Basiscursus Python

Inhoudsopgave

Hoofdstuk 1 – Hello World

Hoofdstuk 2 – Praten met de interpreter
- Hoofdstuk 2-1 – Rekenen
- Hoofdstuk 2-2 – Condities
- Hoofdstuk 2-3 – Variabelen
- Hoofdstuk 2-4 – Samenbrengen van 2-1, 2-2 en 2-3
- Hoofdstuk 2-5 – Andere Datatypes
  - Hoofdstuk 2-5-1 – Floats
  - Hoofdstuk 2-5-2 – Strings

Hoofdstuk 3 – Je eerste echte programma
- Hoofdstuk 3-1 – Het .py bestand
- Hoofdstuk 3-2 – Comments
- Hoofdstuk 3-3 – Introductie functies

Hoofdstuk 4 – Simpele logica
- Hoofdstuk 4-1 – Van datatype veranderen
- Hoofdstuk 4-2 – Een nieuw datatype: booleans
- Hoofdstuk 4-3 – Het if statement en code blocks
- Hoofdstuk 4-4 – Boolean operaties
- Hoofdstuk 4-5 – Elif
- Hoofdstuk 4-6 – Else
- Hoofdstuk 4-7 – De remainder operator

Hoofdstuk 5 – Loops
- Hoofdstuk 5-1 – De While Loop
- Hoofdstuk 5-2 – Break en continue

Hoofdstuk 6 – The Python Standard Library
- Hoofdstuk 6-1 – De random module
- Hoofdstuk 6-2 – De os module

Hoofdstuk 7 – Lijsten
- Hoofdstuk 7-1 – De index operator en de len() functie
- Hoofdstuk 7-2 – Lijst operaties
- Hoofdstuk 7-3 – De in operator

Hoofdstuk 8 – Functies
- Hoofdstuk 8-1 – Functies definiëren

Hoofdstuk 9 – Try en except

Appendix
- Machtsverheffen
- De Python interpreter openen
- Geany openen
- Een nieuw bestand aanmaken
- Een bestand opslaan
- Je programma uitvoeren

Hoofdstuk 1 – Hello World

Het is een oude traditie binnen het programmeren om te beginnen met een Hello World (hallo wereld) programma. Een Hello World programma is niets meer dan een programmaatje dat "Hello World" op het scherm weergeeft.

Een Hello World programma in Python is erg simpel en maar 1 regel lang:

```
print("Hello World")
```

Hoe dit precies werkt leren we verderop. Laten we eerst eens kijken of we dit kleine programma uit kunnen voeren.

Opdracht 1

a) Open de Python interpreter

b) Type `print("Hello World")` en druk op ENTER

c) Kun je bedenken hoe je het programma "Hello Programmer" kunt laten zeggen?

Hoofdstuk 2 – Praten met de interpreter

Hoofdstuk 2-1 – Rekenen

De computer is eigenlijk een hele complexe rekenmachine. Dit betekent dat berekeningen uitvoeren voor de computer ongeveer het makkelijkste is om te doen. Laten we dit verkennen door de Python interpreter een aantal berekeningen te laten uitvoeren:

Opdracht 1

a) Open de Python interpreter

b) Voer de volgende berekeningen uit. Dit doe je door ze in te typen en dan steeds op enter te drukken om de berekeningen uit te voeren
   1. `2+2`
   2. `2-5`
   3. `6555-23382+22211-3`

c) We kunnen uiteraard ook vermenigvuldigen en delen. Vermenigvuldigingen doen we met `*` en delen doen we met `/`
   1. `2*10`
   2. `2/8`
   3. `2*8/10`

d) Net als in wiskunde kunnen we haakjes gebruiken om aan te geven welk gedeelte van een berekening eerst moet worden uitgevoerd
   1. `2+(4-3)`
   2. `2+4-3`
   3. `2*(8-2)`
   4. `2*8-2`

e) We kunnen Python ook makkelijk machten laten uitrekenen. Dit doen we met `**`
   1. `2**2`
   2. `2**10`
   3. `2**1000`

`+`, `-`, `*`, `/`, `**`, en `( )` noemen we **operators**. Python kent nog veel meer operators.

Python volgt de **voorrangsregels** zoals die in de wiskunde zijn om te weten in welke volgorde de berekeningen moeten worden uitgevoerd. De voorrangsregels zijn:

1. Haakjes wegwerken
   Als er haakjes in de som staan worden die eerst weggewerkt

2. Machtsverheffen
   Als er machten in de som staan worden die weggewerkt

3. Vermenigvuldigen en delen
   Daarna delen en vermenigvuldigen van links naar rechts

4. Optellen en aftrekken
   Als laatste optellen en aftrekken van links naar rechts

Dit betekent dat de volgende berekening, net als in wiskunde, als volgt wordt uitgevoerd:

1. `2**2+18-(12+2)*12/12-13+(2+16)*2` *De berekening*
2. `2**2+18-14*12/12-13+18*2` *De berekening nadat de haakjes zijn weggewerkt*
3. `4+18-14*12/12-13+18*2` *De berekening na machtsverheffen*
4. `4+18-14-13+36` *De berekening na vermenigvuldigen en delen*
5. `31` *De oplossing van de berekening*

Overzicht

- Symbolen zoals `+`, `-`, `*`, `/`, `**`, en `( )` noemen we operators
- We gebruiken de `+` operator om getallen op te tellen
- We gebruiken de `-` operator om getallen van elkaar af te trekken
- We gebruiken de `*` operator om getallen met elkaar te vermenigvuldigen
- We gebruiken de `/` operator om getallen door elkaar te delen
- We gebruiken de `**` operator om macht te verheffen
- We gebruiken de `( )` operators om de volgorde van een berekening aan te geven
- Python volgt de voorrangsregels. De voorrangsregels zijn:
    1. Haakjes wegwerken
    2. Machtsverheffen
    3. Vermenigvuldigen en delen
    4. Optellen en aftrekken

Eindopdracht

Dit hoofdstuk heeft geen eindopdracht. Berekeningen uitvoeren is de basis van alle code en het is daarom belangrijk dat je het goed begrijpt. Blijf daarom net zo lang oefenen met de concepten in het overzicht totdat je het gevoel hebt dat ze begrijpt

Hoofdstuk 2-2 – Condities

Naast dat de computer makkelijk berekeningen kan uitvoeren, kan de computer ook makkelijk kijken of bepaalde dingen waar of niet waar zijn. Dit noemen we **condities**. Laten we dit verkennen door de Python interpreter een aantal condities te geven.

Opdracht 1

a) Open de Python interpreter

b) We kunnen vragen of het ene getal groter is dan het andere getal. Dit doen we met de `>` operator. Python vertelt ons dan of dat waar (`True`) of niet waar (`False`) is
   1. `10>2`
   2. `2>10`
   3. `2>0.5*2`

c) We kunnen ook vragen of het ene getal kleiner is dan het andere getal. Dit doen we met de `<` operator
   1. `20<40`
   2. `20<18`
   3. `20<120000`

d) Als we willen weten of een getal groter of gelijk, of kleiner of gelijk is aan een ander getal, kunnen we dit doen met de `<=` en `>=` operators
   1. `18>=20`
   2. `19>=19`
   3. `19<=19`
   4. `23<23`
   5. `23>=1000`
   6. `35<=1000`

e) Wat als we willen weten of twee waardes hetzelfde zijn? Dan gebruiken we de `==` operator (niet te verwarren met de `=` operator uit het volgende hoofdstuk)
   1. `3==3`
   2. `3==4`
   3. `18==2*9`
   4. `18==2000`

f) Of als we willen weten of twee waardes niet hetzelfde zijn, dan gebruiken we de `!=` operator
   1. `3!=3`
   2. `3!=4`
   3. `15!=15*1`
   4. `44!=4*11`

Overzicht

- We gebruiken de < operator om te kijken of de waarde links van de operator kleiner is dan de waar rechts van de operator
- We gebruiken de > operator om te kijken of de waarde links van de operator groter is dan de waar rechts van de operator
- We gebruiken de <= operator om te kijken of de waarde links van de operator kleiner dan is of gelijk is aan de waar rechts van de operator
- We gebruiken de >= operator om te kijken of de waarde links van de operator groter is dan of gelijk is aan de waar rechts van de operator
- We gebruiken de == operator om te kijken of de waarde links van de operator gelijk is aan de waar rechts van de operator
- We gebruiken de != operator om te kijken of de waarde links van de operator niet gelijk is aan de waar rechts van de operator

Eindopdracht

Blijf net zo lang oefenen met condities en de operators in het overzicht totdat je het gevoel hebt dat ze begrijpt

Extra informatie

Als vervanging van de == operator en de != operator kun je ook de **is** operator en de **is not** operator gebruiken

1. `2==2`
   wordt dan
   `2 is 2`
2. `2!=3`
   wordt dan
   `2 is not 3`

Dit verschil is puur voor de vorm en er is geen daadwerkelijk verschil. Het is een andere manier om hetzelfde op te schrijven. Probeer het eens.

Hoofdstuk 2-3 – Variabelen

Variabelen zijn een van de belangrijkste en meest basale elementen van het programmeren.

Een variabele heeft twee eigenschappen:

- Een naam
- Een waarde

Een makkelijke manier om je een variabele voor te stellen is als een emmer met een naam en de waarde als de inhoud van de de emmer. De naam mag geen spaties bevatten en is zoals altijd hoofdlettergevoelig. Laten we dit verkennen door de Python interpreter een paar variabelen te geven.

Opdracht 1

a) Open de Python interpreter

b) We kunnen een variabele aanmaken met de `=` operator (niet te verwarren met de `==` operator)

1. `getal1 = 33`
2. `getal2 = 44`

Wat we hier gedaan hebben is twee variabelen gecreëerd, één genaamd `getal1` en één genaamd `getal2`. `getal1` heeft een waarde van `33` gekregen en `getal2` heeft een waarde van `44` gekregen. Als we dit verplaatsen naar onze emmer-metafoor dan hebben we nu twee emmers. Een emmer genaamd `getal1` met daarin het getal `33` en een emmer genaamd `getal2` met daarin het getal `44`

c) We kunnen de Python interpreter nu vragen wat de waardes zijn van die twee variabelen. Met andere woorden, we kunnen Python vragen wat er in de emmers zit. Dit doen we door de interpreter de naam van de variabele te geven (de naam van de emmer) en Python vertelt ons dan de waarde die bij die variabele hoort

1. `getal1`
2. `getal2`

d) We kunnen de naam van de variabelen niet veranderen, maar de waarde (de inhoud van de emmer) wel. Probeer het volgende maar eens:

1. `getal1 = 555`
2. `getal1`

Je ziet dat `getal1` nu niet meer de waarde `33` heeft maar de waarde `555`

Overzicht

- Een variabele heeft twee eigenschappen
  - een naam
  - een waarde
- Je kunt een variabele zien als een emmer. De naam van de variabele is dan de naam van de emmer en de waarde is wat er in de emmer zit
- De naam mag geen spaties bevatten en is hoofdlettergevoelig
- De waarde van de variabele kun je veranderen, de naam van de variabele niet

Hoofdstuk 2-4 – Herhalen en samenbrengen van 2-1, 2-2 en 2-3

Laten we eens kijken of we de concepten die we geleerd hebben (operators, condities en variabelen) kunnen combineren:

Opdracht 1

a) Open de Python interpreter

b) `getal = 20`

c) `getal + 2`

Aangezien de computer weet dat de variabele `getal` een waarde heeft van `20`, kunnen we de computer vragen hoeveel `getal + 2` is

d) `getal * 3 + getal / 5`

e) `nieuw_getal = getal + 18`

f) `nieuw_getal - getal`

g) `nieuw_getal < getal`

h) `nieuw_getal == getal-18`

i) Zorg dat je de voorbeelden b t/m h goed begrijpt en blijf oefenen totdat je het gevoel hebt dat je de concepten goed snapt.

Hoofdstuk 2-5 – Andere datatypes

Variabelen hebben een **datatype**. Tot nu toe hebben we de hele tijd met één soort datatype gewerkt. Je kunt je een datatype voorstellen als een soort data. Bijvoorbeeld een plaatje is een andere soort data dan een tekstbestand. Vooralsnog hebben we alleen gekeken naar hele getallen. Dit noemen we een in het Engels en in programmeren een **integer**.

Laten we in dit hoofdstuk eens naar een aantal andere datatypes kijken.

Hoofdstuk 2-5-1 – Floats

Een **float** is een decimaal getal. Dat is een getal met cijfers achter de komma. In Python geven we een dergelijk getal aan met een `.`

Opdracht 1

a) Open de Python interpreter

b)
```
float1 = 15.55
```

c)
```
getal1 = 5
```

d)
```
getal1 * 2
```

e)
```
float1 * 2
```

f)
```
product = float1*getal1
```

g)
```
product
```

Hoofdstuk 2-5-2 – Strings

Naast getallen wordt er in het programmeren ook veel met tekst gewerkt. Een stukje tekst noemen we in het programmeren een **string**. Laten we eens kijken hoe dat werkt in de interpreter.

Opdracht 1

a) Open de Python interpreter

b) Je kunt een string maken door je code tussen *dubbele* aanhalingstekens te zetten. Probeer maar eens:
```
"een voorbeeld van een string"
```

c) Je kunt een string ook maken door je code tussen *enkele* aanhalingstekens te zetten:
```
'ook een goed voorbeeld van een string'
```

d) Dit betekent dus ook dat als je in je string een aanhalingsteken wilt gebruiken, dat je in de problemen kunt komen. Zoals in de volgende string:
```
'deze string is vanwege de apostrof in het woord 's morgens afgebroken'
```

e) In de meeste gevallen kun je dit oplossen door een keuze te maken tussen `" "` en `' '`

f) `"Dit is een string met een ' in de string"`

g) `'En dit is een string met een " in de string'`

h) Je kan ook een `\` voor het aanhalingsteken zetten en dan zal deze niet gezien worden als het aanhalingsteken dat de string afsluit. Bijvoorbeeld:
```
'Ik ga \'s morgens hardlopen'
```

i) Kun je een variabele maken genaamd `voornaam` met daarin de waarde van jouw naam?

j) Kun je een tweede variabele maken genaamd `achternaam` met daarin de waarde van jouw achternaam?

k) We kunnen strings ook bij elkaar optellen. Python plakt de strings dan aan elkaar. Als je opdracht i & j goed hebt gedaan kun je nu het volgende doen:
```
voornaam+achternaam
```

l) Kun je ze zo bij elkaar optellen dat er een spatie tussen je voor- en achternaam komt te staan?

m) Je kunt ook vragen of je achternaam langer is dan je voornaam:
```
achternaam > voornaam
```

n) Kun je vragen of je achternaam gelijk is aan je voornaam?

Overzicht

- Een heel getal noemen we een integer
- Een getal met cijfers achter de komma noemen we een float
- Een stuk tekst noemen we een string
- Integers, floats en strings zijn voorbeelden van datatypen, dit zijn verschillende soorten data
- Een float geven we aan met een punt, bijvoorbeeld `1.5`
- Een string geven we aan door middel van een paar dubbele(`"..."`) of enkele(`'...'`) aanhalingstekens, bijvoorbeeld `"Een string"` of `'Een string'`
- Door te kiezen voor de ene soort aanhalingstekens, kun je de andere gebruiken in de string
- Je kunt ook een `\` voor een `"` of `'` in een string zetten om aan te geven dat het aanhalingsteken niet bedoeld is als het eind van de string
- Je kunt strings bij elkaar optellen, Python plakt ze dan aan elkaar
- Je kunt strings vergelijken

Eindopdracht

Schrijf de volgende zin als 1 string en stop deze in een variabele genaamd `lange_zin`

Dries' computer zei “Hello World!” toen hij begon met programmeren.

Hoofdstuk 3 – Je eerste echte programma

Hoofdstuk 3-1 – Het .py bestand

We hebben tot nu toe de hele tijd de interpreter een opdracht tegelijk laten uitvoeren. Een **programma** is een serie van opdrachten. Elke opdracht komt dan op zijn eigen regel te staan. Als we dit hele programma aan de interpreter geven, voert die het programma, stap voor stap, voor ons uit. In Python slaan we zo'n programma op in een .py bestand. Zo'n bestand kun je in elke tekstverwerker maken, maar wij gebruiken een tekstverwerker die speciaal bedoeld is om het programmeren makkelijker te maken. Zo'n tekstverwerker speciaal voor programmeren noemen we een **IDE**. De IDE die wij gebruiken heet **Geany**. Geany laat de tekst in verschillende kleuren zien, zodat we de code makkelijker kunnen lezen.

Laten we het Hello World programma uit Hoofdstuk 1 nog een keer maken, maar nu in Geany:

Opdracht 1

a) Open Geany

b) Eerst maken we een leeg Python programma aan. Dit doen we door een nieuw bestand aan te maken en dan het lege bestand op te slaan als een .py bestand. Geany weet dan dat we een Python programma aan het maken zijn en waar het staat

1. Maak een nieuw bestand aan
2. Sla je bestand op als HelloWorld.py

c) Als alles gelukt is heb je nu een leeg Python programma voor je. Als we hier de enkele opdracht van ons Hello World programma in zetten en dan het bestand opslaan, hebben we een echt programma gemaakt

Schrijf het programma. Weet je het nog?

d) Sla het bestand op

e) Nu het programma af is kunnen we het laten uitvoeren. Dit doen we door op Run (uitvoeren) te klikken. Geany geeft je programma dan automatisch aan de interpreter en je programma wordt geopend in een zwart scherm genaamd LXTerminal. Zorg dat je altijd je programma hebt opgeslagen voor je het uitvoert

Voer HelloWorld.py uit

Overzicht

- Een programma is een serie van opdrachten
- Elke opdracht staat op zijn eigen regel
- Een Python programma slaan we op in een .py bestand
- Een IDE is een tekstverwerker speciaal voor programmeren, onze IDE heet Geany
- Je moet een bestand altijd opslaan voor je het kunt uitvoeren

Hoofdstuk 3-2 – Comments

Omdat het handig is om notities en aantekeningen te maken binnen je code, kun je stukken tekst in je programma markeren als **comment**. Dit doe je door een `#` voor de tekst te zetten. Een comment wordt niet gelezen door de interpreter bij het uitvoeren van je programma.

```
# Dit is een comment
```

Hoofdstuk 3-3 – Introductie functies

Om wat interessantere dingen te kunnen doen, gaan we naar **functies** kijken. Verderop in de cursus gaan we kijken wat functies precies zijn en hoe we ze zelf kunnen maken. Python heeft echter een heleboel ingebouwde functies, daarom is het voor nu genoeg om te weten dat functies iets voor je uitvoeren. We hebben al een voorbeeld van een functie gezien: de `print()` functie. Wat de print functie voor ons doet is datgene tussen de `()` op het scherm printen. In de interpreter hadden we `print()` niet nodig. Alle opdrachten die we gaven werden namelijk meteen geprint. Als je `2+2` als opdracht gaf kreeg je het antwoord `4` meteen op je scherm. In een echt programma, geschreven in een .py bestand, wordt helemaal niks weergegeven tenzij we de print functie gebruiken. Bijvoorbeeld: `print(2+2)`

Het gedeelte tussen de haakjes kun je zien als de input van de functie. De print functie bijvoorbeeld neemt als input datgene dat geprint moet worden. Deze input wordt het **argument** van een functie genoemd. Sommige functies nemen meer dan één argument als input. Er zijn ook functies die helemaal geen argumenten nemen.

Opdracht 1

a) Open Geany

b) Maak een nieuw bestand aan en sla het op als functie_print.py

c) Maak een variabele aan genaamd `cijfers` met een waarde van `500`

d) Op een nieuwe regel, print de variabele `cijfers` met de print functie, gebruik de variabele `cijfers` als het argument

e) Sla functie_print.py op en voer het uit

Als het goed is zie je nu `500` in een verder leeg scherm staan

Laten we eens naar een nieuwe functie kijken: de `input()` functie. `input()` geeft je de mogelijkheid om input van de gebruiker te vragen. Laten we eens kijken hoe dit werkt.

Opdracht 2

a) Open Geany

b) Maak een nieuw bestand aan en sla het op als functie_input.py

c) Type `input()`

d) Sla functie_input.py op en voer het uit

e) Als je het goed gedaan hebt zie je nu een zwart scherm waar je in kan typen. Dit is Python die om jouw input vraagt. Je kunt op enter drukken om te zeggen dat je klaar bent met je input

f) Als je op enter hebt gedrukt is je programmaatje meteen klaar. Dat is logisch, want het enige wat we doen is vragen om de input van de gebruiker, we doen niks met deze informatie. Daarnaast het is nogal onduidelijk wat we precies vragen van de gebruiker.
Laten we de gebruiker om zijn naam vragen. Dit kunnen we doen door de `input()` functie een string als argument te geven die de functie dan aan de gebruiker kan laten zien als we om input vragen.

   Voeg de vraag "Wat is je naam?" toe als als argument in de `input()` functie

g) Voer je programmaatje opnieuw uit

h) Als het goed is vraagt je programmaatje nu wat je naam is en kun je je naam typen en op enter drukken om je input te bevestigen

   Kun je zorgen dat het antwoord dat je typt niet aan je vraag vast zit maar netjes een spatie heeft (en misschien een dubbele punt?)

We hebben nu gezien dat een functie een naam heeft, mogelijk input heeft in de vorm van argumenten en dingen voor ons uitvoert. De laatste stap is dat functies ook een output kunnen hebben. Dit noemen we in het programmeren de **return value**, wat Engels is voor teruggeefwaarde. De input functie is bijvoorbeeld een functie die een waarde teruggeeft. De input functie geeft namelijk de input van de gebruiker terug. In opdracht 1 zou dat het antwoord op de vraag "Wat is je naam?" zijn.

Laten we kijken of we het programmaatje je kunnen laten begroeten met je eigen naam.

Opdracht 3

a) Open Geany

b) Maak een nieuw bestand aan en sla het op als hallo_naam.py

c) Maak een variabele genaamd naam en geef het de waarde die terug komt van de input functie. Dit ziet er zo uit:

```
naam = input("Wat is jouw naam?: ")
```

d) Voeg een regel toe die de tekst "hallo" en dan jouw naam print, gevolgd door een uitroepteken. Als je naam Jan is dan wordt het bijvoorbeeld "Hallo Jan!" Gebruik hiervoor de `print()` functie, de `naam` variabele en strings. Vergeet niet dat je strings kunt optellen

e) Voer hallo_naam.py uit

Overzicht

- Een functie voert iets voor je uit
- Een functie heeft een naam die wordt gevolgd door `()`
- Een functie kan een input hebben in de vorm van één of meerdere argumenten
- Argumenten plaats je tussen de `()`
- Een functie kan een output hebben in de vorm van een return value, de return value is wat de functie je teruggeeft
- Je kunt de return value opslaan in een variabele
- Voorbeelden van functies die we tot nu toe gezien hebben zijn `print()` en `input()`

Eindopdracht

Schrijf een programmaatje dat je voornaam, je achternaam en je lievelingskleur vraagt en de gebruiker vervolgens een welkomstbericht geeft dat zegt: "Hallo voornaam achternaam, jouw lievelingskleur is paars!" waarbij voornaam, achternaam en paars de waarde krijgen van de input van de gebruiker

Hoofdstuk 4 – Simpele logica

In dit hoofdstuk gaan we kijken naar **logica**. Logica stelt ons in staat om de computer te vertellen hoe hij moet omgaan met verschillende omstandigheden. Echter, we gaan eerst naar twee nieuwe functies kijken.

Hoofdstuk 4-1 – Van datatype veranderen

In dit hoofdstuk gaan we kijken hoe we strings om kunnen zetten in integers en andersom met behulp van de `str()` en `int()` functies.

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd int_str.py en sla het op

c) Maak twee variabelen aan:
```
getal = 1
string = "wat een mooie string"
```

d) Laten we eens kijken wat er gebeurt als we de integer bij de string willen optellen:
```
print(string + getal)
```

e) Sla het programma op en voer het uit.
Dit levert een foutmelding op. Dit komt omdat we een twee verschillende datatypes, namelijk een string en een integer, bij elkaar proberen op te tellen en Python weet niet hoe dat moet

f) Om het getal bij de string op te tellen moeten we het getal eerst omzetten in een string. Dit kunnen we doen met de `str()` functie. De `str()` functie zet een getal om in een string.
Bijvoorbeeld:
`str(12)` geeft `"12"` terug

Schrijf opdracht d zo op dat het programmaatje geen foutmelding meer geeft

g) Andersom kan ook, als we een string hebben die gevuld is met cijfers kunnen we deze omzetten in een getal. Dit doen we met de `int()` functie.
Bijvoorbeeld:
`int("1234567890")` geeft `1234567890` terug

Dit werkt alleen als de de string slechts en alleen getallen bevat. Python weet bijvoorbeeld niet hoe hij de letter 'a' moet omzetten in een getal

h) Schrijf een programmaatje dat de gebruiker om een getal vraagt, dat getal met `2222` vermenigvuldigt en de uitkomst op het scherm print.
Tip: de **`input()`** functie geeft een string terug, ook als de input een getal is

Overzicht

- We kunnen de **`str()`** functie gebruiken om een integer om te zetten in een string
- We kunnen de **`int()`** functie gebruiken om een string om te zetten in een integer
- De **`input()`** functie geeft een string terug. Ook als de input een getal is

Hoofdstuk 4-2 – Een nieuw datatype: booleans

Logica werkt met waar (**`True`**) en niet waar (**`False`**). We hebben dit al eerder gezien toen we naar condities aan het kijken waren. We konden bijvoorbeeld vragen of `15` groter was dan `10` (`15>10`) en de computer vertelde ons dan of dat waar (**`True`**) of niet waar (**`False`**) was. **`True`** and **`False`** noemen we **booleans** en zijn hun eigen datatype. **`True`** en **`False`** zijn de enige twee waardes die een boolean kan hebben. Een boolean kunnen we opslaan in een variabele:

```
boolean_variabele = True
boolean_variabele = False
```

Overzicht

- Een Boolean kan slechts 2 waarden aannemen:
  - **`True`** (waar)
  - **`False`** (niet waar)
- We kunnen een boolean opslaan in een variabele

Hoofdstuk 4-3 – Het if statement en code blocks

Het **if** statement is het simpelste logica statement dat we kunnen maken.
Met een if statement kunnen in code schrijven 'als dit, doe dan dat'.
Het if statement ziet er als volgt uit:

```
if conditie:
    ...
```

Hiermee zeggen we: als (**`if`**) de voorwaarde (`conditie`) waar (**`True`**) is, doe dan wat er na de dubbele punt komt.

Bijvoorbeeld:

```
if 1+1 == 2:
    print(True)
```

Wat hier staat is: als één plus één twee is, print dan waar. Aangezien 1+1 altijd 2 is, wordt deze code altijd uitgevoerd.

Zoals je kunt zien is begint het gedeelte na de dubbele punt met een **tab**. Python gebruikt tabs om een **code block** aan te geven. Met de tab zeg je als het ware dat de code bij het if statement erboven hoort. Dit betekent dus dat als je meer dan één opdracht in het if statement wilt stoppen, deze ook allemaal met een tab moeten beginnen.

Bijvoorbeeld:

```
if 1+1 == 2:
    print(True)
    print("Één plus één is altijd twee en daarom wordt dit if statement altijd uitgevoerd")
```

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd if.py en sla het op

c) Probeer beide voorbeelden van hierboven uit en draai het programma

d) Wanneer wordt de variabele `mysterieus_getal` uitgevoerd in het volgende voorbeeld?

```
if 1+1 == 3:
    mysterieus_getal = -0
```

e) Kun je een programmaatje schrijven dat de gebruiker vraagt om een getal en de gebruiker vervolgens vertelt of dit getal groter dan, kleiner dan of gelijk is aan het getal `1000`? Sla het programma op in zijn eigen .py bestand

Overzicht

- We kunnen een if statement gebruiken om te bepalen of iets wel of niet uitgevoerd moet worden
- Een if statement begint met het keyword if gevolgd door een conditie (voorwaarde) en daarna een dubbele punt
- Alle code in het statement moet beginnen met een tab om aan te geven dat de code bij het if statement hoort. Dit noemen we een code block

Hoofdstuk 4-4 – Boolean operaties

Soms hebben we meer dan één conditie (voorwaarde) waaraan voldaan moet worden voordat bepaalde code mag worden uitgevoerd. Hier gebruiken we de **and** (en) operator voor.
Bijvoorbeeld:

```
if 1+1 == 2 and 3 < 4:
      print(True)
```

Hier zeggen we dat 1+1 gelijk aan 2 moet zijn én dat 3 kleiner dan 4 moet zijn om de code `print(True)` uit te voeren. Beide voorwaarden moeten waar zijn. Het volgende voorbeeld wordt daarom niet uitgevoerd. Zie je waarom?

```
if 1+1 == 2 and 3 < 2:
      print(True)
```

We kunnen ook zeggen dat op zijn minst één van de voorwaarden waar moet zijn. Dit doen we met de **or** (of) operator.
Bijvoorbeeld:

```
if 1+1 == 2 or 3 < 2:
      print(True)
```

De eerste voorwaarde (`1+1 == 2`) is wel waar en de tweede (`3<2`) niet. Omdat we de or operator hebben gebruikt is dit voldoende om het if statement uit te voeren.

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd and_or.py en sla het op

c) Kun je een programmaatje schrijven dat de gebruiker om een getal vraagt en de gebruiker vertelt of het getal tussen de `100` en de `1000000` ligt?
Tip: Gebruik de **`and`** operator

d) Kun je een programmaatje schrijven dat de gebruiker om een getal vraagt en de gebruiker vertelt of het getal gelijk is aan 2, 4, 8, 16, of 32?
Tip: Je kunt de **`or`** operator meerdere keren achter elkaar gebruiken

Overzicht

- We kunnen een if statement meer dan één voorwaarde meegeven, dit doen we met de and en or operators
- De and operator gebruik je als beide of alle voorwaarden waar moeten zijn
- De or operator gebruik je als slechts één van de voorwaarden waar hoeft te zijn

Hoofdstuk 4-5 – Elif

Naast het if statement hebben we ook het **elif** statement. Elif is een afkorting voor *else if* wat betekent: anders als. Het werkt precies zo als een if statement, echter het wordt alleen maar bekeken als de conditie van het if statement daarboven niet waar is. Je zegt dus eigenlijk 'als het if statement hier boven niet waar is, maar dit elif statement wel, doe dan dit'

Laten we eens naar een voorbeeld kijken:

```
if 1+1 == 2:
     print("eerste if")
if 2+2 == 4:
     print("tweede if")
```

In dit voorbeeld hebben we twee if statements na elkaar. Beide voorwaarden zijn altijd waar, dus als we dit programma draaien zien we beide strings geprint op het scherm. Laten we het tweede statement eens veranderen in een elif statement:

```
if 1+1 == 2:
     print("eerste if")
elif 2+2 == 4:
     print("tweede if")
```

In dit voorbeeld zijn beide condities waar, echter zal `"tweede if"` nooit geprint worden, omdat het if statement daarboven altijd waar is. In het volgende voorbeeld wordt alleen “tweede if” geprint. Zie je waarom?

```
if 1+1 == 3:
     print("eerste if")
elif 2+2 == 4:
     print("tweede if")
```

Een elif statement moet altijd onder een if statement of een ander elif statement komen te staan.

Opdracht 1

a) Je kunt meerdere elif statements onder elkaar plaatsen. Bijvoorbeeld:

```
if 1+1 == 3:
     print("eerste if")
elif 2+2 == 5:
     print("tweede if")
elif 2+7 == 9:
     print("derde if")
elif 10*2 == 20:
     print("vierde if")
```

Kun je de uitkomst bedenken als je deze code zou draaien?

b) Open de IDE, maak een nieuw .py bestand aan, sla het op en draai de code van vraag a om te kijken of je antwoord klopt

c) Maak een nieuw document aan en sla het op als levensfase.py

d) Schrijf een programmaatje dat de gebruiker vraagt hoe oud hij of zij is en laat het programmaatje de gebruiker vertellen in welke levensfase hij zit. Je zou de levensfases als volgt kunnen indelen:
   1. 0-1: baby
   2. 1-4: kleuter
   3. 4-12: kind
   4. 12-20: puber
   5. 20-40: jonge volwassene
   6. 40-75: volwassene
   7. 75 en hoger: bejaarde

   Je mag ook je eigen indeling maken, zorg er dan voor dat je minimaal 4 levensfases gebruikt

Overzicht

- Het elif statement werkt hetzelfde als een het if statement, met als verschil dat het statement alléén wordt bekeken als de if/elif statements daarboven niet waar blijken te zijn
- Je kunt meerdere elif statements onder elkaar plaatsen
- Een elif statement volgt altijd na een if statement of een ander elif statement

Hoofdstuk 4-6 – Else

We hebben nog een derde logica element. Het **else** (anders) statement. Het else statement is heel simpel. Laten we het eens bekijken in het volgende voorbeeld:

```
if 1+1 == 3:
     print("eerste if")
elif 2+2 == 5:
     print("tweede if")
elif 2+7 == 10:
     print("derde if")
else:
     print("else")
```

Het else statement wordt uitgevoerd als alle statements daarboven niet waar blijken te zijn. Dit betekent dus ook dat een else statement geen voorwaarde nodig heeft. Je zegt eigenlijk 'in alle andere gevallen, doe dit'.
Een else statement komt altijd onder een if statement of een elif statement.

Opdracht 1

a) Open de IDE

b) Maak een nieuw document aan en sla het op als else.py

c) Schrijf een programmaatje dat de gebruiker vraagt om een wachtwoord. Bepaal zelf wat het juiste wachtwoord is en vertel de gebruiker een geheim als hij of zij het juiste wachtwoord geeft. Vertel de gebruiker in alle andere gevallen dat hij of zij het verkeerde wachtwoord heeft gegeven en geen toegang krijg tot het geheim

d) Draai het programma

Overzicht

- Een else statement is een if statement zonder voorwaarde
- Je zegt ermee 'in alle andere gevallen, doe dit'
- Een else statement is altijd het laatste statement in een serie logica
- Een else statement staat altijd onder een if statement of een elif statement

Hoofdstuk 4-7 – De remainder operator

We kunnen Python gemakkelijk vragen wat de rest is van een breuk. De rest is wat je overhoudt als je een getal deelt door een ander getal, zoals bij een staartdeling. Bijvoorbeeld als je 6 door 3 deelt is de rest 0. Het getal 3 past namelijk precies 2 keer in het getal 6. Als we 7 door 3 zouden delen houden we 1 over. De rest van 7/3 is dus 1. We kunnen Python de rest van een breuk laten uitrekenen. Dit doen we met de `%` operator. Dit noemen we de remainder (rest) operator. Bijvoorbeeld:
`7%3` geeft `1`

Opdracht 1

a) Open de interpreter en oefen met de remainder operator. Kun je de uitkomst voorspellen van de volgende opdrachten?
   1. `8%2`
   2. `10%3`
   3. `20%5`
   4. `21%4`
   5. `18%17`

b) Open de IDE

c) Maak een nieuw document aan en sla het op als even_oneven.py

d) Schrijf een programmaatje dat de gebruiker vraagt om een getal en de gebruiker vervolgens vertelt of het getal even of oneven is

e) Draai het programma

Overzicht

- We kunnen de `%` operator gebruiken om de rest van een breuk te vragen. Dit noemen we de remainder operator

Eindopdracht

Schrijf een programma dat de gebruiker drie sommetjes geeft, checkt of de uitkomsten kloppen en als de gebruiker alles goed heeft `"Gefeliciteerd!"` op het scherm print

Hoofdstuk 5 – Loops

Hoofdstuk 5-1 – De While Loop

In dit hoofdstuk gaan we kijken naar loops. Een **loop** is een stukje code dat zich kan herhalen. Het meest simpele loopje dat we kunnen maken is een **while** (terwijl) **loop**. Een while loop is een stukje code dat een conditie (een voorwaarde) meekrijgt en zolang deze conditie waar is, blijft de code in het loopje zich herhalen. Als de conditie niet meer waar is, gaat het programma verder met de code na de while loop. De code in een while loop moet, net als bij een if statement, in zijn eigen code block staan. Laten we eens naar een voorbeeld kijken:

```
i = 10
while i > 0:
      print(i)
      i = i-1
```

Opdracht 1

a) Kun je bedenken wat het voorbeeld hierboven doet?

b) Open de IDE

c) Maak een nieuw bestand aan genaamd while_loop.py en sla het op

d) Type het voorbeeld van hierboven over en voer het uit. Doet het ook daadwerkelijk wat je dacht bij opdracht a?

e) Kun je het programma zo schrijven dat het in plaats van aftelt tot nul, optelt tot 100?

f) Kun je uitleggen waarom `i=10` hierboven buiten de loop staat?

g) Kun je een programmaatje schrijven dat je vijf keer vraagt om een getal en de gebruiker daarna vertelt wat het grootste getal was? Gebruik een while loop!

h) Kun je een infinite (oneindige) while loop maken? Dat wil zeggen een while loop die nooit meer ophoudt

i) Wat is de simpelste infinite while loop die je kunt bedenken?

j) Kun je uitleggen waarom infinite while loops een probleem kunnen zijn?

Overzicht

- Een while loop is een stukje code dat zich herhaalt zolang de conditie waar is
- De code in een while loop moet in zijn eigen code block staan

Extra informatie
In plaats van `i = i + 1` of `i = i - 1` kun je ook de `+=` operator of de `-=` operator gebruiken. Dit ziet er dan zo uit:
`i=i+1` wordt `i+=1`
en
`i=i-1` wordt `i-=1`

Deze operators tellen/trekken het getal rechts van de operator op/af bij het getal rechts van de operator en slaan het vervolgens op in de oorspronkelijke variabele. Er is geen daadwerkelijk verschil tussen de twee manieren. Het een is gewoon een kortere manier van hetzelfde opschrijven.

Hoofdstuk 5-2 – Break en continue

Als we met loops werken dan zijn er twee keywords die we vaak zullen gebruiken. Dit zijn **break** (afbreken) en **continue** (doorgaan). Break zorgt ervoor dat de while loop stopt en niet meer herhaald wordt. Continue zorgt er ook voor dat de loop stopt, echter gaat vervolgens door met de volgende herhaling. Bijvoorbeeld:

```
while True:
      print("hierna stop ik")
      break

while True:
      print("hierna ga ik verder")
      continue
      print("dit print ik dus niet")
```

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd break_continue.py en sla het op

c) Voer het voorbeeld van hierboven uit. Deed het wat je verwachtte?

d) Schrijf een programmaatje dat optelt tot 1000 en elk getal dat eindigt op een 0 weglaat. Gebruik hiervoor een infinite while loop en gebruik het keyword break om uit de loop te stappen

e) Heb je een continue keyword gebruikt om de getallen die eindigen op een 0 niet te printen? Zo nee, zie je hoe je dat kan doen?

Overzicht
- Met het keyword break stop je de while loop. Deze wordt niet meer herhaald
- Met het keyword continue stop je de while loop, deze wordt daarna weer herhaald

Hoofdstuk 6 – The Python Standard Library

Python bevat een grote **library** (blibliotheek) aan functies en code die andere mensen geschreven hebben. Slechts een paar van de functies in Python staan altijd tot onze beschikking, de rest staat in de Python Standard Library. Als we één van de functies uit de standard library willen gebruiken, moeten we deze eerst in onze code importeren. Dit doen we met het **import** keyword. De standard library is opgedeeld in modules (onderwerpen). Het makkelijkst is om de hele module die je nodig hebt te importeren.

Hoofdstuk 6-1 – De random module

Laten we eens kijken hoe de we Python willekeurige getallen kunnen laten generen. Hiervoor moeten we de **random module** importeren. Dit ziet er als volgt uit:

```
import random
```

Nu we de random module geïmporteerd hebben kunnen we alle functies gebruiken die onderdeel zijn van die module. Eén van die functies is de **randrange()** functie. De randrange functie geeft je een willekeurig getal terug. De randrange functie heeft minimaal twee argumenten nodig: start (begingetal) en stop (eindgetal).

Omdat de randrange functie in de random module zit, roep je de randrange functie als volgt aan:

```
random.randrange(0,10)
```

Dit zal ons een getal vanaf 0 en tot 10 geven. Let op: de mogelijke uitkomsten van de code hierboven zijn 0,1,2,3,4,5,6,7,8 en 9. Het getal 0 is wel onderdeel van de range en het getal 10 niet meer. Als je wilt dat 10 ook een mogelijke uitkomst is, zul je dus `random.randrange(0,11)` moeten schrijven.

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd random_module.py en sla het op

c) Schrijf een programmaatje dat een willekeurig getal tussen 100 en 200 print

d) Schrijf een programmaatje dat een willekeurig **even** getal tussen 100 en 200 print

e) Schrijf een programmaatje dat een willekeurig getal tussen 1 en 10 kiest en de gebruiker daarna net zo lang laat raden tot hij het getal geraden heeft

f) Schrijf een programmaatje dat net zo lang willekeurige getallen tussen 0 en 100 op het scherm print tot het laatste getal 44 is

Overzicht

- Modules uit de Python standard library importeren we met het import keyword
- De random module bevat functies die willekeurige resultaten opleveren
- De randrange functie geeft een willekeurig getal terug in een bepaald bereik
- De randrange functie is onderdeel van de random module, dus wordt aangeroepen met `random.randrange()`
- De randrange functie heeft minimaal twee argumenten nodig: start (begingetal) en stop (eindgetal)

Extra informatie

Je kunt de randrange functie ook een derde argument meegeven, step (stapgrootte). De code hieronder heeft als mogelijke uitkomsten bijvoorbeeld 10, 20, 30, 40 en 50:

```
random.randrange(10,51,10)
```

Als je het step argument weglaat, gaat Python er automatisch vanuit dat de stapgrootte 1 is.

Kijk eens of je opdracht d van hierboven nu kunt doen zonder de `%` operator te gebruiken.

Hoofdstuk 6-2 – De os module

Een andere module in de standard library is de **os** (operating system) **module**. Met de os module kun je met het besturingssysteem (operating system) praten. We gaan nu nog niet veel met de os module doen, er is echter een functie die toch wel handig is om te kennen en dat is de **system functie**. Met de system functie kunnen we namelijk het scherm leegmaken. Dit doen we als volgt:

```
import os
os.system('clear')
```

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd os.py en sla het op

c) Schrijf een programmaatje dat aftelt vanaf 100000 en dan het scherm leeg maakt

Overzicht

- Met de os module kun je met het besturingssysteem praten
- Je kunt het scherm leegmaken met `os.system('clear')`

Hoofdstuk 7 – Lijsten

In dit hoofdstuk gaan we naar **lijsten** kijken. Een lijst is een verzameling van objecten met een vaste volgorde. We definiëren een lijst met `[]`. Een lege lijst definiëren we als volgt:

```
een_lege_lijst = []
```

We kunnen ook een lijst definiëren waar al objecten in zitten. Bijvoorbeeld een lijst met drie integers:

```
lijst_met_getallen = [1,2,3]
```

Een lijst kan objecten van verschillende datatypen bevatten:

```
een_gemengde_lijst = [1,2.5,"string",True]
```

Als we een specifiek object uit een lijst willen hebben, kunnen we dat doen met de **index** operator. Dit ziet er als volgt uit:

```
een_gemengde_lijst[0]
```

Dit geeft ons het eerste object in de lijst. In dit geval het getal 1.
LET OP! De computer begint met tellen bij 0. Dit betekent dat in een lijst met 4 objecten de eerste index 0 is en de laatste index 3.

Opdracht 1

1. Open de IDE

2. Maak een nieuw bestand aan en sla het op als lijsten.py

3. Schrijf een programmaatje waarin je een lijst definieert met 5 namen en vervolgens de eerste, de middelste en de laatste naam op het scherm print

Overzicht

- Een lijst is een verzameling objecten met een vaste volgorde
- Een lijst definieer je met `[]`
- Een lijst kan verschillende datatypen bevatten
- We vragen een object uit een lijst op met de index
- Het tellen van de index begint bij 0

Hoofdstuk 7-1 – De index operator en de len() functie

We kunnen de index operator niet alleen gebruiken om een specifiek object uit een lijst te halen, maar ook om een specifiek karakter uit een string te halen. Je kunt je een string in die zin voorstellen als een lijst van karakters met een vaste volgorde. Bijvoorbeeld:

```
een_string = "abcdefg"

een_string[2]
```

Dit geeft ons de derde letter (want we beginnen te tellen bij 0) uit de string `een_string`. In dit geval de letter `'c'`.

Omdat we vaak niet precies weten hoe lang een string of een lijst is, hebben we de **`len()`** functie. Len is kort voor length (lengte) en geeft ons de lengte van de string of lijst die we meegeven als het eerste argument.

Bijvoorbeeld:

```
een_string = "abcdefg"

len(een_string)
```

In dit voorbeeld geeft de **`len()`** functie de lengte van `een_string` terug. In dit geval 7.

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd bijnaam.py en sla het op

c) Schrijf een programmaatje dat de gebruiker om zijn bijnaam vraagt en de gebruiker vervolgens vertelt hoeveel karakters zijn bijnaam heeft

d) Kun je het programmaatje zo schrijven dat het zich blijft herhalen totdat de gebruiker QUIT invoert als naam?

e) Kun je de gebruiker ook vertellen wat de eerste en laatste letter van zijn bijnaam zijn?

Overzicht

- De index operator kun je gebruiken bij lijsten en bij strings
- De **`len()`** functie geeft ons de lengte van een lijst of string

Hoofdstuk 7-2 – Lijst operaties

Lijst operaties zijn operaties die we op lijsten kunnen uitvoeren. Dit houdt operaties in zoals een object aan het einde van de lijst toevoegen, objecten midden in de lijst invoegen, objecten uit de lijst verwijderen, etc.

Lijst operaties zijn functies die onderdeel zijn van de lijst. Functies die onderdeel zijn van een ander object roepen we aan door ze aan het object vast te typen met een een punt, zo dus:

```
object.functie()
```

Laten we eens kijken hoe dat eruitziet met een lijst en de `append()` functie. Met de `append()` functie kunnen we een nieuw object het einde van een lijst toevoegen. Bijvoorbeeld:

```
een_lijst = [1,2,3,4]

een_lijst.append(5)
```

Wat we hier doen is het getal 5 aan de lijst `een_lijst` toevoegen. De lijst is nu: `[1,2,3,4,5]`

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan en sla het op

c) Kun je bijnaam.py van Hoofdstuk 7-1 zo aanpassen dat het programma alle bijnamen die het krijgt opslaat in een lijst en dan als de gebruiker de naam PRINT invoert hij deze lijst te zien krijgt?

Naast append is er nog een aantal lijst operaties.

Voor het toevoegen van objecten hebben we ook nog de `insert()` functie. We kunnen de `insert()` functie gebruiken om een object op een andere plaats dan het einde van de lijst toe te voegen. De insert functie neemt 2 argumenten. Het eerste argument is de index (plaats) binnen de lijst waar het het object moet worden toegevoegd en het tweede argument is het object dat je toe wilt voegen. Dit ziet er als volgt uit:

```
een_lijst = [1,2,3,4]

een_lijst.insert(2, 88)
```

Wat we hier doen is het getal 88 aan de lijst `een_lijst` toevoegen op index 2. De lijst is nu:

```
[1,2,88,3,4]
```

Voor het verwijderen van een object hebben we twee functies: de remove() functie en de pop() functie. Remove haalt het eerste object uit een lijst dat gelijk is aan het argument en pop krijgt een index mee en haalt dan het bijbehorende object uit de lijst. Bijvoorbeeld:

```
een_lijst = [1,2,'Vrijdag',1882, False,'Vrijdag']

een_lijst.remove('Vrijdag')
```

Hiermee halen we de string `'Vrijdag'` uit de lijst. We hoeven niet te weten op welke plek (index) `'Vrijdag'` staat. De tweede `'Vrijdag'` blijft staan. De lijst is nu:

```
[1,2,1882,False,'Vrijdag']
```

Laten we eens kijken wat de `pop()` functie doet:

```
een_lijst.pop(0)
```

Hiermee halen het object met index 0 uit de lijst. De lijst is nu:

```
[2,1882,False,'Vrijdag']
```

Pop geeft het object dat je uit de lijst hebt gehaald terug, probeer het volgende maar eens:

```
jaartal = een_lijst.pop(1)
print(jaartal)
```

De lijst is nu `[2,False,'Vrijdag']` en `jaartal` is `1882`.

Als laatste hebben we de `clear()` functie. Clear maakt de lijst helemaal leeg en is simpel:

```
een_lijst.clear()
```

We hebben de lijst leeg gemaakt. De lijst is nu:

```
[]
```

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand en sla het op als lijst_operaties.py

c) Schrijf een programmaatje dat vijf getallen in een lijst zet en vervolgens de *middelste* teruggeeft

d) Kun je een programmaatje schrijven dat:
– De gebruiker vraagt om te kiezen tussen:
1. Een naam toevoegen aan de lijst
2. Een naam verwijderen uit de lijst
3. De lengte van de lijst geven
4. De lijst de printen
5. De lijst legen

– Doet wat de gebruiker vraagt en opnieuw de vijf opties geeft

e) Kun je het programma van opdracht d laten checken of de gebruiker het zeker weet als hij vraagt om de lijst te legen?

Overzicht

- Lijst operaties roep je aan door de naam van de lijst, een punt en dan de naam van de functie te typen
- De `append()` functie voegt een object toe aan het einde van de lijst en krijgt als argument het object om toe te voegen
- De `insert()` functie voegt een object toe op een specifieke index in de lijst en krijgt als argumenten de index en het object om toe te voegen
- De `remove()` functie verwijdert het eerste voorkomen van een object uit de lijst en krijgt als argument het object om te verwijderen
- De `pop()` functie verwijdert een object uit de lijst en krijgt als argument de index van het object om te verwijderen. De `pop()` functie geeft het verwijderde object terug
- De `clear()` functie maakt de lijst leeg en krijgt geen argumenten

Hoofdstuk 7-3 – De in operator

Met de **in** operator kun je checken of iets in een lijst zit of onderdeel is van een string. Bijvoorbeeld:

```
lijst = [1,10,100,1000]

if 1000 in lijst:
      print(True)
else:
      print(False)
```

of voor een string:

```
string = "abcdefg"

if "bc" in string:
      print(True)
else:
      print(False)
```

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd in_operator.py en sla het op

c) Schrijf een programmaatje dat 10 willekeurige getallen tussen 0 en 100 in een lijst zet en de gebruiker drie keer laat raden en als de gebruiker een getal raadt dat in de lijst zit, dat getal uit de lijst haalt en nog drie keer laat raden, net zolang tot de gebruiker drie keer een getal raadt dat niet in de lijst staat

d) Kun je bijnaam.py van hoofdstuk 7-1 zo aanpassen dat het bijnamen nooit dubbel in de lijst zet?

Overzicht

- De **in** operator kijkt of een bepaald object of karakter in een lijst of string zit

Eindopdracht

Schrijf een programmaatje voor je cijferlijst! Het moet het volgende kunnen:

- Cijfers toevoegen
- Cijfers verwijderen
- Je cijferlijst printen
- Je gemiddelde cijfer berekenen
- Je vertellen wat je hoogste cijfer en je laagste cijfer zijn
- De cijferlijst leeg maken

Bedenk zelf hoe je de gebruiker laat kiezen tussen de opties!

Hoofdstuk 8 – Functies

We hebben al een boel functies gebruikt zoals de `input()` functie en de `print()` functie. In dit hoofdstuk gaan we leren hoe we zelf functies kunnen schrijven. Laten we eens kijken wat we al over functies weten:

- een functie heeft een naam
- een functie doet iets voor ons
- een functie kan input krijgen in de vorm van 1 of meerdere argumenten (wat je tussen de haakjes aan de functie meegeeft)
- een functie kan output terug geven in de vorm van een return value. Dit is wat de functie teruggeeft

Hoofdstuk 8-1 – Functies definiëren

De manier waarop functies werken is door code voor ons uit voeren. Een nieuwe functie definiëren we met het keyword **`def`** (define):

```
def kwadrateer(getal):
    kwadraat = getal * getal
    return kwadraat
```

Deze functie geeft je het kwadraat van `getal`; het argument dat je aan de functie meegeeft.
De functie zal dan overal waar het woord `getal` staat jouw input gebruiken.
Met het keyword **`return`** geef je aan wat de functie terug moet geven.
Je kunt deze functie nu als volgt gebruiken:

```
kwadrateer(12)
```

Dit geeft dan 144 terug.

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd functies.py en sla het op

c) Neem de functie van hierboven over en probeer de functie uit door het kwadraat van 333 op het scherm te printen

d) Kun je een functie schrijven die in plaats van het kwadraat de 10^{e} macht van een getal geeft?

e) Kun je een hello_world functie schrijven? Dat wil zeggen een functie die hello world op het scherm print

f) Kun je een hello_jouw_naam functie schrijven? Dat wil zeggen een functie die je begroet met je naam. Hint: gebruik je naam als argument

g) Kun je een functie schrijven die je vraagt om ja of nee te antwoorden en vervolgens een boolean teruggeeft (`True` of `False`)? Dat wil zeggen `True` teruggeeft als het antwoord ja is, `False` teruggeeft als het antwoord nee is en de vraag herhaalt als het antwoord wat anders dan ja of nee is

h) Kun je een functie schrijven die je een lijst geeft met 10 willekeurige getallen tussen de 20 en de 50?

i) Kun je de functie ook zo schrijven dat je zelf het aantal getallen in de lijst en het bereik van de getallen kunt bepalen? Het bereik houdt in tussen welke getallen de functie een getal kiest. Je functie heeft dan dus drie argumenten

j) Kun je een functie schrijven die de tafel van een willekeurig getal kan printen?

Overzicht

- Een functie heeft een naam
- Een functie doet iets voor ons
- Een functie kan input krijgen in de vorm van argumenten
- Een functie kan output teruggeven in de vorm van een return value
- Een functie definieer je met het keyword `def`
- Wat de functie teruggeeft geef je aan met het keyword `return`

Hoofdstuk 9 – Try en Except

Soms weten we niet zeker of code die we schrijven een **error** gaat opleveren. In dit geval kunnen we gebruik maken van de keywords **`try`** (probeer) en **`except`** (uitzondering). In de appendix kun je even oefenen met errors. Bekijk de volgende code eens:

```
try:
      1 / 0
except:
      print('Kan niet delen door nul!')
```

Als je deze code draait gaat Python "proberen" de code uit te voeren die in het code block van **`try`** staat. Als deze code echter een error oplevert, voert Python de code uit die in het code block van **`except`** staat, maar anders niet. Dit werkt net zoals bij **`if`** en **`elif`**, alleen zeg je hier dus eigenlijk "Probeer dit, en als dat niet lukt, doe dan dit".

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan genaamd try_except.py en sla het op

c) Schrijf een functie genaamd int_input(), die net zolang om input vraagt tot de gebruiker een getal heeft ingevoerd, en dit getal als return value geeft

Overzicht

- Met het keyword **`try`** vraag je Python een code block te proberen uit te voeren
- Met het keyword **`except`** geef je aan wat je wilt dat er gebeurt als de code by **`try`** een error oplevert
- Als de code by **`try`** geen error oplevert, wordt de code by **`except`** overgeslagen

Eindopdracht

Schrijf een programmaatje dat:

- De gebruiker willekeurige sommetjes geeft om op te lossen, zorg dat je optellen, aftrekken en vermenigvuldigen gebruikt, laat delen achterwege
- De gebruiker laat kiezen hoe moeilijk de sommen zijn
- Gebruik maakt van de int_input() functie om te kijken of de gebruiker een getal geeft als antwoord
- Laat zien hoeveel procent de gebruiker goed heeft

Appendix

Machtsverheffen

Machtsverheffen is een operatie uit de wiskunde, net zoals plus, min, keer en gedeeld door. Als we een getal bijvoorbeeld 'tot de macht twee' doen, wil dat zeggen dat we het getal twee keer met zichzelf vermenigvuldigen. Drie tot de macht twee is dus hetzelfde als drie keer drie. Twee tot de macht drie is hetzelfde als twee keer twee keer twee. Zes tot de macht vier is zes keer zes keer zes keer zes, enzovoort. In wiskunde schrijf je ze zo:

Twee tot de macht drie is 2^3
Zes tot de macht vier is 6^4

Machten zijn handig, omdat 12^5 veel korter is om te schrijven dan 12 x 12 x 12 x 12 x 12.

In Python schrijven we machten met `**`, 12^5 is dus `12 ** 5`

Oefenen met errors

Je zult nu wel hebben gemerkt dat het soms gebeurt dat je een **error** (foutmelding) krijgt op het moment dat je code uitvoert die niet werkt. In dit hoofdstuk gaan we kort kijken naar die foutmeldingen en wat we kunnen doen om ze te vermijden of op te lossen. We beginnen met een paar voorbeelden:

Opdracht 1

a) Open de IDE

b) Maak een nieuw bestand aan en sla het op als errors.py

c) Voer de volgende code in:

```
variabele1 = 1 / 0
variabele2 = int('string')
variabele3 = 'string' + 35
```

d) Voer het programma uit

Python geeft je nu als het goed is een error, die ziet er als volgt uit:

```
Traceback (most recent call last):
  File "errors.py", line 1, in <module>
    variabele1 = 1 / 0
ZeroDivisionError: division by zero
```

Als Python een error geeft, vertelt het je de **naam** van het bestand (hier errors.py), de **regel** waarin de error plaatsvindt (hier line 1) en het **soort** error (hier ZeroDivisionError). Python vertelt ons dus dat in het bestand errors.py er een error heeft plaatsgevonden in regel 1 en dat het hier gaat om een deling door nul. Als we de eerste regel nu als volgt opschrijven hebben we het probleem opgelost:

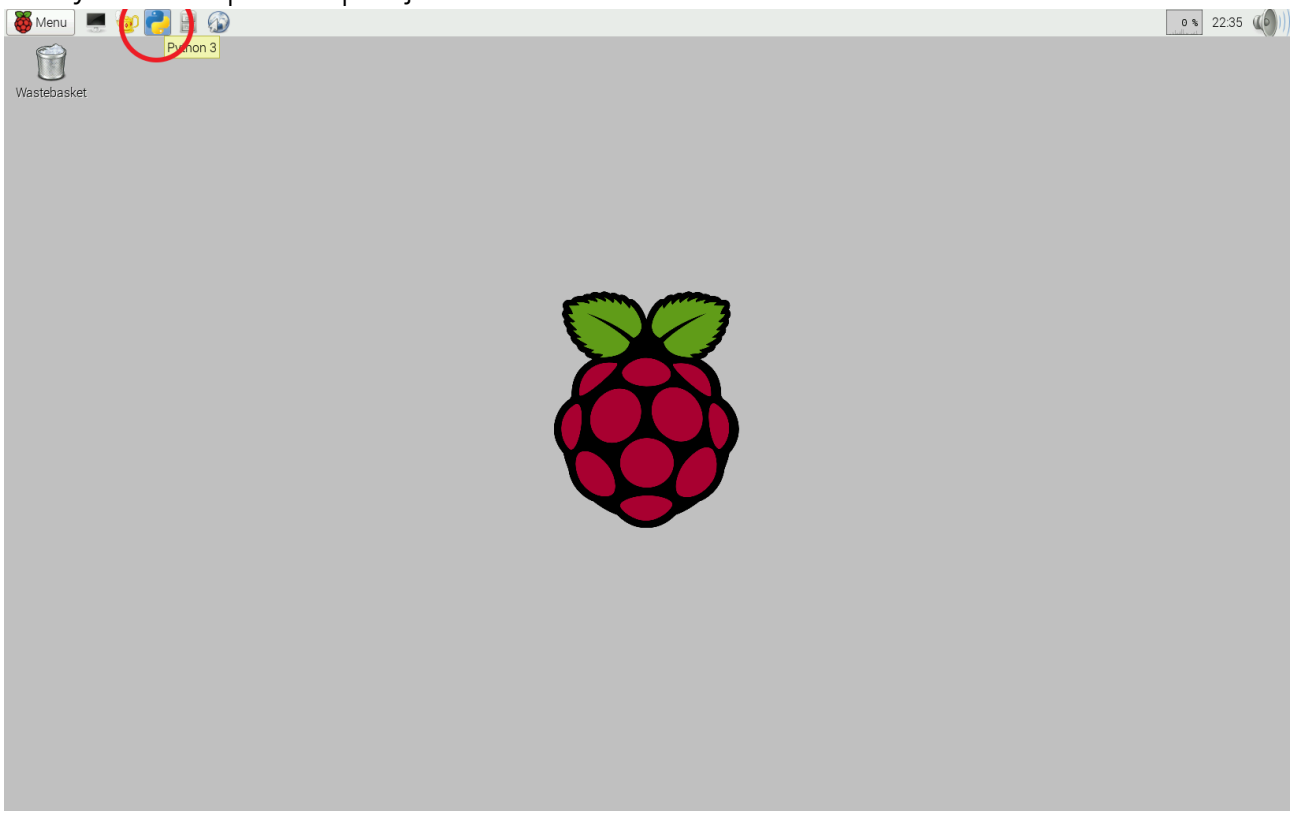
```
variabele1 = 4 / 2
```

Opdracht 2

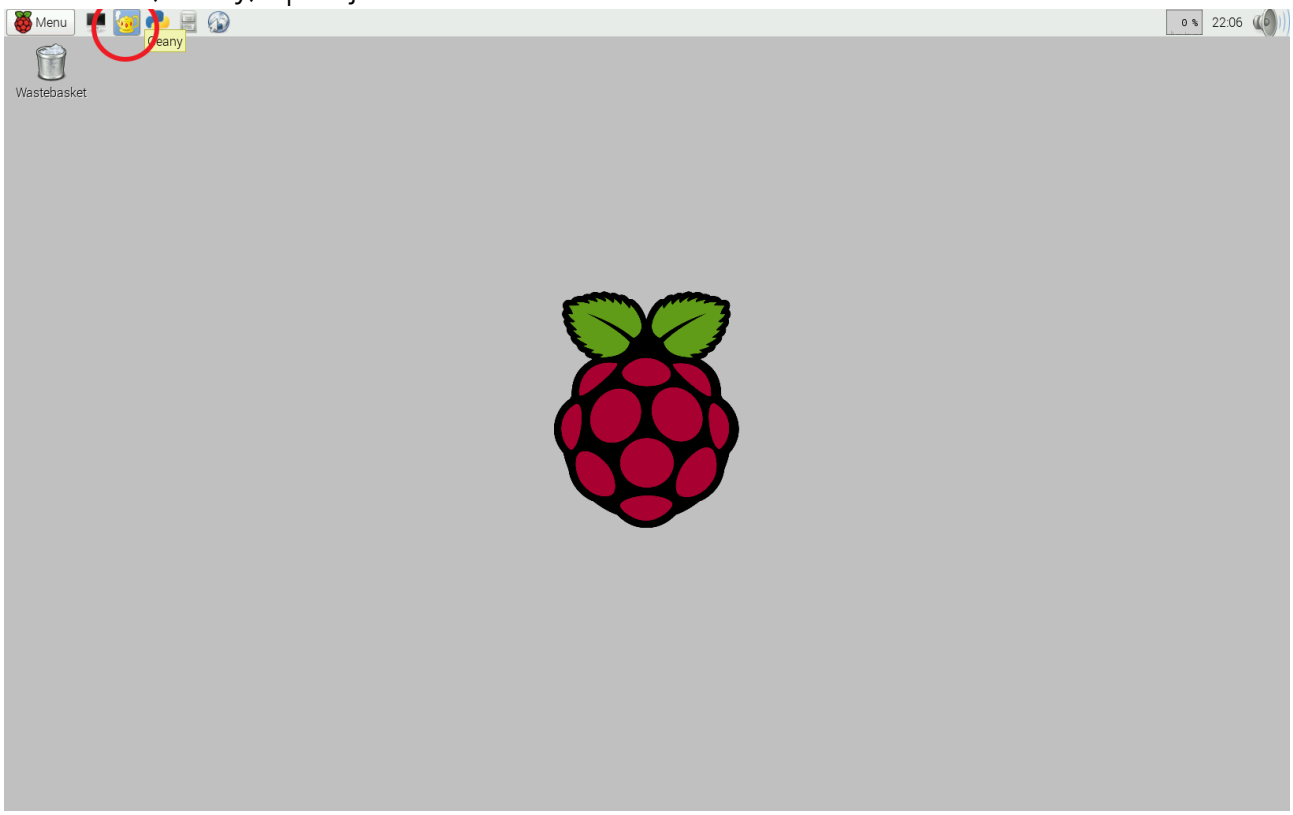
a) Verbeter regel 1 van errors.py zoals hierboven beschreven en draai het opnieuw. Je krijgt nu een andere foutmelding, nu in regel 2. Kun je bedenken wat er fout gaat en hoe je het moet oplossen?

b) Als opdracht a gelukt is, heb je een error gekregen in regel 3. Kun je deze ook verbeteren?

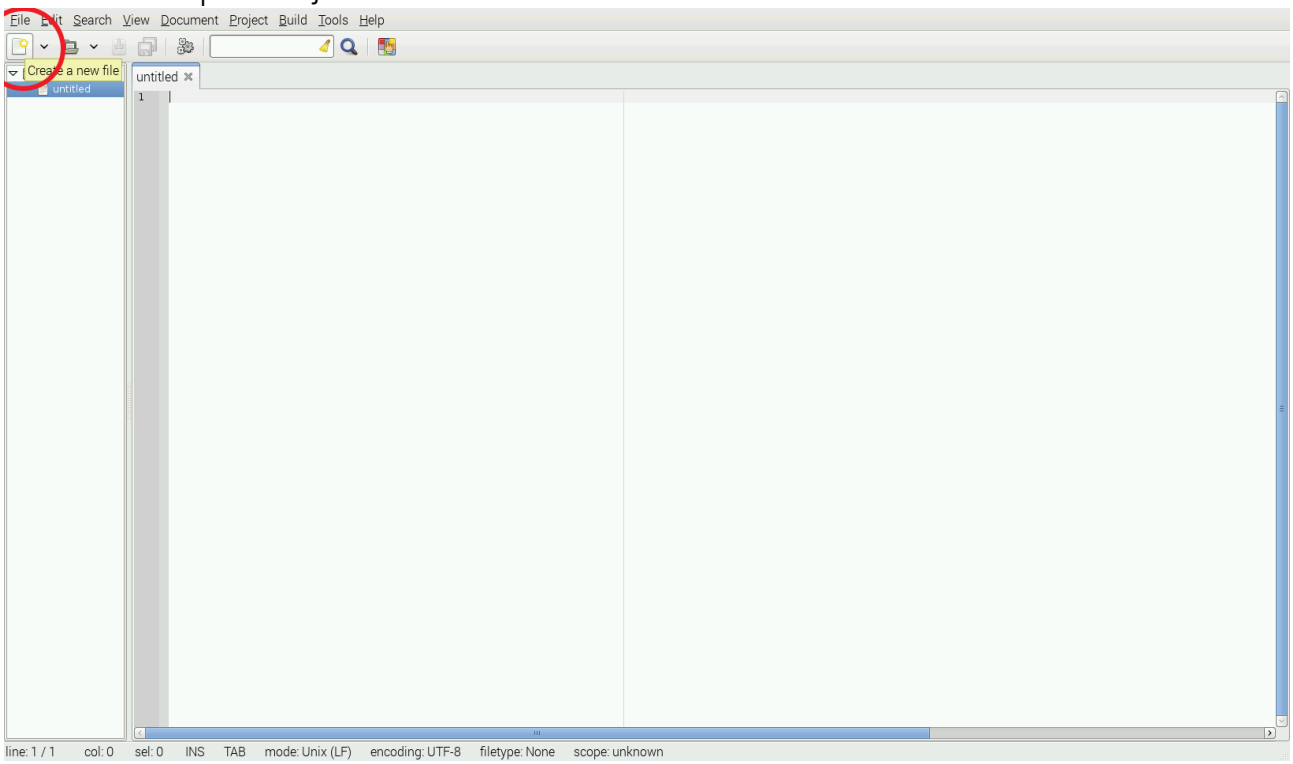
De Python interpreter open je hier:



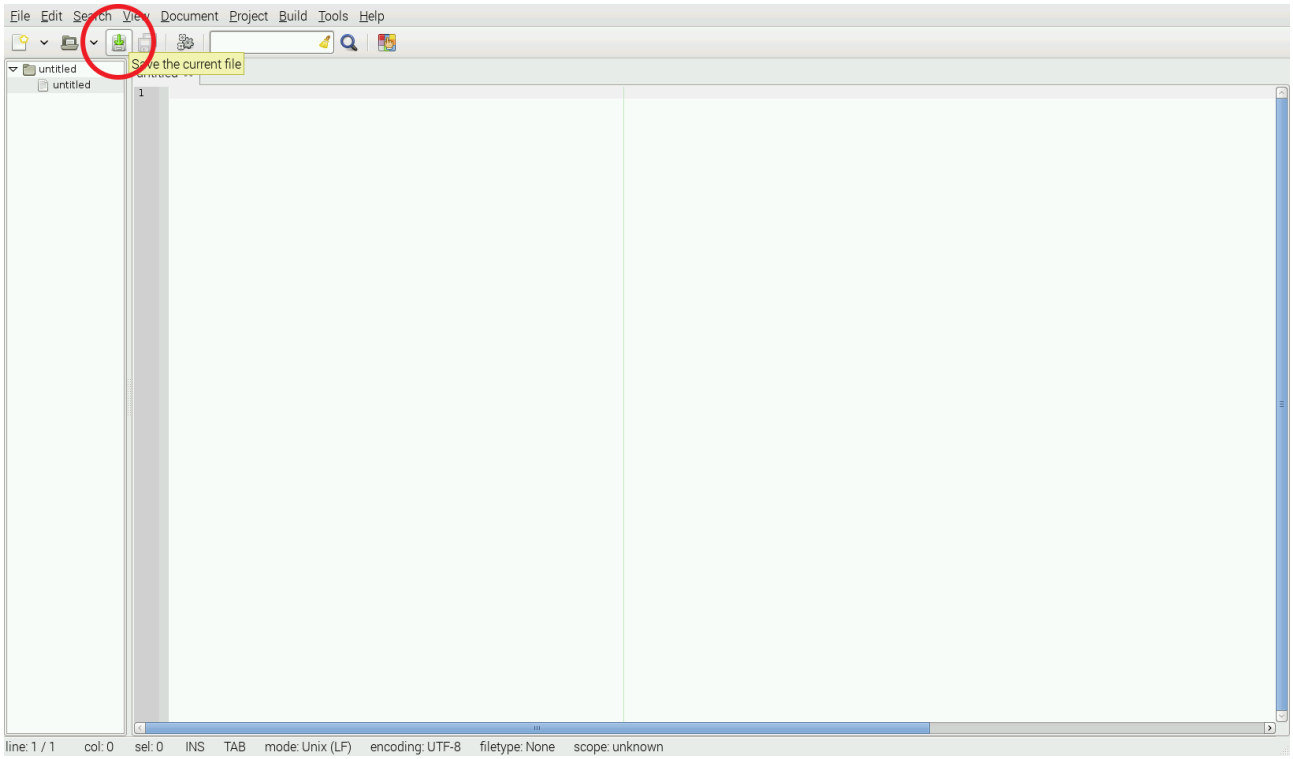
Onze IDE (Geany) open je hier:



Met deze knop maak je een nieuw bestand aan:



Met deze knop sla je je bestand op:



Met deze knop voer je je programma uit: