijk cluster is uiteindelijk mogelijk terug te leiden tot één oorzaak.

- Voorkom dat je onnodig paniek schept.

  Wanneer je log het uitschreeuwt dat er iets aan de hand is, maar er blijkt eigenlijk geen probleem te zijn, dan negeert de lezer van de logs uiteindelijk de berichten die er wel toe doen.

En verder nog een aantal andere tips:

- Zorg dat in een logbericht een tijdstip zit.
- Zorg dat de *severity* van het logbericht helder is.
- Zorg dat een logbericht zelfstandig begrijpbaar is.

  Geef dus niet alleen een foutcode.

- Zorg dat de bron van een logbericht duidelijk is (welk onderdeel van je applicatie levert het bericht).
- Zorg voor een uniform formaat.

  Hiermee maak je het mogelijk dat je filters schrijft die logberichten kunnen analyseren.

Er bestaan bijzonder rijke omgevingen om logs in op te slaan en te analyseren, zie bijvoorbeeld dit artikel om een indruk te krijgen. Wij beperken ons hier tot loggen naar een *log file*.

Voor de applicatie waar wij mee te maken hebben is het voordehandliggend dat de Python applicatie logt. Python heeft een logging framework die uitvoerig beschreven wordt in de “logging HOWTO” (Sajip z.d.b) en de “logging cookbook” (Sajip z.d.a).

8.4 Prototypen

Let op: met ingang van het eerste semester van het schooljaar 2019-2020 is dit hoofdstuk geen onderdeel meer van de toetsstof van de schriftelijke toets van dit vak. Wel komt dit onderdeel aan de orde in het beroepsproduct.

Er wordt wel eens gedacht dat “prototypen” neerkomt op “een beetje aanrommelen” en dat alleen “scrum” staat voor serieus werk. Dit klopt dus niet.

Waar scrum de bedoeling heeft om met zekerheid een product op te leveren waar de klant veel aan heeft, is het doel van prototypen *een onderbouwing te geven op de vraag of klant kan krijgen wat hij wil*, en wanneer dat mogelijk is om dat *onderbouwd aan te geven welke oplossing voor een klant de beste optie is.*

Prototypen is een fase die vaak voorafgaat aan het in productie nemen van een product. Voordat we de machines aanzetten om 100000 eenheden van een product te produceren, willen we eerst wel even weten of het een goed product is, en dat er voor dezelfde kosten geen betere oplossing is.

Zoals je ziet, is het aspect *onderbouwen* nogal een belangrijk thema. Dus, jij zegt me dat ik een probleem het beste op jouw manier kan oplossen? Laat me dan ook maar zien naar welke methoden je ook hebt gekeken en waarop die van jou het beste is... Laat maar eens een test zien waaruit blijkt dat je oplossing inderdaad werkt.

Verslaglegging is een tweede en net zo belangrijk thema. Deze bestaat uit twee onderdelen. Aan de ene kant is de log waarin je tijdens je werk je keuzen en waarnemingen vastlegt. Op een gegeven moment heb je een oplossing, of ben je ervan overtuigd dat wat de klant is niet kan, of heb je een gedeeltelijke oplossing. Het heeft dan geen zin om je log over de schutting van de opdrachtgever te gooien. In plaats daarvan moet je fatsoenlijke overdrachtsdocumentatie leveren.

De klant kan op basis van je overdrachtsdocumentatie een oplossing in productie nemen. Of, als we eerlijk zijn, vaker kan een ander team met je werk verder. Het schema in fig. 7 geeft een handvat om gestructureerd door het proces heen te gaan.

8.4.1 Logboek

In essentie beschrijft je logboek de cyclus van de volgende stappen:

- Wat is het probleem dat je aan het oplossen bent.
- Welke oplossingen zie je voor je probleem.
- Welke oplossing ga je proberen en waarom.

  Het is relevant om achteraf terug te kunnen vinden of je een oplossing probeert omdat je denkt dat het het de meestbelovende optie is, of dat je ergens moet beginnen met proberen en dat de andere opties ook veelbelovend zijn.
- Wat is het resultaat van je werk?

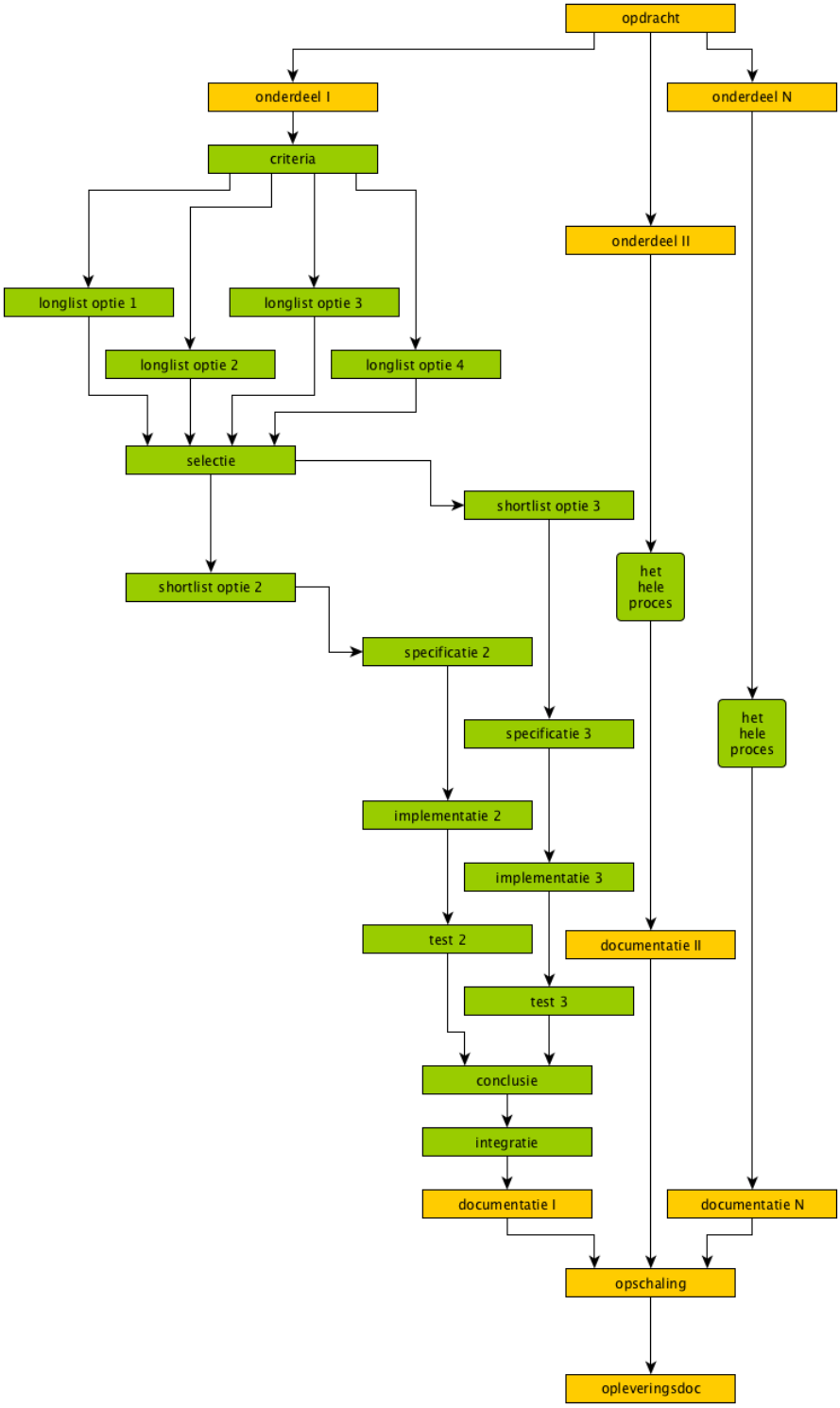
  Het kan zijn dat je oplossing aan de verwachtingen voldoet. Het kan ook zijn dat het niet voldoet. Wees dan helder op welke aspecten het misgaat. Het kan zelfs zijn dat je niet kan uitmaken of je oplossing aan de verwachtingen voldoet. Communiceer dat dan en geef aan welke informatie je mist.
- Hoe ga je verder?

  Eén optie is om gewoon met de oplossing van je probleem verder te gaan.

  Hoewel dit logisch lijkt, hoeft dit helemaal niet de juiste keuze te zijn!

  Mogelijk heb je zoveel aan de oplossing al wel bijgedragen dat de oplossing van een ander probleem meer relevant is geworden. Mogelijk kan je die problematische oplossing nog even uitstellen en aan een ander probleem beginnen in de hoop en verwachting dat je wijzer wordt.

Het logboek is vooral een persoonlijke verslaglegging. Het moet je helpen achteraf een antwoord te geven op de vraag: waarom ben je toen en toen aan dat onderwerp gaan werken en hoe heb je het in je hoofd gehaald om die oplossing uit te proberen en niet de veel meer voor de hand liggende andere optie...



Figuur 7: Schematische weergave van het *prototype* proces.

Maar het moet ook wel een beetje leesbaar zijn voor collega's. Op een goede dag ben jij langdurig op vakantie of met een andere baan begonnen (je snapt wat ik bedoel). Je logboek moet je collega's op precies diezelfde vraag van net ook een hint van een antwoord kunnen geven.

8.4.2 Overdrachtsdocumentatie

Dit is essentie een net uitgewerkt logboek met een nette kop en staart (wat is het op te lossen grote probleem, en waar sta je na afloop van het project).

Op basis van je overdrachtsdocumentatie moet een ander team verder kunnen werken. Ze moeten snappen je werk kunnen reproduceren. Ze moeten snappen waarop je een oplossingsrichting die jij niet bent ingegaan ook niet hoeven in te gaan. Of, dat je gewoon aan sommige opties niet bent toegekomen maar desondanks veelbelovend kunnen zijn.

Je kan voor de indeling van de onderwerpen uitgaan van de onderwerpen die ook je ook in je logboek hebt behandeld. Ook kan je dezelfde indeling hanteren waarin je iedere probleem in je logboek hebt behandeld.

Een logboek ontwikkelt zich gaandeweg je project. De overdrachtsdocumentatie schrijf je achteraf. Dit heeft het voordeel van "met de kennis van nu". Je zal merken dat een aantal problemen die onderweg heel belangrijk waren achteraf veel minder belangrijk zijn, of andersom. Deze kennis neem je mee in je overdrachtsdocumentatie. Zo kan je achteraf je onderwerpen wellicht meer logisch ordenen.

Had ik al duidelijk gemaakt dat logboek en overdrachtsdocumentatie geen succesverhaal hoeven te zijn? (Alles ging zo goed en geweldig...) Je hebt veel meer aan een eerlijk verhaal. De klant heeft er veel meer aan te weten dat jij van een bepaald onderdeel niet zeker bent of de oplossing adequaat is. Stel je de situatie voor dat de klant er pas achterkomt nadat 10 miljoen eenheden van je product bij klanten over de hele wereld liggen ...

8.4.3 Toetsing

Je houdt tijdens je werk aan je eindproduct een logboek bij. Verder moeten de onderdelen die als zodanig in de Eindopdracht gemarkeerd zijn in je overdrachtsdocumentatie aan de orde komen.

Geef beide onderdelen aandacht! Het is echt verdraaid lastig om je logboek in het geval van een herkansing te moet schrijven wanneer je voor je project een voldoende hebt.

9 Bijlagen

9.1 Installatie zonder (verplichte) RPi

In voorgaande uitvoeringen werd gebruik gemaakt van een Raspberry Pi (RPi). Er zijn ondertussen meerdere ontwikkelingen.

1. De RPi is momenteel minder goed beschikbaar, en wanneer beschikbaar, ook duurder dan voorheen.

2. Je kan de RPi koppelen aan Eduroam, maar dit blijkt een foutgevoelig proces te zijn.
3. Windows biedt via WSL2 een eenvoudige manier om een linux-omgeving op je windows-laptop te draaien.

Mocht je gebruik maken van een Mac[47] of een Linux-laptop[48], dan beschik je ook over een linux-achtige omgeving die voldoet voor dit vak.

Heb je een RPi en wil je deze proberen, dan kan je deze installeren volgens de instructies van een van de volgende hoofdstukken: sec. 9.2 beschrijft de installatie met de koppeling van de RPi aan Eduroam, sec. 9.3 beschrijft de thuisinstallatie van de RPi die tijdens Corona is gebruikt. Beide bieden hopelijk voldoende handvatten.

In grote stappen ziet de installatie zonder RPi er als volgt uit:

1. Als windows-gebruiker installeer je WSL (link).
2. Je laptop (iedere variant) gaat deel uitmaken van twee netwerken, met wi-fi blijf je verbonden aan Eduroam, configureer je ethernet-netwerk[49] zo dat deze het vaste adres `192.168.1.1` krijgt, met *network mask* `255.255.255.0`.

   In de *reader* wordt er vanuit gegaan dat de RPi `192.168.1.11` krijgt, de Arduino een hoger adres. Verder wordt de RPi geconfigureerd als DHCP-server die adressen vanaf `192.168.1.51` uitdeelt. Je mag dit voor elkaar proberen te krijgen, maar met alleen vaste adressen (als in de "oude, goede tijd") ga je er ook uitkomen.
3. Je *ethernet shield* van de RPi kan je een vast IP adres geven door de volgende wijzigingen in in je code (bron):

```
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 51 };

void setup()
{
  Ethernet.begin(mac, ip);

  // remainder of your code
}
```

9.2 Installatie RPi en opzet netwerk (sep 2022)

1. Installeer Raspberry Pi Imager op je laptop.

[47] Je zal zelf moeten uitzoeken hoe je nieuwe software installeert. Het in deze *reader* overal gebruikte `apt` doet het niet. Voorheen kwam ik met *homebrew* een heel eind.

[48] De *reader* gaat uit van een Debian-gebaseerde distributie die gebruik maakt van `apt` om software te installeren, installeer je een andere distributie, dan moet je mogelijk zelf uitzoeken hoe je software installeert.

[49] Wanneer je laptop geen aansluiting heeft voor een bedraad netwerk, dan kan je een USB-adapter met ethernet-aansluiting

2. Installeer Raspberry Pi OS op de geheugenkaart

   - Start *imager*
   - In *imager*, selecteer `Raspberry PiOS (other)` en daarin `Raspberry Pi OS Lite (32-bit)`.
   - Kies onder *storage* je *SD card*.
   - Kies *advanced options* (het tandrad symbool), vervolgens:
     - `[x]` Enable SSH

       `(x) use password authentication`
     - set username and password

       De rest van de instructie gaat uit van *username* `pi` maar je mag zelf wat anders kiezen. Let op dat je dan ook je eigen *username* gebruikt in de rest van de instructie.

       Gebruik niet je eduroamwachtwoord, bedenk iets anders.
   - Start met *write*, dit duurt een tijdje.

3. Je laptop heeft meerdere *network adapters*, met je wifi blijf je verbonden met het internet, met de *ethernet* verbinding gaan we een lokaal netwerk maken waar ook de RPi op komt.

   - Geef de *ethernet adapter* (dus, niet je wifi verbinding) van je laptop IP-adres `192.168.1.1` en *subnetmask* `255.255.255.0` (zoek zelf uit hoe je dit voor je eigen OS moet doen, zoek zonodig op iets als "*configure Windows 10 fixed IP adress*").

4. Je RPi moet ook weten ook weten hoe deze moet communiceren met je *laptop*, dat doe je door opties te zetten op de net geschreven geheugenkaart.

   Je geheugenkaart heeft twee partities gekregen, wijzig het volgende op de `boot` partitie:

   - Zet in het bestand `cmdline.txt` de volgende code `ip=192.168.1.11` (tussen `rootwait` en `quiet`). Doe dit met een fatsoenlijke ASCII editor die niets wijzigt aan de manier waarop *end-of-line characters* geschreven worden.

   Verwijder de geheugenkaart en plaats deze in de RPi.

5. Nadat de geheugenkaart in de RPi zit, verbindt deze met de ethernetkabel met je laptop en verbind de RPi met het lichtnet.

   - Wacht een tijdje, er wordt een en ander gewijzigd aan de geheugenkaart (ongeveer een minuut).
   - Controleer of de laptop en je RPi met elkaar kunnen communiceren, voer daartoe de volgende opdracht uit vanaf de *command line* van je *laptop*:

     ```
     $ ping 192.168.1.11
     ```

Let op: de `$` is de *prompt* en geeft aan dat je iets in kan typen. Je typt deze zelf nooit in, dus je typt alleen `ping 192.168.1.11`. Op jouw scherm kan je *prompt* er anders uit zien.

Als je iets als de volgende reactie krijgt:

```
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.400 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.232 ms
64 bytes from 192.168.1.11: icmp_seq=3 ttl=64 time=0.224 ms
```

Dan is de verbinding tot stand gekomen, wacht anders nog een minuut en probeer opnieuw.

6. In deze stap gaan we vanaf je *laptop* inloggen op je RPi, zodat we programma's op je RPi kunnen uitvoeren.

   Met moderne varianten van Windows zou je daarvoor geen software meer hoeven te installeren. Kom je er niet uit, installeer dan putty (*"the only thing which makes windows usable"*)

   - Probeer in te loggen op je RPi, weer vanaf de *command line* (of in *putty* met *hostname* `192.168.1.11`, `port 22`, *connection type* `ssh`)

     ```
     $ ssh pi@192.168.1.11
     ```

     Let erop dat je de *username* gebruikt die je eerder hebt opgegeven.

     Inloggen is gelukt wanneer je op het scherm ziet

     ```
     $ pi@raspberrypi:~ $
     ```

     Iedere opdracht die je achter deze *prompt* uitvoert wordt op je RPi uitgevoerd en niet op je laptop.

7. Ook je RPi heeft twee netwerkadapters, via ethernet heb je verbinding met je laptop, we gaan nu proberen met je RPi's *wifi* op eduroam te komen. Dit is op school extra ingewikkeld omdat we met *eduroam* te maken hebben.

   Voer de volgende opdrachten uit op je RPi (via de ssh toegang die we net geopend hebben).

   a. zet *wifi* aan

      - voer uit `sudo raspi-config`[50]
      - kies optie `5 Localisation Options`, en dan `L4 WLAN Country`, en kies `NL Netherlands`.
      - eventueel met `tab`-toets naar `Ok`

      Sluit af, waarna de RPi *reboot*, het duurt dan weer een tijdje voordat je met `ssh` in kan loggen.

      - Log weer in.

   b. Maak een *hash* van je *eduroam* wachtwoord, deze heb je nodig om je wachtwoord beveiligd op te slaan op de RPi.

[50] Met `sudo`, *super user do* krijg je extra rechten op je systeem. Dat gebruik je alleen als je dat nodig hebt.

```
$ read -s input ; echo -n $input | iconv -t utf16le | openssl md4
```

Het lijkt alsof er dan niets gebeurt, maar de RPi wacht tot je je wachtwoord hebt ingetype. Type je wachtwoord en sluit af met een Enter

Output ziet er ongeveer zo uit:

`(stdin)= f2f271.....f528b2`

kopieer de gegeneerde hash (na de '`(stdin)=`') naar het klembord.

Een typfout levert een ander resultaat op. Herhaal de procedure en kijk of je dezelfde *hash* berekent.

Het voordeel van bovenstaande methode is dat je wachtwoord niet terug te vinden is in je *history*. Het blijkt dat het soms niet de juiste *hash* oplevert. Hier is een alternatief, maar het is belangrijk dat je *met een spatie begint*, nl. commando's die met een spatie beginnen komen ook niet in de *history*:

```
$  echo -n 'bla bla bla' | iconv -t utf16le | openssl md4
# ^ spatie!!!
```

Gebruik *enkele quotes* en vervang `bla bla bla` door je eigen wachtwoord. De tweede regel (`# ^ spatie!!!`) hoef je uiteraard niet in te typen.

Nadat je de *hash* gekopieerd hebt, dan kan je met volgende opdracht je scherm weer opschonen zodat je wachtwoord niet meer leesbaar is.

```
$ clear
```

c. Nu moeten we de configuratie van de RPi aanpassen. We gebruiken de eenvoudige editor `nano`. Let erop dat je muis hier niet werkt.

Open configuratiebestand:

```
$ sudo nano /etc/network/interfaces
```

Voeg de volgende code toe:

```
iface wlan0 inet manual
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

d. Configureer Eduroam

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

voeg toe, en vervang `<jouw code>` door de vijf letter code van je HAN account, en `<de hash die je eerder gegenereerd had>` door de berekende *hash*, dus ook de `<` en `>` vervang je.

```
network={
    ssid="eduroam"
    scan_ssid=1
    proto=RSN
    key_mgmt=WPA-EAP
    pairwise=CCMP TKIP
```

```
    group=CCMP TKIP
    eap=TTLS PEAP
    phase2="auth=MSCHAPV2"
    identity="<jouw code>@han.nl"
    password=hash:<de hash die je eerder gegenereerd had>
    id_str="eduroam"
}
```

e. Tot slot:

```
$ sudo nano /etc/dhcpcd.conf
```

en voeg toe:

```
interface wlan0
env ifwireless=1
env wpa_supplicant_driver=wext,nl80211
```

f. Start je RPi opnieuw op:

```
$ sudo reboot now
```

g. Maak opnieuw verbinding met SSH naar de RPI en controleer of je op Eduroam zit.

Typ `ifconfig`, Wi-fi is goed geconfigureerd als je `wlan0` een geldig IP-adres heeft.

h. Update RPI

Voer uit:

```
$ sudo apt update
$ sudo apt upgrade
```

Als je tijdens de installatie van je RPi hier bent gekomen, dan ben je een heel eind en kan je de RPi grotendeels zinvol gebruiken.

8. We maken van de RPi de DHCP-server van het lokale, bedrade, netwerk. Een eventuele Arduino die op het netwerk komt, kan dan vragen om een IP-adres.

De opdrachten moeten weer op de RPi worden uitgevoerd.

> Dit gedeelte is ongetest gekopieerd uit een eerdere versie van de instructie.

a. Installeer de DHCP server

```
$ sudo apt install isc-dhcp-server
```

Aan het eind van de installatie probeert de DHCP server op te starten. Dit gaat fout, negeer even de rode meldingen.

b. Configureer de DHCP server

```
sudo nano /etc/default/isc-dhcp-server
```

Zoek de regel `INTERFACESv4=""` en verander dat naar

```
INTERFACESv4="eth0"
```

c. Stop de DHCP services

```
sudo systemctl stop isc-dhcp-server.service
```

d. Pas de DCHP configuratie aan

```
sudo nano /etc/dhcp/dhcpd.conf
```

Zoek de regel `#authoritative;`, verwijder `#` zodat er `authoritative;` staat.

Voeg onderaan het bestand het volgende toe:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.51 192.168.1.200;
   option broadcast-address 192.168.1.255;
   default-lease-time 600;
   max-lease-time 7200;
}
```

Start de DHCP service

```
sudo service isc-dhcp-server start
```

9. In voorgaande uitvoeringen is er een probleem geweest met de verbinding tussen de RPi en de *Subversion Server*. Toen was dat als volgt op te lossen.

   De opdrachten moeten weer op de RPi worden uitgevoerd.

   > Dit gedeelte is ongetest gekopieerd uit een eerdere versie van de instructie.

   In `/etc/ssl/openssl.cnf`, wijzig de laatste twee regels in:

   ```
   [system_default_sect]
   #MinProtocol = TLSv1.2
   #CipherString = DEFAULT@SECLEVEL=2
   MinProtocol = TLSv1
   CipherString = DEFAULT@SECLEVEL=1
   ```

10. Je kan thuis op de volgende manier vrij eenvoudig je Raspberry verbinding aan het draadloze netwerk.

    De opdrachten moeten weer op de RPi worden uitgevoerd.

    ```
    $ sudo raspi-config
    ```

    Ga naar optie `1 System Options` en dan weer `S1 Wireless LAN`. Vul in het eerste veld de naam van je draadloze netwerk in, en in het tweede veld je wachtwoord.

    Normaal gesproken voegt dit een netwerk toe naast de eerder gemaakte verbinding met *eduroam*.

9.3 installatiehandleiding voor on-line semesters

9.3.1 Inleiding

Er zijn een aantal redenen waarom we dit semester met een Raspberry Pi (RPi) werken. Een aantal daarvan zijn:

- We hebben onderdelen nodig waarmee we een heterogeen netwerk kunnen bouwen.
- We gebruiken de RPi om ervaring op te doen met een *command line interface* (een "CLI").

Het eerste deel van de coursefase gaat de nadruk naar het tweede punt. Wanneer je nog niet over een RPi beschikt, dan kan je op windows het windows subsystem for linux (WSL) installeren. Heb je een Mac OS of een Linux machine, dan kan je daar de *terminal application* gebruiken.

Ik ga ervanuit dat je een laptop hebt met een *ethernetaansluiting* (een netwerkkabel). Heb je deze niet, bestel dan een USB-ethernet adapter.

9.3.2 Groepsverantwoordelijkheid

Je wordt voor de coursefase van IoT ingedeeld in een groep van vier mensen. Ik heb mijn best gedaan een zo helder mogelijke handleiding te schrijven en toch gaat iedereen tegen problemen aanlopen.[51] Zorg dat je elkaar helpt en dat je als groep ervoor zorgt dat iedereen een werkende installatie krijgt.

Verder, met name de geheugenkaart die met de RPi wordt meegeleverd is foutgevoelig. Ieder jaar zijn er op de hele klas wel een paar studenten bij wie de kaart defect is. Houd je last van onverklaarbare problemen, doe jezelf een lol en koop een nieuwe kaart. Ik denk dat 4GB ruim genoeg is, maar koop voor de zekerheid toch maar een 8GB versie.

9.3.3 Doel

Voor de eerste les zorg je er als groep voor dat iedereen binnen je groep het volgende netwerk thuis heeft draaien zoals weergegeven in fig. 8.

Mocht dit om wat voor reden dan ook niet lukken, dan val je terug op het eerder genoemde WSL.[52]

We maken met de opzet van dit netwerk in deze aftrap een start en werken aan de eerste stap (figuur)

9.3.4 Stap 1: Basissysteem draaiend krijgen

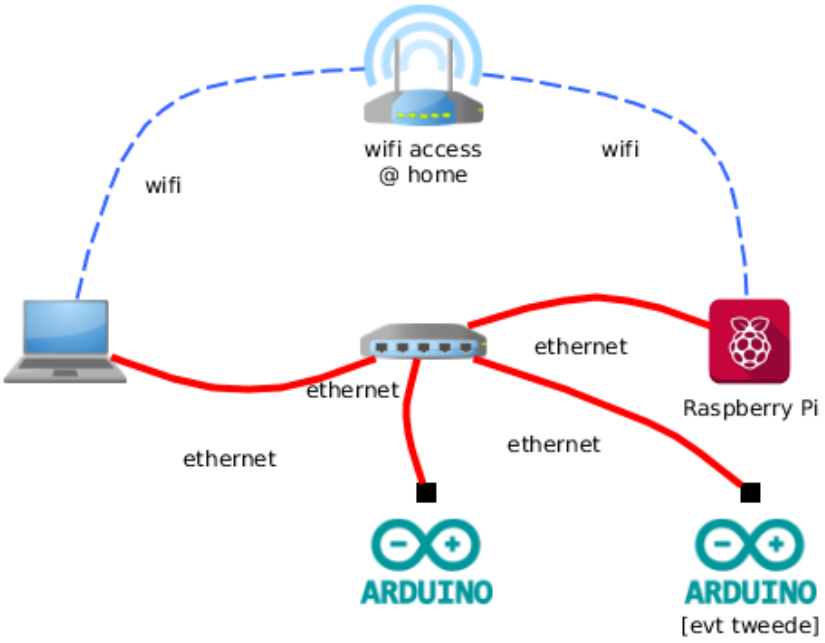
Als groep ben je met deze stap klaar wanneer iedereen binnen de groep in staat is om vanuit je laptop een *secure shell* (`ssh`) verbinding te maken naar de RPi, en via die verbinding het wachtwoord op je raspberry hebt veranderd.

9.3.4.1 Opmerkingen vooraf Deze instructies zijn gebaseerd op een ingewikkelder configuratie van een Raspberry Pi 4.

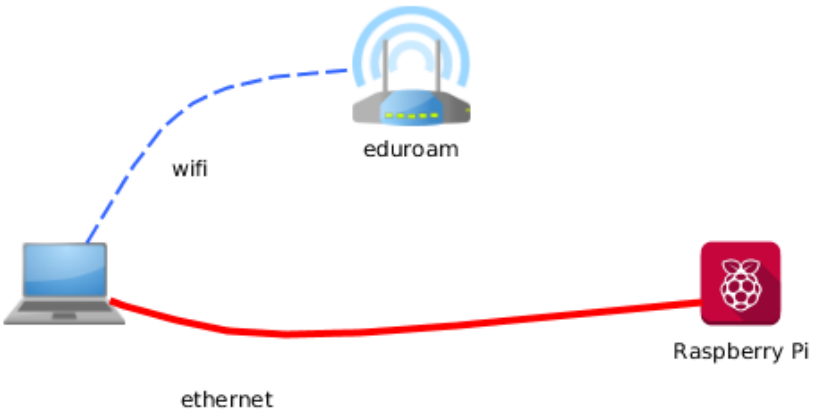
- Lees de instructies goed door. Snel door de instructie heenrennen leidt er in het algemeen toe dat je langer bezig bent.

[51] Het is dan ook geen handleiding die je zonder nadenken kan volgen, je moet er met je kop bijblijven en proberen te snappen waar je mee bezig bent.

[52] De ervaring is dat iedereen WSL zonder veel moeite aan de praat krijgt. Om die reden gaan we in deze handleiding op de installatie hiervan niet verder in.



Figuur 8: Doel installatie.



Figuur 9: De eerste stap

- Heb wat geduld. Het duurt soms even voor een service op de Raspberry Pi actief is. Dus, als `ping`-en of `ssh`-en niet meteen lukt, wacht even en probeer nog een keer.

  Zeker de eerste keer met een nieuwe micro SD kaart opstarten duurt even omdat er een aantal wijzigingen op de kaart worden uitgevoerd.

9.3.4.2 Opzet netwerk

Deze informatie is meer voor referentie achteraf.

De volgende IP-ranges worden gebruikt:

```
192.168.1.1 - 192.168.1.10:  Ethernetcard Laptops
192.168.1.11 - 192.168.1.20: Ethernetcard Raspberry Pi
192.168.1.21 - 192.168.1.30: Ethernetcard Arduino
192.168.1.51 - 192.168.1.200: DHCP (voor IoT-SW, op de Pi)
```

9.3.4.3 Benodigdheden

- Laptop met een ethernet aansluiting (een netwerkkabel). Koop zonodig een USB - ethernet adapter.
- Ethernetkabel.
- SD-card (micro SD inclusief adapter voor in je laptop of card-reader)
- Raspberry PI
- RPi image.

  Vroeger bekend als *Rasbian* heet de distributie nu *Raspberry Pi OS*.[53] Deze is er in meerdere varianten. Voor ons doel voldoet Raspberry Pi OS Lite.[54][55]

  Onderstaande is getest met lite versie van Raspbian. Als je een image downloadt krijg je een zip-bestand, dit bestand eerst uitpakken zodat je een img-bestand krijgt.

Draait je laptop Windows:

- SSH: Putty https://putty.org/
- Notepad++ https://notepad-plus-plus.org/ of een andere fatsoenlijke ascii-editor die onderscheid kent tussen windows en unix *end of line* characters.
- SD-card beschrijven: Etcher https://www.balena.io/etcher/

Draait je laptop Linux:

- SD-card beschrijven: Etcher https://www.balena.io/etcher/[56]

[53] Ik hoop dat behalve de naam niets wezenlijks gewijzigd is. De tijd zal het leren.

[54] Heb je een scherm en een toetsenbord over, dan kan het interessant zijn om te experimenteren met een van de andere versies die een GUI hebben.

Ook wanneer je serieuze installatieproblemen hebt en een scherm dan heeft het zin je scherm aan te sluiten omdat je dan mogelijk verder geholpen wordt met de foutmelding die je krijgt.

[55] Let op, dit zijn links in de tekst, er zitten meer links in de tekst die je gebruiken kan.

[56] Ik kon etcher niet meer eenvoudig op een Debian installatie installeren omdat de `appImage` formaat niet standaard ondersteund wordt.

Een alternatief is gebruik te maken van `dd`, zie de instructies, dit wordt verder niet uitgelegd in deze instructies.

- SSH: zit standaard in de terminal.
- Ascii-editor.

Draait je laptop MacOS:

- SD-card beschrijven: Etcher https://www.balena.io/etcher/
- SSH: zit standaard in de terminal.[57]
- Ascii-editor.

9.3.4.4 Stappenplan

1. Geef de ethernet card (en dus niet je WiFI verbinding) van de laptop het IP `192.168.1.1` en subnetmask `255.255.255.0`. Zoek zelf info (Zoek op "assign static ethernet IP address in <operating system>"), ik hoop dat de volgende links nog relevante info geven.
   - Windows: https://www.howtogeek.com/howto/19249/how-to-assign-a-static-ip-address-in-xp-vista-or-windows-7/
   - Linux: https://www.howtoforge.com/debian-static-ip-address (Kan tegenwoordig ook vaak via de configuratiescherm)
   - MacOS: https://www.howtogeek.com/howto/22161/how-to-set-up-a-static-ip-in-mac-os-x/

   Geef, zonodig, je wifi verbinding op je laptop een hogere prioriteit dan je ethernet verbinding.
2. Schrijf de RPI-image op de SD-card met etcher.
3. Open van de RPI-image de `boot`-partitie:
   - Zet in het bestand `cmdline.txt` de volgende code `ip=192.168.1.11` (tussen `rootwait` en `quiet`). *Windows*: gebruik Notepad++ en geen kladblok/notepad van Windows zelf.
   - Maak in de `boot`-partitie een bestand met de naam `ssh` (geen extensie)
   - Verwijder de SD-card (gebruik *altijd* veilig verwijderen voordat je de SD-card er uithaalt)
4. Setup RPI:
   - SD-card erin
   - Ethernetkabel tussen laptop en RPI aansluiten
   - Sluit RPi aan op het lichtnet

   Het rode en groene lampje beginnen te knipperen, zo gauw het groene lampje niet meer knippert is de RPI waarschijnlijk opgestart. De allereerste keer duurt het wat langer voordat de RPI opgestart is.

[57] Of is MacOs zover afgegleden?

5. check connectie met ping vanaf de laptop
   - Open een command line op je laptop:
     - Windows: type `cmd`
     - Linux: Open een terminal
     - MacOS: Open een Terminal
   - Type op de command line `ping 192.168.1.11` (dus het IP-nummer van de RPI)

     Als de RPI goed is geïnstalleerd (en je netwerkkaart is correct geconfigureerd) geeft de RPI antwoord:

     ```
     ping 192.168.1.11
     PING 192.168.1.11 (192.168.1.11): 56 data bytes
     64 bytes from 192.168.1.11: icmp_seq=0 ttl=64 time=0.639 ms
     64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.723 ms
     64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.711 ms
     64 bytes from 192.168.1.11: icmp_seq=3 ttl=64 time=0.810 ms
     ```

     Bij sommige OS's stopt `ping` na een aantal keer, bij andere moet je het onderbreken met `ctrl-C`.
6. Inloggen op de RPI met SSH
   - Windows:
     - Open `putty`
     - Hostname `192.168.1.11`
     - Port `22`
     - Connection type: `ssh`
     - klik op `Open`
   - Linux / MacOS
     - Open `terminal`
     - type `ssh pi@192.168.1.11`

   De eerste keer dat je een SSH-verbinding maakt moet het IP-adres van de RPI toegevoegd worden aan de lijst van veilige verbindingen, je krijgt hierover de volgende melding:

   ```
   The authenticity of host '192.168.1.11 (192.168.1.11)' can't be established.
   ECDSA key fingerprint is SHA256:9EcBt8u3uYZprTXN/1nTNFHOaoCSIS8jsT0nJUG5UYo.
   Are you sure you want to continue connecting (yes/no)?
   ```

   Kies voor `yes` en ga verder met inloggen.

   De gebruikersnaam is `pi` (heb je al meegegeven met de ssh-verbinding door pi@ te doen), het wachtwoord is `raspberry`. Let op, dat wachtwoord kun je maar beter wijzigen (Robillard 2012)

7. Wijzig het wachtwoord voor `pi`.

   Typ in je *ssh sessie*,

   ```
   $ passwd
   ```

   en volg de instructie. Gebruik niet het wachtwoord dat je voor eduroam gebruikt. Maak het ook niet te ingewikkeld, je gaat het veel nodig hebben.

8. Pas het volgende aan op je RPi zodat je RPi en de subversion server met elkaar kunnen praten.

   In `/etc/ssl/openssl.cnf`, wijzig de laatste twee regels in:

   ```
   [system_default_sect]
   #MinProtocol = TLSv1.2
   #CipherString = DEFAULT@SECLEVEL=2
   MinProtocol = TLSv1
   CipherString = DEFAULT@SECLEVEL=1
   ```

   Een manier om het bestand te editen is:

   ```
   $ cd /etc/ssl
   $ sudo nano openssl.cnf
   ```

   Je komt nu in een heel simpele ascii-editor op de RPi, onderaan staan veelgebruikte opdrachten.

   Let op, met `$` bedoel ik de zgn. *opdrachtprompt* waarachter je een commando intypt en uitvoert, deze ziet er bij jou waarschijnlijk anders uit. Punt is dat je zelf `$` niet typt, alleen de tekst die erachter staat.

   Let nog een keer op, de opdracht `sudo` staat voor *super user do*, je krijgt ermee onbeperkte macht op je RPI waarin je veel moois maar ook veel kapot kan maken. Je gebruikt dit alleen wanneer je iets in het systeem verandert wat buiten je eigen *home directory* als gebruiker `pi` ligt.

   Om maar vast een veelgemaakte fout te voorkomen: wanneer je straks tijdens de course code schrijft, doe je dat niet met `sudo`, want, het is *jouw* code en niet dat van de *super user*.

9.3.5 Stap 2: internetverbinding maken

Voor het tweede deel van de installatie heb je je huisnetwerk nodig (een draadloze verbinding met Eduroam kan, maar is behoorlijk complex, zie daarvoor de installatiehandleiding van IoT Software[58].

9.3.5.1 Wifi thuis

Zet eerst de wifi aan

- voer op de terminal `sudo raspi-config` uit.
- kies optie 4: “Localisation Options”, kies dan “WLAN Country” en selecteer `NL` (selecteer OK en daarna finish).

  Wacht met rebooten.

Vervolgens kan je koppelen met je eigen draadloze netwerk:

[58] Bookmark deze pagina maar vast, je gaat deze nog vaak nodig hebben.

- Ga naar optie 1 (system options), kies dan “Wireless LAN”, vul de naam van je netwerk in. Vul het wachtwoord van je wi-fi verbinding in het volgende veld in.
- Reboot je RPi door in de terminal achter de *command prompt* `sudo reboot` in te vullen.

Heb je problemen of wil je een uitgebreidere configuratie, kijk dan op https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

Je kan je verbinding controleren door vanuit de Raspberry verbinding te maken met bijvoorbeeld google:

```
$ ping google.com
```

Als alles goed is, zie je dat je Raspberry de server van google kan vinden.

9.3.6 Update je systeem

Voer uit:

```
$ sudo apt update
$ sudo apt upgrade
```

(hier gaat mogelijk wat tijd overheen)[59]

9.3.7 Raspberry Pi als DHCP server

Wanneer de Raspberry een DHCP server op de (bedrade) ethernetverbinding is, is het voor de Arduino's eenvoudiger om verbinding te maken met de Raspberry: zij zullen de Raspberry om een IP adres vragen.

1. Log in op de RPI via SSH
2. Installeer de DHCP server

   `sudo apt install isc-dhcp-server`

   Aan het eind van de installatie probeert de DHCP server op te starten. Dit gaat fout, negeer even de rode meldingen.
3. Configureer de DHCP server

   `sudo nano /etc/default/isc-dhcp-server`

   Zoek de regel `INTERFACESv4=""` en verander dat naar

   `INTERFACESv4="eth0"`

   Sluit af met ctrl-X en bewaar de wijzigingen
4. Stop de DHCP services

   `sudo systemctl stop isc-dhcp-server.service`
5. Pas de DCHP configuratie aan

   Open het configuratiebestand:

[59] In mijn laatste test ging het juist snel omdat er pas over 5 dagen kon worden geupdate (?).

`sudo nano /etc/dhcp/dhcpd.conf`

Zoek de regel `#authoritative;`, verwijder het `#` zodat er `authoritative;` staat.

Voeg onderaan het bestand het volgende toe:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.51 192.168.1.200;
   option broadcast-address 192.168.1.255;
   default-lease-time 600;
   max-lease-time 7200;
}
```

Start de DHCP service

`sudo service isc-dhcp-server start`

6. Instellingen controleren

   Herstart de RPI `sudo reboot now`

   Na het herstarten, doe een ping naar de RPI (`192.168.1.11`)

   Log in op je RPI (ssh) en ping naar je laptop (zoek eerst op wat het IP-nummer is van je Ethernet card). Als je antwoord krijgt, is alles goed ingesteld.

9.3.8 Conclusie

Je hebt je systeem nu volledig voorbereid op de coursefase van IoT. Via de bedrade verbinding kan je met je laptop inloggen op je Raspberry. Je kan nu bijvoorbeeld je Raspberry instrueren om via de wifi-verbinding nieuwe software te installeren. Dat zullen we regelmatig doen.

Referenties

'Apache Thrift'. 2018. https://en.wikipedia.org/wiki/Apache_Thrift.

'Apache Thrift - home'. z.d. Geraadpleegd 5 juni 2018. https://thrift.apache.org/.

'Atom (bestandsformaat)'. 2017. https://nl.wikipedia.org/wiki/Atom_(bestandsformaat).

'AttachInterrupt, Arduino Reference'. z.d. Geraadpleegd 24 maart 2021. https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/.

'Augmented BNF for Syntax Specifications: ABNF'. 2008. https://tools.ietf.org/html/rfc5234.

'BSON (Binary JSON) Serialization'. z.d. Geraadpleegd 5 juni 2018. http://bsonspec.org/.

'Classes'. z.d. Geraadpleegd 8 april 2021. https://docs.python.org/3.9/tutorial/classes.html.

Earl, Bill. z.d. 'Memories of an Arduino'. Geraadpleegd 19 april 2021. https://learn.adafruit.com/memories-of-an-arduino?view=all.

Engo, Nana. z.d. 'Matrix multiplication revisited'. Geraadpleegd 15 april 2021. https://nl.overleaf.com/articles/matrix-multiplication-revised/qkvxttgrsr

qh.
'Extensible Markup Language'. 2018. https://nl.wikipedia.org/wiki/Extensible_Markup_Language.
'Flask Tutorial'. z.d.a. Geraadpleegd 8 april 2021. https://flask.palletsprojects.com/en/0.12.x/tutorial/.
———. z.d.b. Geraadpleegd 8 april 2021. https://flask.palletsprojects.com/en/1.1.x/tutorial/.
Foord, Michael. z.d. 'Fetch Internet Resources Using The `urllib` Package'. Geraadpleegd 8 maart 2021. https://docs.python.org/3/howto/urllib2.html.
Gammon, Nick. z.d. 'Gammon Forum: Interrupts'. Geraadpleegd 24 maart 2021. http://gammon.com.au/interrupts.
Hofstede, Herman. z.d. 'De Inverse van een Matrix'. Geraadpleegd 13 juni 2019. https://www.hhofstede.nl/modules/inversematrix.htm.
Holzmann, Gerard J. 2006. 'The power of 10: Rules for Developing Safety-Critical Code'. 2006. https://www.researchgate.net/publication/220477862_The_Power_of_10_Rules_for_Developing_Safety-Critical_Code.
'HTTP Status Codes'. 2017. http://www.restapitutorial.com/httpstatuscodes.html.
'Hypertext Transfer Protocol (HTTP) Method Registry'. 2017. https://www.iana.org/assignments/http-methods/http-methods.xhtml.
'Hypertext Transfer Protocol (HTTP) Status Code Registry'. 2017. https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml.
'Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing'. 2014. 2014. https://tools.ietf.org/html/rfc7230.
'Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content'. 2014. https://tools.ietf.org/html/rfc7231.
'Hypertext Transfer Protocol Version 2 (HTTP/2)'. 2015. https://tools.ietf.org/html/rfc7540.
'interrupts() Arduino Reference'. z.d. Geraadpleegd 24 maart 2021. https://www.arduino.cc/reference/en/language/functions/interrupts/interrupts/.
Jeremiah, Jerry. z.d. 'Simple C example of doing an HTTP POST and consuming the response'. Geraadpleegd 8 maart 2021. https://stackoverflow.com/a/22135885.
'JSON'. 2005. https://nl.wikipedia.org/wiki/JSON.
Kernighan, Brian, en Dennis M. Ritchie. 1990. *C Handboek*. Prentice Hall.
Klemens, Ben. 2015. *21st Century C - C Tips from the New School*. O'Reilly.
'List of HTTP header fields'. 2018. https://en.wikipedia.org/wiki/List_of_HTTP_header_fields.
'Media type'. 2018. https://en.wikipedia.org/wiki/Media_type.
'Media Types'. 2018. https://www.iana.org/assignments/media-types/media-types.xhtml.
Mellis, David A. z.d. 'Arduino Webserver'. Geraadpleegd 8 maart 2021. https://www.arduino.cc/en/Tutorial/LibraryExamples/WebServer.
'MessagePack'. 2018. https://en.wikipedia.org/wiki/MessagePack.
'MessagePack home page'. z.d. Geraadpleegd 5 juni 2018. https://msgpack.org/.
'Monkey Island (series)'. z.d. Geraadpleegd 9 maart 2021. https://en.wikipedia.org/wiki/Monkey_Island_(series).
'Our Documentation: Python'. z.d. Geraadpleegd 6 februari 2023. https://www.python.org/doc/.
'Protocol Buffers'. z.d. Geraadpleegd 5 juni 2018. https://developers.google.co

m/protocol-buffers/.
'Pytest helps you write better programs'. z.d. Geraadpleegd 8 april 2021. https://docs.pytest.org/.
'Representational state transfer'. 2017. https://nl.wikipedia.org/wiki/Representational_state_transfer.
Robillard, Steve. 2012. 'operating systems - What should be done to secure Raspberry Pi? - Raspberry Pi Stack Exchange'. https://raspberrypi.stackexchange.com/questions/1247/what-should-be-done-to-secure-raspberry-pi/#1250.
Sajip, Vinay. z.d.a. 'Logging Cookbook'. Geraadpleegd 4 februari 2019. https://docs.python.org/3/howto/logging-cookbook.html.
———. z.d.b. 'Logging HOWTO'. Geraadpleegd 4 februari 2019. https://docs.python.org/3/howto/logging.html#logging-basic-tutorial.
Steen, Marc. 2023. *Ethics for people who work in tech*. CRC Press.
Taylor, Richard N. 2000. 'Representational State Transfer (REST)'. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
'The Atom Syndication Format'. 2005. https://tools.ietf.org/html/rfc4287.
'The Python Tutorial'. z.d. Geraadpleegd 2 april 2021. https://docs.python.org/3/tutorial/index.html.
'Volatile Type Qualifier'. z.d. Geraadpleegd 24 maart 2021. https://en.cppreference.com/w/c/language/volatile.
Xebia. z.d. 'Keep your logs clean'. Geraadpleegd 4 februari 2019. http://essentials.xebia.com/clean-logs/.
'YAML'. 2018. https://en.wikipedia.org/wiki/YAML.
'YAML Ain't Markup Language (YAML™) Version 1.2'. 2009. http://yaml.org/spec/1.2/spec.html.