Python

D. Leeuw

8 april 2025
v.0.1.0

Over dit Document

Inhoudsopgave

Over dit Document i

1 Wat is Python? 1
- 1.1 Opdrachten . . . 2
- 1.2 Installatie van de Python-interpreter . . . 2
  - 1.2.1 Windows . . . 2

2 Python als rekenmachine 5
- 2.1 Opdrachten . . . 6
- 2.2 Verdieping modulo en floor division . . . 6
  - 2.2.1 Opdrachten . . . 8

3 IDE 9
- 3.1 Visual Studio Code . . . 9

4 Variabelen 11
- 4.1 int, float, complex . . . 11
- 4.2 string . . . 11
- 4.3 boolean . . . 11
- 4.4 Casting . . . 11

5 Operators 13

6 If-statements (Condities) 15

7 Ordenen van data 17
- 7.1 list, tuple, range . . . 17
- 7.2 Dictionaries (Key/Value pairs) . . . 17
- 7.3 set, frozenset . . . 17

8 For-statements 19

9 While-statements **21**

10 For, While **23**
10.1 Verschillen tussen for en while 23
10.1.1 For-loop 23
10.1.2 While-loop 24
10.2 Samenvatting 24
10.3 Vragen 24
10.4 Opdrachten 25

11 Break, Continue **27**
11.1 Break: ontsnappen aan de loop 27
11.2 Continue: stappen overslaan 27
11.3 Opdrachten 28

12 Functies **29**
12.1 De functie definitie 29
12.1.1 Functie parameters 30
12.1.2 Functie return-waarden 31
12.2 Samenvatting 31
12.3 Opdrachten 31

13 Werken met bestanden **33**
13.1 Schrijven naar een bestand 33
13.2 Lezen van een bestand 33

14 Error afhandeling (Try/Except) **35**

15 Objects en Classes (OOP introductie) **37**
15.1 Opdrachten 38

16 Antwoorden van de opdrachten **39**
16.1 Wat is Python? 39
16.2 Python als rekenmachine 39
16.2.1 Verdieping modulo en floor division 40
16.3 For, While 40
16.4 Break, Continue 41
16.5 Functies 42
16.6 Objects en Classes (OOP introductie) 42

Index **43**

Hoofdstuk 1

Wat is Python?

Python is een programmeertaal. Met een programmeertaal kan je een computer dingen voor je laten doen. Doordat de computer dan werk voor je doet kan jij andere dingen doen (nee, niet gamen). Tijdens het programmeren gebruik je een programmeertaal om programma-code of kortweg code te schrijven. De programma-code schrijf je in een tekst-bestand zonder opmaak, we gebruiken daarom geen Word om een computer programma te schrijven maar een editor. Een editor is een heel simpele tekstverwerker zonder alle speciale functies als het invoegen van plaatjes, het nummeren van pagina's of het maken van hoofdstuk-koppen. De meeste programmeurs gebruiken een IDE (Integrated Development Environment).

Er zijn verschillende talen om een computer te vertellen wat hij moet doen. Python is er daar één van. Programmeertalen kunnen grofweg in twee soorten verdeeld worden. We hebben de scripting-talen en de talen die eerst gecompileerd moeten worden.

Talen die eerst gecompileerd moeten worden worden ook geschreven in een editor of een IDE, maar nadat het programma geschreven is gaat deze eerst door een compiler. Deze compiler maakt van de programma-code een binary. Op Windows systemen is dit vaak een bestand met de extensie .exe van executable. Een executable is een programma dat direct door een computer uitgevoerd kan worden. De binary bevat 1-en en 0-en die direct door de computer begrepen worden. Een executable kan dan ook op elke computer, met dezelfde processor en hetzelfde operating system, uitgevoerd worden zonder dat de programma-code aanwezig is.

Scripting talen hebben geen compiler, maar een interpreter. De taal waarin het programma is geschreven wordt op de computer door de interpreter omgezet naar machinetaal, of wel de 1-en en 0-en, om dan uitgevoerd te worden. Op elke computer waarop je het programma wilt draaien moet dus een interpreter en de code aanwezig zijn om de code te kunnen uitvoeren.

Python is een scriptingtaal en heeft dus een Python-interpreter nodig om gebruikt te kunnen worden.

1.1 Opdrachten

1. Zoek op Internet naar 3 programmeertalen die gecompileerd moeten worden en welke compiler daarvoor gebruikt kan worden

2. Zoek op Internet naar 3 programmeertalen die net als Python ook scripting-talen zijn en welke interpreter daarvoor nodig is

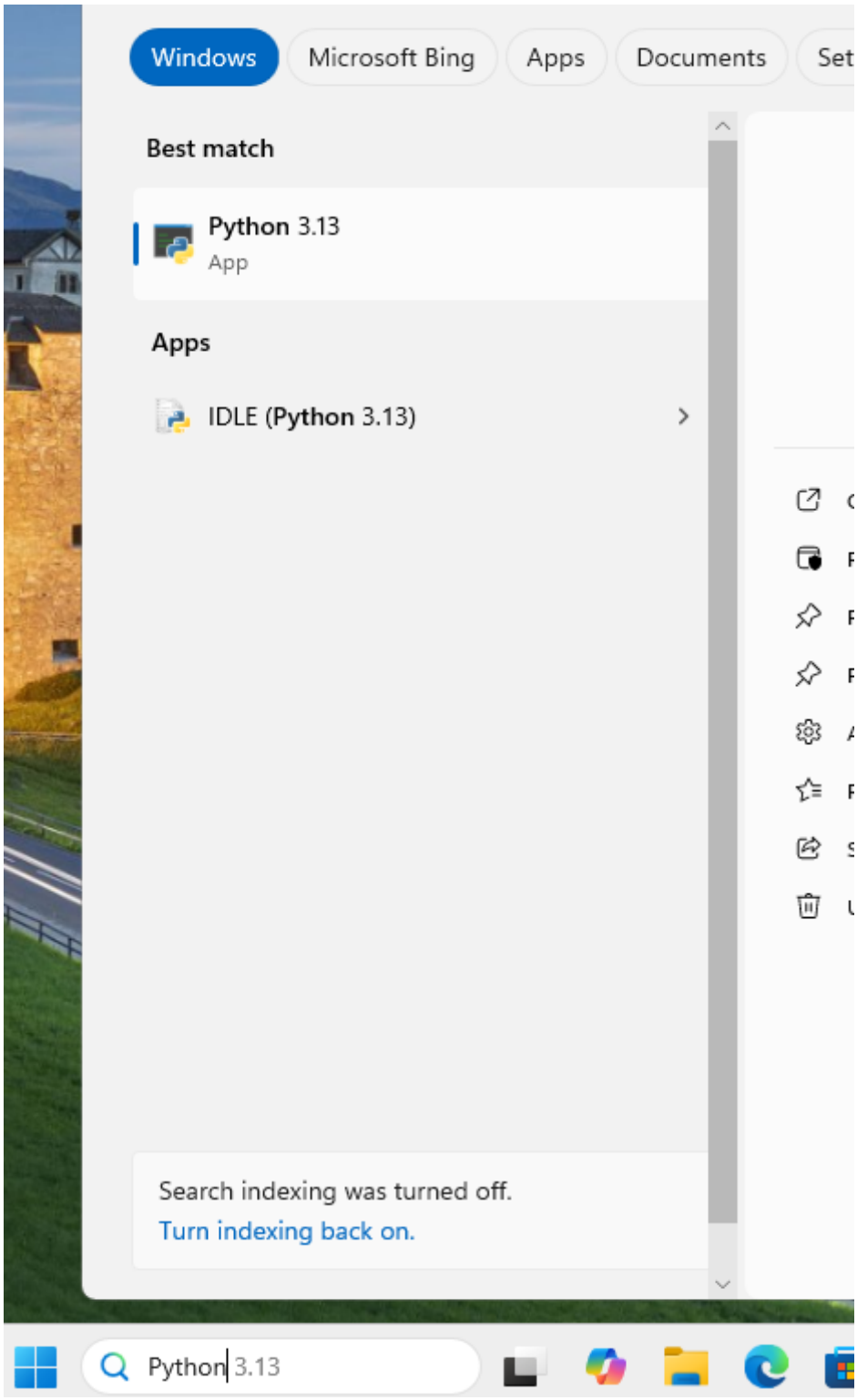
3. Zijn compilers en interpreters binaries? Leg je antwoord uit.

1.2 Installatie van de Python-interpreter

1.2.1 Windows

Volg de instructies op https://learn.microsoft.com/en-us/windows/python/scripting om Python te installeren. Alleen het kopje "Install Python". Je installeert nu de Python-interpreter.

Als Python geïnstalleerd is kan je via de zoekbalk Python opzoeken en opstarten, zoals je kan zien in 1.1



Figuur 1.1: Zoek en start Python

Om de Python-interpreter af te sluiten type je `exit()`. Denk om de haakjes aan het einde.

Hoofdstuk 2

Python als rekenmachine

Programmeertalen kunnen ook gebruikt worden om te rekenen, daarbij kunnen ze de traditionele rekenmachine vervangen. Start de Python-interpreter en gebruik Python als simpele rekenmachine zoals dat hieronder staat:

```
Python 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 6+7
13
>>> 7-6
1
>>> 6-7
-1
>>> 7*6
42
>>> 7/6
1.1666666666666667
>>>
```

We zien heel simpele zaken als optellen, aftrekken, vermenigvuldigen en delen.

Zo kunnen we ook machtsverheffen en worteltrekken:

```
Python 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 5**2
25
>>> 25**(1/2)
5.0
>>>
```

We gebruiken hier () om het delen van 1/2 voorrang te geven op het machtsverheffen. We kunnen met haakjes dus bepalen wat er voorrang heeft boven de normale rekenregels:

```
>>> 5+6*7+8
55
>>> (5+6)*(7+8)
165
```

Vermenigvuldigen gaat voor optellen. Bij de eerste som wordt dus eerst 6*7 gedaan waarna er 5 en 8 bij opgeteld worden. Als we haakjes gebruiken worden eerst 5+6 en 7+8 uitgevoerd waarna de uikomsten van de beide optellingen (11 en 15) met elkaar vermenigvuldigd worden.

Bij het worteltrekken zit er een kleine onnauwkeurigheid als we bijvoorbeeld met derdemachten werken, gelukkig komen die bij ons niet vaak voor:

```
>>> 5**3
125
>>> 125**(1/3)
4.999999999999999
```

De onnauwkeurigheid wordt veroorzaakt omdat 1/3 geen mooi rond getal is en dus zelf al een onnauwkeurigheid heeft. Er zijn functies in Python om wel nauwkeurig te werken met wortels, maar daar gaan we hier niet nader op in.

2.1 Opdrachten

Gebruik Python om de volgende sommen uit te rekenen.

1. Joris is jarig en hij heeft 20 taartjes meegenomen. In zijn klas zitten 17 medeleeringen en er is een juf. Iedereen krijgt 1 taartje. Hoeveel taartjes houdt Joris over?

2. Ahmed viert Suikerfeest met zijn familie. Er komen 24 familieleden langs en Achmed moet brood halen voor bij de maaltijd. Elk familielid eet ongeveer een half brood. Hoeveel brood moet Achmed halen?

2.2 Verdieping modulo en floor division

Bij het rekenen hebben we ook nog de geheeltallige deling (floor division) en de bijkomende modulo- of klok-rekenen. Om deze begrippen uit te leggen beginnen we met een voorbeeld.

Stel we hebben bij het begin van een nieuw schooljaar 213 studenten die willen gaan studeren. Er kunnen uit praktische overwegingen 30 studenten in een klas. Hoeveel klassen kunnen we dan volledig vullen en hoeveel studenten blijven er over?

Als we dit willen uitrekenen kunnen we het zo doen:

```
>>> 213/30
7.1
>>> 30*7
210
>>> 213-210
3
```

We hebben dus 7 klassen van 30 studenten en dus houden we 3 studenten over. Dit kan ook korter:

```
>>> 213//30
7
>>> 213%30
3
```

De dubbele slash // is de "floor division" ofwel die geeft weer hoeveel keer het volledige aantal in het totaal past. Met het % character berekenen we de modulo, kortom wat blijft er over na de vorige berekening.

Het voorgaande lijkt misschien een voorbeeld waar je niet zo vaak mee te maken krijgt, toch komen dit soort berekeningen redelijk vaak voor, wat duidelijk gemaakt wordt door de term klok-rekenen. Stieken doen we namelijk behoorlijk vaak aan modulo-rekenen, zonder dat we het weten. Als we horen dat we om 13:00 uur een afspraak hebben dan weten we dat dat om 1 uur is. Eigenlijk is onze "groep"-grote 12 en trekken we 12 van 13 af om op 1 uur uit te komen. Zo ook bij 16:00 uur waar we 12 van 16 aftrekken om op 4 uur uit te komen. Omdat we er altijd alleen maar 12 vanaf hoeven te trekken als het om de klok gaat valt het niet direct op, dat de echte wiskundige berekening feitelijk is:

```
>>> 16/12
1.3333333333333333
>>> 12*1
12
>>> 16-12
4
```

Om nog een stapje verder te gaan, het volgende voorbeeld. Stel we weten dat een auto er 6455768 seconden over doet om een bepaalde afstand af te leggen. Hoeveel uur, minuten en seconden heeft auto er dan over gedaan. Dat kunnen we berekenen met floor division en modulo-rekenen. We weten dat er 60 seconden in een minuut zitten en 3600 seconden in een uur. We gaan eerst de uren brekenen:

```
>>> 6455768//3600
1793
>>> 6455768%3600
```

```
968
```

We hebben dus 1793 uur gereden en er zijn 968 seconden over. Nu gaan we uitzoeken hoeveel minuten dat is en wat er dan nog aan seconden over blijft:

```
>>> 968//60
16
>>> 968%60
8
```

Er zijn dus 16 minuten verstreken en 8 seconden over. Het totaal komt dan uit op 1793 uur, 16 minuten en 8 seconden.

2.2.1 Opdrachten

1. Pieter heeft een duurloop gedaan en hij heeft daar 145476 seconden over gedaan. Hoeveel dagen, uren, minuten en seconden heeft Pieter hardgelopen?

Hoofdstuk 3

IDE

3.1 Visual Studio Code

Hoofdstuk 4

Variabelen

4.1 int, float, complex

4.2 string

4.3 boolean

4.4 Casting

Hoofdstuk 5

Operators

Hoofdstuk 6

If-statements (Condities)

Hoofdstuk 7

Ordenen van data

7.1 list, tuple, range

7.2 Dictionaries (Key/Value pairs)

7.3 set, frozenset

Hoofdstuk 8

For-statements

Hoofdstuk 9

While-statements

Hoofdstuk 10

For, While

Zowel de for-loop als de while-loop zijn manieren om bepaalde handelingen in een programma te herhalen onder bepaalde voorwaarden. Waarom zijn er twee manieren om dat te doen?

10.1 Verschillen tussen for en while

Er zijn een aantal redenen om waarom we in de ene situatie een for-loop gebruiken en in de andere een while-loop. Dit heeft te maken van de constructie van de twee verschillende loops. Bij een for-loop geef je op welke waarden doorlopen moeten worden, bij de while-loop geef je een conditie op die bereikt moet worden.

10.1.1 For-loop

- Als je van te voren weet hoe vaak je door een loop moet lopen, dan gebruik je de for-loop
- De for-loop volgt een sequentie (list, tuple, string, range) en voert het code block uit voor elk item in de sequentie
- De loop variable is beschikbaar in de loop
- Een for-loop zal doorgaan totdat alle elementen doorlopen zijn.
- Een for-loop stopt als het laatste item is bereikt.

10.1.2 While-loop

- De while-loop wordt gebruikt als je niet van te voren weet hoe vaak je een block code moet doorlopen, maar je weet aan welke conditie voldaan moet worden om de loop te laten eindigen.
- Bij een while-loop is het van belang dat de gegeven eind conditie ooit bereikt wordt, anders heb je een oneindige loop.
- Een while-loop zal doorgaan tot een bepaalde conditie is bereikt.
- Een while-loop stopt zodra de opgegeven conditie niet meer geldig is.

10.2 Samenvatting

Het belangrijkste verschil tussen een for- en while-loop is dat de for-loop kan worden gebruikt als het aantal iteraties bekend is en de while-loop kan worden gebruikt als het aantal iteraties niet bekend is.

	for-loop	while-loop
Loop variabele	Gedefinieerd in de loop aan het begin	Gedefinieerd buiten de loop, moet expliciet gedaan worden
Conditie	Controle voor elke iteratie	Controle voor elke iteratie
Update	Gedaan na elke iteratie	Gedaan in de loop, moet expliciet gedaan worden
Scope	Wordt bepaald door de loop body (voor gedefinieerd)	Moet expliciet gedaan worden
Gebruik	Als het aantal iteraties bekend is	Als het aantal iteraties niet bekend is, of als er aan een bepaalde conditie voldaan moet worden

10.3 Vragen

1. Achmed heeft een script geschreven om zijn servers automatisch te controleren. Via de verschillende IP-adressen kan hij via `ping` zien of de servers online zijn. De lijst met servers is gegevens als:

```
server_ips = [192.168.10.1, 192.168.10.4, 192.168,10.5,
    192.168.10.10, 192.168.10.55, 192.168.10.56 ]
```

(a) Wat zou achmed moeten gebruiken om alle IP-adressen via ping te controleren een for- of een while-loop?

(b) Zou Achmed de lijst ook kunnen doorlopen met een range(57)? Waarom wel/niet?

10.4 Opdrachten

1. Schrijf een script dat door de IP-adressen van Achmed loopt en een ping uitvoert naar elke server. Als de server aanwezig is geeft het script UP terug en als de server niet gevonden kan worden dan wordt DOWN teruggegeven.

2. Neem de volgende code over in je editor, sla de code op:

```
while num in range(55):
    print(f"Nummer is {num}")
```

run de code, welke foutmelding krijg je? En hoe los je deze op?

Hoofdstuk 11

Break, Continue

11.1 Break: ontsnappen aan de loop

De for- en de while-loop kunnen voortijdig afgebroken worden door het break-statement. Een for-loop hoeft dan bijvoorbeeld niet alle items te doorlopen en een while-loop kan gestopt worden als bijvoorbeeld de loop een aantal keer doorlopen is.

Voorbeeld

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

11.2 Continue: stappen overslaan

Het kan ook voorkomen dat je uit een reeks 1 bepaalde waarde niet mee wil nemen in je for-loop. Om door te gaan naar de volgende waarde in de reeks gebruiken we continue.

Voorbeeld

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

11.3 Opdrachten

1. Schrijf een script, gebruikmakend van range(), om de IP-adressen 1 t/m 10 te pingen van het 192.168.22.0 netwerk waarbij de IP-adressen 5 en 7 worden overgeslagen.

Hoofdstuk 12

Functies

Een functie is een herbruikbaar stuk code dat een specifieke taak of actie uitvoert. Het gebruik van functies bevordert een modulaire aanpak voor het ontwerpen van software, waardoor de code duidelijker, leesbaarder en herbruikbaarder wordt.

12.1 De functie definitie

Een functie wordt in Python gedefinieerd met het sleutelwoord "def", gevolgd door de naam van de functie, een paar haakjes, met daar tussen 0 of meer paramters en de functie-definitie wordt afgesloten met een dubbele punt. De code die de functie uitvoert, komt op de volgende regels en is geïndenteerd.

```
def naam_van_de_functie(parameters)
    # code van de functie
```

De naam van de functie mag willekeurig gekozen worden, zolang als deze maar niet overeenkomt met een bestaande functie.

Een functie staat los van het hoofdprogramma. Als je de functie aanroept in je hoofdprogramma, dan springt de computer naar de functie om als hij klaar is met de functie terug te keren naar waar die vandaan kwam:

```
# Functie, wordt niet uitgevoerd als het script start
def naam_van_de_functie(parameters):
    # code van de functie

# Hoofdprogramma, hier start het script
argument = "tekst"

# Programma springt naar functie
naam_van_de_functie(argument)
```

```
# Nadat de functie be-eindigd is gaat het programma hier verder
exit()
```

12.1.1 Functie parameters

Het is vaak nodig dat je gegevens (data) meestuurt vanuit het programma naar een functie. De meegegeven data kan dan in variabelen binnen de functie gebruikt worden. Deze variabelen hebben een speciale naam: we noemen ze parameters. Elke paramater kan een argument bevatten. De argumenten zijn de daadwerkelijke data die we meegeven hebben. Laten we dat eens bekijken in een voorbeeld:

```
def groet(naam):
    print(f"Hallo, {naam}")
```

De functie heeft als naam "groet" gekregen. Tussen haakjes staat de parameter (variabele) met de naam "naam" en de variable wordt in de functie gebruikt door het `print` statement.

Wanneer je de functie "groet" aanroept met een naam als argument, zal het die naam gebruiken om je te begroeten. De complete code zou er zo uit kunnen zien:

```
def groet(naam):
    print(f"Hallo, {naam}")

groet("Dennis")
```

Dus "Dennis" is hier het argument.

Een functie mag geen of meerdere parameters hebben. Als er geen parameters zijn dan staat er niets tussen de haakjes. De haakjes moeten er wel zijn! Wil je meer dan één parameter meegeven dan moeten de verschillende parameters gescheiden worden door een komma:

```
def groet(voornaam, achternaam):
    print(f"Hallo, {voornaam} {achternaam}")

groet("Dennis", "Leeuw")
```

Een vereiste is dat als er 2 parameters door de functie gevraagd worden er ook twee argumenten geleverd moeten worden. Niet meer en ook niet minder. Komt het aantal argumenten niet overeen met het aantal parameters dan geeft Python een error melding.

12.1.2 Functie return-waarden

Een functie kan waarden retourneren naar het hoofdprogramma door gebruik te maken van het "return"-sleutelwoord:

```
def optellen(a, b)
    return a + b
```

Er kan slechts een enkele waar terug gegeven worden. Return waarden mogen van elk data-type zijn. Dus als je meer waarden terug wilt sturen kan dat via een list, tuple of dictionatry zijn.

12.2 Samenvatting

Functies kunnen om een aantal verschillende redenen gebruikt worden:

Onderhoud Wijzigingen in de code hoeven maar in één functie gedaan te worden, in plaat van op verschillende plekken in het programma waar dezelfde code gebruikt wordt

Herbruikbaarheid Je kunt dezelfde functie meerdere keren in een programma gebruiken (met bijvoorbeeld verschillende argumenten).

Leesbaarheid Door stukken code in functies te stoppen kan het hoofdprogramma beter leesbaar worden. Kleine specialistische functies zijn makkelijker te lezen en omdat ook het hoofdprogramma korter wordt is ook dat makkelijker leesbaar.

Een functie heeft een naam en kan voorzien worden van data via argumenten die meegeven worden aan parameters.

De functie kan data terug geven via de return-value.

12.3 Opdrachten

1. Herschrijf het volgende programma gebruikmakend van een functie

```
lst_coureur = []

print("We starten een formule 1 race, daarvoor hebben 3 rijders
    nodig. Geef je naam van je rijder op:")

coureur = input("Gebruiker 1: Welke coureur ben jij? ")
lst_coureur.append(coureur)

while True:
```

```
    coureur = input("Gebruiker 2: Welke coureur ben jij? ")
    if coureur not in lst_coureur:
        lst_coureur.append(coureur)
        break
    print(f"Coureur {coureur} is al in de lijst")

while True:
    coureur = input("Gebruiker 3: Welke coureur ben jij? ")
    if coureur not in lst_coureur:
        lst_coureur.append(coureur)
        break
    print(f"Coureur {coureur} is al in de lijst")

print(lst_coureur)
```

Hoofdstuk 13

Werken met bestanden

Met Python kan je ook data van bestanden lezen en data naar bestanden schrijven dit onderdeel gaat over deze mogelijkheden. We gaan eenvoudige data lezen en schrijven van en naar een bestand.

Het werken met bestanden bestaat eruit dat we een bestand eerst moeten openen, daarna moeten we ervan lezen of naar schrijven, waarna we het bestand weer moeten sluiten. Om niet elke keer te hoeven aangeven om welk bestand het gaat gebruiken we een zogenaamde file handle. Je geeft in je script één keer aan welk bestand je wilt openen en koppelt daaraan een file handle en daarna gebruik je alleen nog deze file handle om te lezen of te schrijven. Tot slot moet je die file handle sluiten.

In Python ziet dat er ongeveer zo uit:

```
# File handle korten we af als fhdl
fhdl = open("testbestand.txt")
fhdl.write("Regel in bestand")
fhdl.close()
```

13.1 Schrijven naar een bestand

Write

13.2 Lezen van een bestand

Read

Hoofdstuk 14

Error afhandeling (Try/Except)

Hoofdstuk 15

Objects en Classes (OOP introductie)

Stel dat we een formule 1 racespel willen maken. We hebben voor dat spel een aantal auto's nodig die een aantal rondjes over een circuit racen. We zouden een script kunnen maken per auto en aan dat script het merk van de automodel, het team waarvoor gereden wordt, en de coureur mee kunnen geven. Die scripts kunnen we afzonderlijk aanroepen, maar daarmee hebben we nog geen race. We zouden ook een functie kunnen maken die we auto noemen en deze elke keer aanroepen met de gewenste eigenschappen. Ook dat lijkt nog niet echt op de werkelijheid. Wat we zouden willen is dat er een auto template is van waaruit we alle auto's kunnen maken.

Met object georienteerd programmeren kunnen we dat maken. In OOP (Object Oriented Programming) programmeer je een Class. Een Class is een template, daarin beschrijf je de eigenschappen van de auto en de functies die de auto kan uitvoeren. Als het programma draait kiezen de spelers hun auto, team en coureur en maak je van de Class een Object door bepaalde waarden te zetten. We zeggen dan dat we een Instance van een Class hebben. Het Object is een specifiek object met bepaalde waarden (variabelen) gezet die het "uniek" maken. Deze waarden die gezet kunnen worden noemen we Instance Variables. Door voor elke speler zijn eigen instance van de class te maken hebben we dus voor elke speler zijn eigen object gebaseerd op zijn eigen instance variabelen.

Tot slot willen we dat de auto een aantal rondjes gaat rijden op een circuit. Hiervoor hebben we Methodes. Methodes zijn functies die het object kan uitvoeren. We zouden een Methode kunnen hebben die zegt dat een auto een rondje moet rijden en als waarde geven we aan deze Methode 64 mee. De auto gaat nu 64 rondjes rijden. Er wordt dus een opdracht uitgevoerd.

Er zijn ook zaken die voor elke auto hetzelfde zijn, omdat de regels zijn

die formule 1 horen. Zo heeft elke auto 4 wielen, 1 stuur, is maximaal 200 cm breed en de hoogte van de auto is 950 cm gemeten vanaf de bodemplaat. Dit zijn waarden die voor elke auto hetzelfde zijn en kunnen dus niet gewijzigd worden. Dit zijn de zogenaamde Class Variables.

De Class voor dit object zouden we zo kunnen beschrijven:

- Instance variables: coureur, team, automodel
- Class variables: wielen=4, stuur=1, auto_breedte=200, auto_hoogte=950
- Methode(s): rondjes(aantal)

15.1 Opdrachten

1. Voor een emulatie programma willen we een virtuele server kunnen maken. De server hangt in een (virtueel) 19-inch rek en is aangesloten op een (virtuele) 1G ethernet switch. Jij moet een Class voor de server verzinnen, beantwoord de volgende vragen voor deze Class:
   (a) Wat zijn mogelijke Instance Variabelen (noem er 4)?
   (b) Wat zijn de Class Variables (noem er minimaal 2)?
   (c) Welke Methode(s) zou je aanbieden?
2. Een hostingbedrijf heeft je op hulp gevraagd. Zij willen hun klanten een extra dienst aanbieden. Alle web-sites die zij hosten voor hun klanten willen ze monitoren en zodra er een probleem is moet er een e-mail naar de klant gaan. Aan jou de taak om een Class te bedenken die een webserver zou kunnen monitoren.
   (a) Wat zijn mogelijke Instance Variables (noem er minimaal 2)?
   (b) Wat zijn mogelijke Class Variables (noem er minimaal 2)?
   (c) Welke Methode(s) zou je aanbieden (noem er minimaal 2)?

Hoofdstuk 16

Antwoorden van de opdrachten

16.1 Wat is Python?

1. Er zijn verschillende antwoorden mogelijk, tussen haakjes staan de mogelijke compilers: C (gcc), C++ (gcc), Pascal (gcc), Cobol (gcc), C#(?), Java (javac)

2. Perl (perl), PHP (php), JavaScript (javascript), Java (jre), sh (bash, sh), PowerShell (PowerShell)

3. Beide zijn binaries en moeten gecompileerd zijn voordat ze gebruikt kunnen worden. Zowel een compiler als een interpreter is gecompileerd voor de hardware en het OS waarop het moet kunnen functioneren, het moet het onderliggende systeem "kennen".

16.2 Python als rekenmachine

1. Joris mag zelf ook een taartje, zijn 17 medeleeringen krijgen er 1 en de juf ook 1:

```
>>> 20-(1+17+1)
1
```

   Er blijft er dus 1 over.

2. Achmed zijn familieleden eten elk een half brood, dus Achmed moet:

```
>>> (1/2)*24
12
```

   12 broden halen.

16.2.1 Verdieping modulo en floor division

1. Pieter heeft er 1 dag, 16 uur, 24 minuten en 36 seconden over gedaan.

```
>>> 145476//(3600*24)
1
>>> 145476%(3600*24)
59076
>>> 59076//3600
16
>>> 59076%3600
1476
>>> 1476//60
24
>>> 1476%60
36
```

16.3 For, While

Vragen

1. Lijst met IP-adressen
   (a) Een for-loop, want hij heeft een lijst (fixed aantal items)
   (b) Zou kunnen maar dan zijn er een hoop IP-adressen die DOWN geven, het maakt het niet overzichtelijker.

Opdrachten

1.

```
#!/usr/bin/python

# 2025 03 10 DL Check for OS (platform)
# 2025 03 09 DL Eerste versie

import os
import platform

server_ips = ["127.0.0.1", "192.168.10.1", "192.168.10.4",
    "192.168.10.5", "192.168.10.10", "192.168.10.55",
    "192.168.10.56"]

# Als windows gebruik -n anders -c (linux) als optie voor ping
my_platform = platform.system().lower()
if my_platform=='windows':
    param = '-n'
else:
    param = '-c'
```

```
# Loop door de ip adressen, we gebruiken FOR omdat we een lijst
    hebben.
for ip in server_ips:
    # Enkelvoudige ping naar CMD
    response = os.system(f"ping " + param + "1 " + ip)

    # Controleer de RETURN code
    # 0 is foutloos (ping returns)
    # iets anders is er is een fout
    if response == 0:
        print(f"{ip} is UP")
    else:
        print(f"{ip} is DOWN")

# END
```

2. De error-melding is:

```
NameError: name 'num' is not defined. Did you mean: 'sum'?
```

vervang while voor for om het probleem op te lossen.

16.4 Break, Continue

1.

```
import os
import platform

# Netwerk address
netaddr = "192.168.22"

# Check hoe we ping moeten aanspreken
if platform.system().lower() == 'windows':
    param = '-n'
else:
    param = '-c'

# Loop door de IP adressen 1 t/m 10 van ons netwerk
for addr in range(1,11):
    if addr == 5 or addr == 7:
        continue
    ip_addr = netaddr + "." + str(addr)

    response = os.system(f"ping " + param + "1 " + ip_addr)
```

16.5 Functies

1. Het script zou als volgt herschreven kunnen worden

```
lst_coureur = []

def vraag_coureur(nummer):
    while True:
        coureur = input(f"Gebruiker {nummer}: Welke coureur ben
    jij? ")
        if coureur not in lst_coureur:
            return(coureur)
        print(f"Coureur {coureur} is al in de lijst")

print("We starten een formule 1 race, daarvoor hebben 3 rijders
    nodig. Geef je naam van je rijder op:")

for i in range(1,4):
    antw = vraag_coureur(i)
    lst_coureur.append(antw)

print(lst_coureur)
```

16.6 Objects en Classes (OOP introductie)

1. De server Class:
   (a) Mogelijke Instance variables kunnen zijn: CPU, memory, harddisk, hoogte van de serverkast (in U)
   (b) Mogelijke Class variables kunnen zijn: Breedte van de server is 19-inch, Ethernet port is 1gb
   (c) Mogelijke Methodes kunnen zijn: service met een waarde als webserver, dhcp, dns, etc.
2. De webmoniting Class:
   (a) Mogelijke Instance variables kunnen zijn: IP-adres (server up/-down), domeinnaam (domein van de klant), e-mail adres (waar moet de storing heen)
   (b) Mogelijke Class variables kunnen zijn: port 80 en port 443
   (c) Mogelijke Methodes kunnen zijn: ping (server up/down), port check (normaal en ssl), get index.html (doet de site het)

Index

aftrekken, 5

binary, 1

Class, 37
Class Variables, 38
code, 1
compiler, 1

delen, 5

editor, 1
executable, 1

gecompileerd, 1

IDE, 1
Instance, 37
Instance Variables, 37
Integrated Development Environment, 1
interpreter, 1

machinetaal, 1
machtsverheffen, 5
Methodes, 37

Object, 37
Object Oriented Programming, 37
OOP, 37
optellen, 5

programma-code, 1
programmeertaal, 1
programmeren, 1

rekenen, 5
    aftrekken, 5
    delen, 5
    machtsverheffen, 5
    optellen, 5
    vermenigvuldigen, 5
    worteltrekken, 5

scripting, 1

vermenigvuldigen, 5

worteltrekken, 5