FACULTEIT
BIO-INGENIEURSWETENSCHAPPEN

PROGRAMMEREN I:

Syllabus & Werkcolleges

Prof. dr. ir. Jan VERWAEREN
Dr. Gauthier VANHAELEWYN
Dr. Demir Ali KÖSE
Ir Lander DE VISSCHER

Bachelor in de biowetenschappen
Academiejaar 2020–2021

UNIVERSITEIT
GENT

Inhoudsopgave

Voorwoord

Genomics, precisielandbouw, precision lifestock farming, robotisatie, big data, artificiële intelligentie en industrie 4.0 zijn voorbeelden van hoog-technologische disciplines en sectoren waar zeer veel (industrieel) ingenieurs in de biowetenschappen in contact mee komen. Bovendien kennen al deze sectoren een hoge graad van informatisering en automatisatie. Een gevolg daarvan is dat ingenieurs vaak zeer grote hoeveelheden ruwe gegevens moeten kunnen verwerken, en op basis van deze verwerking beslissingen nemen, in een beperkte tijdspanne. Enkele voorbeelden:

- Een bedrijf dat sla teelt in hydrocultuur heeft de laatste tijd te kampen met valse meeldauw. Je wordt gevraagd hiervan de oorzaak te achterhalen en beschikt onder andere over om metingen van temperatuur, luchtvochtigheid, CO_2 concentratie, fotosynthetisch-actieve-lichtintensiteitsmetingen in de serres op basis van tientallen sensoren van de laatste jaren (meer dan 10000 metingen) . . . en uiteraard ook je biologische kennis.
- Een plantenveredelingsbedrijf wenst na te gaan of een nieuw tomatenras een verhoogde resistentie heeft tegen bladvlekkenziekte. Om dit gedetailleerd te kunnen opvolgen wordt een proef aangelegd met 200 planten in potten. Deze potten worden op een transportband geplaatst en gedurende 10 weken wordt elke plant twee maal per dag automatisch gefotografeerd. Op basis van de resulterende 28000 foto's wenst men een gedetailleerd beeld te krijgen van resistentie tegen bladvlekkenziekte tijdens de ontwikkeling van de plant.
- Om inzicht te krijgen de infectieproces van *Fusarium graminearum* (een pathogeen op verschillende granen) wenst men een nieuwe mutant te maken waarin men een bepaald gen uitschakelt. Om dit gen gericht te kunnen uitschakelen moet je uiteraard op zoek gaan naar de plaats van dit gen in het genoom van dit organisme (ongeveer 36 miljoen basenparen).

De hoeveelheden gegevens die moeten verwerkt worden om de bovenstaande problemen het hoofd te kunnen bieden zijn van een grootte-orde waarbij een aanpak met pen-en-papier geen optie is. Vaak doet men dan ook beroep op computers om deze problemen op te lossen. Het gebruik van spreadsheet (MS Excel) of specifieke software kan soms een oplossing bieden, maar vaak zijn problemen dermate specifiek dat er geen software bestaat (of dat deze software niet flexibel genoeg is) om de nodige verwerking uit te voeren. In dergelijke gevallen zal je zelf software (al is de term script hier meer gepast, zie verder) moeten schrijven die je kan helpen bij deze analyses. In deze cursus wordt de basis gelegd voor de programmeervaardigheden die je daarvoor nodig hebt.

Concreet leer je in deze cursus de basisprincipes van gestructureerd, modulair en deels object-georiënteerd programmeren, alsook deze principes toe te passen om problemen op een tijdsefficiënte

en geautomatiseerde manier op te lossen. Tijdens de lessen wordt aangeleerd hoe een taak die geformuleerd wordt in een natuurlijke taal kan omgezet worden in een programma dat door een computer kan uitgevoerd worden. Hierbij wordt gebruik gemaakt van de programmeertaal Python. Deze cursus kan je dan ook in de eerste plaats beschouwen als een inleiding tot programmeren in Python 3.

Merk op dat deze cursusnota's deel uitmaken van een lessenpakket, waarbij ook lesslides horen die beschikbaar zijn via het Elektronisch Leerplatform.

DEEL I

SYLLABUS

1

Computers, programma's en Python

Nagenoeg iedereen maakt dagelijks gebruik van een computer. Met bepaalde computers kan je surfen op het web, computerspelletjes spelen, een stuk tekst schrijven, enz. tot zelfs het weer voorspellen. De vragen 'Wat is een computer precies?' of 'Waarom kan een computer zo veel verschillende taken uitvoeren?' gebruiken we in als startpunt in deze inleidende programmeercursus.

1.1 Een flexibele machine

Een moderne computer kan men definiëren als een '*Machine die informatie opslaat en manipuleert onder controle van een wijzigbaar programma*'. We onderscheiden in deze definitie twee elementen. Ten eerste is een computer een toestel dat informatie manipuleert. Dit wil zeggen dat we informatie in een computer kunnen invoeren en dat de computer deze informatie kan manipuleren en transformeren en tenslotte het resultaat kan tonen aan de gebruiker (bijvoorbeeld op het scherm). In die zin verschilt een computer weinig van een eenvoudig rekentoestel of een digitale chronometer. Ook daarop kan je informatie invoeren, verwerken en het resultaat bekijken.

Het belangrijkste verschil tussen een rekentoestel en een (universele) computer is het deel 'controle door een wijzigbaar programma' in de definitie. De taken die een eenvoudig rekentoestel kan uitvoeren zijn vaak beperkt tot het optellen van een aantal getallen. De verschillende programma's (of software) die op een computer aanwezig zijn daarentegen, stellen dit toestel in staat om meerdere taken, die sterk van elkaar kunnen verschillen, uit te voeren. De mogelijkheid om verschillende programma's te kunnen uitvoeren is dus een belangrijk aspect, maar wat bedoelt men hier precies met 'een programma'?

Een *computerprogramma* is een gedetailleerde, stap-voor-stap omschreven, sequentie van instructies die ondubbelzinnig beschrijven wat een computer moet doen. Wanneer een programma gewijzigd wordt zal de computer een andere sequentie van instructies uitvoeren en bijgevolg een ander taak uitvoeren. Het schrijven van deze sequentie van instructies in een taal die voor een mens leesbaar is, maar, mits een aantal tussenstappen, ook door een computer uitvoerbaar is noemt men *programmeren*.

Software en hardware

In de definitie van een moderne computer die hierboven gegeven werd, staat de mogelijkheid om verschillende programma's of *software* uit te voeren op een centrale plaats. De computer wordt in deze definitie beschreven op basis van zijn functionaliteiten (wat het toestel moet kunnen). Om deze functionaliteiten te voorzien is er *hardware* nodig. De hardware is de verzameling van alle fysieke onderdelen van een computer waaronder de processor, het moederbord, het RAM-geheugen, de harde schrijf, het scherm, het toetsenbord, enz. De belangrijkste onderdelen zijn elektronische onderdelen en zijn opgebouwd uit geleiders, transistoren, condensatoren, enz., die vrij ingewikkelde schakelingen vormen.

Basisinzicht in de elektronische schakelingen die een computer bevat en de mogelijkheden en beperkingen die deze met zich meebrengen is belangrijk wanneer men leert programmeren. De opslag en verwerking van informatie die vermeld wordt in de definitie, wordt immers door deze schakelingen verwezenlijkt. Een overzicht van de hardware componenten en hun link met informatieopslag en -verwerking kan teruggevonden worden in de lesslides.

Belangrijke aanvulling op theorieslides!

Bekijk de theorieslides voor een inleiding tot de hardware van een computer en de link met data en gegevenstypes.

1.2 De programmeertaal Python

Python is een *programmeertaal*, dit is een formele taal waarin men opdrachten kan schrijven die door een computer kunnen worden uitgevoerd. Net als een natuurlijke taal (zoals bv. het Nederlands) heeft een programmeertaal een woordenschat en een aantal grammaticale regels, waarnaar men vaak verwijst als de syntax(is) van de taal. Iemand die de programmeertaal Python aanleert, moet dus onder andere leren om deze regels correct toe te passen. Een collectie, door een mens leesbare, instructies die men in een programmeertaal neerschrijft noemt noemt men *broncode*. Het onderstaande codefragment is een voorbeeld van broncode die geschreven werd in Python 3.

Fragment 1.1: `voorbeeldSom.py`

```
1 a = 5.0
2 b = 6.0
3 c = a + b
4 print(c)
```

Wanneer deze broncode wordt uitgevoerd (zie hierna), zal de waarde 11 op het scherm verschijnen.

Waarom Python?

De keuze voor een (eerste) programmeertaal hangt af van meerdere factoren. Belangrijke factoren zijn de toepassing(en) waarvoor de taal zal gebruikt worden, de leercurve van de taal (hoe moeilijk

een taal aan te leren is), de (voor)kennis van de programmeur. Zonder in detail te gaan worden hieronder een aantal redenen aangehaald die een keuze voor Python motiveren .

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. (overgenomen van `https://www.python.org/`*)*

1.3 De Python interpreter

Zoals Codefragment 1.1 illustreert, is broncode platte tekst die kan worden opgeslagen op de harde schrijf van een computer in een tekstbestand. Om aan te duiden dat zo'n file Python broncode bevat gebruikt men vaak de extensie `.py` in de bestandsnaam van deze bestanden (zoals bijvoorbeeld `voorbeeldSom.py` in het voorbeeld). Indien men de instructies die in een broncodebestand effectief door de computer wil laten uitvoeren, dan moet men gebruikmaken van een Python 3 interpreter. De interpreter is een computerprogramma dat broncode, die geschreven werd in de programmeertaal Python 3, uitvoert. Om deze reden noemt men Python een geïnterpreteerde taal (het interpreteren en uitvoeren van de broncode gebeurt in één stap), in tegenstelling tot gecompileerde talen waarbij broncode eerst wordt gecompileerd naar machine code en daarna pas wordt uitgevoerd. In de volgende sectie wordt beschreven hoe je de Python interpreter kan installeren op je toestel.

1.4 Installatie van de Python interpreter

De *Python 3 interpreter* is een computerprogramma dat broncode, die geschreven werd in de programmeertaal Python 3, uitvoert. Om gebruik te kunnen maken van Python moet de interpreter worden geïnstalleerd. In deze cursus gebruiken we de CPython interpreter (dit is de meest populaire interpreter en werd geschreven in de programmeertaal C, vandaar de letter C in de titel). De meest recente officiële versie van deze interpreter kan je vinden op `https://www.python.org/downloads/`. Voor het algemeen gebruiksgemak is het echter handig om, samen met de interpreter, een aantal populaire aanvullende modules te installeren (zoals bv. `numpy`, `skimage`, ...) samen met de interpreter. In de cursus raden we aan om gebruik te maken van de Anaconda distributie van Python 3.6 (of hoger). Deze distributie bevat een Python 3.6 interpreter alsook een aantal modules of bibliotheken die in dit opleidingsonderdeel vaak zullen gebruikt worden. Bovendien is deze distributie *gratis*, dit geldt tevens voor de meeste Python-tools. Je kan de **installer downloaden op**:

`https://www.anaconda.com/download/`

De **installatie** wijst zichzelf uit[1], maar kan even duren. Bij problemen kan je de installatie-instructies terugvinden op:

`http://docs.anaconda.com/anaconda/install/`

Nagaan of Anaconda correct werd geïnstalleerd kan je als volgt doen.

- Windows: Ga naar *Start* > selecteer *Anaconda Navigator*. Indien de navigator geopend wordt is de installatie geslaagd.
- macOS: Ga naar de *Launchpad* > selecteer *Anaconda Navigator*. Indien de navigator geopend wordt is de installatie geslaagd.

1.5 Keuze voor een IDE

Python broncode kan je schrijven en aanpassen in een eenvoudige teksteditor zoals bijvoorbeeld *Notepad* (Kladblok) en bewaren als een tekstbestand (vaak met de extensie `.py`). Korte, eenvoudige programma's kunnen op deze manier ontwikkeld worden, maar vaak is het aan te raden om gebruik te maken van een *Integrated Development Environment (IDE)*. Dit is software die speciaal ontworpen werd om het schrijven en testen van broncode te vereenvoudigen. Enkele voorbeelden van IDEs zijn:

- Eclipse, met de PyDev plugin (zie `http://www.pydev.org/`)
- PyCharm (zie `https://www.jetbrains.com/pycharm/`)
- **Spyder** (zie `https://www.spyder-ide.org/`)

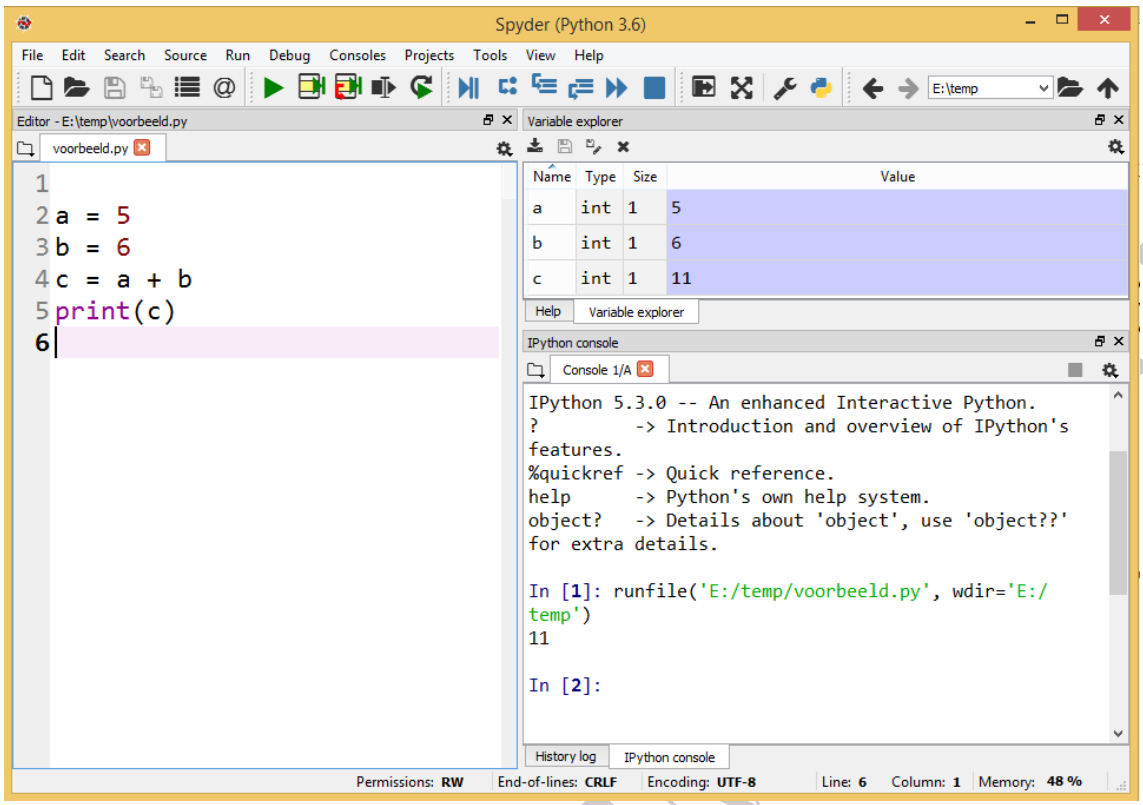
Elk van deze IDEs heeft een aantal voor- en nadelen. In een wetenschappelijke context is **Spyder** een populaire IDE. Bovendien wordt deze IDE automatisch geïnstalleerd bij de installatie van Anaconda. In deze cursus wordt daarom aangeraden om gebruik te maken van Spyder.

1.6 Eerste gebruik van de Spyder IDE

De Spyder IDE kan geopend worden via de Anaconda launcher. Na het opstarten van Spyder krijg je een venster te zien zoals getoond wordt in Figuur 1.1. Drie belangrijke deelvensters (*panes*) zijn de Editor (links), Variable explorer (rechtsboven) en de IPython Console (rechtsonder).

- **Editor pane**: Dit deelvenster is een eenvoudige text-editor waarin je tekst kan invoeren. Merk op dat de ingevoerde tekst automatisch wordt opgemaakt (dit heet syntax highlighting) om broncode meer leesbaar te maken. Broncode die hier wordt ingevoerd kan vervolgens worden opgeslagen via *File* > *Save*. De broncode uitvoeren kan via *Run* > *Run*, via de

[1] Indien mogelijk installeer je best voor alle gebruikers.



Figuur 1.1: Screenshot van de Spyder IDE.

sneltoets *F5* of via het *Play*-icoontje in de werkbalk. Indien je deze broncode uitvoert, dan zal de volgende instructie in de IPython Console verschijnen (waarbij `voorbeeld.py` de naam is van het codebestand):

```
runfile('E:/temp/voorbeeld.py', wdir='E:/temp')
```

Een bestaand script openen kan via *File* > *Open*.

- **IPython Console**: In deze console kan je rechtstreeks Python instructies invoeren. Deze worden dan onmiddellijk uitgevoerd. Ook het uitvoeren van code in het *Editor pane* (zoals hierboven getoond) gebeurt automatisch in deze console. Merk op dat in Figuur 1.1 de waarde 11 in de IPython console staat. Dit wil zeggen dat de code die in de Editor pane te zien is reeds werd uitgevoerd voor het maken van het screenshot.
- **Variable explorer**: In dit deelvenster krijg je een overzicht van alle variabelen die in het werkgeheugen aanwezig zijn. Merk op dat in het werkgeheugen drie variabelen te zien zijn (`a`, `b` en `c`) omdat deze variabelen werden gedeclareerd in de uitgevoerde broncode.

1.7 De Spyder IDE via Athena

De Spyder IDE kan ook gebruikt worden via Athena (`https://athena.ugent.be`). In dit geval hoef je de Python interpreter of Anaconda niet te installeren op je toestel. **Let op**: voordat je

Athena voor de eerste keer gebruikt dient een Citrix client geïnstalleerd te worden op je toestel. Installatie-instructies kan je vinden op `https://helpdesk.ugent.be/athena/ica.php`.

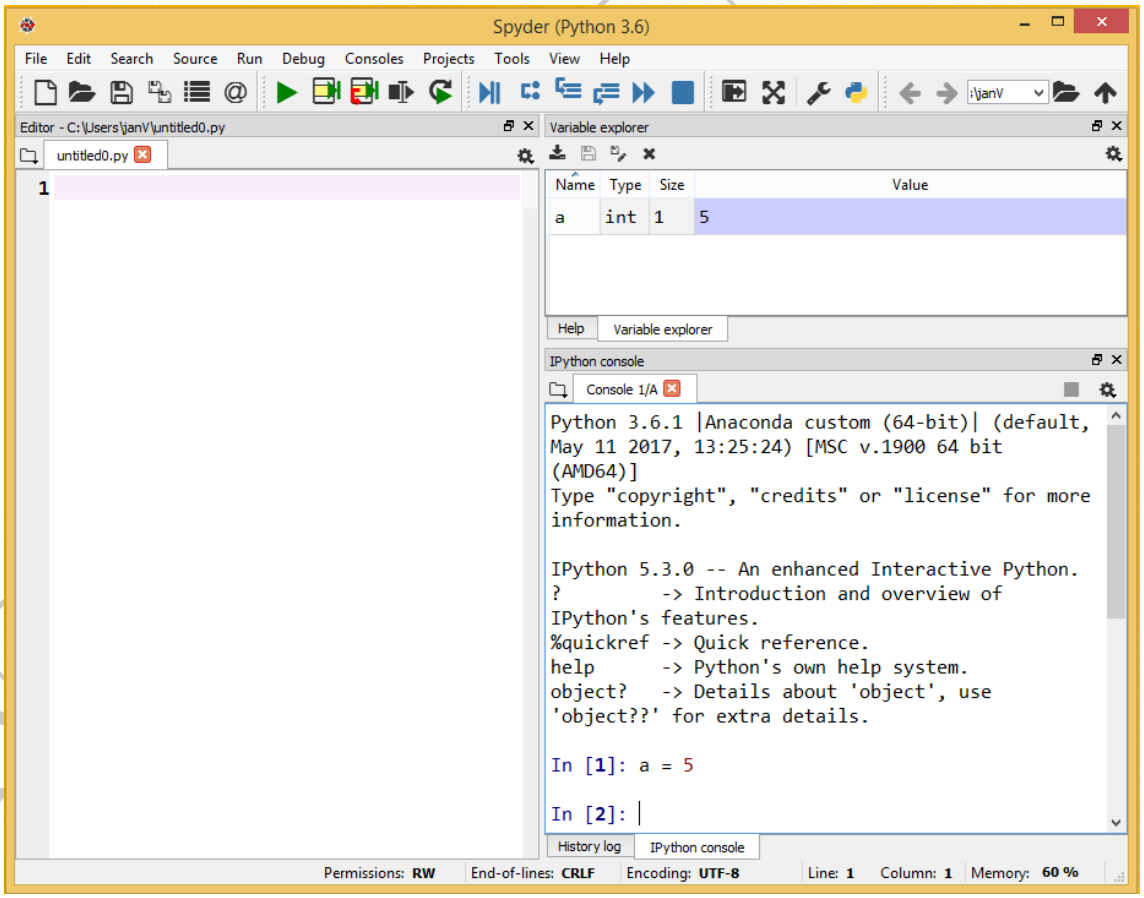
1.8 Gebruik van de IPython console

Codefragment 1.1 is een voorbeeld van een Python script. Hoe een script wordt uitgevoerd, werd reeds geïllustreerd in Sectie 1.6. Eenvoudige instructies kan men ook rechtstreeks invoeren in de *IPython console* (het venster rechtsonder in Figuur 1.1).

In de volgende hoofdstukken worden instructies vaak geïllustreerd d.m.v. IPython console sessies. Stel dat men bedoelt dat *een variabele a moet worden gecreëerd met waarde 5* in een console. Dan wordt dit al volgt geïllustreerd:

```
>>> a = 5
```

De prompt-tekens >>> geven aan dat een instructie wordt uitgevoerd in de console. Figuur 1.2 toont een screenshot van diezelfde instructie in de console (rechtsonder). Een instructie wordt uitgevoerd door ze in te typen en vervolgens op *Enter* te drukken.



Figuur 1.2: Illustratie van het gebruik van de IPython console.

Merk op dat:

- Ingevoerde instructies niet (of toch moeilijk) bewaard kunnen worden.
- Uitgevoerde instructies niet ongedaan gemaakt kunnen worden (men kan uiteraard wel de console heropstarten). Stel dat, per vergissing, `a = 50` werd ingegeven, dan kan dit wel rechtgezet worden door de correcte instructie `a = 5` daarna opnieuw in te voeren.
- Na het uitvoeren van de bovenstaande instructie de variabele `a` en de bijhorende waarde `5` werden opgenomen in de *Variabele explorer*. Dit wil zeggen dat deze variabele en zijn waarde in daarna volgende instructie kunnen gebruikt worden. Bijvoorbeeld:

```
>>> b = a + 2
>>> print(b)
7
```

Op basis van deze demonstratie kunnen de eenvoudige instructies die aan bod komen in het eerste deel van het volgende hoofdstuk correct worden uitgevoerd.

2

Python bouwstenen

In Hoofdstuk 1 werd een computerprogramma omschreven als *een gedetailleerde, stap-voor-stap omschreven, sequentie van instructies die ondubbelzinnig beschrijven wat een computer moet doen.* In dit hoofdstuk wordt beschreven hoe de meest eenvoudige instructies worden opgebouwd.

2.1 Objecten, gegevenstypes en variabelen

2.1.1 Numerieke en tekstuele gegevens: een voorbeeld

Als een startpunt hernemen we Codefragment 1.1 uit de inleiding.

```
1 a = 5.0
2 b = 6.0
3 c = a + b
4 print(c)
```

Wanneer dit codefragement wordt uigevoerd, zullen de instucties van-boven-naar-onder worden uitgevoerd. Dit codefragment heeft als output:

```
11
```

In dit codefragment noemen we `a`, `b` en `c` **variabelen**. De variabele `a` verwijst naar (een object met) de **waarde** `5.0` en de variabele `b` verwijst naar (een object met) de waarde `6.0`. Om de waarde van `c` te bepalen zullen de waarden van `a` en `b` opgehaald worden en vervolgens worden opgeteld. Het resultaat (de waarde `11.0`) wordt toegekend aan de variabele `c`.

Het bovenstaande codefragment toont dat variabelen kunnen verwijzen naar numerieke waarden en dat deze variabelen kunnen gebruikt worden in bewerkingen. Variabelen kunnen ook verwijzen naar tekstuele gegevens zoals getoond wordt in het onderstaande codefragment:

```
1 zin = "Het is mooi weer vandaag"
2 print(zin)
3 lengte = len(zin)
4 print(lengte)
```

Dit codefragment heeft als output:

```
Het is mooi weer vandaag
24
```

In dit codefragment verwijst de variabele `voorbeeld` naar (een object met) de waarde `Het is mooi weer vandaag`. Deze waarde is tekstueel. Om dit duidelijk te maken aan de interpreter wordt deze op regel 1 tussen `" "` geplaatst. In regel 3 wordt bepaald hoeveel karakters de zin bevat waarnaar de variabele `voorbeeld` verwijst (dit zijn 24 karakters inclusief spaties).

Opmerking: commentaar in code
Vaak is het aangewezen om broncode te voorzien van **commentaar**. Dit kan je doen door deze commentaar te laten voorafgaan door een `#`. Wat volgt na deze hashtag wordt niet geïnterpreteerd door de interpreter. Code voorzien van commentaar ziet er als volgt uit:

```
1 a = 5.0    # variabele a verwijst naar object met waarde 5.0
2 b = 6.0
3 c = a + b
4 print(c)   # print functie print waarde van c op scherm
```

Opmerking: de functie `print`
In de eerste codefragmenten wordt vaak gebruikgemaakt van de functie **`print`**. Deze functie zorgt ervoor dat de waarde van een variabele (die tussen haakjes geplaatst wordt) naar het scherm wordt geprint bij het uitvoeren van het codefragment.

2.1.2 Objecten en gegevenstypes

Python is een object-georiënteerde programmeertaal. Men zegt vaak dat in Python *alles* een object is. Een gevolg daarvan is na het uitvoeren van de instructie

```
a = 5.0
```

de variabele `a` verwijst naar een **object** met als **waarde** `5.0`. Een object is een vrij abstract begrip en tot op dit moment hebben we nog maar één belangrijk aspect van een object vernoemd: zijn waarde.
De onderstaande instructie maakt een object aan met als waarde `Het is mooi weer vandaag`:

```
voorbeeld = "Het is mooi weer vandaag"
```

De voorgaande voorbeelden tonen aan dat Python objecten zowel numeriek als tekstueel kunnen zijn. Echter, de manier waarop numerieke gegevens bewaard worden in het werkgeheugen verschilt sterk van de manier waarop tekstuele gegevens bewaard worden. Bovendien zijn de bewerkingen die kunnen uitgevoerd worden op beide types gegevens verschillend. Een object met een tekstuele waarde heeft dan ook een ander **gegevenstype** dan een object met een numerieke waarde. Het gegevenstype is dan ook tweede belangrijk aspect van een object (naast zijn waarde). Drie belangrijke gegevenstypes zijn:

- **het gegevenstype `float`**: Een belangrijk gegevenstype voor numerieke gegevens in Python, en in het bijzonder **decimale getallen**[1]. Het statement `a = 5.0` zorgt ervoor dat in het werkgeheugen een object van het type float wordt gecreëerd met als waarde `5.0`. Je kan floats optellen, vermenigvuldigen, enz.
- **het gegevenstype `int`** (lees *integer*): Een belangrijk gegevenstype voor numerieke gegevens in Python, en in het bijzonder **gehele getallen**. Het statement `b = 7` zorgt ervoor dat in het werkgeheugen een object van het type int wordt gecreëerd met als waarde `7`.
- **het gegevenstype `str`** (lees *string*): Het standaard gegevenstype voor tekstuele gegevens in Python. Het statement `voorbeeld = "een test"` creëert een object van het type string in het werkgeheugen met als waarde `een test`. Tekstuele gegevens zijn in Python een opeenvolging van karakters. Van een string object kan je bijvoorbeeld nagaan hoeveel karakters het bevat. Na het uitvoeren van de instructie `print(len(voorbeeld))` zal 8 op het scherm verschijnen.

Belangrijke aanvulling op theorieslides!

Bekijk de theorieslides voor een inleiding tot gegevenstypes en binaire voorstellingen.

Objecten aanmaken in Python

Waarden zullen in Python steeds worden ondergebracht in objecten (in het werkgeheugen) van een gepast type. Bij het uitvoeren van de instructie `a = 5`, wordt het karakter `5` herkend door de interpreter als een getal met waarde 5 dat vervolgens wordt ondergebracht in een object van het type `int`. Hetzelfde geldt voor de instructie `a = "hallo"`. De karakters `"hallo"` worden erkend door de interpreter en leiden ertoe dat een string object wordt gecreëerd met waarde `hallo`. Onderdelen van broncode (zoals `5` en `"hallo"` in de voorgaande voorbeelden) die de interpreter rechtstreeks herkent en aanleiding geven tot de creatie van nieuwe objecten noemt men **literals**.

In de voorbeelden tot nu werden de gecreëerde objecten steeds toegekend aan een variabele. De variabele verwijst dan naar het object zodat je dit object later in je code kan oproepen. Het toekennen van een object aan een variabele is echter niet noodzakelijk. Je kan een object aanmaken een door een literal rechtstreeks in de Python console te typen en op *Enter* te drukken (zie Sectie 1.8 voor een demonstratie van het gebruik van de IPython console), bv.

[1] Het gegevenstype float wordt gebruikt om rationale getallen efficiënt voor te stellen in het werkgeheugen. LET OP: niet elk rationaal getal kan voorgesteld worden als een float. Indien dit niet mogelijk is wordt automatisch afgerond (de afrondingsfout is globaal genomen vrij klein).

```
>>> "ACAAGA"
'ACAAGA'

>>> 6.02214129e+23
6.02214129e+23
```

De eerste instructie maakt een object van het type **`str`** aan en de tweede instructie een object van het type **`float`**. LET OP: omdat er geen variabelen verwijzen naar deze objecten kan je ze nadien niet meer aanspreken, maar het kan een handige manier zijn om na te gaan of een bepaalde waarde kan geïnterpreteerd worden.
Indien er een foutmelding optreedt kan het object niet aangemaakt worden.

```
>>> ACAAGA
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ACAAGA' is not defined

>>> 1 2 3 4 5
  File "<stdin>", line 1
    1 2 3 4 5
      ^
SyntaxError: invalid syntax
```

Opdracht 2.1

Voer de volgende waarden (literals) in in een Python console, bestudeer het resultaat en ga na welke waarden succesvol kunnen worden ingevoerd:

- `101300`
- `1013e2`
- `1013hPa`
- `1.0e0.1`
- `1.0e+308`
- `1.0e+309`
- `-122`
- `--122`
- `+122`
- `0b01100110`
- `012`

Opmerking

De notatie `0b01100110` is een voorbeeld van een binaire voorstelling van een getal.

Het gegevenstype opvragen: de functie `type`

Elk object heeft een gegevenstype (*strongly typed*). In Python zijn er verschillende soorten gegevenstypes. Voor een oplijsting van de gegevenstypes belangrijk in deze cursus, zie gedeelte 7.9. Het gegevenstype van een object kan nagegaan worden met de functie **`type()`**, bv.

```
>>> type("ACAAGA")
str

>>> type(101300)
int

>>> type(0.0089102)
float
```

toont aan dat de objecten `"ACAAGA"`, `101300` en `0.0089102` respectievelijk van het type **`str`** (*string*), **`int`** (*integer*) en **`float`** (*float*) zijn.
Indien gebruik gemaakt wordt van variabelen, dan kan het type van een object opgeroepen worden via de variabele die ernaar verwijst. Dit is vaak de meest aangewezen manier van werken.

```
>>> dna = "ACAAGA"
>>> type(dna)
str

>>> x = 101300
>>> type(x)
int
```

Opdracht 2.2

Ga na wat de gegevenstypes zijn van de objecten die Python creëert voor onderstaande literals (je zal een aantal nieuwe gegevenstypes zien, die later nog aan bod komen). Stemmen de resultaten overeen met je verwachtingen?

- `159`
- `159.0`
- `5+8j`
- `[1, 2, 3, 4, 5]`
- `[]`
- `"B"`
- `"BBBBBBB"`
- `" "`
- `""`
- `(5.332, 'z')`
- `{"Belgium":11144420, "Netherlands":16802463, "Luxembourg": 536761}`
- `True`
- `False`

2.1.3 Variabelen

In de voorgaande secties werd meermaals de term **variabele** gebruikt. Een variabele is een naam die verwijst naar een object. De toekenning van een naam aan een object gebeurt d.m.v. de toekenningsoperator (=). Een instructie waarin een dergelijke toekenning gebeurt noemt men een **toekenningsstatement** (**assignment statement**). Voorbeelden van toekenningsstatements zijn:

```
>>> a = 7                                   # int met waarde 7
>>> zin = "Het is mooi weer vandaag!"       # str
>>> b = 18.2 + 15.1                         # float met waarde 33.3
>>> b_plus_een = b + 1.0                    # float met waarde 34.3
```

De laatste twee regels in het bovenstaande voorbeeld illustreren tevens de **kracht van variabelen**. Ze **verwijzen naar een object zodat je dit object** (en dus ook zijn waarde) **nadien kan gebruiken in andere instructies**.

Uit het voorgaande blijkt dat de naam van een variabele kort of lang kan zijn. Daarnaast kan een naam ook bijzondere tekens bevatten zoals underscores. Er zijn echter een aantal regels waaraan de naam van een variabele moet voldoen.

1. Elke naam moet beginnen met een letter of de underscore karakter (_):
   - Een cijfer is niet toegelaten als eerste karakter.
   - Meervoudige woordnamen kunnen gelinkt worden met de underscore karakter (_), bv. `veld_tarwe`, `veld_mais`.
   - Voorlopig zullen we de underscore (_) niet gebruiken als eerste karakter (variabele-namen die starten met een underscore hebben vaak een bijzondere betekenis).
2. Na de eerste letter mag de naam een willekeurige combinatie zijn van letters, cijfers en underscores:
   - De naam mag geen zgn. *keyword* zijn zoals in Tabel 2.1, zie gedeelte 7.9.
   - Er mogen geen scheidingstekens (*delimiters*), punctuaties of operatoren voorkomen in een naam.
3. Een naam mag een willekeurige lengte hebben.
4. 'HOOFDLETTERS' zijn verschillend van 'kleine letters'
   - `mijn_naam` is verschillend van `mijn_Naam` of `Mijn_naam`

Een voorbeeld van het aanmaken van een geldige variabelnaam in de Python console:

```
>>> c_plus_plus = "C++ is een programmeertaal."    # bevat operator +
```

Een voorbeeld van een ongeldige variabelnaam:

```
>>> c++ = "C++ is een programmeertaal."
  File "<stdin>", line 1
    c++ = "C++ is een programmeertaal."
        ^
SyntaxError: invalid syntax
```

Opdracht 2.3

Welke van de onderstaande variabelenamen zijn toegelaten in Python. Bij variabelenamen die niet geldig zijn, geef de reden(en).

- `xyzzy`
- `2deVariabele`
- `rich&bill`
- `zeer_lange_naam`
- `good2go`
- `python-input`

Controleer je antwoord door elk van de variabelen aan te maken en te laten verwijzen naar een object, bv. een *integer*, *float* of *string*.

Python keywords

Keywords zijn speciale woorden in Python die niet mogen gebruikt worden als variabelenaam (Tabel 2.1). Deze woorden hebben voor de Python interpreter dan ook een bijzondere betekenis (verder in de cursus wordt zullen een aantal van deze keywords aan bod komen).

Tabel 2.1: Overzicht van Python *keywords*

and	as	assert	break	class	continue
def	del	elif	else	except	finally
for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass
raise	return	try	while	with	yield
False	None	True			

Opdracht 2.4

Welke van de onderstaande instructies zijn toegelaten in Python. Bij uitdrukkingen die niet geldig zijn, geef de reden(en).

- `define = "#ELEM = 7"`
- `global = 0b0101`
- `boolean_var = "True"`
- `local = 8`

- `True_or_Flase = 0`
- `yield = 8.0405e+6`

Controleer je antwoord door elk van de instructies in een Python console uit te voeren.

Link tussen een variabele(naam) en object kan wijzigen

De link tussen een naam en het object waar die naam naar verwijst kan wijzigen, zoals het volgende codefragment illustreert.

```
1 a = 15.1
2 b = 10.1
3 print("De waarden van a en b")
4 print(a)
5 print(b)
6 a = "Hallo"
7 print("De nieuwe waarde van a")
8 print(a)
```

Wanneer dit codefragment wordt uitgevoerd, wordt de volgende output bekomen:

```
De waarden van a en b
15.1
10.1
De nieuwe waarde van a
Hallo
```

De variabele `a` verwijst eerst naar het **`float`** object met waarde `15.1` en nadien naar het **`str`** object met waarde `Hallo`.

De functie `id` en aliasing

Telkens wanneer in Python een object aangemaakt wordt, krijgt dit object een identificatienummer (ID). Dit ID kan je terugkrijgen met de ingebouwde functie **`id()`**. Zolang een object bestaat in het werkgeheugen blijft zijn ID ongewijzigd en bovendien uniek[2].

In het volgende voorbeeld wordt een variabele `a` aangemaakt die verwijst naar een *integer* object met als waarde `7`. Het type wordt nagegaan met **`type(a)`** en dit stemt inderdaad overeen met **`int`** (*integer*). Het object ID wordt nagegaan met **`id(a)`** en is in dit geval `10455232`.

```
>>> a = 7
>>> type(a)
int
>>> id(a)
10455232
```

De volgende instructies genereren een *float*-object met waarde `2.5` en variabele `b` die naar dit object verwijst.

[2] Je kan zo'n identificatienummer principieel vergelijken met het rijksregisternummer van een persoon.

```
>>> b = 2.5
>>> type(b)
float
>>> id(b)
140672328648360
```

Merk op dat de IDs van beide objecten verschillend zijn. Zet nu het vorige voort met

```
>>> a = b
>>> type(a)
float
>>> id(a)
140672328648360
```

De variabelen `a` en `b` verwijzen nu naar hetzelfde object (de `id` is immers dezelfde). Dit principe heet **aliasing** en treedt op wanneer een variabelenaam gebruikt wordt aan de rechter zijde van de toekenningsoperator zoals in het toekenningsstatement `a = b`.

Het toekenningsstatement `a = b` leidt er bovendien toe dat `a` nu verwijst naar het *float*-object met als inhoud `2.5`, immers `a` is nu van het type *float* met hetzelfde ID als `b`. Er is nu geen verwijzing meer naar het oorspronkelijke *integer* object met als inhoud `7` dat niet meer beschikbaar is voor het programma en automatisch uit het werkgeheugen van de computer verwijderd zal worden. Dit automatisch vrijmaken van geheugenplaatsen waarnaar geen variabelen meer verwijzen heet **garbage collection**.

Belangrijke aanvulling op theorieslides!

Bekijk de theorieslides voor een inleiding tot variabelen, het toestandsdiagram en de *name space*.

Opdracht 2.5

- Voer de volgende instructies uit en bekijk de IDs van de aangemaakte variabelen.

```
>>> a = 5
>>> b = 6
```

- Zet het vorige voort met de volgende instructie en bekijk de inhoud en de IDs van de variabelen. Zoek zelf een verklaring voor de inhoud van de objecten en hun IDs.

```
>>> a = b
```

- Zet het vorige voort met de volgende instructie en bekijk opnieuw de inhoud en de IDs van de variabelen. Zoek zelf een verklaring voor de inhoud van de objecten en hun IDs.

```
>>> b = 10
```

Opdracht 2.6

Voer de volgende instructies in in een console.

```
>>> dier = "poes"
>>> meubel = "stoel"
>>> getal = 4
>>> dier = meubel
>>> meubel = getal
>>> getal = dier
```

- Wat zijn de waarden van de variabelen `dier`, `meubel` en `getal` geworden zijn.
- Verklaar waarom er niet meer naar het object `poes` verwezen wordt.

Opdracht 2.7

1. Creëer een variabele `zin` die verwijst naar de *string*
   `"We behandelen het eerste hoofdstuk."`
2. Creëer een variabele met naam `getal` die verwijst naar de *float*
   `3.14159`
3. Creëer een variabele `tijdelijk`, die een alias is voor `zin`.
4. Update `zin` zodat het een alias wordt voor `getal`.
5. Update `getal` zodat het een alias wordt voor `tijdelijk`.
6. Bekijk de waarden van `zin` en `getal`, deze zouden omgewisseld moeten zijn.

Opmerking:
Het proces dat we in deze opdracht expliciet uitgevoerd hebben heet ***swapping*** (van het Engels *to swap*: wisselen). Bij het "swappen" wordt de inhoud van twee variabelen verwisseld. Dit gebeurt typisch d.m.v. een tijdelijke variabele.

2.2 Expressies en statements

2.2.1 Expressies

In de voorgaande voorbeelden hebben we beschreven dat waarden in Python worden ingekapseld in objecten en dat variabelen verwijzingen zijn naar deze objecten. Op deze objecten kunnen vervolgens bewerkingen worden uitgevoerd (bv. twee *floats* optellen). Het resultaat van deze bewerkingen is een nieuw object. Dergelijke instructies worden in Python expressies genoemd. Formeel wordt een **expressie** gedefinieerd als: een combinatie van waarden, variabelen en operatoren.

Voorbeelden van expressies zijn:

```
53.2 + 12.5  % resultaat is de som van 53.2 en 12.5
9 * 6        % resultaat is product van 9 en 6

a = 3        % dit is een toekenningsstatement (geen expressie)
5 + a        % resultaat is som van 5 en 3 (3 is de waarde
             % van het object waarnaar a verwijst)
```

We beschouwen de expressie `53.2 + 12.5` nu in detail:

- Het plusteken (`+`) geeft aan dat de getallen `53.2` en `12.5` moeten opgeteld worden en is een voorbeeld van een **operator**[3]. In de expressie `9 * 6` is `*` de (vermenigvuldigings)operator[4].
- De waarden waarop de operator inwerkt, in dit voorbeeld `53.2` en `12.5`, zijn de **operanden**.
- Wanneer de expressie `53.2 + 12.5` uitgevoerd wordt door de interpreter, dan zeggen we dat deze **geëvalueerd** wordt.
- Wanneer een expressie geëvalueerd wordt, dan wordt een nieuw object gegenereerd. Men zegt dat dit object **geretourneerd** wordt door de expressie.

De expressie `5 + a` bevat ook een variabele. Wanneer deze expressie geëvalueerd wordt, zal het volgende gebeuren:

- De variabele `a` wordt automatisch vervangen door het object waarnaar ze verwijst.
- De expressie wordt verder geëvalueerd met deze waarde.

Wanneer een expressie wordt ingevoerd in een Python console, dan wordt de waarde van het geretourneerde object onmiddellijk getoond.

```
>>> 47 + 53          # expressie: combinatie getallen en operator
100                  # resultaat van de expressie 47 + 53
```

```
>>> x = 4            # statement
>>> x**2 + x - 20    # expressie (de operator ** is de machtoperator)
0                    # resultaat van de expressie
```

Operatoren voor strings: concatenatie (+) en duplicatie *

In de voorgaande voorbeelden waren de operanden steeds numeriek (type `float` of `int`). Bepaalde operatoren kunnen echter ook inwerken op strings. Twee belangrijke voorbeelden zijn de operatoren `+` en `*`.

Hieronder wordt de expressie `"mooi" + "weer"` geëvalueerd in een Python console.

[3] Ook in de wiskunde noemt men dit een operator. Meer precies is `+` een *binaire* operator. Men noemt deze operator binair omdat hij inwerkt op *twee* waarden, de operanden.

[4] In de volgende sectie volgt een overzicht van alle mogelijke operatoren in Python

```
>>> "mooi" + "weer"  # expressie
mooiweer             # resultaat van de expressie
```

De oorspronkelijke strings `"mooi"` en `"weer"` worden in dit voorbeeld **geconcateneerd** (samengevoegd) door de `+`. Deze expressie retourneert dus de nieuwe string `"mooiweer"`.

Wanneer de `*` operator wordt toegepast in een expressie waarin één van de operanden een *string* is en de andere een *integer*, dan wordt de string gedupliceerd.

```
>>> "mooi" * 5                  # expressie
mooimooimooimooimooi            # resultaat van de expressie

>>> 3 * "hallo"
hallohallohallo
```

2.2.2 Statements

Een **statement** is een instructie die de Python interpreter kan uitvoeren, maar retourneert (in tegenstelling tot een expressie) GEEN waarde. Een belangrijk type statement is het **toekenningsstatement** zoals bijvoorbeeld `a = 5`.

Ook het resultaat van een expressie kan onmiddellijk worden toegekend aan een nieuwe variabele. Dit geheel noemt men ook een toekenningsstatement. Voorbeelden van dergelijke statements zijn:

```
>>> a = 5
>>> b = 2 + a * 3       % voorrangsregels cfr. wiskunde (zie verder)
>>> woord = "hallo"
>>> woorden = woord * a
```

De waarden van de objecten waarnaar `b` en `woorden` verwijzen kunnen bekeken worden met de print functie.

```
>>> print(b)
17
>>> print(woorden)
hallohallohallohallohallohallo
```

Het print statement

Wanneer de functie **`print`** gebruikt wordt in een instructie (om de waarde van een variabele op het scherm te printen), dan spreekt men van een print statement. De print functie retourneert zelf niets[5] [6], maar toont enkel een waarde op het scherm.

[5] Strikt genomen retourneert elke functie, dus ook de functie **`print`**, een object. De functie **`print`** retourneert een object van het (gegevens)type `NoneType`. Het `NoneType` object vormt een manier om instructies die niets wezenlijks retourneren toch een waarde te laten retourneren. Het toekenningsstatement **`resultaat = print("Hallo")`** kan dus geïnterpreteerd worden door de interpreter. De variabele **`resultaat`** zal dan verwijzen naar een `NoneType`

[6] Omdat de functie print een waarde retourneert, is de oproep van de print functie strikt genomen een expressie! Er

```
>>> print(56)          # statement: print het getal 56 naar het scherm
56                     # resultaat van print(56)

>>> print("Faculteit: FBW")    # statement
Faculteit: FBW                 # resultaat van print("Faculteit: FBW")
```

```
>>> mijn_int = 5
>>> print(mijn_int)
5
>>> print("De getalwaarde is:", mijn_int)  # een print statement dat twee
De getalwaarde is: 5                       # objecten op het scherm toont (zie verd
```

Opdracht 2.8

Welke van de onderstaande instructies zijn *statements* en welke zijn *expressies*. Controleer je antwoord door elk van de instructies in een Python console uit te voeren. Bemerk dat sommige instructies gescheiden zijn door een punt-komma (;) om ze op één regel te krijgen.

- `y = "hahaha"+ "hihihi"`
- `5 * (1 + 1 / (1 + 8))**3`
- `x = 2; y = 3; x*y`
- `abs(-6.249)`
- `len("Hoe lang is deze zin?")`
- `m = 66.0; g = 9.81; print("F =", m*g, "N")`

Opdracht 2.9

Bekijk de volgende instructies en tracht te verklaren waarom een foutmelding optreedt.

```
>>> x = 2
>>> print(x + 5)          # expressie x + 5 wordt geevalueerd en
7                         # resultaat wordt argument van print
>>> print(y = x + 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'y' is an invalid keyword argument for this function
```

2.3 Operatoren en operanden

In de voorgaande secties werden reeds een aantal operatoren besproken die kunnen inwerken op floats, integers en strings. In deze sectie wordt dit overzicht vervolledigd.

wordt nog vaak verwezen naar print als een statement. Deze foute terminologie is nog een restant van de terminologie in Python 2, waar print wel een statement was.

2.3.1 Binaire operatoren voor numerieke operanden

Binaire operatoren zijn operatoren die inwerken op twee operanden. De meest gekende binaire operatie (of bewerking) is waarschijnlijk de optelling. In de bewerking

```
5 + 3
```

stelt het plusteken de operator (die de optelling symboliseert) voor en zijn `5` en `3` de operanden.

Voor de wiskundige bewerkingen optellen, aftrekken, vermenigvuldigen, delen en machtsverheffen zijn in Python operatoren voorzien. Tabel 2.2 geeft een overzicht van de binaire operatoren (aan de hand van hun symbolen) die in Python gedefiniëerd zijn voor numerieke (type **`float`** en **`int`**) operanden.

Tabel 2.2: Overzicht van binaire operatoren in Python (met operanden van het type *int* of *float*).

Symbool	Bewerking	Opmerking
`+`	optelling	
`-`	aftrekking	
`*`	vermenigvuldiging	
`/`	deling	
`//`	gehele deling (quotiënt)	`17 // 5` heeft als resultaat `3`
`%`	rest (modulus operator)	`17 % 5` heeft als resultaat `2`
`**`	machtsverheffing	`2 ** 3` lees je als twee tot de derde macht (2^3)

Het gegevenstype van het resultaat van een expressie die één van deze operatoren bevat hangt af van het type van de operanden. Hierna volgt een overzicht.

Minstens één operand van het type `float`
Indien minstens één van de operanden van het type **`float`** is, dan zal het object dat het voor elke operator ook het resultaat van het type **`float`** zijn.

```
>>> a = 3.1 + 2.5        # optelling van twee floats
>>> a
5.6
>>> type(a)
float                    # type is float
```

```
>>> b = 6.0 / 2          # deling van float en integer
>>> b
3.0
>>> type(b)
float                    # type is float
```

Beide operanden van type `int`

Indien beide operanden van het type **`int`** zijn, is het type van het resultaat afhankelijk van de gebruikte operator.

De operatoren `+`, `-`, `*`, `//`, `%` en `**` hebben als resultaat een object van het type **`int`**, voorbeelden:

```
>>> 8//3            # gehele deling
2                   # resultaat van 8//3
>>> type(8//3)      # controleer het type
int                 # type is inderdaad 'int'
```

```
>>> 8**3            # machtsverheffing met twee integers
512                 # resultaat 8**3
>>> type(8**3)      # controleer het type
int                 # type is inderdaad 'int'
```

De operator `/` heeft steeds een object van het type **`float`** als resultaat, bv.:

```
>>> 8/2             # (gewone)deling
4.0                 # resultaat (een kommagetal!)
>>> type(8/2)       # expliciete controle van het type
float               # type is inderdaad 'float'
```

Opdracht 2.10

Stel dat `var_int = 42` en `var_float = 42.0`, voorspel de waarde en het type van de objecten die geretourneerd worden door de volgende expressies en controleer in een Python console.

- `var_int * 5`
- `var_int * 5.0`
- `var_float + 5.0`
- `var_int / 7`
- `var_float // 6.0`
- `var_float % 5`
- `var_int % 5`

2.3.2 Unaire operatoren voor numerieke operanden

Unaire operatoren werken in op één operand. Voorlopig beschouwen we uitsluitend de unaire min (-) en de unaire plus[7] (+).

[7]Dit is in Python ook een unaire operator, maar het gebruik ervan bij een numeriek operand heeft weinig nut.

```
>>> a = 5
>>> -a      # de unaire min veroorzaakt een tekenwissel
-5          # het gegevenstype blijft behouden
```

2.3.3 Binaire operatoren voor het type `str`

De bewerkingen die gedefiniëerd zijn voor strings in Python zijn concatenatie (operator `+`) en de duplicatie (operator `*`).

De **concatenatie** (operator `+`) werkt in op twee operanden van het type **`str`** en heeft als resultaat een object van het type **`str`** waarin de twee oorspronkelijke strings bij elkaar zijn gevoegd, zoals geïllustreerd wordt in het volgende voorbeeld:

```
>>> woord1 = "hallo"
>>> woord2 = "test"
>>> woord3 = woord1 + woord2  # concatenatie
>>> print(woord3)
hallotest
```

Duplicatie (operator `*`) werkt in op één operand van het type **`str`** en één operand van het type **`int`**. Het resultaat is steeds een object van het type **`str`**.

```
>>> "hallo" * 3
hallohallohallo
>>> 4 * "test"
testtesttesttest
```

Opdracht 2.11

Stel dat `mijn_str = "Hello"` en `jouw_str = "World"`, ga na in een Python console wat het resultaat is van de volgende expressies:

- `mijn_str + jouw_str`
- `mijn_str - jouw_str`
- `mijn_str * jouw_str`
- `jouw_str + mijn_str`
- `mijn_str + ' ' + jouw_str`
- `mijn_str + ', hello ' + jouw_str + '!'`

Opdracht 2.12

Stel dat `mijn_str = "Hello"`, ga na in een Python console wat het resultaat is van de volgende expressies:

- `mijn_str * 3`
- `3 * mijn_str`
- `(mijn_str + ' ') * 3`
- `mijn_str + 3`
- `6 * '#'+ ' ' + mijn_str + ' ' + 6 * '#'`
- `'5' * 3`
- `5 * 3`

2.3.4 Voorrangsregels

Indien een expressie meerdere operatoren bevat, dan moet je rekening houden met voorrangsregels (zie Tabel 2.3) die de volgorde bepalen waarin de operatoren inwerken op hun operanden. Voor het afwijken van de standaardvolgorde, en soms voor de duidelijkheid, worden haakjes gebruikt. Bewerkingen die in de tabel op gelijke hoogte staan, zoals optellen en aftrekken, zijn gelijkwaardig. Gelijkwaardige bewerkingen worden van links naar rechts uitgevoerd.

Tabel 2.3: Voorrangsregels van de operatoren en haakjes, geordend volgens prioriteit, van hoog naar laag.

`()`	haakjes
`**`	machtsverheffing
`+x, -x`	unaire min of plus
`*, /, %, //`	vermenigvuldiging, deling, rest, gehele deling
`+, -`	optellen, aftrekken

Hierna volgen enkele voorbeelden

```
>>> 3.2 * 5 + 2
18.0
>>> 3.2 * (5 + 2)
22.400000000000002   # merk ook de afronding tgv float op
>>> 5**2*3
75
>>> -5**2*3
-75
>>> 5 * 3 // 2
7
```

Opdracht 2.13

Gegeven de volgende *expressie*: `30 - 3 ** 2 + 8 // 3 ** 2 * 10`

(a) Wat is het resultaat van de *expressie*? Controleer je antwoord in een Python console.

(b) Rekeninghoudend met de voorrangsregels, geef dezelfde bewerking met haakjes weer om de uitdrukking te verduidelijken. Controleer dat je hetzelfde resultaat bekomt als in (a).

Opdracht 2.14

Voorspel het resultaat en controleer dit in een Python console.

(a) `2**2**3`

(b) `2**(2**3)`

(c) `(2**2)**3`

Waarom hebben twee van de bovenstaande *expressies* dezelfde uitkomst?
Herschrijf *expressie* (c) met één machtsverheffingsoperator (`**`) en één vermenigvuldigingsoperator (`*`).

Opdracht 2.15

Je wenst het volgende uit te rekenen: $6^{3/2}$

1. Welke van de onderstaande *expressies* levert het juiste resultaat op?
2. Welke twee *expressies* zijn gelijkwaardig?

(a) `6**3/2`

(b) `(6**3)/2`

(c) `6**(3/2)`

(d) `6**(3//2)`

2.4 Eerste codescripts – programma's

Tot nu toe hebben we alle instructies rechtstreeks via een Python console (interactief) ingegeven. In dit geval zal Python de ingevoerde instructies onmiddellijk uitvoeren en het resultaat tonen in de console. Dit is handig als je een reeks instructies interactief wil uitvoeren of indien je Python gebruikt als een uitgebreide rekenmachine.

In vele gevallen wens je meerdere instructies te bundelen in een broncode-bestand om deze (eventueel ook op een later moment) sequentiëel te laten uitvoeren door de interpreter. In deze gevallen kan je de instructies onder elkaar plaatsen in een tekstfile (en deze opslaan met de extensie `.py`). Een dergelijke file noemt men een **script**. We beschouwen ter illustratie het volgende probleem.

Een elektrisch verwarmingselement wordt aangesloten op een gelijkspanning van $U = 120.0V$. In de kring meet je een stroomsterkte van $I = 4.8\,A$. Bereken de weerstand van het verwarmingselement,

het vermogen en de hoeveelheid warmte-energie geproduceerd in $1h$. De weerstand R, het vermogen P en de warmte-energie Q worden gegeven door de volgende formules:

$$R = \frac{U}{I} \qquad P = U \cdot I \qquad Q = P \cdot \Delta t$$

Het onderstaande codefragment is een voorbeeld van een Python script, dat werd opgeslagen in de tekstfile `warmte_energie.py` en deze drie grootheden berekent:

Fragment 2.1: `warmte_energie.py`

```
1  U = 120.0   # spanning in V
2  I = 4.8     # stroomsterkte in A
3  dt = 1      # tijdsduur in uur (1 h = 3600 s)
4
5  R = U / I   # berekening van de weerstand in Ohm
6  print("Weerstand R =", R, "Ohm")
7
8  P = U * I   # berekening van het vermogen in W
9  print("Vermogen P =", P, "W")
10
11 Q = P * dt  * 3600   # warmte-energie in J
12 print("Warmte-energie Q =", Q, "J")
```

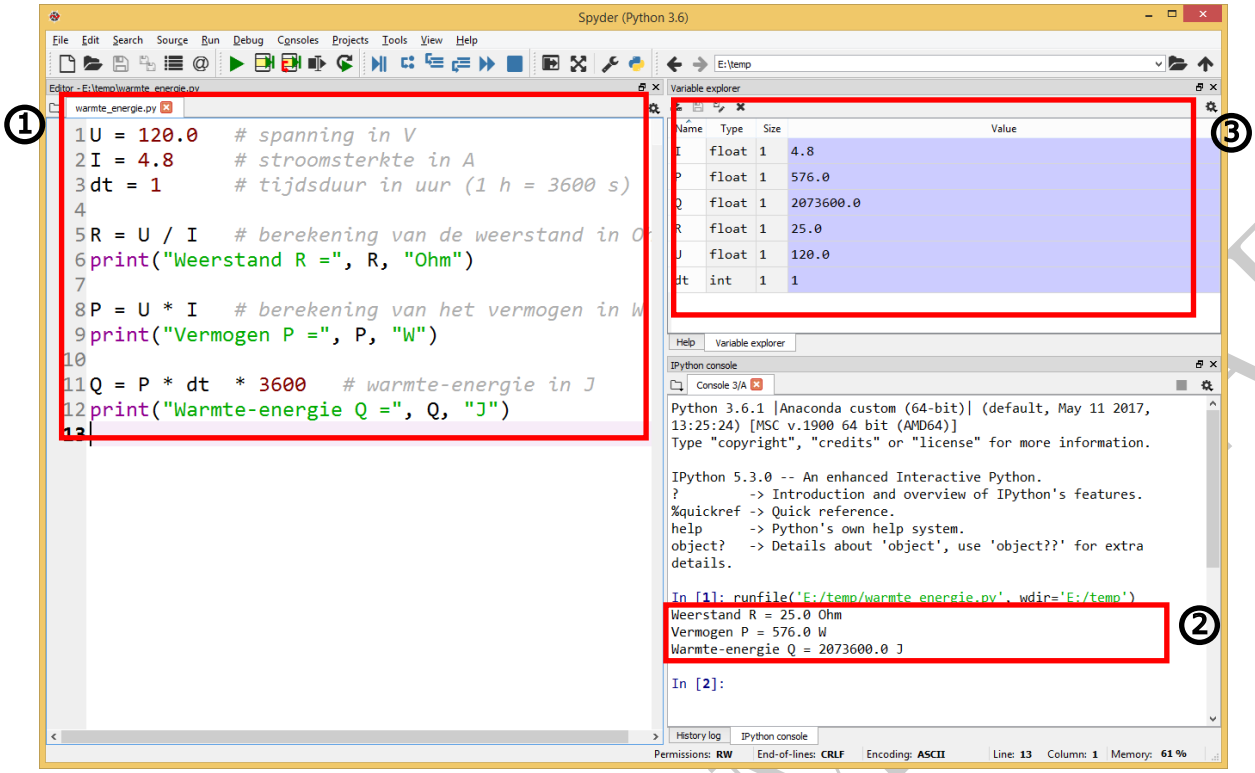
Opmerking

- De lijnnummers aan de linker kant van de code maken geen deel uit van de script (code).
- Open een nieuw bestand (ga in Spyder naar *File* > *New file...* of maak gebruik van de sneltoets *Ctrl+N*) en voer de bovenstaande code in dit bestand in. Sla dit bestand op in je werkmap onder de naam `warmte_energie.py`. Ga in Spyder naar *Run* > *Run* of maak gebruik van de functietoets F5 om dit script uit te voeren (alle instructies worden dan van boven naar onder uitgevoerd).

Het programma `warmte_energie.py` levert volgende output:

```
Weerstand R = 25.0 Ohm
Vermogen P = 576.0 W
Warmte-energie Q = 2073600.0 J
```

Figuur 2.1 toont hoe dit script en zijn output er uit zien nadat het werd uitgevoerd in Spyder.



Figuur 2.1: Screenshot van Spyder IDE na uitvoeren van het script `warmte_energie.py`. (1) de script file; (2) de console, met de output van het script; en (3) de variable explorer die een overzicht geeft van de variabelen en hun waarden die aanwezig zijn in het werkgeheugen nadat het script werd uitgevoerd.

Opdracht 2.16 (`weerstand_serie_parallel.py`)

Schrijf een script met als naam `weerstand_serie_parrallel.py` dat de vervangingsweerstand (substitutieweerstand) berekent van twee weerstanden $R_1 = 3.0\,\Omega$ en $R_2 = 9.0\,\Omega$ in het geval van serie- (R_s) en parallelschakeling (R_p). Generieke formules voor het berekenen van de vervangingsweerstand van N serie- of parallelgeschakelde weerstanden zijn de volgende:

$$R_s = \sum_{i=1}^{N} R_i \qquad R_p = \left(\sum_{i=1}^{N} R_i^{-1} \right)^{-1}$$

Wanneer je dit script uitvoert moet de volgende informatie het scherm verschijnen:

```
Waarden elektrische weerstand:
R1 = 3.0 Ohm
R2 = 9.0 Ohm
Vervangingsweerstanden:
Rs = 12.0 Ohm  (seriesschakeling)
Rp = 2.25 Ohm  (parallelschakeling)
```

Voor je begint.

- Maak een map `informaticaI` aan op je persoonlijke H-schijf (via Windows explorer).
- Maak in deze map een nieuw Python script `weerstand_serie_parallel.py` aan (in Spyder

ga je naar `File > New File`) en werk daarin verder.

- Voor de berekening van de vervangingsweerstanden, werden de generieke formules gegeven voor N weerstanden. In deze opdracht hebben we maar twee weerstanden, schrijf deze formules expliciet uit voor $N = 2$.

$$R_s = \ldots\ldots\ldots\ldots\ldots, \qquad R_p = \ldots\ldots\ldots\ldots\ldots$$

- Gebruik Fragment 2.1 als leidraad voor deze opdracht.

Opdracht 2.17 (`massacentrum.py`)

Schrijf een script met als naam `massacentrum.py` dat de positie x_C van het massacentrum berekent voor drie (punt)massa's $m_1 = 5.3\ kg$, $m_2 = 2.6\ kg$ en $m_3 = 1.7\ kg$ die zich respectievelijk op posities $x_1 = 0.9\ m$, $x_2 = 1.3\ m$ en $x_3 = -0.2\ m$ langs een as bevinden. De generieke formule voor het berekenen van de positie van het massacentrum van (punt)massa's (m_i) op posities (x_i) voor $i = 1, \cdots, N$ langs een as is:

$$x_C = \frac{\sum_{i=1}^{N} m_i x_i}{\sum_{i=1}^{N} m_i}$$

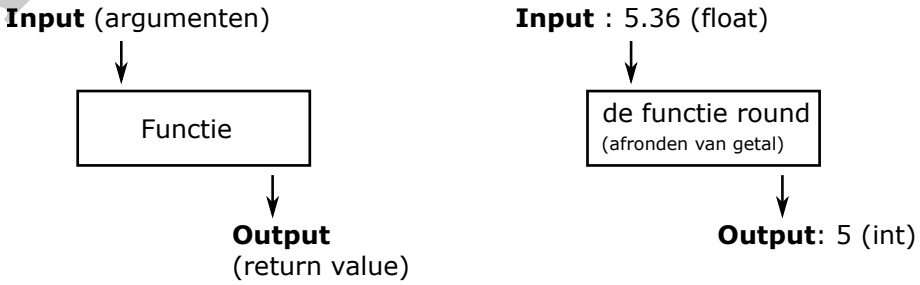
Wanneer je dit script uitvoert moet de volgende informatie het scherm verschijnen:

```
xC = 0.81 m
```

2.5 Gebruik van functies

2.5.1 Functies oproepen in Python

Een Python functie is een benoemde sequentie van instructies (een stuk broncode) dat bij naam aangeroepen kan worden in een programma en dat een specifieke taak uitvoert. Een functie kan gezien worden als een structuur die **input** (bijvoorbeeld een `float` of `str`) aanvaardt en verwerkt tot **output**. Figuur 2.2 illustreert dit concept.



Figuur 2.2: (links) Schematische voorstelling van een functie als een structuur die input vertaalt naar output. (rechts) Toepassing van dit schema voor de functie `round` die een decimaal (type `float`) getal afrondt tot een geheel getal (type `int`).

In Python wordt de functie `round` als volgt opgeroepen:

```
>>> round(5.36)
5
```

Wanneer een functie als onderdeel van een expressie gebruikt wordt, dan zegt men dat deze functie wordt **opgeroepen** (*function-call*). Bij een functie-oproep zijn drie aspecten van een fucntie belangrijk. Dit zijn de naam, input en output. Deze worden in Python als volgt benoemd.

- Elke functie heeft een **naam**, in het voorbeeld is `round` de naam van de functie. Een functie wordt steeds opgeroepen op basis van zijn naam.
- De objecten (of de variabelen die ernaar verwijzen) die de input vormen van de functie, zoals de `float` `5.36` in het voorbeeld, noemen we de **argumenten** van de functie. De argumenten worden bij de functie-oproep tussen ronde haakjes `()` geplaatst na de naam. Indien er meerdere argumenten zijn, worden deze gescheiden door komma's.
- Het object (steeds één object) dat de output vormt noemen we de **return value**.

In het voorbeeld wordt de functie `round` opgeroepen met als argument 5.36 en is de return value 5.

Een ander voorbeeld van een functie is `abs`. Van een numeriek argument (type `float` of `int`) zal deze functie de absolute waarde berekenen en retourneren, waarbij het type van de *return value* hetzelfde is als dat van het argument.

```
>>> abs(-5.36)
5.36
```

Functies met meerdere argumenten

De meeste functies in Python kunnen meerdere argumenten aanvaarden. Een eenvoudig voorbeeld is de functie `min`. Deze functie bepaalt de kleinste waarde van zijn argumenten en geeft deze terug als return value. Indien er **meerdere argumenten** worden meegegeven, worden deze steeds **gescheiden door komma's**.

```
>>> min(3.16, 4.23, 1.2)
1.2
```

MERK OP dat niet elke functie meerdere argumenten kan aanvaarden. Indien het aantal meegegeven argumenten niet overeenstemt met het verwachte aantal argumenten, wordt een foutmelding opgeworpen.

```
>>> abs(-5, -3)
TypeError: abs() takes exactly one argument (2 given)
```

Gebruik van variabelen bij de functie-oproep.

In de voorgaande voorbeelden zijn de argumenten bij de functie-oproep waarden. De argumenten kunnen ook variabelen zijn en de return value kan worden toegekend aan een nieuwe variabele. Dit wordt geïllustreerd in het volgende voorbeeld:

```
>>> a = 5.36
>>> b = round(a)
>>> a
5.36              # waarde van a blijft ongewijzigd
>>> b
5                 # variabele b verijst naar nieuw object met waarde 5
```

Merk op dat de waarde van de variabele die als argument wordt gebruikt niet wijzigt! Dit is bij zeer veel functies in Python het geval.

Een functie-oproep met meerdere argumenten laat eenzelfde manier van werken toe.

```
>>> a = 5.36
>>> b = 3.2
>>> c = min(a, b)
>>> c
3.2
```

In zijn meest algemene vorm is de syntaxis van een functie-oproep de volgende:

De functie-oproep:

```
returnValue = functieNaam(argument1, argument2, argument3, ...)
```

Nesten (of samenstellen) van functie-oproepen

In de wiskunde kan men functies samenstellen. Wenst men bijvoorbeeld aan te geven dat de sinus van de absolute waarde van de variabele x moet bepaald worden, dan kan men dit noteren als $\sin(|x|)$. Eerst zal dan de absolute waarde bepaald worden van x en van deze absolute waarde wordt vervolgens de sinus berekend. Ook in Python kan men deze functies gaan samenstellen. Men noemt dit het nesten van functie-oproepen.

Als voorbeeld bepalen we $\min(|-5|, |-3|)$:

```
>>> min(abs(-5), abs(-3))
3
```

Vergelijk dit met $|\min(-5, -3)|$:

```
>>> abs(min(-5, -3))
5
```

Opdracht 2.18 (`gebruikfuncties.py`)

Schrijf een script met als naam `gebruikfuncties.py` waarin het volgende gebeurt.

- Maak twee variabelen a en b aan die verwijzen naar twee `float` objecten met waarden die je zelf mag kiezen.
- Bepaal het minimum van $(a\sqrt{|b|}+1)^2$ en $(a^2+1)(|b|+1)$ en ken het resultaat toe aan de variabele c.
- Breng de volgende boodschap op het scherm: `Het minimum is ....`

Test je code: voor $a = 1.2$ en $b = -2.3$ is het minimum bij benadering 7.95178.

Voor je begint.

- Lees eerst sectie 2.5.1

Ter info (niet gebruiken): Als gevolg van de ongelijkheid van Cauchy-Schwarz zou het minimum steeds de eerste van deze twee uitdrukkingen moeten zijn.

2.5.2 De function signature, parameters, positionele en keyword argumenten

Python bevat een beperkt aantal ingebouwde (*built-in*) functies (68 in versie 3.4.9)[8]. De modules die in een basis Anaconda installatie beschikbaar zijn verhogen het aantal beschikbare functies tot enkele duizenden. Het is dus, zelfs voor een ervaren programmeur, onmogelijk om alle functies te kennen. Bij het eerste gebruik van een functie is het dan ook belangrijk de functiedocumentatie te raadplegen. In de eerste plaats kan dit door gebruik te maken van de functie `help`. De `help` functie kan eenvoudig worden opgeroepen in de console met als argument de naam van de functie.

`help`(*functieNaam*)

Voor de functie `round` bekomen we de volgende informatie:

```
>>> help(round)
Help on built-in function round in module builtins:

round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

Uit de woordelijke beschrijving is het duidelijk dat de functie `round` een getal afrondt. Daarnaast is vooral de volgende regel belangrijk:

[8] De volledige lijst van de ingebouwde functies (Python 3) vind je terug op:
`https://docs.python.org/3.4/library/functions.html`

```
round(number[, ndigits]) -> number
```

Deze regel noemt men vaak de **function signature** omdat hij aangeeft welke (types) argumenten een functie kan aanvaarden en wat het type van de return value zal zijn[9]. We ontleden deze regel nu stap-voor-stap.

De parameters van `round`.
De functie `round` heeft twee **parameters**: `number` en `ndigits`. Dit zijn twee namen die tijdelijk (en automatisch) worden toegekend aan de argumenten tijdens de functie-oproep. Merk op dat `, ndigits` tussen vierkante haken (`[]`) staat, maar dit negeren we voorlopig. Dit wil zeggen dat de functie `round` twee argumenten kan aanvaarden:

- `number`: het getal (`float` of `int`) dat men wenst af te ronden.
- `ndigits`: (staat voor number of digits) het aantal decimalen (cijfers na de komma), een `int`.

De volgorde waarin deze parameters voorkomen bepaalt ook de volgorde van de argumenten. De link tussen parameters en argumenten wordt gelegd op basis van positie. We noemen deze argumenten daarom **positionele argumenten**:

```
>>> round(5.32434, 2)
5.32
```

Een alternatieve manier om argumenten door te geven aan een functie maakt gebruik van **keyword argumenten**. Dit gebeurt via de toekenning `parameter = waarde`.

```
>>> round(number = 5.32434, ndigits = 2)
5.32
```

Het resultaat is hetzelfde, maar de functie-oproep wordt iets langer. Een voordeel is dat de broncode mogelijks duidelijker wordt omdat de programmeur duidelijk aangeeft wat de functie is van de argumenten. Bovendien is het mogelijk om ook de positie van de argumenten vrij te kiezen.

```
>>> round(ndigits = 2, number = 5.32434)
5.32
```

De optionele argumenten van `round`.
`, ndigits` staat tussen vierkante haken om aan te geven dat dit een **optioneel argument** is. Dat wil zeggen dat er een standaard (default) waarde wordt toegekend aan de parameter `ndigits` indien er geen waarde wordt voorzien bij de functie-oproep. In de documentatie lees je dat wanneer maar één argument wordt meeggeven de return value een `int` is. Merk het subtiele verschil op wanneer `ndigits = 0` wordt meegegeven (de return value is nu een `float`).

[9] Merk op dat de term *function signature* in Python iets soepeler dient te worden geïnterpreteerd dan in bv. Java of C. In deze talen ligt het gegevenstype van de argumenten en de return value vast. In Python is dit niet noodzakelijk het geval. De functie `abs` kan bijvoorbeeld zowel een argument van het type `int` als van het type `float` aanvaarden.

```
>>> round(5.32434)
5
>>> round(5.32434, ndigits = 0)
5.0
>>> round(5.32434, ndigits = None)
5
```

Het voorgaande voorbeeld geeft aan dat de default waarde voor `ndigits` niet 0 is, maar `None`[10]. Om aan te geven dat een parameter een default waarde heeft, wordt, in plaats van `[]`, ook gebruik gemaakt van de volgende notatie:

```
round(number, ndigits = None)
```

Opdracht 2.19 (`waterdebiet.py`)

De functies `pow()` en `round()` zijn ingebouwde functies. Schrijf een script met de naam `waterdebiet.py`, waarin deze functies gebruikt worden, dat het debiet door een open kanaal berekent. In een open kanaal wordt het waterdebiet Q (in $m^3 s^{-1}$) berekend met de fomule:

$$Q = \frac{A^{5/3} \cdot S_0^{1/2}}{n \cdot P^{2/3}}$$

waarbij $A = 18.0\ m^2$ de dwarsdoorsnede is van het water, $S_0 = 0.001$ de relatieve helling, $n = 0.015$ een getal afhankelijk van het soort kanaal en de wand, $P = 13.94\ m$ de zgn. natte omtrek.
Bereken het waterdebiet Q tot op 1 cijfer na de komma en geef je resultaat weer op het scherm gevolgd door zijn eenheid. Wanneer je dit script uitvoert moet de volgende informatie op het scherm verschijnen:

```
Q = 45.0 m^3/s
```

Uitbreiding:

Als alternatief op de ingebouwde functie `pow()` kan je uiteraard ook de machtverheffingsoperator (`**`) gebruiken. Kopieer en herschrijf de expressie voor het berekenen van `Q` door gebruik te maken van `**` i.p.v. `pow` voor het berekenen van de machten. Let op de voorrangsregels en gebruik haakjes (cf. Opdracht 2.15)! Je resultaat zou identiek moeten zijn.

Voor je begint.

- Lees Sectie 1.5 en Subsectie 1.5.1 aandachtig.
- Bekijk de specificaties van de functies `pow()` en `round()`. Je kan dit doen door `help(pow)` in te typen in de console. Alternatief kan je `pow` intypen in de editor of de console, met de cursor op de naam gaan staan en *Ctrl+i* drukken. In een helpvenster zou je informatie moeten te zien krijgen over deze functie en hoe je deze moet gebruiken. Informatie over een functie kan je ook online zoeken op bv. `www.google.com` met de zoekterm *python 3*

[10] `None` is in Python een bijzonder object dat je het makkelijkst kan interpreteren als *niets*.

pow.

Opdracht 2.20 (`dieselmotor.py`)

In een dieselmotor ontsteekt de brandstofnevel door temperatuurstijging als gevolg van samendrukking door een zuiger in een cilinder. Zij V_i het initieel volume van de cilinder vóór de samendrukking bij een temperatuur $T_i = 293\ K$, en zij V_f het finaal volume van de cilinder na de samendrukking bij een temperatuur $T_f = 773\ K$ is. Indien je voor zo'n proces de volgende uitdrukking mag gebruiken: $T_i V_i^{\gamma-1} = T_f V_f^{\gamma-1}$, met $\gamma = 1.4$, bereken dan de verhouding V_i/V_f.

- Leid zelf een uitdrukking af voor het berekenen van verhouding V_i/V_f en schrijf een script dat deze verhouding berekent en het resultaat op het scherm toont afgerond zonder cijfers na de komma. Wanneer je dit script uitvoert moet de volgende informatie op het scherm verschijnen:

```
Vi/Vf = 11
```

2.5.3 De functie `print`

De functie print maakt het mogelijk om in broncode aan te geven dat objecten of hun waarden op het scherm moeten worden weergegeven. Deze functie kan één argument aanvaarden:

```
>>> print("Hallo")
Hallo
```

of meerdere argumenten:

```
>>> print("een", "twee", "drie")     # drie argumenten (gescheiden door ,)
een twee drie
```

Indien meerdere argumenten (van het type **`str`**) worden meegegeven als argumenten, worden deze naast elkaar op het scherm getoond. Merk op dat automatisch een spatie wordt tussengevoegd als **separator**. Ook numerieke objecten kunnen deel uitmaken van de argumenten.

```
>>> print("Ik ben", 45, "jaar oud.") # drie argumenten (gescheiden door ,)
Ik ben 45 jaar oud.
```

Via de **`help`** bestuderen we de function signature van **`print`**.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Merk op dat de documentatie vrij beperkt is. `value` is de parameter is die de waarde(n) voorstelt die geprint zullen worden. Met ... geeft men aan dat het aantal *values* dat kan geprint worden variabel is (zoals uit de voorbeelden ook blijkt). Daarnaast zijn er de volgende parameters:

- `sep` met default `' '`, bepaalt wat de separator (het scheidingsteken) is tussen de values die geprint worden. De default waarde is een spatie, maar deze kan een **`str`** naar keuze zijn.
- `end` met default `'\n'`, is een **`str`** die geprint wordt nadat alle values zijn geprint. De default value `'\n'` stelt een nieuwelijnkarakter voor. Dit wil zeggen dat, indien meerdere print statements na elkaar worden uitgevoerd, elk statement op een nieuwe regel zal starten.
- **`file`** met als default `sys.stdout` bepaalt de *stream* waarnaar wordt geprint. De default waarde stelt (meestal) het scherm voor, maar men kan deze ook vervangen door een file object (zie later) zodat de values worden opgeslagen in een bestand.

Het onderstaande codefragment illustreert een aantal mogelijkheden

```
1 a = "een"
2 b = "twee"
3 c = "drie"
4 print(a, b, c)
5 print(a, b, c, sep = "x")
6 print("---------")
7 print(a, b, c, end = "xx")
8 print("---------")
```

De output van dit codefragment is:

```
1 een twee drie
2 eenxtweexdrie
3 ---------
4 een twee driexx---------
```

Merk op dat in regel 2 van de output de woorden gescheiden worden door x'en (in plaats van spaties). Omdat bij de eerste drie print statements de optie `end` de default waarde `'\n'` heeft, wordt telkens een nieuwe regel gestart voor de volgende printopdracht. Bij het voorlaatste print statement wordt `end = "xx"`, dit heeft tot gevolg dat de karakters `xx` achteraan worden toegevoegd. Bovendien wordt geen nieuwe lijn gestart. Daarom wordt de streepjeslijn die volgt op dezelfde lijn toegevoegd.

Opdracht 2.21 (`printoefening.py`)
Geef de volgende instructies in in een script, maar vul de ... aan zodat de output van het script dezelfde is als de onderstaande.

```
a = "een"
b = "twee"
c = "drie"
print(a, b, c, sep = ......, end = .......)
print(a, b, c, sep = ......, end = .......)
print("-einde-")
```

```
een+twee+drie = een plus twee plus drie!
-einde-
```

Opdracht 2.22
Welke foutmelding treedt op bij het uitvoeren van de onderstaande instructie (en waarom)?

```
print("de" 3, "biggetjes")
```

Wat verschijnt op het scherm bij het uitvoeren van volgende instructie (en waarom)?

```
print("Het is\n mooi weer vandaag")
```

2.5.4 De functie `len`

Een string is een opeenvolging van karakters. Vaak is het handig om te kunnen bepalen uit hoeveel karakters een string bestaat. De functie **`len`** biedt deze functionaliteit aan[11]. Ook spaties `' '` en nieuwelijnkarakters `'\n'` zijn karakters en dragen dus bij tot de lengte.

```
>>> len("abc")  # lengte van de string "abc"
3
```

[11] Later zullen we zien dat **`len`** ook de kan gebruikt worden om de lengte van andere types te bepalen zoals **`list`**s.

```
>>> len("Wie niet sterk is moet slim zijn.")  # lengte van de string
33
```

Opdracht 2.23

Bepaal (eerst zonder Python) het resultaat van de volgende expressies:

- `len("het is\nmaandag")`
- `len("abc") * 2 * "abc"`

2.6 Python modules

2.6.1 Modules in Python

Naast een beperkt aantal *built-in* functies bevat de *Python Standard Library*[12] een groot aantal functies die worden aangeboden via **modules**. Een module kan (voorlopig) beschouwd worden als een verzameling van functiedefinities die door de programmeur kunnen gebruikt worden. Vaak situeren de functies in een module zich rond eenzelfde topic. Zo zijn er bijvoorbeeld modules die voornamelijk (elementaire) wiskundige functies bevatten (zoals `math`) of functies om webpagina's in te laden (zoals `urllib`) of functies om te werken met datums en tijd (zoals `time`)[13]. Om de functionaliteiten van een module te kunnen gebruiken moet deze eerst geïmporteerd worden. Dit kan met een import statement. Dit is een statement gevormd door het keyword `import` gevolgd door de naam van de module.

Deze import-statements worden vaak bovenaan in het broncodebestand geplaatst. Expressies en statements die onder deze imports staan, kunnen gebruikmaken van deze modules. Hieronder volgen enkele voorbeelden van import-statements.

```
import math         # importeren math library
import urllib       # importeren urllib library
import time         # importeren time library
```

De functies die via deze modules beschikbaar zijn, kunnen worden opgeroepen door hun naam te laten voorafgaan door de naam van de module waartoe ze behoren, gevolgd door een punt. Zo kunnen de functies `sin` en `cos`, die behoren tot de `math` module en gebruikt worden om de sinus en cosinus van een hoek (in radialen) te berekenen, als volgt worden opgeroepen.

[12] De Python Standard Library is de bibliotheek die bij alle installaties van de Python interpreter wordt geïnstalleerd. De programmeur kan er dus van uit gaan dat alle gegevenstypes, functies en modules die tot de *standard library* behoren beschikbaar zijn op het systeem waarop de code zal worden uitgevoerd.

[13] De beschikbare *modules* zijn te vinden via de zgn. *Python Module Index* op `https://docs.python.org/3/py-modindex.html`

```
import math         # importeren math library
a = math.sin(0.6)   # oproepen van de functie sin, hoek 0.6 in radialen
b = math.cos(0.8)   # oproepen van de functie cos, hoek 0.8 in radialen
print(a, b)
```

Dit codefragment heeft als output:

```
0.5646424733950354 0.6967067093471654
```

Gebruik van modules.

- Importeren: `import` *naam_module*
- Functie-oproep: *naam_module*.*naam_functie*(*argument1*, *argument2*, ...)

2.6.2 Voorbeeld 1: de module `math`

De module math bevat een aantal (elementaire) wiskundige functies en constanten. Deze kunnen onderverdeel worden in:

- Functies uit de getallenleer, bv. `floor`, `ceil`
- Machtsfuncties en logaritmische functies, bv. `log`, `exp`
- Trigonometrische functies, bv. `sin`, `asin`
- Omzetting van hoeken, bv. `degrees`, `radians`
- Hyperbolische functies, bv. `cosh`, `acosh`
- Bijzondere functies, bv. `erf`, `gamma`
- Wiskundige constanten, bv. `pi`, `e`

De *signature* van elke van deze functies kan worden opgevraagd met de `help` functie. Hieronder wordt ter illustratie de sinus berekend van $\pi/3$ in een console.

```
>>> import math           # importeren math library
>>> math.sin( math.pi/3 ) # hoek als argument van sin staat in radialen
0.8660254037844386
```

Wens je de sinus van een hoek in graden te berekenen met de `sin` functie, dan moet je die eerst in radialen omzetten met de functie `radians`:

```
>>> import math
>>> math.sin( math.radians(60) )
0.8660254037844386
```

Een hoek in radialen omzetten naar een hoek in graden kan met de functie `degrees`:

```
>>> import math
>>> math.degrees( math.pi/3 )
59.99999999999999
```

Het volgende codefragment illustreert het gebruik van de `math` module in een uitgebreider script, waarin het geluidsniveau (in decibel dB) berekend wordt voor een gemeten geluidsintensiteit $I = 10^{-4} Wm^{-2}$. De referentie intensiteit $I_0 = 10^{-12} Wm^{-2}$. Het geluidsniveau in dB wordt gedefinieerd als

$$B = 10 \log \left(\frac{I}{I_0} \right)$$

Fragment 2.2: `geluidsniveau_db.py`

```
1 import math
2
3 I0 = 1.0e-12    # referentie intensiteit
4 I  = 1.0e-4     # gemeten intensiteit
5
6 B = 10 * math.log10(I / I0)   # geluidsniveau in dB
7
8 print("De geluidsintensiteit is", I, "W/m^2")
9 print("Het geluidsniveau is", B, "dB")
```

Dit levert het volgende resultaat

```
De geluidsintensiteit is 0.0001 W/m^2
Het geluidsniveau is 80.0 dB
```

Opdracht 2.24 (`brekingsindex_labo.py`)

Een monochromatische lichtstraal gaat over van lucht naar een bepaalde glassoort. De invalshoek in $\theta_1 = 43°$ en de brekingshoek is $\theta_2 = 26°$

$$n_{glas} = \frac{\sin(\theta_1)}{\sin(\theta_2)}$$

Schrijf een script met als naam `brekingsindex_labo.py`. Wanneer je dit script uitvoert moet de volgende info op het scherm verschijnen.

```
theta1 = 43.0 graden (invalshoek)
theta2 = 26.0 graden (brekingsshoek)
De brekingsindex van het glassoort:
n_glas = 1.5558
```

Voor je begint.

- Lees Sectie 2.6 aandachtig.
- Gebruik Codefragment 2.2 als voorbeeld.

2.6.3 Voorbeeld 2: de module `random`

De module `random` kan gebruikt worden om (pseudo-)random getallen te genereren uit een groot aantal distributies. Een voorbeeld is de simulatie van een worp met een dobbelsteen. Het resultaat van een worp is een geheel getal tussen 1 en 6 (type `int`), waarbij elk getal eenzelfde kans op voorkomen moet hebben. De functie `randint` uit de `random` module kan daarvoor gebruikt worden.

We importeren eerst de random module en bekijken de documentatie van `randint`.

```
>>> import random
>>> help(random.randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

Uit deze beschrijving blijkt dat deze methode twee argumenten aanvaardt en een integer retourneert.

```
>>> worp = random.randint(1, 6)
>>> print(worp)
5
```

2.7 Typeconversie

De waarden (strikt genomen noemen we dit *literals*) `5`, `5.0` en `"5"` stellen allen het getal vijf voor. De literal `5` zal door Python echter geïnterpreteerd worden als een integer waarde, `5.0` als een floating point waarde en `"5"` als een string. Men kan echter een object van het ene type omzetten in een (nieuw) object van een ander type met dezelfde waarde[14]. Dergelijke omzetting noemt men een **typeconversie**.

De functies `int`, `float` en `str` kunnen gebruikt worden om een object om te zetten naar respectievelijk een integer, float of string. Het oorspronkelijke object wordt daarbij niet gewijzigd. We illustreren het gebruik van deze functies hieronder.

[14] In sommige gevallen kan, bv. door afronding of een beperking in het aantal decimalen, de geconverteerde waarde minimaal afwijken van de oorspronkelijke

```
>>> a = "5"
>>> b = int(a)
>>> c = float(a)

>>> type(a)      # type van a blijft str
str
>>> type(b)      # type van b is int
int
>>> type(c)      # type van c is float
float

>>> b
5                # waarde van b is 5
>>> c
5.0              # waarde van c is 5.0
```

```
>>> d = 5**(1/2)
>>> str(d)
'2.23606797749979'    # aantal decimalen kan voorspeld
                      # worden, maar niet triviaal
```

Merk op dat het object dat moet omgezet worden eenvoudig wordt meegegeven als argument aan de functie die de typeconversie uitvoert.

Typeconversie is echter niet steeds mogelijk, zo kan een een string die geen geheel getal voorstelt niet geconverteerd worden naar een int.

```
>>> int("hallo")       # str stelt geen getal voor
ValueError: invalid literal for int() with base 10: 'hallo'

>>> int("5.1234")      # str stelt geen wel getal voor, maar niet geheel
ValueError: invalid literal for int() with base 10: '5.1234'
```

Anderzijds kan de waarde na conversie sterk afwijken van de oorspronkelijke waarde.

```
>>> int(5.123)   # decimaal deel valt weg
5
>>> int(5.789)   # decimaal deel valt weg
5                # volgt NIET de afrondingsregels
```

Opdracht 2.25 (`typeconversieOefening.py`)

Vul het onderstaande codefragment aan met de nodige functies voor typeconversie (gebruik tevens de functie **`round`**) zodat de weergegeven output correct op het scherm verschijnt.

```
temp_juni = 16.57
```

```
temp_juli = "18.66"

b1 = "Gemiddelde temperatuur juni: " + ..... + " graden."

b2 = "Na afronding is dit " + ..... + " graden."

b3 = "Gemiddelde temperatuur juli: " + ..... + " graden."

b4 = "Na afronding is dit " + ..... + " graden."

print(b1, b2, b3, b4, sep = "\n")
```

Output:

```
Gemiddelde temperatuur juni: 16.57 graden.
Na afronding is dit 16.6 graden.
Gemiddelde temperatuur juli: 18.66 graden.
Na afronding is dit 18.7 graden
```

2.8 Input van keyboard

Beschouw het volgende script waarin de snelheid van een voorwerp, uitgedrukt in kmh^{-1} wordt omgezet naar een snelheid in ms^{-1}.

```
snelheid_km_h = 120.0
snelheid_m_s = round(120 / 3.6, 2)
print("De snelheid in SI-eenheden is:", snelheid_m_s, "m/s")
```

Merk op dat de programmeur hier zelf de waarde 120.0 heeft gekozen. Zonder de broncode te wijzigen is het niet mogelijk om deze omzetting ook uit te voeren voor andere snelheden.

De functie `input` laat toe om, bij het uitvoeren van een script, de gebruiker om input te vragen. Deze input kan worden ingevoerd via het keyboard. Wanneer de input-functie wordt uitgevoerd zal de gebruiker een boodschap te zien krijgen op het scherm. De uitvoer wordt vervolgens onderbroken tot de gebruiker input heeft ingevoerd via het keyboard. Wanneer de gebruiker op de Enter-toets drukt wordt de uitvoer weer hervat.

Beschouw ter illustratie het volgende codefragment.

```
woord = input("Typ een woord: ")
print("Je voerde", woord, "in.")
```

Bij het uitvoeren van dit fragment verschijnt de volgende output.

```
Typ een woord in: Snackbar     # gebruiker voert Snackbar in en drukt Enter
Je voerde Snackbar in.         # print statement wordt uitgevoerd
```

Zoals het voorbeeld illustreert, heeft **input** de volgende signature:

- De naam van de functie: **input**.
- De parameter(s) prompt: boodschap (type **str**) die de gebruiker te zien krijgt bij de vraag om input.
- De return value: een **str** object met daarin de gebruikersinput.

Gebruikersinput die numeriek is van aard (getallen) wordt door **input**-functie ook als een *str* geretourneerd! Je moet deze waarden dus converteren naar *int* of *float* om er bv. bewerkingen mee uit te voeren.

```
>>> getal_str = input("Typ een getal: ")
Typ een getal: 56.3
>>> type(getal_str)
str                                    # getal_str is van type str
>>> getal_float = float(getal_str)     # converteer naar float
>>> type(getal_float)                  # getal_float is van type float
'float'
```

Het onderstaande script laat de gebruiker toe om zelf een snelheid in te geven in kmh^{-1}, zet deze om in ms^{-1} met twee cijfers na de komma en geeft deze weer op het scherm.

Fragment 2.3: snelheid_km_ms.py

```
1 # vraag naar de snelheid in km/h
2 snelheid_km_str = input("Geef een snelheid in km/h: ")
3 # converteer str naar float
4 snelheid_km_float = float(snelheid_km_str)
5 # zet om naar m/s en rond af tot 2 cijfers na de komma
6 snelheid_ms_float = round(snelheid_km_float / 3.6, 2)
7 # print het resuiltaat naar het scherm
8 print("De snelheid in SI-eenheden is:", snelheid_ms_float, "m/s")
```

Dit levert het volgende resultaat indien je 120 ingeeft als snelheid

```
Geef een snelheid in km/h: 120
De snelheid in SI-eenheden is: 33.33 m/s
```

De volgende twee (kortere) coderingsmogelijkheden hebben dezelfde output.

Fragment 2.4: `snelheid_km_ms.py`

```
snelheid_km_float = float(input("Geef een snelheid in km/h: "))
snelheid_ms_float = round(snelheid_km_float / 3.6, 2)
print("De snelheid in SI-eenheden is:", snelheid_ms_float, "m/s")
```

Fragment 2.5: `snelheid_km_ms.py`

```
snelheid_ms_float \
= round(float(input("Geef een snelheid in km/h: ")) / 3.6, 2)
print("De snelheid in SI-eenheden is:", snelheid_ms_float, "m/s")
```

De backslash (\) wordt gebruikt om een instructie op een volgende regel verder te zetten. Zo is het statement

```
x = 2 + 5
```

hetzelfde als

```
x \
= 2 \
+ 5
```

Opdracht 2.26 (`schuine_zijde.py`)
Implementeer een script dat achtereenvolgens de lengtes vraagt van de rechthoekszijden van een rechthoehige driehoek (op een interactieve manier m.b.v. de **`input`** functie), de lengte van de schuine zijde berekent en deze vervolgens op het scherm weergeeft. Gebruik **`sqrt`** en **`pow`** uit de `math`-module. Een mogelijke output is:

```
Geef de lengte van de rechthoekszijde 1: 3
Geef de lengte van de rechthoekszijde 2: 4
De lengte van de schuine zijde is: 5.0
```

Voor je begint.

- Lees Sectie 2.8 aandachtig.
- Gebruik codefragment 2.3 als voorbeeld.

2.9 Het gegevenstype bool en logische expressies

2.9.1 Proposities of beweringen

Een **propositie** of bewering is een declaratieve zin die **waar** (`True`) of **vals** (`False`) kan zijn. De volgende proposities zijn **waar**:

'Een schaap is een zoogdier.'

'Het getal 5 is groter dan het getal 2.'

'Na het uitvoeren van de toekenningsstatements `x = 2` *en* `y = 2` *zijn de waarden waar* `x` *en* `y` *naar verwijzen gelijk.'*

De volgende proposities zijn daarentegen **vals**:

'Een paard is een vis.'

'Het getal 7 is kleiner dan het getal 3.'

Het al dan niet waar (`True`) of vals (`False`) zijn van een propositie noemt men de **waarheidswaarde** van de propositie.

2.9.2 Het gegevenstype `bool`

Het aangeven van de waarheidswaarde van een expressie gebeurt in Python met het gegevenstype `bool` (afkomstig van *boolean*). Dit is een gegevenstype dat maar twee mogelijke waarden onderscheidt: `True` en `False` (let op de hoofdletters `T` en `F`). Expressies die als resultaat een object van het type `bool` hebben, noemt men **logische expressies**. Logische expressies bevatten (nagenoeg) altijd vergelijkingsoperatoren of relationele operatoren. Dit zijn operatoren die de relatie tussen twee objecten bekijken. Zo kan bekeken worden of de waarden van twee objecten gelijk zijn aan elkaar met de `==` operator of kan men bekijken of een waarde groter is dan een andere waarde met de `>` operator. Een compleet overzicht wordt gegeven in Tabel 2.4.

Tabel 2.4: Tabel met relationele operatoren.

`<`	kleiner dan
`>`	groter dan
`<=`	kleiner dan of gelijk aan
`>=`	groter dan of gelijk aan
`==`	gelijk aan
`!=`	niet gelijk aan (verschillend van)

Het onderstaande codefragment illustreert het gebruik van deze operatoren voor numerieke operanden (type `float` en `str`).

```
>>> 5 > 3
True
>>> 6.0 > 10.0
False
>>> 7 == 7
True
```

Het resultaat van een logische expressie kan men toekennen aan een nieuwe variabele.

```
>>> a = 5 >= 3
>>> print(a)
True
>>> type(a)
bool
```

De literals `True` en `False` zijn tevens Python keywords die kunnen gebruik worden om een object van het type `bool` aan te maken.

```
>>> a = True
>>> print(a)
True
>>> type(a)
bool
>>> b = False
>>> print(b)
False
```

Opdracht 2.27

Controleer in een Python console welke waarde geretourneerd wordt door de volgende logische expressies

- `3 > 2`
- `5+3 < 3-2`
- `'1' < 2`
- `1 + 2 == 3`
- `1.0 + 2.0 == 3.0`
- `1.1 + 2.2 == 3.3`

Kan je verklaren waarom `1.1 + 2.2 == 3.3` de waarde `False` teruggeeft? Tip: bekijk eens het resultaat van `1.1 + 2.2` afzonderlijk.

2.9.3 Samengestelde proposities

De voorgaande proposities waren steeds enkelvoudige proposities. Men kan deze proposities echter samenstellen door gebruik te maken van voegwoorden. Deze voegwoorden hebben in deze samengestelde proposities de rol van een operator, we noemen deze dan ook **booleaanse operatoren**.

Voegwoord EN (Engels `and`).
De propositie

$$\underbrace{\text{Een schaap is een zoogdier}}_{p} \ \textbf{en} \ \underbrace{\text{het getal 5 is groter dan het getal 2}}_{q}$$

is een voorbeeld van een samengestelde propositie, waarbij p en q werden samengesteld met het voegwoord **en**. We noemen dit de *conjunctie* van p en q. Omdat zowel p als q waar zijn, is ook de conjunctie p en q waar.

De conjunctie van twee proposities is waar indien beide samenstellende delen waar zijn. Zodra minstens één van deze delen vals is, is de conjunctie vals. Dit wordt samengevat in waarheidstabel 2.5.

Tabel 2.5: Waarheidstabel van de **`and`** operator.

p	q	p and q
True	True	True
True	False	False
False	True	False
False	False	False

De conjunctie wordt in Python geïmplementeerd door de **`and`** operator.

```
>>> a = (5 > 3) and (7 > 1)   # True and True -> True
>>> print(a)
True

>>> b = (5 == 7) and (9 < 11) # False and True -> False
>>> print(b)
False
```

Voegwoord OF (Engels `or`).
De propositie

$$\underbrace{\text{Een schaap is een vis}}_{p} \ \textbf{of} \ \underbrace{\text{het getal 5 is groter dan het getal 2}}_{q}$$

is een voorbeeld van een samengestelde propositie, waarbij p en q werden samengesteld met het voegwoord **of**. We noemen dit de *disjunctie* van p en q. Omdat minstens één van p en q waar zijn, is ook de disjunctie p of q waar.

De disjunctie van twee proposities is waar indien minstens één van beide samenstellende delen waar zijn. Indien beide samenstellende delen vals zijn, is de disjunctie ook vals. Dit wordt samengevat in waarheidstabel 2.6.

Tabel 2.6: Waarheidstabel van de `or` operator.

p	q	p or q
True	True	True
True	False	True
False	True	True
False	False	False

De disjunctie wordt in Python geïmplenteerd door de `or` operator.

```
>>> a = (5 < 3) or (7 > 1)      # False or True -> True
>>> print(a)
True

>>> b = (5 != 5) or (9 >= 11)  # False or False -> False
>>> print(b)
False
```

Negatie[15]**: NIET (Engels `not`)**.
De propositie

$$\underbrace{\text{het getal 7 is}}_{p}\ \textbf{niet}\ \underbrace{\text{groter dan het getal 10}}_{p}$$

is een voorbeeld van een propositie die een negatie bevat. De negatie van een propositie p is waar indien p vals is (en omgekeerd). Dit wordt samengevat in waarheidstabel 2.7.

Tabel 2.7: Waarheidstabel van de `not` operator.

p	not p
True	False
False	True

De negatie wordt in Python geïmplenteerd door de `not` operator.

```
>>> 5 > 7
False
>>> not (5 > 7)
True
```

[15] `not` is strikt genomen geen voegwoord.

2.9.4 Combineren van numerieke en logische operatoren

Rekenkundige (bv. `+`, `/`), relationele (bv. `>`, `==`) en booleaanse operatoren (**`and`**, **`or`**, **`not`**) kunnen samen voorkomen in een expressie. Bij het evalueren van deze expressies worden voorrangsregels gevolgd die worden weergegeven in Tabel 2.8

Tabel 2.8: Samenvatting voorrangsregels van hoogste naar laagste prioriteit.

Operator	Beschrijving
`()`	Haakjes
`**`	Machtsverheffing
`+x, -x`	Unaire plus/min
`*, /, //, %`	Vermenigvuldiging, Deling, Rest
`+, -`	Som, Verschil
`<, <=, >, >=, !=, ==`	Vergelijking
`not` `x`	*Boolean* NOT
`and`	*Boolean* AND
`or`	*Boolean* OR

In de onderstaande expressie zijn `a`, ..., `g` variabelen die verwijzen naar numerieke objecten.

```
a < b and c + d == e or f <= g
```

In deze expressie komen drie types operatoren voor. De volgende expressie heeft hetzelfde resultaat, maar maakt gebruik van haakjes die de prioriteit van de operatoren benadrukken.

```
((a < b) and (c + d == e)) or (f <= g)
```

Opdracht 2.28

Ga het resultaat na van de volgende expressies

- `(0 <= a_int) and (a_int <= 5)` waarbij `a_int = 5`
- `(X > 2) or (X > 5)` waarbij `X = 3`
- `2 > 5 or 6 * 2 <= 12 and 7 > 3`
- `True and False or True and False`

Algemene tip: gebruik steeds haakjes!

2.9.5 Relationele operatoren voor operanden van het type string

De operanden van relationele operatoren kunnen in Python ook van het type **`str`** zijn. Zo kan men van **twee strings nagaan of ze gelijk zijn** met de `==` operator, zoals wordt geïllustreerd hieronder.

```
>>> "Hallo" == "Test"
False                    # strings hebben verschillende waarde
>>> "Hallo" == "Hallo"
True                     # strings hebben dezelfde waarde
>>> "Hallo" == "hallo"
False                    # HOOFDLETTERGEVOELIG
```

Ook de operatoren `<`, `>`, `<=` en `>=` kunnen inwerken op operanden van het type **`str`**. Bij het vergelijken van strings wordt gewerkt volgens de **regels** die gebruikt worden bij het **alfabetisch rangschikken van woorden**, maar met een **uitgebreid alfabet** dat naast letters ook karakters zoals komma's, punten, enz. bevat. De plaats van deze karakters in het alfabet wordt gepaald door hun ascii/Unicode code[16][17]. Leestekens en cijfers hebben een eerder lage ascii/Unicode code (de spatie heeft de laagste ascii code van de printbare karakters), gevolgd door hoofdletters (volgens alfabet) en tenslotte kleine letters (volgens alfabet).

```
>>> "olifant" > "muis"
True                     # o heeft hogere ascii waarde dan m (idem alfabet)
>>> "kip" < "ei"
False                    # k heeft hogere ascii code dan e (idem alfabet)
>>> "Kip" < "ei"
True                     # K (hoofdletter) heeft lagere ascii code dan e
>>> "plaats" < "plas"
True                     # a (positie 4) heeft lagere ascii code dan s
                         #                                (idem alfabet)
```

Het onderstaande voorbeeld toont de invloed van spaties, die een zeer lage ascii waarde hebben.

```
>>> "Paul Van den Berghe" < "Paul Vanden Berghe"
True                # spatie tussen 'Van den' heeft lage ascii
```

Indien men het Unicode *code point* expliciet wenst op te vragen kan dit met de functie **`ord`**.

```
>>> ord("a")
97
>>> ord("b")
98
>>> ord("A")
65
>>> ord(" ")   # spatie heeft lage ascii/unicode
32
```

[16] Meer precies wordt Unicode gebruikt, wat een uitbreiding van ascii (die alle gangbare karakter bevat) is en waarbij de eerste 128 *code points* de ascii karakters bevatten.

[17] Voor een volledige lijst zie **`https://en.wikipedia.org/wiki/ASCII`**

2.9.6 Typeconversie

Objecten van het gegevenstype `bool` kunnen geconverteerd worden naar numerieke objecten of stringobjecten, met de vertrouwde functies voor typeconversie `int`, `float` en `str`. Bij dergelijke conversies, die in eenvoudige Python code niet vaak voorkomen, worden volgende regels gebruikt.

- `True` wordt bij omzetting naar een numeriek gegevenstype één.
- `False` wordt bij omzetting naar een numeriek gegevenstype nul.
- Typeconversie van de booleans `True` en `False` naar een string resulteert in de strings `'True'`, resp. `'False'`.

```
>>> int(True)
1
>>> str(False)
'False'
```

Typeconversie naar een object van het gegevenstype `bool` kan met de functie `bool`. Daarbij worden voor alle numerieke gegevenstypes **niet-nul waarden** geconverteerd naar `True`. **Nullen** worden geconverteerd naar `False`.

```
>>> bool(12)
True
>>> bool(0.0)
False
```

Indien een string object geconverteerd wordt naar een `bool` object, dan zal het resultaat `True` zijn. De enige uitzondering is de *lege* string `""` die wordt geconverteerd naar `False`

```
>>> bool("hallo")
True
>>> bool("")
False
```

Opdracht 2.29 (`even_getal.py`)
Een manier om na te gaan of een geheel getal (*integer*) even is, bestaat erin dit getal te delen door 2 en de rest te bekijken. Schrijf een Python script `even_getal.py` dat:

- De gebruiker vraagt een geheel getal in te geven.
- Controleert of dit getal kleiner is dan of gelijk aan 100.
- Controleert of dit getal even is.
- Indien beide voorwaarden voldaan zijn de waarde 1 op het scherm toont. Indien minstens een voorwaarde niet voldaan is, moet 0 op het scherm verschijnen.

Een mogelijke input/output is

```
Geef een geheel getal: 23
0
```

Voor je begint:

- Bekijk Sectie 2.9.
- Creëer een variabele `a` die verwijst naar een integer object met een waarde naar keuze.
- Ken het resultaat van de logische expressie `a <= 10` toe aan de variabele `b`.
- Voer de instructie `print(b)` uit.
- Voer de instructie `print(int(b))` uit.
- Gebruik de verworven inzichten om de opdracht uit te voeren.

2.10 Meer effciënt gebruik van Spyder

In secties 1.6–1.8 werd het gebruik van Spyder reeds kort gedemonstreerd. Op Ufora vinden jullie een paar korte videotutorials, waarin iets dieper wordt ingegaan op het gebruik van deze IDE.

Opdracht 2.30

Bekijk de twee videotutorials over het gebruik van Spyder op Ufora en vul de onderstaande tekst aan.

Bij het opstarten van de Spyder IDE, zien we 3 vensters openstaan: de Editor, IPython Console en ..... In de Console kan code rechtstreeks worden ingegeven en uitgevoerd, waarna het aan de ..... wordt toegevoegd. Via de ..... kan ook snel worden genavigeerd door reeds uitgevoerde code. Bovendien beschikt Spyder over zogenaamde *autocomplete* functie, wat kan worden geactiveerd d.m.v. de .....-toets. In de Editor kan broncode worden ingevoerd en opgeslagen als script, wat we kunnen uitvoeren door de sneltoets ..., het ...-icoontje of via het ...menu. Daarnaast kan een geselecteerd deel van de broncode afzonderlijk worden uitgevoerd via de ...sneltoets.

Twee andere handige en dus veelgebruikte vensters zijn de *Variable explorer* en *File explorer*. In de Variable explorer vinden we alle reeds gedefinieerde variabelen terug, met de ....., het ....., de .....en de .....als bijhorende informatie. De file explorer vereenvoudigt het instellen van de huidige ....., wat erg belangrijk is voor wanneer we bestanden willen ..... (iets wat verder in de cursus nog aan bod komt). Om een bepaalde map of bestand te bekijken met een externe verkenner (zoals bijvoorbeeld de Windows verkenner) moeten we .....op de gewenste map/bestand en .....aanklikken.

2.11 Gemengde opdrachten

Opdracht 2.31 (kogel_max_hoogte.py)
Een kogel wordt verticaal omhoog geschoten met een beginsnelheid van $v_0 = 300\,ms^{-1}$. De maximale hoogte wordt berekend met:

$$h_{max} = \frac{v_0^2}{2 \cdot g}$$

waarbij $g = 9.81\,ms^{-2}$ de valversnelling is. Schrijf een programmaatje die deze hoogte berekent. Een mogelijke output is:

```
Een kogel wordt verticaal afgevuurd.
Begin snelheid:  300 m/s
Valversnelling:  9.81 m/s^2
Maximale hoogte: 4587.155963302752 m
```

Opdracht 2.32 (rente_na_10_jaar.py)
Een spaarrekening heeft een rente van 0.50 % per jaar. Je plaatst een initieel bedrag van €10000 op deze spaarrekening. Hoe groot is het bedrag na 10 jaar?

$$A_n = A_0 \left(1 + \frac{p}{100}\right)^n$$

hierin is A_0 het initieel bedrag, n het aantal jaren, p de rente in % en A_n het eindbedrag. Een mogelijke output is:

```
Rente:           0.5 %
Initieel bedrag: 10000.0 euro
Aantal jaar:     10 jaar
Eind bedrag:     10511.401320407896 euro
```

Opdracht 2.33 (twee_getallen.py)
Schrijf een programmaatje die achtereenvolgens twee getallen vraagt en vervolgens die twee afzonderlijke getallen en hun som, verschil, product, quotiënt weergeeft op het scherm. Een mogelijke output is:

```
Geef het eerste getal: 9
Geef het tweede getal: 4
9.0 + 4.0 = 13.0
9.0 - 4.0 = 5.0
9.0 * 4.0 = 36.0
9.0 / 4.0 = 2.25
```

Opdracht 2.34 (uren_minuten_seconden.py)
Een dag bevat $86400\,s (= 24 \times 60 \times 60\,s)$. Schrijf een programmaatje die een tijd in seconden tussen 0 en 86400 vraagt en deze omzet naar uren, minuten en seconden. Een mogelijke input/output is

```
Geef een tijd in seconden tussen 0 en 86400 s: 56335
56335 seconden is 15 uren, 38 minuten, en 55 seconden.
```

Opdracht 2.35 (oppervlakte_trapezium.py)
Een oppervalkte A van een trapezium wordt berekend met

$$A = \frac{1}{2} \cdot h \cdot (a + b)$$

waarbij a en b de lengtes zijn van de basis en h de hoogte. Schrijf een programmaatje dat vraagt naar deze lengtes (in meter), de oppervlakte berekent en deze weergeeft naar het scherm. Een mogelijke input/output is

```
Geef de lengte (in m) van de ene basis: 5.0
Geef de lengte (in m) van de andere basis: 4.0
Geef de hoogte (in m): 3.0
De oppervlakte van de trapezium is 13.5 m^2
```

Opdracht 2.36 (hoeken_driehoek.py)
In een driehoek geldt $c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(C)$ met a, b en c de lengtes van de zijden en C de hoek gevormd door a en b. Schrijf een script dat de lengtes a, b en c vraagt aan de gebruiker vraagt als input. Op basis van de gebruikersinput moet eerst nagegaan worden of de ingegeven lengtes een driehoek kunnen vormen. Vervolgens moet de hoek C berekend worden. Een mogelijke input/output is:

```
Geef lengte van zijde a: 3
Geef lengte van zijde b: 7
Geef lengte van zijde c: 9
Controle geldige driehoek: True
Hoek tussen zijden a en b is: 123.20382252997027 graden

Geef lengte van zijde a: 6
Geef lengte van zijde b: 7
Geef lengte van zijde c: 1
Controle geldige driehoek: False
Hoek tussen zijden a en b is: 0.0 graden
```

Voor je begint.

- Zoek (op het Web) op aan welke voorwaarden a, b en c moeten voldoen opdat ze de lengtes van de zijden van een driehoek zouden kunnen zijn.
- Implementeer deze voorwaarden als een (vrij uitgebreide) logische expressie (je hebt mogelijks de logische operatoren `and`, `or` en `not` nodig).

Opdracht 2.37 (`isbn01.py`)

De International Standard Book Number (ISBN) is een unieke numerieke commerciële boek identificatienummer. Als dit nummer toegekend is aan een boek na 1 januari 2007, dan is dit ISBN-nummer 13 cijfers lang (ISBN-13). De eerste 12 daarvan geven informatie over het boek zelf, terwijl het laatste louter een controlecijfer is dat dient om foutieve ISBN-13 codes te detecteren. Indien x_1, x_2, ..., x_{12} de eerste 12 cijfers van een ISBN-13 code voorstellen, dan wordt het controle cijfer x_{13} als volgt berekend:

$$x_{13} = (10 - (x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 + 3x_{10} + x_{11} + 3x_{12}) \mod 10) \mod 10$$

x_{13} kan m.a.w. een waarde aannemen tussen 0 en 9 ($0 \leq x_{13} \leq 9$). Schrijf een programma dat achtereenvolgens de eerste 12 cijfers vraagt van een ISBN-13 code en het 13de cijfer berekent en weergeeft naar het scherm. Een mogelijke input/output is:

```
9
7
8
1
2
9
2
0
2
1
0
3
Controle cijfer:  4
ISBN-13 code:  9781292021034
```

Opdracht 2.38 (`ontwikkelingsindex.py`)

De index van de menselijke ontwikkeling (ontwikkelingsindex) of Human Development Index (HDI) van de Verenigde Naties meet voornamelijk armoede, analfabetisme, onderwijs en levensverwachting in een bepaald land of gebied. De index werd in 1990 ontwikkeld door de Pakistaanse econoom Mahbub ul Haq en wordt sinds 1993 door VN-Ontwikkelingsprogramma gebruikt in haar jaarlijks rapport. Noorwegen staat vaak op de eerste plaats. In 2014 was Nederland goed voor een vierde plaats en stond België op de 21ste plaats. Onderaan staan de Afrikaanse landen Tsjaad, de Centraal Afrikaanse Republiek, Congo en Niger.

De index meet de gemiddelde prestaties van een land, opgedeeld in drie categorieën:

- **Volksgezondheid**: deze wordt gemeten aan de hand van de levensverwachtingsindex (Life Expectancy Index; LEI) die de gemiddelde levensverwachting bij geboorte uitdrukt

$$LEI = \frac{LE - 20}{82.3 - 20}$$

waarbij LE (Life Expectancy) staat voor de levensverwachting bij de geboorte.

- **Kennis**: deze wordt gemeten aan de hand van de onderwijsindex (Education Index; EI) die een maat is voor het analfabetisme en het deel van de bevolking dat primair, secundair en tertiair onderwijs doorloopt

  $$EI = \frac{\sqrt{MYSI \cdot EYSI}}{0.951}$$

  hierbij wordt de gemiddelde scholingsjarenindex (Mean Years of Schooling Index; $MYSI$) gegeven door

  $$MYSI = \frac{MYS}{13.2}$$

  waarbij MYS (Mean Years of Schooling) staat voor het gemiddeld aantal jaren scholing voor een 25-jarige, en wordt de verwachte scholingsjarenindex (Expected Years of Schooling Index; $EYSI$) gegeven door

  $$EYSI = \frac{EYS}{20.6}$$

  waarbij EYS (Expected Years of Schooling) staat voor het verwacht aantal jaren scholing voor een 5-jarige.

- **Levensstandaard**: deze wordt gemeten aan de hand van de inkomensindex (Income Index; II) die het bruto nationaal product uitdrukt per hoofd van de bevolking, in koopkrachtpariteit in dollars

  $$II = \frac{\ln(GNI) - \ln(100)}{\ln(107721) - \ln(100)}$$

  waarbij GNI (Gross National Income at purchasing power parity per capita) staat voor het bruto nationaal inkomen per hoofd van de bevolking, en ln staat voor de natuurlijke logaritme.

De uiteindelijke index is het meetkundig gemiddelde van de drie genormaliseerde indices:

$$HDI = \sqrt[3]{LEI \cdot EI \cdot II}$$

Schrijf een programma die de volgende invoer vraagt:

- Naam van het land
- Levensverwachting bij geboorte (LE)
- Gemiddeld aantal jaren scholing voor een 25-jarige (MYS)
- Verwacht aantal jaren scholing voor een 5-jarige (EYS)
- Bruto nationaal inkomen per hoofd van de bevolking ($GNIpc$)

waarbij je LE, MYS, EYS en $GNIpc$ beschouwt *floats*. Het programma moet uiteindelijk een regel op het scherm weergeven waarin de naam van het land staat en de HDI als *float*. Een mogelijke input/output is

```
Naam land: Belgie
LE:  80.548
MYS: 10.86875064
EYS: 16.2
GNI: 39470.90422
De HDI van Belgie bedraagt 0.8896358328268209
```

Bereken de ontwikkelingsindex van enkele andere landen:

Land	LE	MYS	EYS	GNI
Noorwegen	81.5	12.6	17.6	63909
Nederland	81.0	11.9	17.9	42397
Duitsland	80.7	12.9	16.3	43049
Groot Brittannië	80.5	12.3	16.2	35002

2.12 Belangrijkste concepten – samenvatting

- Object, (gegevens)type, variabele
  - Types, bv.: *int*, *float*, *str*, *bool*
- Geldige variabelnamen voldoen aan bepaalde regels
  - begin met een letter; anders letters, cijfers en underscore
  - beginnend met een underscore heeft een speciale betekenis voor later
- *Keywords*: Tabel 2.1
- *Statement*
  Een *statement* is een instructie; geeft geen waarde terug.
- Expressie
  Een expressie is een combinatie van waarden, variabelen en operatoren; geeft een waarde terug.
- Operatoren voor operanden *int* en *float*: Tabel 2.2
  - Voorrangsregels voor rekenkundige bewerkingen: Tabel 2.3
- Operator voor operanden *string* en *string*: `+`
- Operator voor operanden *string* en *int*: `*`
- Functies (eerste kennismaking)
  - *Built-in* functies: `https://docs.python.org/3.6/library/functions.html`
  - `math`-module: `http://docs.python.org/3.6/library/math.html`
- Typeconversiefuncties: `int()`, `float()`, `str()`, `bool()`
- Input van keyboard en output naar scherm: `input()`, `print()`
- Gegevenstype *Boolean*: `True`, `False`
- *Boolean* vergelijkingsoperatoren: Tabel 2.4

- Basis *Boolean* operatoren: **`and`**, **`or`**, **`not`**
- Samenvatting voorrangsregels: Tabel 2.8
- De *string*-methoden: *str*.`isalpha()`, *str*.`isnumeric()`
  (`https://docs.python.org/3.6/library/stdtypes.html#string-methods`)
- Andere te beheersen functies:
  - *Built-in Functions*:
    `abs()`, **`complex()`**, **`id()`**, **`len()`**, **`max()`**, **`min()`**, **`pow()`**, **`round()`**, **`pow()`**, **`type()`**
    (zie `https://docs.python.org/3.6/library/functions.html`)
  - *math - Mathematical functions*:
    `math.fabs(x)`, `math.exp(x)`, `math.log(x)`, `math.log10(x)`, `math.pow(x, y)`, `math.sqrt(x)`,
    `math.acos(x)`, `math.asin(x)`, `math.atan(x)`, `math.cos(x)`, `math.sin(x)`, `math.tan(x)`, `math.degrees(x)`
    `math.radians(x)`, `math.pi`, `math.e`
    (zie `https://docs.python.org/3.6/library/math.html`)