INFORMATICAWETENSCHAPPEN

PYTHON PROGRAMMEREN

UHASSELT

DOOR JURGEN NIJS

Inhoudopgave

Inleiding

Welkom bij deze cursus voor het leren van Python. Deze cursus is ontworpen in overleg met Prof. Dr. Frank Neven (UHasselt) en is speciaal ontworpen ter ondersteuning van het vak informaticawetenschappen.

In dit handboek zullen we je begeleiden op een boeiende reis door de wereld van programmeren en computationeel denken met behulp van de veelzijdige programmeertaal Python.

Python is niet alleen een krachtige taal die wordt gebruikt door professionals in de industrie, maar ook een uitstekende keuze voor beginners vanwege zijn leesbaarheid en eenvoudige syntax. In dit handboek zullen we je stap voor stap door de basisconcepten van programmeren leiden, zoals variabelen, datatypes, conditionele verklaringen en lussen.

Ons doel is niet alleen om je vertrouwd te maken met Python als programmeertaal, maar ook om je te helpen bij het ontwikkelen van het analytisch denken en probleemoplossend vermogen dat essentieel is binnen de informaticawetenschappen. Elk hoofdstuk bevat praktische voorbeelden en oefeningen die zijn afgestemd op de uitdagingen die je tegenkomt in je studie.

Dit handboek biedt je een crash course aan. Door het beperkt aantal uren dat ter beschikking is voor informaticawetenschappen (i22n) zijn er drastisch keuzes gemaakt. Het aantal instructies werd beperkt, maar met doel met een zo klein mogelijk set van instructies, een zo groot mogelijk aantal problemen te kunnen oplossen. We hopen dan ook dat dit handboek je voldoende competent maakt en van Python laat proeven om zelf de ontdekkingstocht verder te zetten.

Dus pak je toetsenbord en laten we samen de fascinerende wereld van Python en informaticawetenschappen verkennen!

Deze cursus is geschreven door Nijs Jurgen, educatief medewerker van de School voor Educatieve Studies van UHasselt in samenwerking met Prof. Dr. Frank Neven, informatica faculteit UHasselt.

Het algoritme van dit handboek

Deze cursus is opgebouwd uit verschillende modules.

De eerste module draait rond de meest eenvoudige scriptstructuur, sequentie. Instructies worden één voor één, steeds in dezelfde volgorde uitgevoerd. We gaan zien hoe we data kunnen invoeren in het script, de data gaan toewijzen aan een variabele, berekeningen doen met de data, en tenslotte de resultaten uitvoeren naar het scherm.

Module twee draait rond selectie. Bij deze structuur, moet een keuze gemaakt worden welke weg het script uitgaat. De keuze hangt af van een **conditie** die gesteld wordt. De conditie is een ja/nee-vraag.

In module drie behandelen we iteratie. Bij een iteratie wordt een gedeelte van het script een aantal keren herhaald. Het aantal keren dat de herhaling uitgevoerd wordt hangt af van de conditie die gesteld wordt.

In module 4 bekijken we een complexere datastructuur: lijsten. We bekijken hoe we deze kunnen opstellen, bewerken en gebruiken.

In elke module gaan we eerst op verkenning. We analyseren een probleem, en lossen dit op zonder gebruik te maken van de computer. De oplossing van het voorbeeld leidt tot een algoritme, een stappenplan. Dit stappenplan wordt daarna grafisch voorgesteld in een Nassi-Shneiderman-diagram. Tenslotte leren we de instructies die nodig zijn om het algoritme om te zetten in een script.

De nieuwe instructies worden dan ingeoefend met kleine opdrachten. Focus ligt daar op het juiste gebruik, de syntax en minder op de analyse. In deze fase kan er gebruik gemaakt worden van het Dodona platform. Een groot aantal oefeningen worden aangeboden. Nadruk ligt niet op het maken van alle oefeningen maar het beheersen van de syntax. Eenmaal je deze beheerst, mag je overstappen naar de verdiepingsfase.

Voordat we kunnen starten moeten we zorgen dat we de juiste programmeeromgeving hebben. Twee mogelijke programmeeromgevingen worden behandeld in module 0.

Module 0: De Bouwstenen

Een Python-script schrijf je in elke willekeurige teksteditor naar keuze. Deze tekst vertegenwoordigt jouw creatieve visie en logica. Maar hoe zet je die tekst om in een werkend script? Hier komt de interpreter in beeld. Een interpreter vertaalt je geschreven script naar instructies die je computer kan uitvoeren.

Om je ervaring te vergemakkelijken, stellen we je voor aan de IDE (Integrated Development Environment). Dit is een alles-in-één omgeving die zowel een krachtige teksteditor en meestal ook een interpreter omvat. Met een IDE kun je naadloos je script's schrijven, testen en uitvoeren, en dat zowel offline als online.

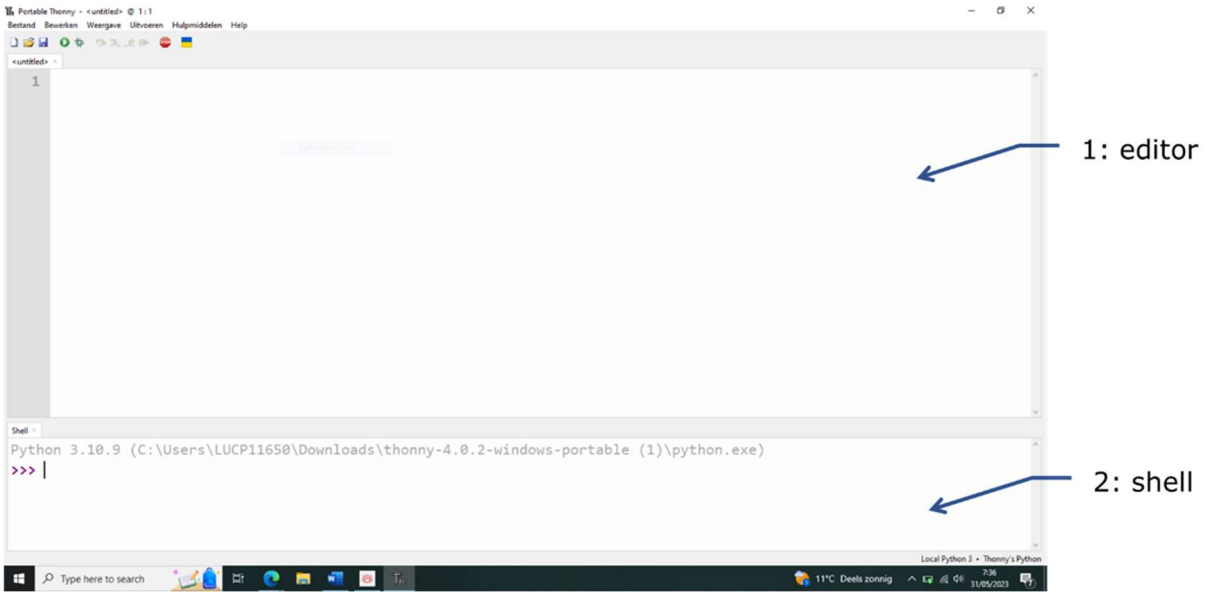
De Keuze van Omgeving

In deze keuze maken we gebruik van de Thonny IDE.

Thonny is een gebruiksvriendelijke programmeeromgeving die speciaal is ontworpen voor beginners. De interface is schoon, overzichtelijk en biedt een minimalistische benadering van programmeren. Je vindt alle benodigde tools en functies op één plek, waardoor je snel aan de slag kunt.

Je kan Thonny downloaden via de volgende URL: www.thonny.org

De eerste keer dat je de Thonny programmeer omgeving opent, staan er twee schermen open.



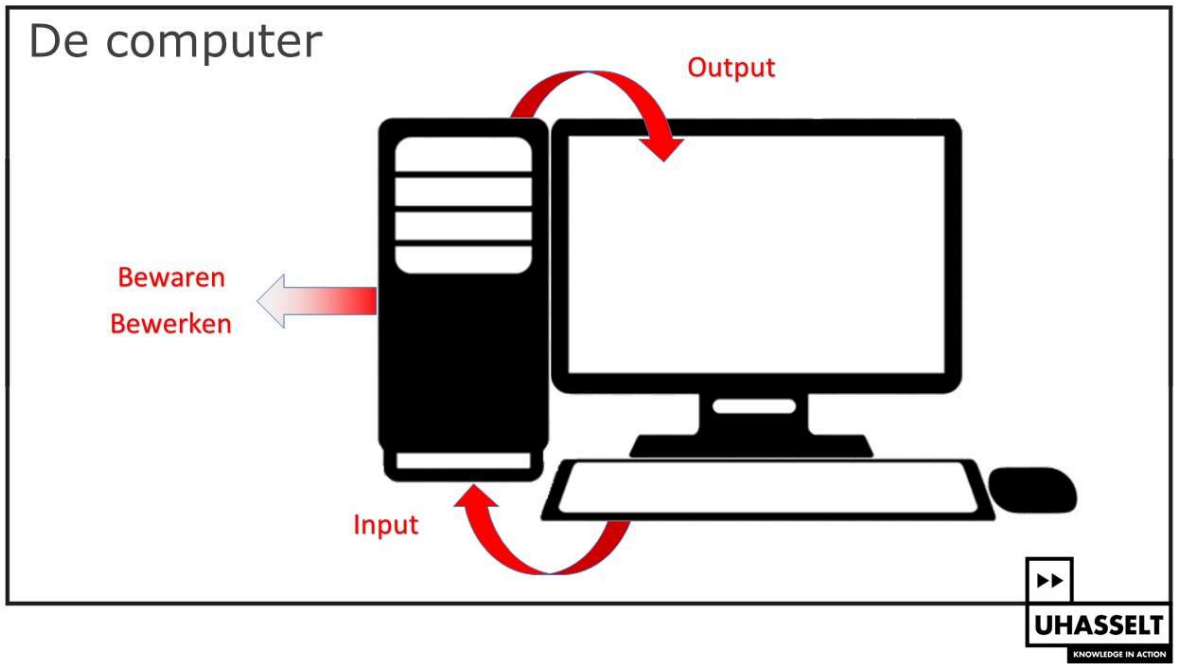
1. ***Editor***: Dit is het primaire scherm van Thonny waar je je Python-code kunt schrijven, bewerken en opmaken. Het biedt functies zoals syntax highlighting, automatische indentatie (inspringen) om je te helpen bij het efficiënt schrijven van code.

2. ***Shell***: Het Shell-scherm in Thonny is een interactieve Python-omgeving waarin je Python-code direct kunt uitvoeren en de resultaten kunt zien. Het werkt als een interactieve console waarin je regels code kunt invoeren en de uitvoer kunt bekijken. Dit is handig om korte codefragmenten te testen of om te experimenteren met Python-functies en -expressies.

Module 1: Sequentie

Wat behandelen we in deze module?

Een computer kan worden beschouwd als een apparaat dat informatie ontvangt, meestal via het toetsenbord (input). Deze informatie wordt opgeslagen, verwerkt en uiteindelijk wordt er output naar een scherm gestuurd.



In deze module zullen we je leren hoe je in Python gegevens kunt invoeren via het toetsenbord. De ingevoerde gegevens worden opgeslagen in variabelen, bewerkt met basis wiskundige operatoren en uiteindelijk wordt er output weergegeven op het scherm.

We beginnen met eenvoudige voorbeelden die je eerst op de traditionele manier, met pen en papier, zult oplossen. Van daaruit zul je een algoritme afleiden dat je vervolgens zult omzetten naar een Python-script.

In de verkenningsfase ga je experimenteren met de syntaxis van de code, waarna je verder gaat vervolmaken met verschillende oefeningen.

Verkenning 1:

Doel:

In deze verkenningsfase ga je een probleem analyseren. We starten met zeer eenvoudige problemen, kijken hoe we dit met pen en papier oplossen (analyse), zetten dit om naar een stappenplan (algoritme), we proberen de code uit in de shell, en tenslotte plaatsen we de code in de editor en testen het script uit.

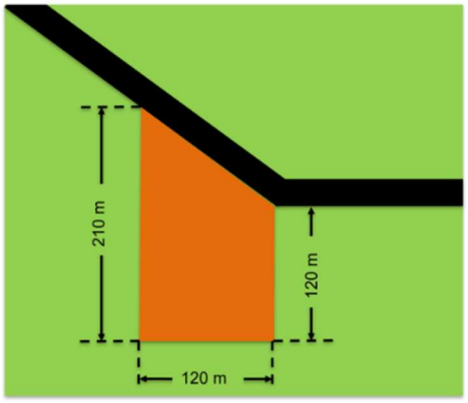
In deze verkenning ga je:

- een probleem analyseren en omzetten in een algoritme
- getallen bewaren in een variabele (in het geheugen)
- kennismaken met de rekenkundige basisbewerkingen
- output geven met de print-opdracht

Probleemoplossend denken:

Opdracht 1.1:

Een boer heeft een akker in de vorm van een rechthoekig trapezium. De afmetingen van de akker zijn aangeduid op de bijgevoegde figuur. Hij wilt de akker omvormen tot een boomgaard. Hij leest op de website www.landleven.nl dat een appelboom een oppervlakte van 8 x 8 m nodig heeft. Hoeveel bomen moet de boer aankopen?



Neem een blad en papier en bereken op een gestructureerde manier het aantal bomen dat de boer moet aankopen.

Hou je oplossing goed bij, want later ga je hiermee verder werken.

Algoritme

Opdracht 1.2:

Veronderstel dat je broer of zus een machine is die je moet programmeren om een boterham met choco te smeren.

Maak eens (in je eigen woorden) een stappenplan waarmee je broer of zus deze taak tot een goed einde kan brengen.

Bekijk daarna het filmpje: ***Leer programmeren met boterham met choco.***

Zou jou stappenplan wel tot een goed resultaat geleid hebben?

bit.ly/BoterhamChoco

Van analyse tot algoritme

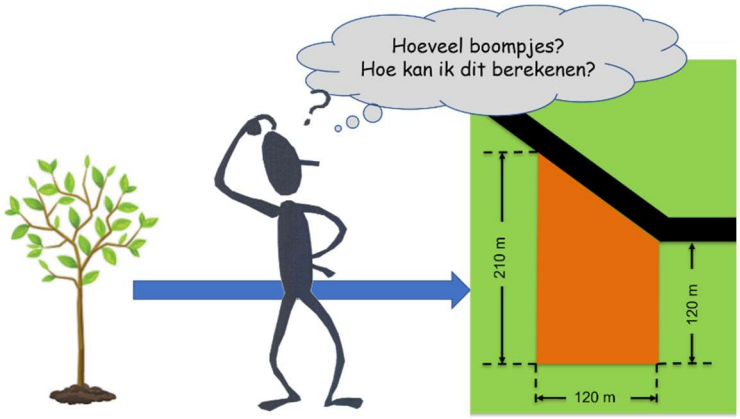
In de wiskunde en wetenschappen krijg je wel dikwijls te maken met een stappenplan dat leidt tot een oplossing, maar in het dagelijkse leven kunnen die dikwijls zeer complex zijn.

Meestal worden complexe problemen opgelost door deze op te delen in kleinere minder complexe problemen die wel oplosbaar zijn. Elk van deze kleine problemen wordt dan apart opgelost en uitgetest. Op het einde worden al deze oplossingen samengebracht om zo de oplossing van het complexe probleem te vormen. Dit is computationeel denken.

Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation or simulation.

Janet Wing – 2006

Opdracht 1.3:
Schrijf eens in je eigen woorden op welke stappen je ondernomen hebt om het aantal boompjes te berekenen dat de boer moet aankopen. Jij bent geen machine, dus je mag acties zoals pen en papier nemen overslaan.



Basisbewerkingen

Als ik een tekst in het Nederlands zou schrijven met fouten erin, zou je nog steeds in staat zijn om de tekst te begrijpen. Maar als we AI even buiten beschouwing laten, kan een computer dat niet.

Daarom is de syntaxis van een computertaal van groot belang. Gelukkig is de syntaxis van Python over het algemeen vrij eenvoudig en levert het leesbare script's op. Dit betekent dat je meer aandacht kunt besteden aan het denkproces achter het programmeren.

Laten we eens kijken naar de basis wiskundige bewerkingen bij het verkennen van de syntaxis van de code. Je hebt waarschijnlijk al ervaring met het uitvoeren van berekeningen in spreadsheetscript's zoals Excel of Google Spreadsheets. Op basis van die ervaring, kun je vermoeden welke symbolen Python gebruikt voor de basisbewerkingen: optelling, aftrekking, vermenigvuldiging, deling en machtsverheffing.

Opdracht 1.4:
Vul onderstaande lijst aan met de symbolen die je denkt dat overeenkomen met de bewerking.

- Optelling: .....
- Aftrekking: .....
- Vermenigvuldiging:.....
- Deling:.....
- Machtsverheffing: .....

Controleer of jouw symbolen juist zijn door onderstaande bewerkingen uit te voeren in de shell. Je typt gewoon de bewerking en in plaats van het gelijkheidsteken druk je op ***ENTER***..

- $8 + 2$
- $8 - 2$
- $8 \cdot 2$
- $\frac{8}{2}$
- 8^2

Algoritme uitvoeren in de shell

Opdracht 1.5:
Controleer je algoritme van opdracht 1.3 door het stap voor stap uit te voeren in de shell.

Je geeft één voor één de bewerkingen in. Sluit de bewerkingen af door op ***ENTER*** te drukken.

Grafische voorstelling van een algoritme

Er zijn verschillende manieren om een algoritme voor te stellen. Enkele voorbeelden zijn: psuedo code, stroomdiagram, scriptstructuur diagram (Nassi–Shneiderman diagram).

Nassi-Schneidermann diagrammen (NSD) zijn een goed middel om algoritmen voor te stellen, doordat het automatisch leidt tot gestructureerd geheel.

Let op! Een algoritme wordt onafhankelijk gemaakt van de programmeertaal die je gaat gebruiken. We gebruiken daarom in het algoritme geen syntax specifiek voor een taal. Voor bewerkingen gebruiken we gewoon de wiskundige notatie. Voor de toekenning (assignment) van een waarde aan een variabele maken we gebruik van een pijl.

Het voorbeeld van het algoritme bestaat uit 6 stappen die achter elkaar moeten uitgevoerd worden. Dit wordt een sequentie genoemd.

Aantal bomen nodig
$b \leftarrow 120$
$h \leftarrow 120$
$H \leftarrow 210$
$oppervlakte\_boom \leftarrow 8 \cdot 8$
$A \leftarrow \frac{h+H}{2} \cdot b$
$aantal\_bomen \leftarrow \frac{A}{oppervlakte\_boom}$
Output: $aantal\_bomen$
EINDE

Meer informatie over Nassi-Sheiderman diagramma's of scriptstructuur diagram vind je op Wikipedia.

Link: bit.ly/NSD_diagram

Het juiste antwoord is 309 (bomen).

Waarschijnlijk kom je als resultaat 309,375 bomen uit. Dit komt omdat je "/" gebruikt hebt als wiskundige operator. Verander nu het symbool "/" met "//" (twee maal het deelteken) en controleer wat je nu krijgt.

Het symbool "//" wordt gebruikt voor de gehele deling. De gehele deling geeft als uitkomst het gehele getal dat kleiner of gelijk is aan de deling.

Hieronder zie je het verschil tussen beide bewerkingen:

```
Shell
Python 3.10.9 (C:\Users\LUCP11650\Downloads\thonny-4.0
>>> (210 + 120)/2*120
19800.0
>>> 19800/(8*8)
309.375
>>> 19800//(8*8)
309
>>>
```

Variabelen

In Python is een variabele een naam die aan een waarde is gekoppeld. Het is een container waarin je gegevens kunt opslaan en later kunt gebruiken in je script. Variabelen in Python zijn dynamisch getypeerd, wat betekent dat je ze kunt gebruiken om verschillende soorten gegevens op te slaan.

Enkele belangrijke gegevenstypen die vaak worden gebruikt in Python zijn:

1. ***Integer (int)***: Dit is een geheel getal, bijvoorbeeld 5, -3, 0. Het wordt gebruikt voor het opslaan van numerieke waarden zonder decimalen.
2. ***Float***: Dit is een drijvendekommagetal, bijvoorbeeld 3.14, -0.5, 2.0. Het wordt gebruikt voor het opslaan van numerieke waarden met decimalen.
3. ***String (str)***: Dit is een reeks tekens, bijvoorbeeld "Hallo", "Python", "123". Het wordt gebruikt voor het opslaan van tekstuele gegevens.
4. ***Boolean (bool)***: Dit is een datatype dat slechts twee waarden kan bevatten, True of False. Het wordt gebruikt voor het opslaan van waarheidswaarden.

Deze zijn slechts enkele belangrijke gegevenstypen in Python.

Variabele namen

Bij het oplossen van Fysica vraagstukken maak je ook gebruik van variabelen, meestal bestaande uit één letter of symbool. Passen we dezelfde methode toe in een script (script), dan is het later moeilijk te achterhalen waarvoor deze variabele gebruikt werd.

We geven daarom onze variabele een betekenisvolle naam..

Er zijn welke enkele regels die we moeten volgen.

Een variabele naam .....

- mag enkel bestaan uit letters, cijfers en "underscores" (_).
- mag niet beginnen met een cijfer.
- mag geen gereserveerd woord (keyword) van Python zijn.

De gereserveerde woorden zijn:

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

Conventies:

- Programmeurs kiezen nooit een variabele naam die ook de naam is van een functie (of het nu een standaard Python functie betreft of een functie die ze zelf hebben geschreven). Als je dat doet, loop je de kans dat de functie niet langer door de code gebruikt kan worden, wat kan leiden tot uitermate vreemde fouten.
- Kies een betekenisvolle naam (oppervlak_boom, grote_zijde, kleine_zijde, basis, aantal_bomen,...)
  Uitzondering: Wegwerp variabelen
  Dit zijn variabelen die slechts in een klein deel van de code gebruikt worden en daarna niet meer.
- Om verwarring te vermijden, gebruiken je best alleen kleine letters in variabele namen.
- Als een variabele naam uit meerdere woorden bestaat, plaats je underscores tussen die woorden. Dit zorgt ervoor dat je variabele leesbaarder wordt
- Het gebruik van namen die starten met een underscore is voorbehouden aan de auteurs van Python.

Toewijzing

Met het toekenningsteken “=” kunnen we een waarde gaan toewijzen aan een functie. Dit is hetzelfde teken dat we uit de wiskunde kennen als gelijkheidsteken, maar heeft wel een andere betekenis.

Onderstaand voorbeeld lees je niet als “lange_zijde is gelijk aan 210” maar als “de variabele lange_zijde krijgt waarde 210”

```
Lange_zijde = 210
```

Onderstaand voorbeeld toont het verschil tussen het gelijkheidsteken en het toekenningsteken.

```
aantal = 5
aantal = aantal + 1
```

De variabele “aantal” kan (wiskundig) nooit gelijk zijn aan aantal +1.

In onze code krijgt de variabele aantal de waarde 5. In de tweede regel gaat het script de waarde van variabele aantal ophalen, en gaat hierbij 1 optellen. De variabele waarde krijgt het resultaat van deze bewerking.

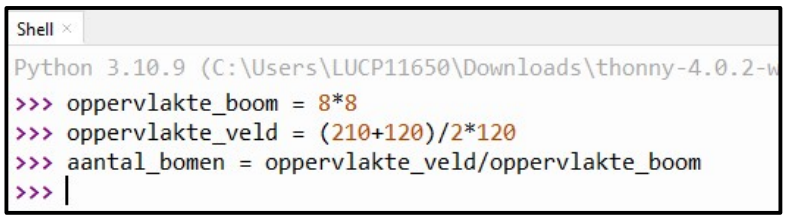
Bij een toewijzing wordt dus eerst de bewerking rechts van het toewijzingsteken eerst uitgevoerd, en het resultaat van deze bewerking wordt dan toegekend aan de variabele links van het toewijzingsteken.

Python instructie: print(…)

We hebben zonet ons algoritme gecontroleerd door rechtstreeks de berekeningen in te typen.

Je kan echter ook gebruik maken van variabelen, zoals aangegeven in het algoritme.

Als je dit in de shell intypt, dan krijg je onderstaand beeld:



Je merkt hier dat je geen uitvoer krijgt van je berekeningen.

Wanneer je gewoon een berekening in de shell typt, wordt deze berekening uitgevoerd, en wordt het resultaat getoond.

Wanneer je echter **aantal_bomen = oppervlakte_veld/oppervlakte_boom** typt, dan vraag je niet om de berekening uit te voeren en het resultaat toe te wijzen aan de variabele N.

Wil je het resultaat weten, of de waarde van een variabele, dan kan dit met de **print()**-instructie.

Wanneer je de print()-opdracht gebruikt, dan vraag je om uitvoer te geven naar het scherm.

Wil je dus het resultaat, het aantal bomen kennen, dan kan dit met

```
print(aantal_bomen)
```

Je moet niet steeds gebruik maken van een variabele, maar kan ook rechtstreeks de bewerking tussen de haaktjes plaatsen:

```
print((210+120)/2*120)
```

Of je kan verschillende gegevens combineren door ze te scheiden met een komma.

```
print("Oppervlakte veld:",(210+120)/2*120)
```

Opmerking:

- Belangrijk is op te merken dat na een print()-opdracht er steeds een nieuwe lijn gestart wordt. Hoe je dit kan voorkomen leren we later.
- Je kan tekst weergeven door de tekst te plaatsen tussen dubbele of enkele aanhalingstekens. Let op, het begin en einde van je string moeten wel beide ofwel enkelvoudige, of beide dubbele aanhalingstekens zijn.

```
Python 3.10.9 (C:\Users\LUCP11650\
>>> print("Hello World!")
 Hello World!
>>> print('Hello World')
 Hello World
>>> print("'s avonds laat")
 's avonds laat
>>>
```

Maximaal gebruik van variabelen

We kunnen ook alle gegevens toewijzen aan variabelen. In de berekeningen worden dan de namen van deze variabelen gebruikt.

```
lange_zijde = 210
korte_zijde = 120
breedte = 120
oppervlakte_boom = 8*8
oppervlakte_veld = (lange_zijde + korte_zijde)/2*breedte
aantal_bomen = oppervlakte_veld/oppervlakte_boom
print("Aantal bomen:",aantal_bomen)
```

In de shell ziet dit er zo uit:

```
Python 3.10.9 (C:\Users\LUCP11650\Downloads\thonny-4.0.2-windows-
>>> lange_zijde = 210
>>> korte_zijde = 120
>>> breedte = 120
>>> oppervlakte_boom = 8*8
>>> oppervlakte_veld = (lange_zijde + korte_zijde)/2*breedte
>>> aantal_bomen = oppervlakte_veld/oppervlakte_boom
>>> print("Aantal bomen:",aantal_bomen)
 Aantal bomen: 309.375
>>>
```

Gaan we eenmaal een script schrijven, dan wordt deze methode geprefereerd. Deze manier van werken zorgt ervoor dat bij het wijzigen van een waarde, je de wijziging slechts op één plaats in je script moet doorvoeren.

Opdracht 1.6

Onderzoek welk type variabele (integer of float) je krijgt bij het uitvoeren van een deling (/) en een gehele deling (//). Probeer alle combinaties uit van integers en floats.

Opdracht 1.7

Bekijk onderstaande instructies. Noteer wat je verwacht dat op de stippellijnen onder elke print-instructie op het scherm getoond gaat worden.

Controleer je antwoorden door de code in de shell in te typen.

```
>>> a = 4.8
>>> b = 4
>>> c = -2
>>> print(a/b)
.....
>>> print(a//b)
.....
>>> print(b/c)
.....
>>> print(b//c)
.....
>>> print(a/c)
.....
>>> print(a//c)
.....
```

Python script schrijven

We kennen nu alle code die we nodig hebben om een script te schrijven. Plaatsen we de code van het voorbeeld met de variabelen gewoon in de editor, dan is ons Python script klaar.

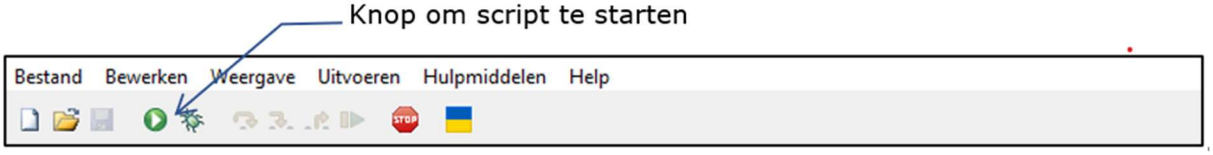
```
1  # Aantal bomen berekenen
2
3  lange_zijde = 210
4  korte_zijde = 120
5  breedte = 120
6
7  oppervlakte_boom = 8*8
8  oppervlakte_veld = (lange_zijde+korte_zijde)/2*breedte
9  aantal_bomen = oppervlakte_veld/oppervlakte_boom
10
11 print("Aantal bomen:",aantal_bomen)
12
```

Nu moeten enkel nog het script uitvoeren.

Hieronder kan je zien hoe je, zowel in Thonny als in Repl.it je script kan staten.

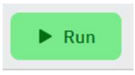
Thonny

Je kan je script starten door de sneltoets F5 te drukken of te klikken op de startknop.



REPL.IT

In **REPL.IT** kan je het script starten door de toetsencombinatie **CTRL+ENTER** te drukken of door op **RUN** te klikken.



In de shell krijg je onderstaande uitvoer (de gegeven uitvoer is de uitvoer in Thonny maar REPL.IT is zeer vergelijkbaar)

Opmerkingen:

Documenteer je script.

Het is belangrijk dat je script leesbaar is en daarom van voldoende commentaar voorzien wordt. Op die manier kan iemand de gedachtegang van de programmeur volgen. Dit maakt het makkelijker om later aanpassingen aan het script aan te brengen.

Je kan opmerkingen toevoegen door deze vooraf te laten gaan door de hashtag (#).
Alles na # wordt genegeerd en is louter ter info.

Gebruiksvriendelijke interface

Je interactie met de gebruiker van je script moet ook gebruiksvriendelijk zijn.

Dit kan je bereiken door meer print-opdrachten toe te voegen .

Voorbeeld

Script met verzorgde output en extra uitleg voor/opmerkingen

```
# Aantal bomen berekenen

lange_zijde = 210
korte_zijde = 120
breedte = 120

oppervlakte_boom = 8*8

# Oppervlakte veld berekenen
oppervlakte_veld = (lange_zijde+korte_zijde)/2*breedte

# Aantal bomen berekenen
aantal_bomen = oppervlakte_veld/oppervlakte_boom

# Output
print("Gegevens:")
print("Lange zijde:",lange_zijde)
print("Korte zijde:",korte_zijde)
print("Breedte:",breedte)
print()
print("Resultaat:")
print("Aantal bomen aan te kopen:",aantal_bomen)
```

De output kan er dan zo uitzien:

```
>>> %Run -c $EDITOR_CONTENT

  Gegevens:
  Lange zijde: 210
  Korte zijde: 120
  Breedte: 120

  Resultaat:
  Aantal bomen aan te kopen: 309.375
  Aantal bomen: 309.375

>>>
```

Let wel op!

Het openen en sluiten moet met hetzelfde soort aanhalingstekens gebeuren. Anders krijg je een foutmelding.

```
11 # Aantal bomen berekenen
12 aantal_bomen=A/oppervlakte_boom
13
14 # Output
15 print("Gegevens:')
16 print("Korte zijde:",h)
```

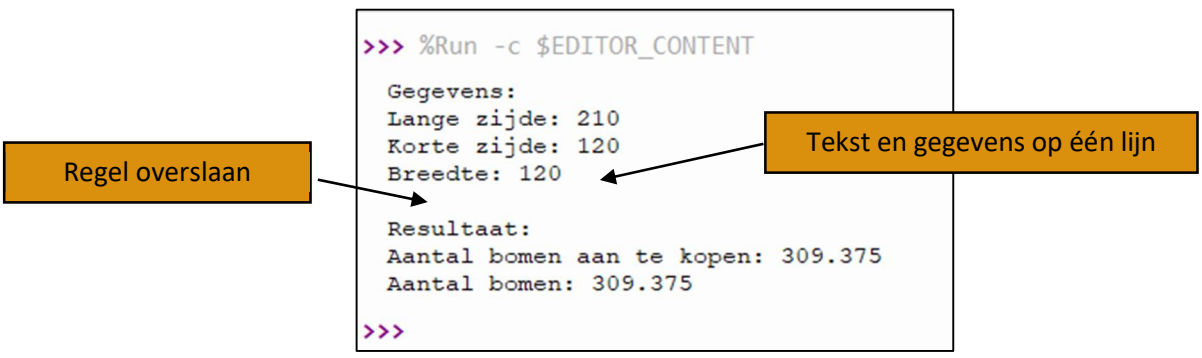
```
Shell
>>> %Run bomen.py
  Traceback (most recent call last):
    File "C:\Users\LUCP11650\Downloads\bomen.py", line 15
      print("Gegevens:')
            ^
  SyntaxError: unterminated string literal (detected at line 15)
>>>
```

Finaliseren

Bestudeer de code en bekijk de output. Los daarna onderstaande vragen op:
Hoe kan je in Python….

- een regel overslaan, zoals tussen “Breedte: 120” en “Resultaat”?
- twee gegevens zoals “Breedte” en “120 “ (de waarde van ***breedte***) op één lijn printen?

Output:



Code:

```
#Gegevens
```

```
korte_zijde=120
lange_zijde=210
breedte=120
oppervlakte_boom=8*8

#Oppervlakte berekenen
Oppervlakte_veld=(lange_zijde+korte_zijde)/2*breedte

#Aantal bomen berekenen
aantal_bomen=A/oppervlakte_boom

#Output
print("Gegevens:")
print("Korte zijde:",korte_zijde)
print("Lange zijde:",lange_zijde)
print("Breedte:",breedte)
print()
print("Resultaat:")
print("Aantal bomen aan te kopen:",aantal_bomen)
```

Verkenning 2

Doel:

In deze verkenningsfase ga je onderzoeken hoe je programma (script) kan vragen om via het toetsenbord data in te geven.

In deze verkenning ga je:

- de input-instructie onderzoeken
- je kennis verdiepen van drie belangrijke data types
- leren de data van het ene type om te vormen naar het andere type.

Python instructie: input(..)

Opdracht 1.8:
Python is een zeer leesbare taal. Wanneer je alle Engelse instructies zou vertalen naar het Nederlands, dan kan je al een goed idee krijgen wat het script gaat doen.

Bestudeer onderstaande scripts.

- Wat denk je dat er gaat gebeuren?
- Typ de code over en controleer je antwoord.

Script 1

```
getal=input("Geef getal in:")
print("Het ingegeven getal was",getal)
```

Script 2

```
getal=input("Geef getal in:")
dubbel_getal=2*getal
print("Het ingegeven getal was",getal)
print("Het dubbele van het getal is", dubbel_getal)
```

Converteren tussen variabel types

Je merkt dat je soms verplicht bent om gegevens van het ene variabel type te converteren naar het andere. Gegevens die toegewezen worden met de input-instructie zullen steeds van het string type zijn. Om hiermee wiskundige bewerkingen te doen, ga je deze moeten converteren naar een integer of een float.

Let wel op, sommige conversie gaan wel een effect hebben op “de waarde” van je variabele. Zo zal een omzetting van float naar integer ertoe leiden dat alle cijfers na de komma wegvallen.

- $2.1 \xrightarrow{naar\ integer} 2$
- $-2.1 \xrightarrow{naar\ integ} -2$
- $2 \xrightarrow{naar\ float} 2.0$

De instructie die je gebruikt om te converteren hangt enkel af van het type naar waar je converteert, en niet van het oorspronkelijke variabel type:

- Converteren naar een integer:

```
geheel_getal = int(gegeven).
```

- Converteren naar een float:

```
decimaal_getal = float(gegeven)
```

- Converteren naar een string:

```
tekst = str(gegeven)
```

Als Python niet in staat is om het gegeven te converteren, dan krijg je een foutmelding.

Opdracht 1.9

Bekijk onderstaande instructies. Noteer wat je verwacht dat op de stippellijnen onder elke print-instructie op het scherm getoond gaat worden.

Controleer je antwoorden door de code in de shell in te typen.

```
>>> a = 9
>>> b = 2
>>> c = -2
>>> print(a/b)
.....
>>> print(a//b)
.....
>>> print(int(a/b))
.....
>>> print(a/c)
.....
>>> print(a//c)
.....
>>> print(int(a/c))
.....
>>>
```

Opdracht 1.10

Pas onderstaande code aan zodat het script het dubbele van het getal berekent en dit als output geeft.

```
getal=input("Geef getal in:")
dubbel_getal=2*getal
print("Het ingegeven getal was",getal)
print("Het dubbele van het getal is", dubbel_getal)
```

Een voorbeeld van een mogelijke output wordt hieronder getoond.

```
>>> %Run -c $EDITOR_CONTENT
  Geef getal in:13
  Het ingegeven getal was 13
  Het dubbele van het getal is 26
>>>
```

Finaliseren

Opdracht 1.11:

Onderstaande code is een mogelijke oplossing voor opdracht 1.1.
Pas deze, of je eigen code, aan zodat de gebruiker verschillende zijden kan ingeven zonder er iets aan de code moet veranderd worden.

```
# Aantal bomen berekenen

# Gegevens
lange_zijde = 210
korte_zijde = 120
breedte = 120

oppervlakte_boom = 8*8

# Oppervlakte veld berekenen
oppervlakte_veld = (lange_zijde+korte_zijde)/2*breedte

# Aantal bomen berekenen
aantal_bomen = oppervlakte_veld/oppervlakte_boom

# Output
print("Gegevens:")
print("Lange zijde:",lange_zijde)
print("Korte zijde:",korte_zijde)
print("Breedte:",breedte)
print()
print("Resultaat:")
print("Aantal bomen aan te kopen:",aantal_bomen)
```

Verkenning 3:

Doel:

In deze verkenningsfase gaan we voornamelijk bestaande kennis uitbreiden.

We gaan:

- er op letten dat ons script voldoende informatie bevat, zodat je later makkelijk het script kan aanpassen (gebruik van #)
- met de basisbewerkingen die we kennen gaan we ook (vierkants)wortels berekenen
- een bibliotheek importeren om de wortel met een speciale functie te berekenen.

Wortels...

Opdracht 1.12

Onze boer is heel gelukkig met zijn boomgaard, en hij begint al aardig wat appelen op te leveren. Spijtig genoeg ligt zijn boomgaard langs een toeristisch fietspad. Zeer veel fietsers stoppen voor een appel, en gaan zelfs met volle fietstassen naar huis.

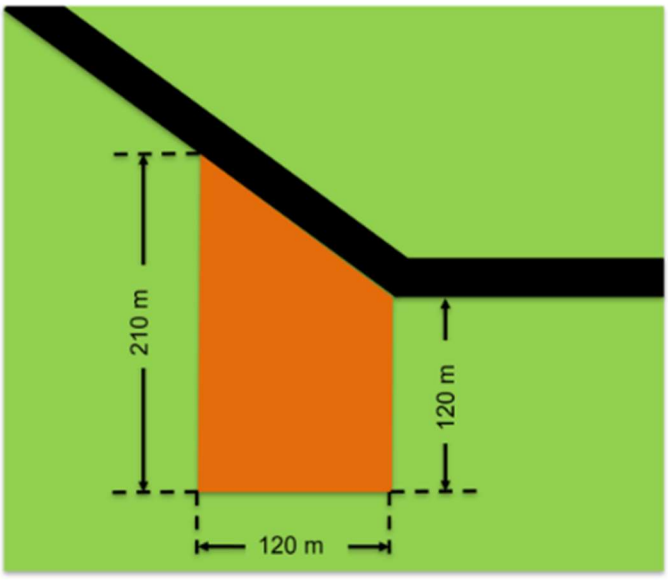
Hij besluit daarom langs de weg een omheining te plaatsen.

In de omheining moet een opening zijn van 4 m voor een poort, en de omheining wordt gemaakt van metalen draadhekken zoals op de foto is aangegeven. De lengte van 1 draadhek is 2,03 m.

Maak een algoritme en zet dit om naar een script dat flexibel kan ingezet worden om het nodige aantal draadhekken en palen te berekenen.

Tip:

Je script van opdracht 1.11 kan je gebruiken als start-punt.



Python instructie: sqrt(…)

Inleiding

Een Python-bibliotheek is een verzameling van code en functies die is ontwikkeld om bepaalde taken en functionaliteiten gemakkelijk uit te voeren in Python-script's. Deze bibliotheken bevatten vooraf geschreven code die programmeurs kunnen gebruiken om specifieke problemen op te lossen zonder ze helemaal opnieuw te moeten programmeren.

Modules in de Python-standaardbibliotheek moeten niet te worden geïnstalleerd en daarom is het importeren ervan bovenaan onze script's voldoende om aan de slag te gaan.

Een bekend voorbeeld van dergelijke module is ***math***. De wiskundige module biedt toegang tot algemene wiskundige functies.

Met behulp van deze functies kunnen we verschillende wiskundige uitdrukkingen uitvoeren, zoals het vinden van de vierkantswortel van een getal. De instructie hiervoor is **math.sqrt(….)**
(sqrt = squarerooth, vierkantswortel).

Gebruik

Aan het begin van het script geef je aan dat je gebruik wenst te maken van de math module. Dit doe je via

```
import math
```

Nadat je dit gedaan hebt kan je alle functies uit de module in je script gebruiken.

Je script wordt dan

```
# Aantal hekelementen berekenen

#importeren math module van de standaard bibliotheek

import math

# Titel op scherm (wat doet het script)

print("Aantal palen en hekken berekenen")
print()

# Gegevens ingeven

lange_zijde = float(input("Lengte lange zijde: "))
korte_zijde = float(input("Lengte korte zijde: "))
breedte_akker = float(input("Breedte van de akker: "))

breedte_poort = float(input("Breedte van de poort: "))
lengte_element = float(input("Lengte van 1 draadhek: "))


# Berekenen van de lengte langs de weg met Phytagoras

lengte_verschil = lange_zijde-korte_zijde

lengte_weg = math.sqrt(lengte_verschil**2+breedte_akker**2)
```

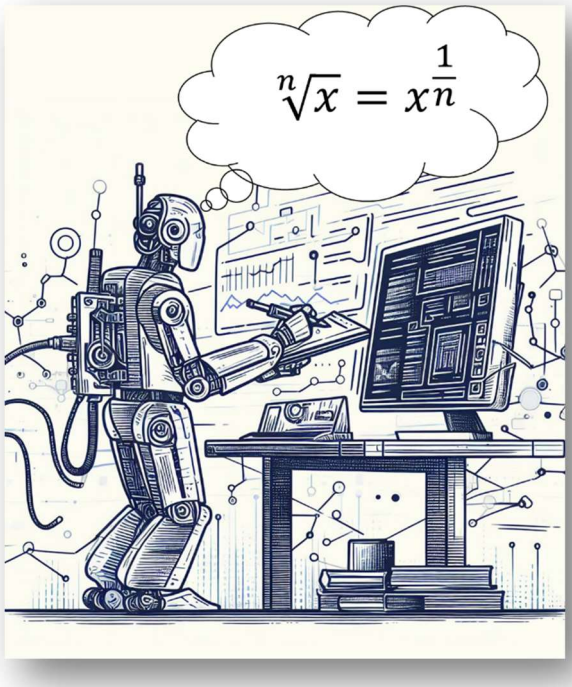
```
# Berekenen aantal hekken

aantal_hekken = int(lengte_weg//lengte_element + 1)
#int om een geheel getal te krijgen
# +1 extra element nodig dat je moet inkorten

# Berekenen aantal palen

aantal_palen = aantal_hekken + 2

# Links en recht van de poort een hek. Telkens 1 extra paal nodig

#uitvoer

print()
print("Aantal hekken:",aantal_hekken)
print("Aantal palen:",aantal_palen)
```

Finaliseren

Opdracht 1.13

Schrijf een script dat aan de gebruiker vraagt om het volume van een kubus in te geven. Het script berekent dan de zijde.

```
Bereken zijde van een kubus uit het volume
Volume van de kubus: 144
De zijde is 5.241482788417793

>>>
```



Inoefeningen

Oefening 1.1

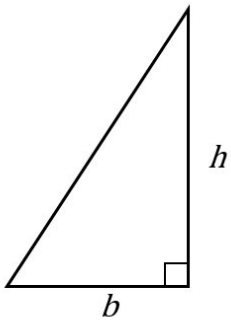
Schrijf een script dat aan de gebruiker vraagt om 3 gehele getallen in te geven. Het script berekent dan de som van de getallen en geeft dit als output op het scherm.

Voorbeeld:

```
Som van drie gehele getallen

Het eerste gehele getal: 12
Het eerste gehele getal: -5
Het eerste gehele getal: 0
De som van de drie getallen is 7

>>>
```

Oefening 1.2

Bereken de oppervlakte van een rechthoekige driehoek. De gebruiker geeft de twee rechthoekszijden in en geeft als output de oppervlakte van de driehoek.



$$A = \frac{1}{2}bh$$

Voorbeeld:

```
Oppervlakte berekening rechthoekige driehoek
--------------------------------------------
Zijde: 4
Hoogte: 2.5
Oppervlakte A = 5.0

>>>
```

Oefening 1.3

Gegeven twee tijdstippen op dezelfde dag (bv. 12:34:56 en 21:43:07).

Het tweede tijdstip valt later dan het eerste tijdstip (laatste tijdstip 23:59:59)

Schrijf een script dat berekent hoeveel seconden er tussen beide tijdstippen zitten en geeft het resultaat op het scherm.

Voorbeeld:

```
Tijdsduur tussen 2 tijdstippen
------------------------------
Tijdstip 1:
   uur: 8
   minuten: 12
   seconden: 16
Tijdstip 2:
   uur: 12
   minuten: 9
   seconden: 51
Tijdsduur in seconden = 14255

>>>
```

Oefening 1.4

Bereken de schuine zijde van een rechthoekige driehoek als de twee rechthoekszijden gegeven zijn.

Voorbeeld:

```
Schuine zijde rechthoekige driehoek
Rechthoekszijde 1: 3
Rechthoekszijde 2: 4
De schuine zijde is 5.0

>>>
```

Verdieping:

Oefening 1.5: Hoekpunten van een driehoek

Schrijf een script waarin de gebruiker de coördinaten ingeeft van de hoekpunten van een driehoek. Het script berekent dan de oppervlakte van de driehoek.

Tip:

Hoe kan je de afstand van een punt tot een rechte berekenen?

Google: **wiskunde interactief afstand rechte**

Oefening 1.6: Zijde van een kubus met afronding van resultaat

Schrijf een script dat aan de gebruiker vraagt om het volume van een kubus in te geven. Het script berekent dan de zijde en rond het resultaat af tot op 2 cijfers na de komma.

Voorbeeld:

```
Bereken zijde van een kubus uit het volume
Volume van de kubus: 142
De zijde is 5.21

>>>
```

Oefening 1.7: Zijde van een kubus met afronding van resultaat

Zelfde oefening als oefening 1.6 maar vraag nu ook aan de gebruiker op hoeveel decimalen er moet afgerond worden.

Voorbeeld:

```
Bereken zijde van een kubus uit het volume
Volume van de kubus: 144
Op hoeveel tekens na de komma (decimale punt) afronden? 2
De zijde is 5.24

>>>
```

Oefening 1.8: Tijdsduur tussen 2 tijdstippen

Herhaal oefening 1.3 maar geef je resultaat in uren, minuten en seconden.

Voorbeeld:

```
Tijdsduur tussen 2 tijdstippen
------------------------------
Tijdstip 1:
   uur: 12
   minuten: 0
   seconden: 0
Tijdstip 2:
   uur: 13
   minuten: 10
   seconden: 12
Tijdsduur tussen de twee tijdstippen: 1:10:12

>>>
```

Module 2: Selectie en conditie

Doel :

Een computer kan worden beschouwd als een apparaat dat gegevens ontvangt, meestal via het toetsenbord (input). Deze gegevens worden opgeslagen, verwerkt en uiteindelijk wordt er output naar een scherm gestuurd. Soms vereist het oplossen van problemen meer dan alleen een rechtlijnige aanpak. Er zijn momenten waarop beslissingen genomen moeten worden op basis van specifieke situaties of voorwaarden die wel of niet vervuld zijn. In de wereld van programmeren wordt dit een 'conditie' genoemd.

In deze module gaan we kijken hoe Python in staat is om beslissingen te nemen. We beginnen met een bekend probleem en ontwikkelen daarvoor een algoritme. Vervolgens zullen we leren hoe we een conditie grafisch kunnen voorstellen in een Nassi–Shneiderman diagram (NSD). Ten slotte zullen we ontdekken welke instructies gebruikt kunnen worden om een conditie te programmeren in Python.

Met deze kennis zullen we in staat zijn om onze code dynamischer te maken en specifieke acties te laten plaatsvinden op basis van verschillende scenario's. Laten we nu samen op ontdekkingsreis gaan naar de fascinerende wereld van conditionele programmering met Python.

Verkenning 1

Opdracht 2.1

In de wiskunde is een vierkantsvergelijking of kwadratische vergelijking een vergelijking van de vorm $ax^2 + bx + c = 0$.

Hierbij zijn a, b en c reële getallen (kan ook complex zijn, maar dit laten we hier buiten beschouwing).

Het kan zijn dat in eerste instantie deze vergelijking niet deze vorm heeft, maar als we alle termen naar hetzelfde lid brengen, dan herleidt deze wel naar deze vorm.

Het oplossen van een vierkantsvergelijking is aan de orde bij het zoeken naar nulpunten.

Maak een algoritme dat de nulpunten geeft van een vierkantsvergelijking in zijn standaardvorm. Maak dit in pseudo code (gewoon Nederlandse taal).

Selectie: grafische voorstelling

Om te bepalen welk deel van het script wordt uitgevoerd, formuleren we een **conditie**.

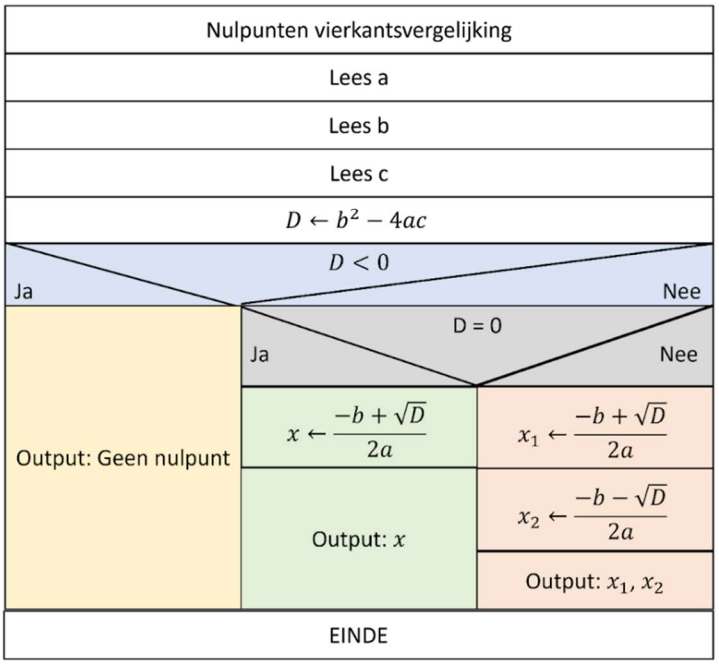
De conditie is steeds een JA/NEE vraag.

Ons algoritme gaat dus moeten opgesplitst worden in 2 kollommen. De ene kolom zal gevolgd worden als het antwoord op de vraag JA is, de andere als het antwoord NEE is.

Hieronder is het algoritme gegeven in pseudo code. Daarnaast is hetzelfde algoritme gegeven als NSD. Met kleuren is aangegeven hoe de conditie wordt omgezet in het NSD.

1. Lees de coëfficiënten a, b en c
2. Bereken de discriminant $D = b^2 - 4ac$
3. **Als** $D < 0$ **dan**
   a) geef als ouput: "Geen nulpunten"
4. **Anders**
   a) Als $D = 0$ **dan**
   - Bereken $x = \frac{-b+\sqrt{D}}{2a}$
   - Output: "Een nulpunt", x

   b) Anders
   - Bereken $x_1 = \frac{-b+\sqrt{D}}{2a}$
   - Bereken $x_2 = \frac{-b-\sqrt{D}}{2a}$
   - Output: "Twee nulpunten", x_1, x_2



Python code: if… else…

We hebben reeds opgemerkt dat een conditie bestaat uit een JA/NEE vraag.

In het geval van JA moet er een bepaalhet blok aan code uitgevoerd worden.
Als het antwoord NEE is moet er een andere blok code uitgevoerd worden.

We gaan dus moeten kijken wat de python code is om de conditie te testen, en hoe we een blok aan code kunnen maken.

Hieronder vind je de algemene structuur.

```
if <conditie>:
  <instructie 1>
  <instructie 2>
  ....
else:
  <instructie A>
  <instructie B>
  .....
# Ongeacht wat het resultaat is van de conditie, hier loopt het script
terug verder
<instructie X>
```

Om de conditie te testen gebruiken we dus ***if*** als instructie, gevolgd door de conditie.

Achter de conditie start dan het blok met code die moet uitgevoerd worden als het antwoord op de conditie positief is.
Dit blok start met : (dubbel punt) waarna de instructies van het blok inspringen (indentatie).

Wanneer je slechts één instructie moet uitvoeren, dan zal het blok bestaan uit één instructie.

Via het keyword **else** geven we aan welke code moet uitgevoerd worden wanneer de conditie een negatief antwoord heeft

Wanneer er bij een negatief antwoord niets moet uitgevoerd worden mag je het else blok weglaten

```
if <conditie>:
  <instructie 1>
  <instructie 2>
  ....

# Ongeacht wat het resultaat is van de conditie, hier loopt het script
terug verder
```

Vergelijkende operatoren

We gaan in onze conditie een vergelijking moeten maken tussen 2 gegevens (bijvoorbeeld A en B)
Mogelijke vergelijkingen zijn

- A **groter dan** B : $A > B$
- A **kleiner dan** B: $A < B$
- A **gelijk aan** B: $A = B$
- A **niet gelijk aan** B: $A \neq B$
- A **groter of gelijk aan** B: $A \geq B$
- A **kleiner of gelijk aan** B: $A \leq B$

Voor groter dan en kleiner dan kunnen we gebruik maken van de wiskundige tekens > en <.

Voor kleiner of gelijk aan, groter of gelijk aan, niet gelijk aan vindt je standaard geen teken op je toetsenbord en gaan we dus op zoek moeten gaan naar een andere oplossing.

Ook gelijk aan levert problemen op. Je zou verwachten dat we hiervoor het gelijkheidsteken (=) zouden gebruiken. Dit teken wordt echter al gebruikt om een waarde toe te kennen aan een variabele.

De oplossing voor deze problemen vind je hieronder

- A groter dan B

```
>
```

- A kleiner dan B

```
<
```

- A gelijk aan B

```
= =
```

- A niet gelijk aan B

```
!=
```

- A groter of gelijk aan B

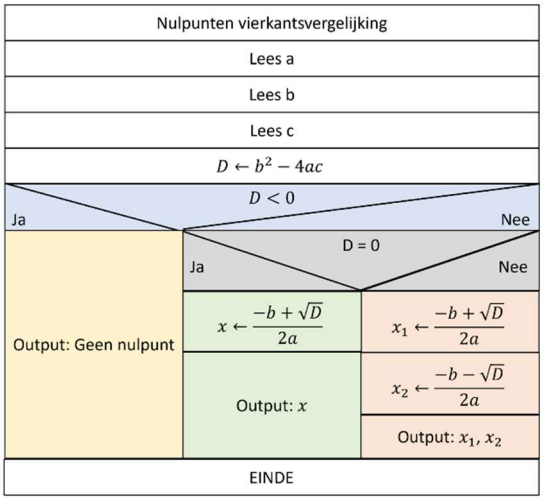
```
>=
```

- A kleiner of gelijk aan B

```
<=
```

Van algoritme naar code

We kennen nu alle instructies om ons algoritme om te zetten naar code.




```
a=int(input("a =")) # ingeven a en omzetten naar integer
b=int(input("b =")) # ingeven a en omzetten naar integer
c=int(input("c =")) # ingeven a en omzetten naar integer

D=b**2-4*a*c

if  D<0 :
    print("Geen nulpunten")

else:
    if  D==0 :
        x=(-b+D**0.5)/(2*a)
        print("Het nulpunt van de vergelijking is",x)

else:
        x1=(-b+D**0.5)/(2*a)
        x2=(-b-D**0.5)/(2*a)
        print("De nulpunten van de vergelijking zijn",x1,"en",x2)
```

Dit is een basisscript. Onderstaand zie je het script met een gebruiksvriendelijke interface.

```
#Nulpunten vierkanstsvergelijking
print("De vierkantsvergelijking heeft de vorm ax²+bx+c=0")
print("Geef de coëfficiënten in")
print()
a=int(input("a ="))
b=int(input("b ="))
c=int(input("c ="))
print()
D=b**2-4*a*c
if D<0:
    print("Geen nulpunten")
else:
    if D==0:
        x=(-b+D**0.5)/(2*a)
        print("Het nulpunt van de vergelijking is",x)
    else:
        x1=(-b+D**0.5)/(2*a)
        x2=(-b-D**0.5)/(2*a)
        print("De nulpunten van de vergelijking zijn",x1,"en",x2)
```

Mogelijke outputs zie je dan hieronder

Voorbeeld 1:

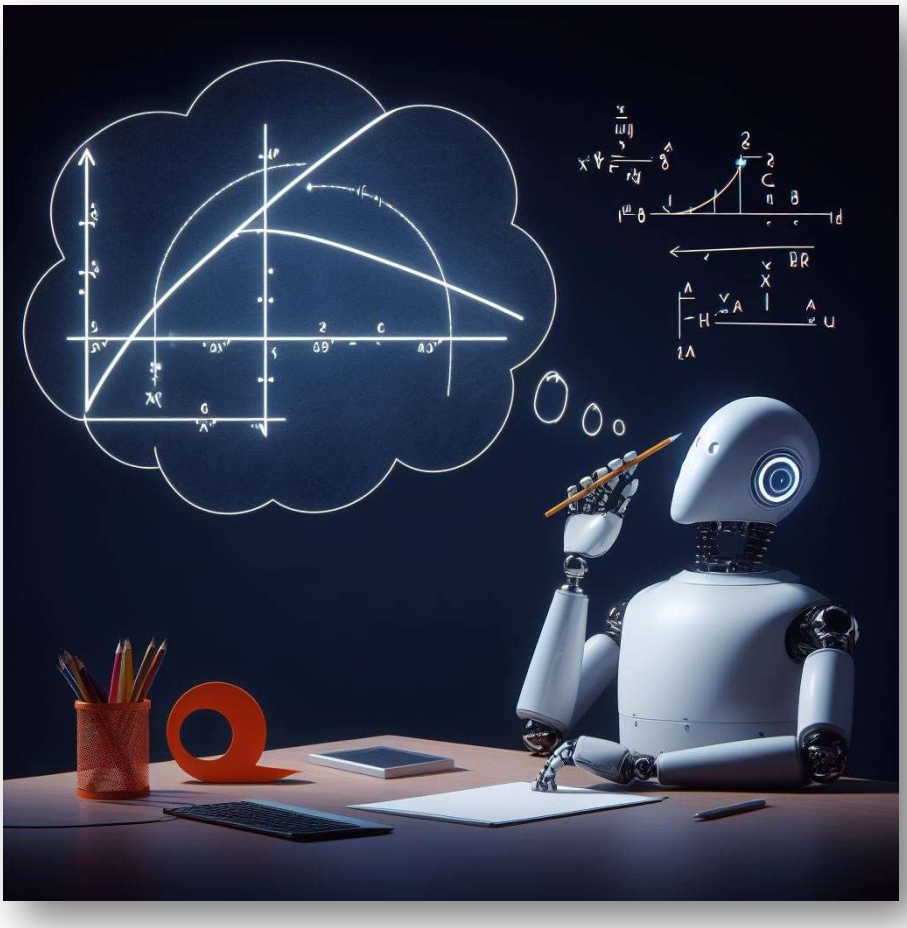
```
>>> %Run -c $EDITOR_CONTENT
  De vierkantsvergelijking heeft de vorm ax²+bx+c=0
  Geef de coëfficiënten in

  a =1
  b =0
  c =-1

  De nulpunten van de vergelijking zijn 1.0 en -1.0
>>>
```

Voorbeeld 2:

```
>>> %Run -c $EDITOR_CONTENT
  De vierkantsvergelijking heeft de vorm ax²+bx+c=0
  Geef de coëfficiënten in

  a =1
  b =-2
  c =1

  Het nulpunt van de vergelijking is 1.0
>>>
```

Voorbeeld 3:

```
>>> %Run -c $EDITOR_CONTENT
  De vierkantsvergelijking heeft de vorm ax²+bx+c=0
  Geef de coëfficiënten in

  a = 9
  b =2
  c =3

  Geen nulpunten
>>>
```



Verkenning 2

Opdracht 2.2

Ontwerp een algoritme en schrijf een script dat aan de gebruiker een jaartal vraagt en dan bepaalt of het jaar dat behoort tot het jaartal een schrikkeljaar is of niet.

Wanneer is een jaar een schrikkeljaar?

Elk jaar dat evenredig deelbaar is door 4 is een schrikkeljaar: bijvoorbeeld 1988, 1992 en 1996 zijn schrikkeljaren.

Er is echter nog steeds een kleine fout die moet worden verantwoord. Om deze fout te elimineren, bepaalt de Gregoriaanse kalender dat een jaar dat gelijkmatig deelbaar is door 100 (bijvoorbeeld 1900) alleen een schrikkeljaar is als deze ook gelijkmatig deelbaar is door 400.

Voorbeeld:

```
Geef een jaartal in: 1996
Schrikkeljaar

>>>
```

```
Geef een jaartal in: 2000
Schrikkeljaar

>>>
```

```
Geef een jaartal in: 1900
Geen schrikkeljaar

>>>
```

TIP!

Bedenk eerst welke **condities** je moet controleren, en bij welke combinatie van condities het een schrikkeljaar is, en wanneer niet.

Opdracht 2.3

In je tuin staat een parasol. Je gebruikt de parasol enkel om onder te zitten als de zon schijnt voor de schaduw, of als het regent om niet nat te worden.

Condities:

- **Conditie 1:** Het regent (?)
- **Conditie 2:** De zon schijnt (?)

Vul onderstaande tabel aan.

- Schrijf in de tabel onder conditie 1:
    - 1: als aan conditie 1 voldaan is. (Als het regent)
    - 0: als aan de conditie niet voldaan is. (Als het niet regent)
- Doe hetzelfde voor conditie 2
- En tenslotte 1 als je onder de parasol gaat zitten, of 0 als je niet onder de parasol gaat zitten

Weer	Conditie 1	Conditie 2	Onder de parasol?
Bewolkt zonder regen			
Blauwe lucht zonder regen en stralende zon			
Het regent pijpenstelen met een donker wolkendek			
De regen en de zon laten een mooie regenboog zien.			

Opdracht 2.4

Controleer of de steden in de tabel Europese hoofdsteden zijn.

Schrijf 2 condities op waaraan een stad moet voldoen om een Europese hoofdstad te zijn.

Vul daarna de tabel aan.

Stad	Conditie 1	Conditie 2	Europese hoofdstad?
Rio de Janeiro			
Hasselt			
Ottawa			
Brussel			

Logische operatoren

Python kent 3 logische operatoren. Deze kunnen gebruikt worden

- **AND**: het resultaat zal waar (True) zijn als aan beide condities voldaan is.
- **OR**: het resultaat zal waar (True) zijn als aan minimum één van beide condities voldaan is.
- **NOT**: deze logische operator gaat geen condities combineren, maar gaat het resultaat van een conditie omkeren, dus True wordt False, en False wordt True.

Opdracht 2.5

In het studentenrestaurant van UHasselt krijgen volgende categorieën korting:

- Studenten van UHasselt
- Werknemers van UHasselt

1) Vul onderstaande tabel aan.

Persoon	Student (?)	Werknemer (?)	Korting (?)
Een student geneeskunde			
Een praktijkassistente fysica			
Een gepensioneerd docent			
Een praktijkassistente die ook een extra masteropleiding volgt			

2) Ontwerp daarna een algoritme dat bepaald of iemand korting krijgt of niet. Je mag wel maar één selectie hebben in je algoritme.

3) Zet je algoritme om in een Python script.

Finaliseren

Meerdere condities combineren kan leiden tot complexe structuren. Nemen we terug het voorbeeld van opdracht 3.2

Je hebt een schrikkeljaar als het jaartal:

- Deelbaar is door 4 **EN** niet deelbaar door 100

OF

- Deelbaar door 400

Passen we dit toe in een Python script dan krijgen we:

```
#Schrikkeljaar bepalen
jaar=int(input("Geef een jaartal in: "))

# Bepaal de rest van jaar/4
quotient = int(jaar/4)
rest4 = jaar - quotient*4

# Bepaal de rest van jaar/100
quotient = int(jaar/100)
rest100 = jaar - quotient*100

# Bepaal de rest an jaar/400
quotient = int(jaar/400)
rest400 = jaar - quotient*400

if (rest4 == 0 and rest100 != 0) or rest400 == 0:
    print("Schrikkeljaar")
else:
    print("Geen schrikkeljaar")
```

Inoefenen

Oefening 2.1: Kleinste van 2 getallen

Gegeven twee gehele getallen.

Schrijf een script dat het kleinste getal afdrukt op het scherm.***Voorbeeld:***

```
Kleinste van 2 gehele getallen
Eerste getal: 12
Tweede getal: -3
Het kleinste getal is -3

>>>
```

Oefening 2.2: Even of oneven

Gegeven een geheel getal.

Schrijf een script dat ons vertelt of het getal “EVEN” of “ONEVEN” isVoorbeeld:

Voorbeeld:

```
Is het getal even of oneven?
Geef getal in: 13
Het getal is ONEVEN

>>>
```

Oefening 2.3: Het zwarte schaap

Gegeven 3 gehele getallen, waarvan er 2 aan elkaar gelijk zijn.

Schrijf een script dat laat weten welk van de 3 getallen het afwijkende getal (het zwarte schaap) is.

Er zijn dus 3 mogelijke antwoorden:

- “Het eerste getal, nl. … is het zwarte schaap.”
- “Het tweede getal, nl. … is het zwarte schaap.”
- “Het derde getal, nl. … is het zwarte schaap.”

Voorbeeld:

```
Welk getal is het zwarte schaap?
--------------------------------

Geef 3 gehele getallen in.
Twee van deze getallen moeten gelijk aan elkaar zijn,
één getal moet verschillend zijn
Geef het eerste getal: 12
Geef het tweede getal: 13
Geef het derde getal: 12
Het tweede getal, nl. 13 is het zwarte schaap

>>>
```

Oefening 2.4: Getal van 3 cijfers

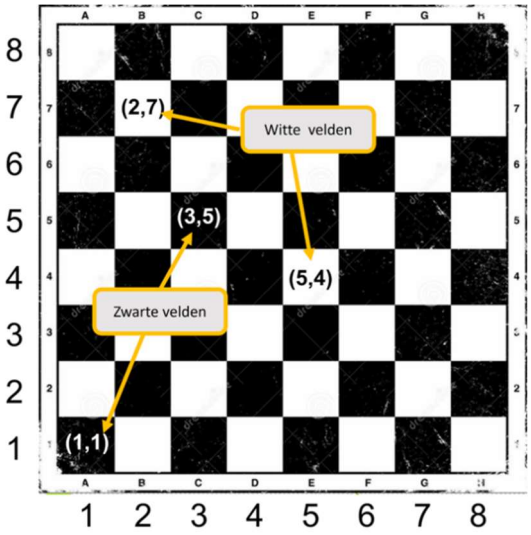
Gegeven een natuurlijk getal.

Schrijf een script dat het antwoord is op de vraag “Bestaat het ingevoerde getal uit exact 3 cijfers?”

Verdieping: het schaakspel

Achtergrond informatie

Bekijk bijgevoegde tekening.



Een schaakbord bestaat uit zwarte (donkere) en witte (lichte) velden.

Elk veld wordt aangeduid door een combinatie van kolomnummer (k) en rijnummer (r). Bij het verwijzen naar een veld geven we steeds eerst het kolomnummer door en dan pas het rijnummer.

Opgelet!

De kolom helemaal links heeft kolomnummer 1, de kolom helemaal rechts heeft kolomnummer 8.

De rij helemaal onderaan heeft rijnummer 1, de rij helemaal bovenaan heeft rijnummer 8.

Het veld (1,1), het veld links, onderaan is een 'donker' veld.

Het veld (5,4), het veld in de 5de kolom, en 4de rij is een 'licht' veld.

Oefening 2.5: Donker of licht veld

Gegeven een veld op het schaakbord.

Schrijf een script dat antwoord geeft op de vraag "Is het een 'donker' of 'licht' veld?".

Er zijn twee mogelijke antwoorden:

- "Donker veld", als het opgegeven veld donker is
- "Licht veld", als het opgegeven veld licht is.

Oefening 2.6: Zelfde kleur

Gegeven twee velden op een schaakbord.

Schrijf een script dat antwoord geeft op de vraag "Hebben beide velden dezelfde of een verschillende kleur?"

Er zijn twee mogelijke antwoorden:

- "Zelfde kleur", als beide velden dezelfde kleur hebben
- "Verschillende kleur", als beide velden een verschillende kleur hebben.

Verdieping: Spelen met getallen

Oefening 2.7: Minstens één teken negatief

Gegeven twee getallen.

Schrijf een script dat antwoord geeft op de vraag “Is er minstens één negatief getal ingegeven?”

Er zijn twee mogelijke antwoorden:

- “Minstens één getal is negatief”, als één of beide getallen negatief is.
- “Geen negatieve getallen”, als beide getallen positief zijn.

Oefening 2.8: Zelfde teken

Gegeven twee getallen.

Schrijf een script dat antwoord geeft op de vraag “Hebben beide getallen hetzelfde teken?”

Er zijn twee mogelijke antwoorden:

- “De getallen hebben hetzelfde teken”, als de getallen beide negatief of beide positief zijn.
- “De getallen hebben verschillende tekens”, als één van de getallen positief, en het andere negatief is.

Oefening 2.9: sorteren

Vervolledig onderstaande code zodat de getallen die de gebruiker ingeeft gesorteerd op het scherm komen.

De gegeven code mag niet aangepast worden, je mag enkel code toevoegen.

```
# oefening 2.9

print("Geef 3 gehele getallen in")
getal1=int(input("   getal 1: "))
getal2=int(input("   getal 2: "))

# code aanvullen

# Output
print(getal1,"is kleiner dan",getal2)
```

Voorbeeld:

```
Geef 2 gehele getallen in
   getal 1: 9
   getal 2: 5
5 is kleiner dan 9

>>>
```

Verdieping: palindroom

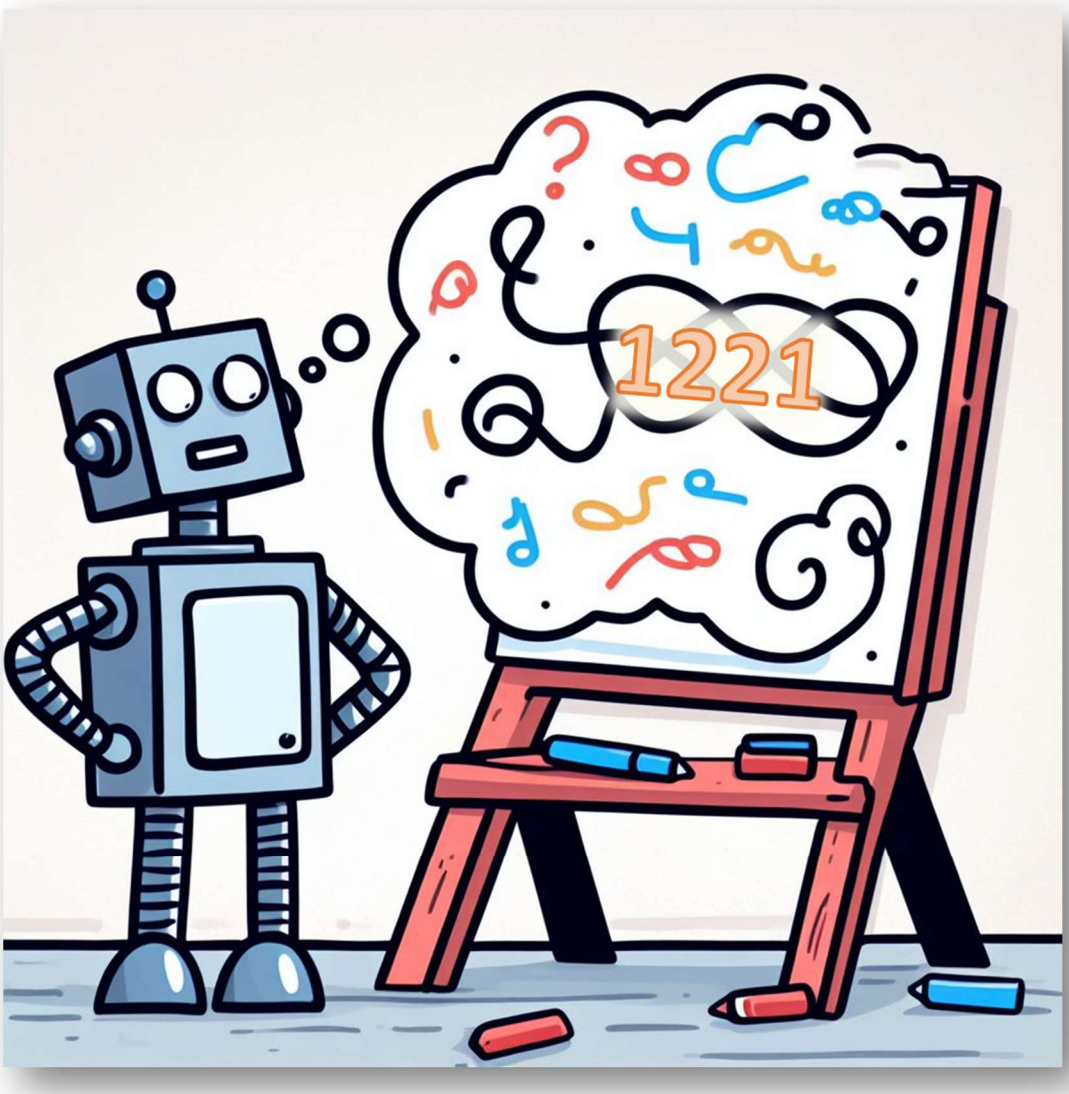
Oefening 2.10

We noemen een integer een cijferpalindroom wanneer het zowel van voor naar achter als van achter naar voor kan gelezen worden
(voorbeeld: 1991).

Gegeven een natuurlijk getal van 4 cijfers.

Schrijf een script dat laat weten of het ingevoerde getal al dan niet een cijferpalindroom is.

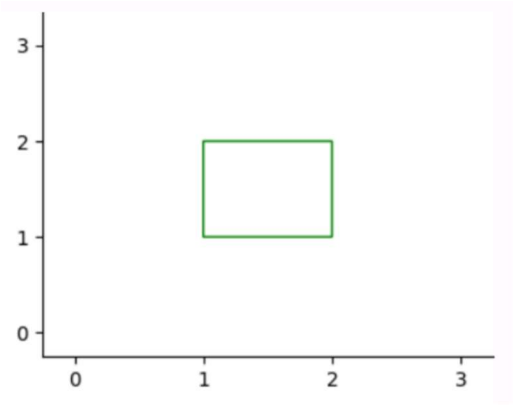
De mogelijke antwoorden zijn “EEN CIJFERPALINDROOM” en “GEEN CIJFERPALINDROOM”.



Verdieping: Hoekpunten van een rechthoek

Oefening 2.10: Vierde hoekpunt

Gegeven: 3 hoekpunten van een rechthoek



Schrijf een script dat de coördinaten van het 4de hoekpunt vindt.

We gaan uit van een rechthoek, waarvan de zijden evenwijdig zijn met de assen van een rechthoekig assenstelsel.

Van de drie gekende hoekpunten, ontvangen we de x- en y-coördinaten onder de vorm van zes gehele getallen..

Oefening 2.11: Het vierde hoekpunt bis

In de vorige oefening stonden waren de zijden van de rechthoek evenwijdig met de assen van het assenstelsel. Pas het script aan zodat dit niet meer noodzakelijk is.

Tip:

- Chat GPT gaf op 3/11/2023 geen juiste code.
- Bepaal eerst met welk punt je samen met de twee andere een rechte hoek kunt vormen.

Module 3: Iteratie (lussen)

Wat behandelen wij in deze module?

Bij het gestructureerd programmeren zijn er drie programmeercontructies die steeds terugkomen. We vinden deze dan ook terug in het NSD.
Deze structuren zijn:

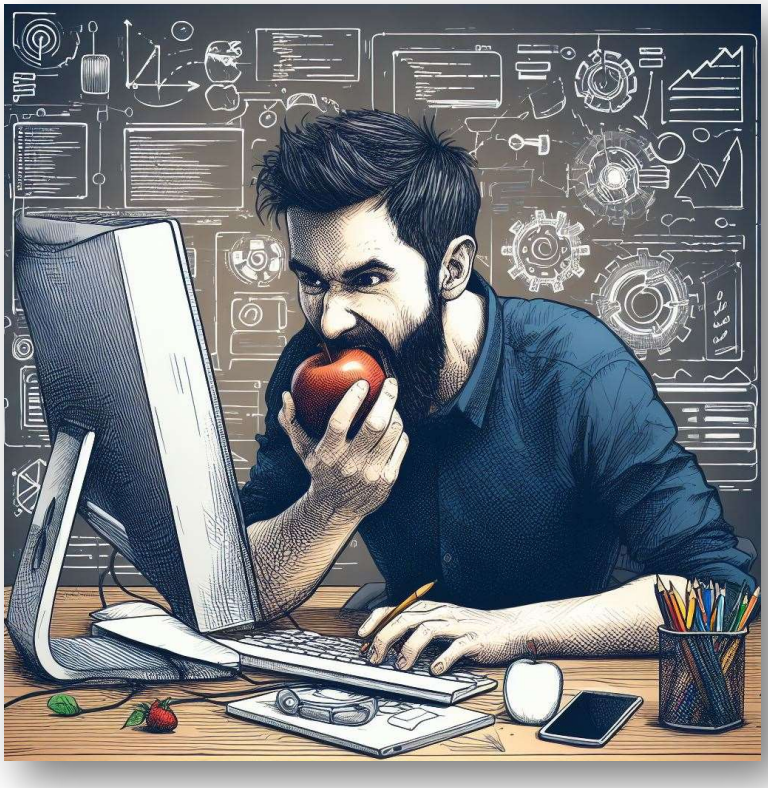
- sequentie
- selectie (met conditie)
- iteratie

De eerste twee van deze structuren hebben we reeds behandeld. In Module 3 gaan we naar de iteratie kijken.

Een iteratie verwijst naar een herhaling van een bepaald stuk code of een reeks instructies. Het stelt ontwikkelaars in staat om een bepaalde actie of set van acties meerdere keren uit te voeren, vaak op basis van bepaalde voorwaarden.

De tekening hieronder illustreert dit. Om de appel op te eten gaat de programmeur in de appel moeten blijven bijten totdat hij op is, daarna mag hij de appel weggooien.

Meestal wordt een iteratie gedaan met behulp van een zogenaamde "lus" (loop) in de programmeertaal. Een lus bevat een blok code dat wordt herhaald totdat aan een bepaalde voorwaarde is voldaan. Bij elke iteratie wordt de code binnen de lus uitgevoerd, en wanneer de voorwaarde niet meer waar is, eindigt de lus en gaat het script verder met de volgende instructies.



Verkenning

Opdracht 3.1

Een priemgetal is een natuurlijk getal dat groter is dan 1 en alleen door zichzelf en door 1 gedeeld kan worden.

Onderzoek of de getallen 15 en 17 priemgetallen zijn.

Noteer goed welke stappen je onderneemt om dit te achterhalen.

Wanneer ga je stoppen met controleren of het een priemgetal is?

Iteratie

Bij opdracht 3.1 zal je een aantal keren dezelfde stap(pen) doorlopen hebben.

Je bent waarschijnlijk beginnen kijken of 71 deelbaar is door 2, dan door 3, … enz.

Zou je dit in een algoritme plaatsen, dan gaat een gedeelte van dit algoritme doorlopen worden zolang er aan een conditie voldaan is.

Je zou bijvoorbeeld voor alle getallen van 2 tot 70 kunnen controleren of 71 door dit getal deelbaar is, maar je stopt als je een deler gevonden hebt.

Een mogelijk algoritme is:

1) getal ← 15 of 17
2) deler ← 2
3) **Zolang** getal/deler geen rest heeft herhaal
    - deler ← deler +1
    - **Als** rest gelijk is aan 0 dan
4) **Als** deler gelijk is aan getal is
    - Output: Priemgetal

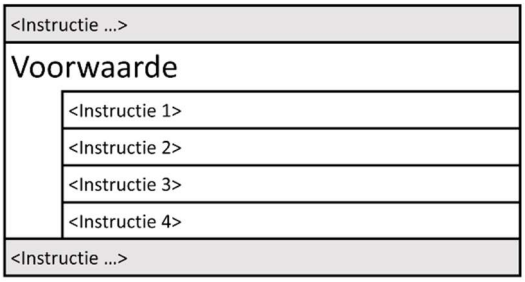
    Anders
    - Output: Geen priemgetal

Grafische voorstelling iteratie

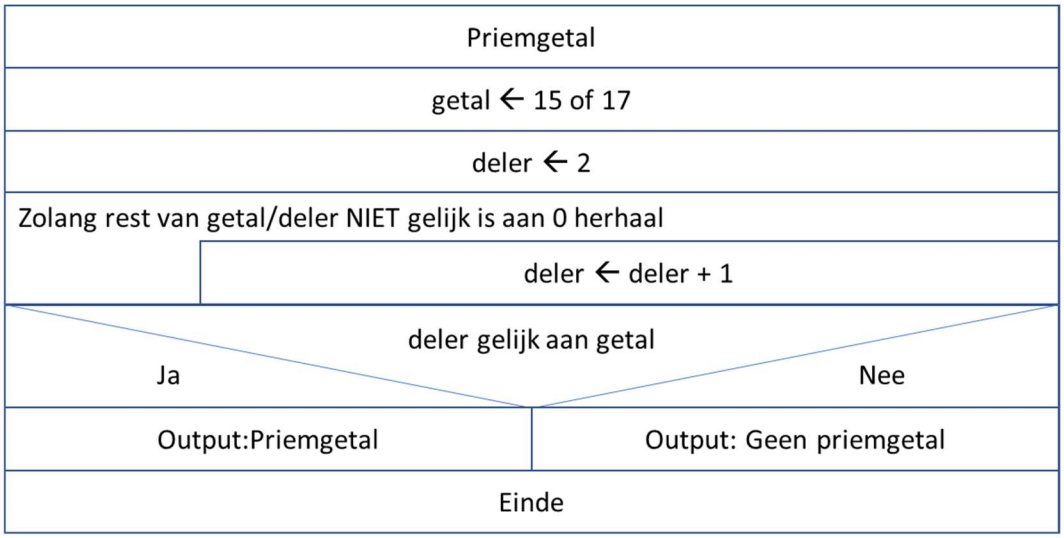
iernaast wordt getoond hoe en iteratie (soms ook loop genoemd) in een NSD voorgesteld .

het voorbeeld zijn instructies 1 tot/met 4 een sequentie die zal herhaald worden (iteratie) zolang de voorwaarde voldaan is.

De voorwaarde zal eerst getest worden vooraleer de instructies uitgevoerd worden. Het is dus mogelijk dat de lus niet doorlopen wordt.



Passen we dit toe op de opdracht 3.1.



Python code: while....

Er zijn verschillende manieren om in Python een iteratie of loop te programmeren.

Wij beperken on in eerste instantie tot de **while-loop**.

```
while <conditie> :
  <instructie 1>
  <instructie 2>
  ....
```

De code "**while**" is gewoon het Engelse woord voor **zolang**, waarna de conditie volgt.
Hierna start het blok met instructies die gaat uitgevoerd worden, zolang de conditie vervuld is. Om dit aan te geven wordt er na de conditie een dubbele punt (:) geplaatst. De instructies van het blok die moeten herhaald worden, worden dan geïntendeerd (springen in).

Zetten we dit om naar Python code, dan krijgen we

```
# Priemgetallen

getal=int(input("Geef een natuurlijk getal in:"))

deler = 2

while getal/deler > getal//deler:
    deler  =deler+1

if deler == getal:
    print("Priemgetal")
else:
    print("Geen priemgetal")
```

Input controleren (beperken)

Je kan aan een gebruiker wel zeggen waaraan een data moet voldoen, maar dat wilt niet steeds zeggen dat de gebruiker zich daaraan houdt.

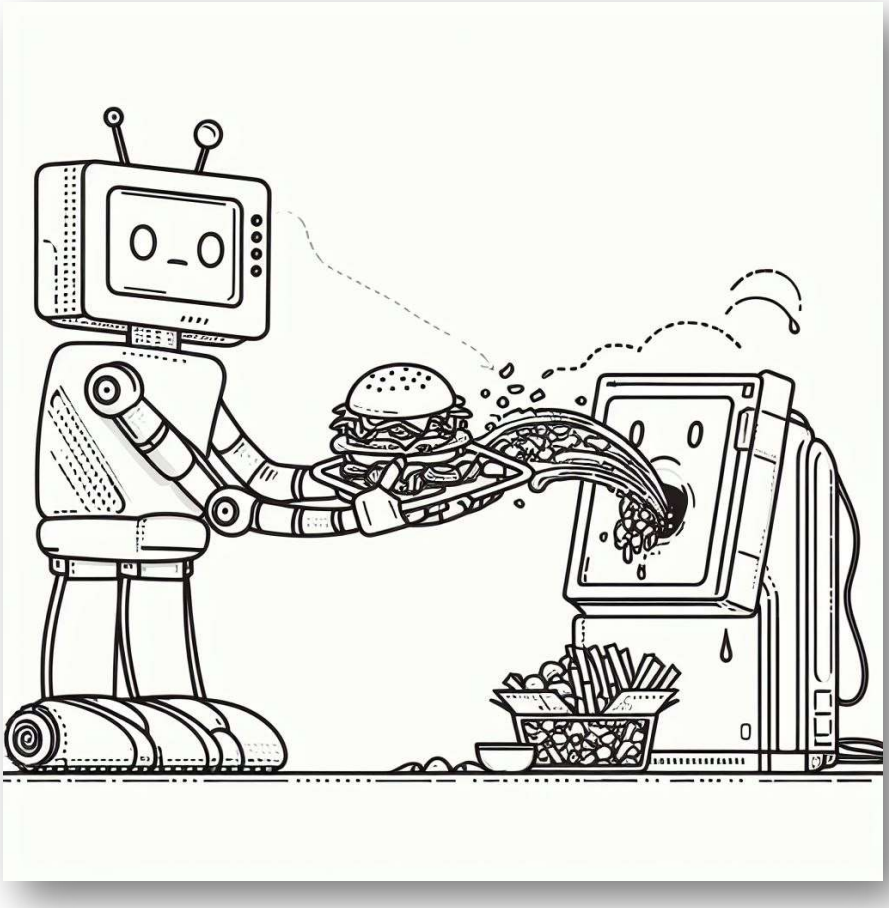
Je gaat daarom moeten controleren of een input van de gebruiker voldoet aan de gestelde voorwaarden, en indien niet zo, vragen om opnieuw de data in te geven.

We gaan in de volgende opdracht eerst een script schrijven waarbij we de input niet gaan controleren, in de daaropvolgende opdracht gaan we dan een controle invoeren.

Opdracht 3.2
Schrijf een algoritme en een script dat aan de gebruiker vraagt een getal in te geven.

Geef alle delers van het getal.

Opdracht 3.3
Herhaal opdracht 3.2, maar zorg er nu voor dat enkel een getal van 10 t/m 20 geaccepteerd wordt.



Inoefen

Oefening 3.1

Vraag de gebruiker om een natuurlijk getal. Druk alle getallen af van 0 tot en met dat getal

Voorbeeld:

```
>>> %Run -c $EDITOR_CONTENT
  Geef een natuurlijk getal: 5
  0
  1
  2
  3
  4
  5

>>>
```

Oefening 3.2

Vraag aan een gebruiker om twee natuurlijke getallen in te geven. Druk alle even getallen af vanaf het kleinste van de twee getallen tot en met het grootste.

Voorbeeld;

```
>>> %Run 'even getallen in range.py'
  Geef eerste getal in: 12
  Geef tweede getal in: 16
  12
  14
  16
>>>
```

```
>>> %Run 'even getallen in range.py'
  Geef eerste getal in: 17
  Geef tweede getal in: 12
  12
  14
  16
>>>
```

```
>>> %Run 'even getallen in range.py'
  Geef eerste getal in: 11
  Geef tweede getal in: 16
  12
  14
  16
>>>
```

Oefening 3.3

Vraag aan de gebruiker welke tafel van vermenigvuldiging je moet printen en print deze.

Voorbeeld:

```
>>> %Run -c $EDITOR_CONTENT
  Welke tafel? 5
  1 x 5 = 5
  2 x 5 = 10
  3 x 5 = 15
  4 x 5 = 20
  5 x 5 = 25
  6 x 5 = 30
  7 x 5 = 35
  8 x 5 = 40
  9 x 5 = 45
  10 x 5 = 50
>>>
```

Oefening 3.4

Met onderstaande code kan je de computer een willekeurig natuurlijk getal laten generen tussen de opgegeven getallen. In ons geval 1 en 6.

```
# Script om een getal van 1 tot en met 6 te genereren
# Importeer de random module
import random
x = random.randint(1,6)
# Vul hieronder de code zelf aan
```

Laat de gebruiker een getal ingeven van 1 tot 10.

Laat de computer dan zoveel keren een getal genereren en print dit getal op het scherm.

Verdieping: Getallen raden

Oefening 3.5:

Schrijf een script dat aan de gebruiker een natuurlijk getal vraagt tussen 1 en 20.
Als de gebruiker een getal ingeeft dat niet voldoet aan de voorwaarde moet het script een nieuwe ingave vragen, anders geeft het als output: “Voldoet”.

```
  Geef een natuurlijk getal in van 1 tot/met 20: 5.2
  Geef een natuurlijk getal in van 1 tot/met 20: -3
  Geef een natuurlijk getal in van 1 tot/met 20: -3
  Geef een natuurlijk getal in van 1 tot/met 20: -3.5
  Geef een natuurlijk getal in van 1 tot/met 20: 4
  Voldoet

>>>
```

Oefening 3.6:

Laat de computer een getal van 1 tot 20 genereren;

Laat de gebruiker dan het getal raden.

Bij elke poging geeft de computer aan of het getal te hoog of te laag is.

Als de gebruiker het getal geraden heeft, geeft de computer het aantal pogingen dat nodig was op het scherm weer.

Voorbeeld:

```
>>> %Run -c $EDITOR_CONTENT
  Welk getal denk je dat we zoeken? 10
  Het getal dat we zoeken is groter dan 10

  Welk getal denk je dat we zoeken? 15
  Het getal dat we zoeken is kleiner dan 15

  Welk getal denk je dat we zoeken? 12
  Het getal dat we zoeken is kleiner dan 12

  Welk getal denk je dat we zoeken? 11
  Het getal dat we zoeken is groter dan 11

  Proficiat! Het getal dat we zochten was 11
  Je had 4 pogingen nodig.
>>>
```

Oefening 3.7:

Draai oefening 3.6 nu om. Laat de gebruiker een getal ingeven van 1 tot/met 20 en laat de computer raden naar het getal.

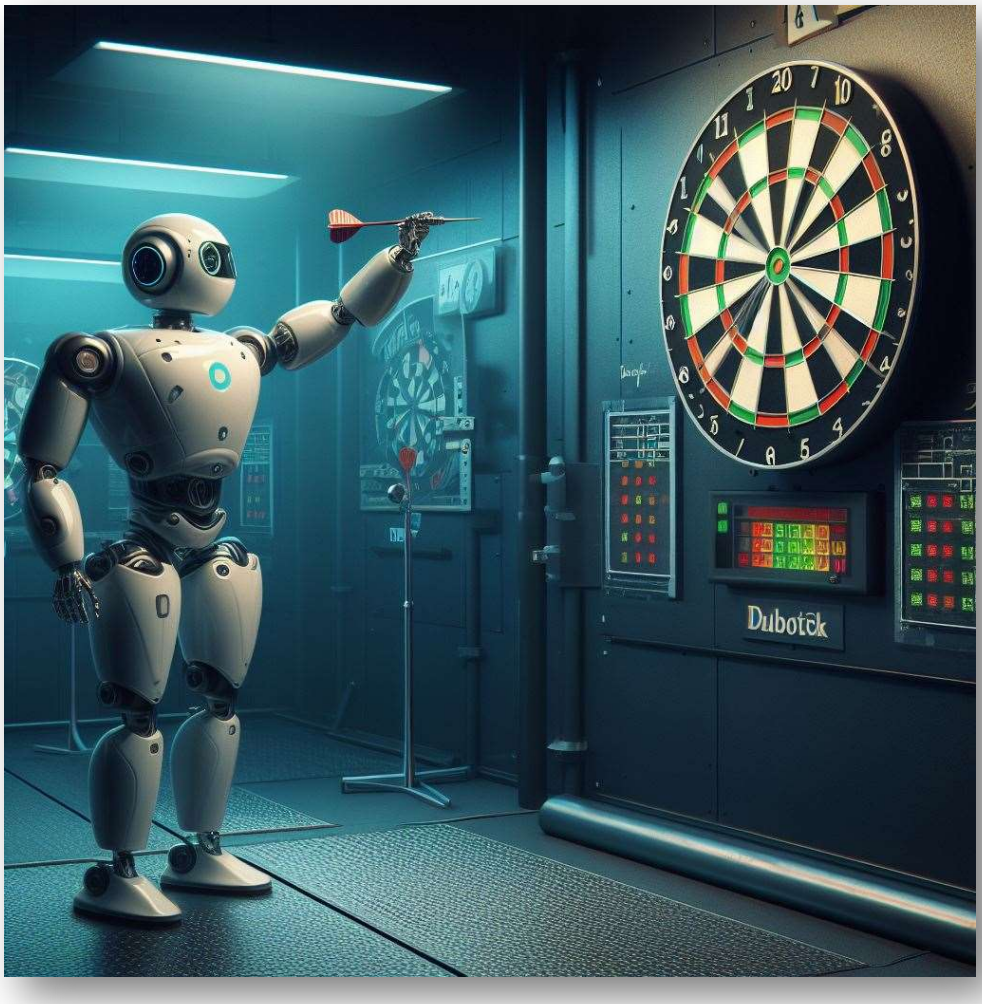
De gebruiker geeft steeds aan of het getal dat gezocht wordt groter (G), kleiner (K) of gelijk(=) aan het getal dat gezocht wordt.

Als de computer het getal geraden heeft, geeft de computer het aantal pogingen dat nodig was op het scherm weer.

Je kan de computer natuurlijk gewoon laten gokken, maar er bestaat een algoritme waarbij de computer in maximaal 5 stappen het juiste getal raadt.

Voorbeeld:

```
>>> %Run -c $EDITOR_CONTENT
  Welk getal kies je ? 13

  Ik kies: 10
  Is het getal dat gezocht wordt groter (G), kleiner (K) of gelijk (=) aan dit getal? G

  Ik kies: 15
  Is het getal dat gezocht wordt groter (G), kleiner (K) of gelijk (=) aan dit getal? K

  Ik kies: 12
  Is het getal dat gezocht wordt groter (G), kleiner (K) of gelijk (=) aan dit getal? G

  Ik kies: 13
  Is het getal dat gezocht wordt groter (G), kleiner (K) of gelijk (=) aan dit getal? =

  Joepie! Het getal dat ik zocht was 13
  IK had 4 pogingen nodig.
>>>
```



Verdieping: Fibonacci in de natuur

Inleiding

Wiskunde en natuur komen samen als we tussen al het groen speuren naar de Rij van Fibonacci en die nog vinden ook!

In zijn boek 'Liber Abaci' presenteerde Leonardo van Pisa (ook wel Fibonacci genoemd) in de dertiende eeuw een bijzondere rij cijfers. De rij is tegenwoordig beter bekend als de Rij van Fibonacci. Hoewel de naam doet vermoeden dat Leonardo van Pisa de rij ontdekte, is dat onterecht. In India waren wiskundigen al veel eerder op de bijzondere rij gestuit.

De rij

De Rij van Fibonacci ziet er als volgt uit: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,…

De rij wordt verkregen door de twee getallen die aan x voorafgaan bij elkaar op te tellen. Dus: 2 en 3 maakt 5, 3 en 5 maken 8, 5 en 8 maken 13, enzovoort.

Voorbeelden uit de natuur

- **Wilde bertram**: De plant Achillea ptarmica (Wilde bertram) volgt met zijn takken netjes de Rij van Fibonacci.
- **Dennenappel**: De spiralen van een dennenappel in beide richtingen leveren netjes getallen uit de Rij van Fibonacci op. Bij de zonnebloem zien we datzelfde
- **Ons lichaam**: Ons DNA 34 ångström (1 ångström is 0,1 nanometer) lang en 21 ångström breed.

Dat Fibonacci-getallen regelmatig in de natuur opduiken en elkaar ook vaak vergezellen, moge duidelijk zijn. Maar waarom is dat nu eigenlijk? Soms zal het misschien toeval zijn. Maar in veel gevallen is het gewoon de beste manier om zaken (bijvoorbeeld zaden) te rangschikken. Neem bijvoorbeeld de zonnebloem. Door deze manier van rangschikken kan de bloem in het hart de meeste zaden kwijt. En hoe meer zaden, hoe groter de kans op een succesvolle voortplanting! En planten die hun blaadjes volgens de Rij van Fibonacci rangschikken, doen dat vaak om zoveel mogelijk zonlicht te vangen. Voor hen is deze wiskundige regel een zaak van levensbelang.

Oefening 3.8:

Schrijf een script dat de eerste 20 getallen van Fibonacci berekent en toont op het scherm.

Uitbreiding: Geneste lussen

Soms kan je in een lus een andere lus hebben. Dit worden geneste lussen genoemd.

Dit is een simpel uitspraak, maar soms een moeilijk concept om te begrijpen.

Neem er even het voorbeeld van de priemgetallen bij. In plaats van het script telkens opnieuw op te starten, kan je de code laten herhalen totdat de gebruiker 0 ingeeft als getal.

De extra code is aangeduid in het rood.

```
# Priemgetallen - tot nul ingegeven

getal = 1

while getal > 0:
  getal=int(input("Geef een natuurlijk getal in:"))

  deler = 2
  priemgetal = True

  while priemgetal==True:
        quotient = getal//deler
        rest = getal - quotient*deler
        if rest == 0:
              priemgetal = False
        else:
              deler=deler+1

        if deler == getal:
              print("Priemgetal")
        else:
              print("Geen priemgetal")
```

Lus A

Lus B

Je ziet hierbij dat “Lus B” volledig omsloten is door “Lus A”. Voordat er een nieuw cyclus van Lus A gestart wordt, zal Lus B een aantal keren herhaald worden totdat de conditie van deze lus (priemgetal == True) niet meer voldaan is. Daarna gaat de volgende cyclus van Lus A gestart worden.

Opdracht 3.4

Waarom staat **priemgetal == True** in Lus A, maar voordat Lus B gestart wordt?

Inoefenen geneste lussen

Oefening 3.8

Schrijf een script dat aan de gebruiker vraagt een natuurlijk getal in te geven van 1 tot en met 10.

Het script toont dan alle tafels vermenigvuldiging die kleiner of gelijk zijn aan het ingegeven getal.

Een voorbeeld is gegeven.

```
Geef een natuurlijk getal van 1 tot/met 10: 2
Tafel van 1
1 x 1 = 1
2 x 1 = 2
3 x 1 = 3
4 x 1 = 4
5 x 1 = 5
6 x 1 = 6
7 x 1 = 7
8 x 1 = 8
9 x 1 = 9
10 x 1 = 10
Tafel van 2
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20

>>>
```

Oefening 3.9

Schrijf een script dat aan de gebruiker een natuurlijk getal vraagt. Het script print dan alle priemgetallen, kleiner of gelijk aan dit getal.

Het script blijft dit herhalen totdat je getal “0” ingeeft.

Oefening 3.10

Elk getal kan voorgesteld worden als sommen van Fibonacci-getallen, maar er is slechts één representatie die elk Fibonacci-getal maximaal één keer gebruikt en opeenvolgende paren van Fibonacci-getallen vermijdt; deze unieke voorstelling staat bekend als de Zeckendorf-voorstelling.

Schrijf een script dat aan de gebruiker een natuurlijk getal vraagt en dan de Zeckendorf-voorstelling van dit getal geeft.

Module 4: lijsten

Doel:

In deze module introduceren we lijsten, lists, We leggen je uit hoe je een lijst kunt maken en kunt modificeren door elementen toe te voegen, er uit te verwijderen en meer.

In de praktijk zul je bij het werken met data veel lijsten tegenkomen. De lengte van deze lijsten kan ook enorm verschillen. Het aantal mogelijke lijsten is oneindig, hierbij echter een aantal voorbeelden van mogelijke lijsten:

- Diersoorten die voorkomen in een bepaald verblijf
- Namen van leden uit een team
- Ingrediënten voor een recept
- Finishers Iron Man 2025 in chronologische volgorde
- Het aantal keer dat een Lottoballetje gevallen is.

Je kunt lijsten gebruiken om allerlei soorten informatie te achterhalen. Python is hier uitermate geschikt voor.

Lijsten kunnen zeer complexe vormen aannemen. In deze cursus richten wij op het verkrijgen van een goede basis (ten koste behandelde leerstof). Om die reden beperken wij ons hier tot eendimensionale lijsten.

Net zoals in de vorige modules gaan we eerst van start met blad en papier. Deze “old school” oplossing vormt het startpunt van onze analyse. We hopen hierbij de noodzaak aan te tonen van deze datastructuur, waarna we verder op zoek gaan naar de verschillende manipulatie mogelijkheden.

Verkenning

Opdracht 4.1

Neem een dobbelsteen en gooi er 20 keren mee.

Hou bij hoeveel keer elk aantal ogen gegooid is.

Als je geen dobbelsteen hebt, dan kan je met onderstaande code een dobbelsteen simuleren.

```
Dobbelsteen

import random

tel = 0

while tel<20:
    print("Beurt: ",tel)
    print("Aantal ogen gegooid:",random.randint(1,6))
    wacht=input("druk op ENTER voor volgende beurt")
    print()
    tel=tel+1

print("Einde")
```

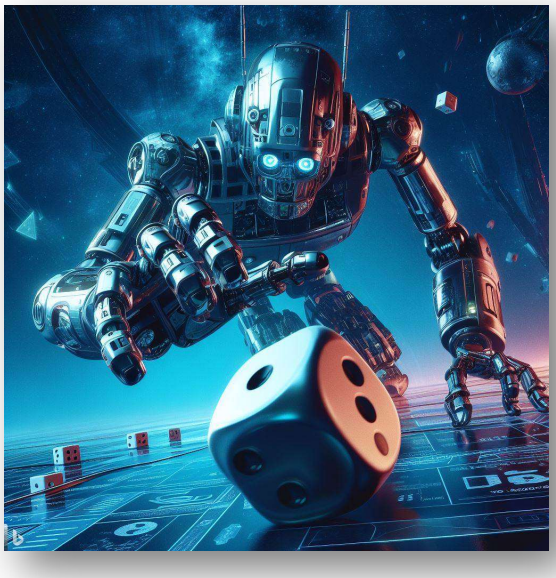
Je kan gebruik maken van onderstaande tabel om de aantallen bij te houden.

Aantal ogen	Aantal keer gegooid
1	
2	
3	

Aantal ogen	Aantal keer gegooid
4	
5	
6	

Opdracht 4.2

Ontwerp een algoritme en vertaal dit in een Python script waarmee je dezelfde opdracht als opdracht 4.1 kunt uitvoeren. Na het ingeven van de data print het script een overzicht van het aantal keren dat elk aantal ogen gegooid is.



Inleiding

Als je niet stiekem vals gespeeld hebt en op het internet, of in deze cursus op zoek geweest bent naar een oplossing, dan lijkt je script waarschijnlijk een beetje op onderstaand script.

```
#opdracht 4.2

tel = 0
aantal_1 = 0
aantal_2 = 0
aantal_3 = 0
aantal_4= 0
aantal_5 = 0
aantal_6 = 0

while tel<20:
    tel = tel + 1
    print("Beurt:",tel)
    ogen = int(input("Aantal ogen gegooid: "))
    if ogen==1:
        aantal_1 = aantal_1 + 1
    else:
        if ogen==2:
            aantal_2 = aantal_2 + 1
        else:
            if ogen==3:
                aantal_3 = aantal_3 + 1
            else:
                if ogen==4:
                    aantal_4 = aantal_4 + 1
                else:
                    if ogen == 5:
                        aantal_5 = aantal_5 + 1
                    else:
                        aantal_6 = aantal_6 + 1

print()
print("Aantal keren gegooid:")
print("  1 oog: ",aantal_1)
print("  2 ogen:",aantal_2)
print("  3 ogen:",aantal_3)
print("  4 ogen:",aantal_4)
print("  5 ogen:",aantal_5)
print("  6 ogen:",aantal_6)
```

Je gebruikt hierin 6 verschillende variabelen, ééntje voor elk aantal ogen.

Hoewel je waarschijnlijk voelt dat deze oplossing niet de ideale oplossing is, is deze werkwijze met 6 tellers nog werkbaar.

Willen we nu hetzelfde gaan doen met de balletjes van de Lotto, dan ga ik al moeten werken met 45 tellers, en dan is deze werkwijze niet meer werkbaar. We gaan een andere structuur moeten gebruiken, namelijk lijsten.

Lijsten

Een lijst (list) is een verzameling (of “collectie”) elementen.

De elementen van een lijst zijn geordend. Omdat ze geordend zijn, kun je ieder element van een lijst benaderen via een index

In Python kun je lijst herkennen aan het feit dat de elementen van een lijst tussen vierkante haken ([]) staan.

Voorbeelden:

```
aantal = [5,4,3,3,2,3]
namen = ["Jan","Piet","An","Linda","Peter"]
boodschappen =["melk",2,"eiëren",12,"boter",1]
```

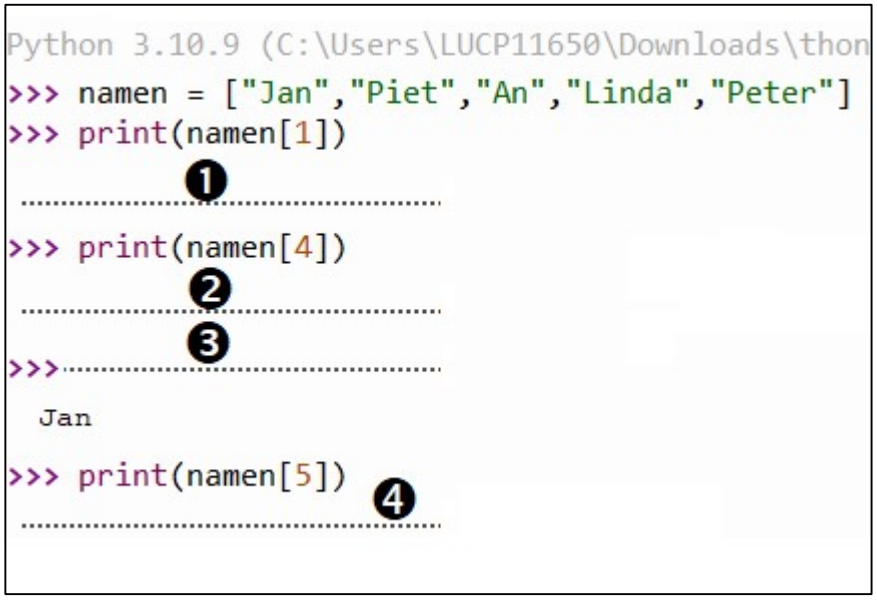
Je kan de elementen van een lijst individueel benaderen via hun index. Je gebruikt daarvoor de naam van de lijst gevolgd door de index van het element geplaatst tussen vierkante haken.

```
naam_lijst[index]
```

Opdracht 4.3

Bekijk onderstaande instructies. Schrijf op positie ❶❷❹ wat je verwacht dat geprint gaat worden.

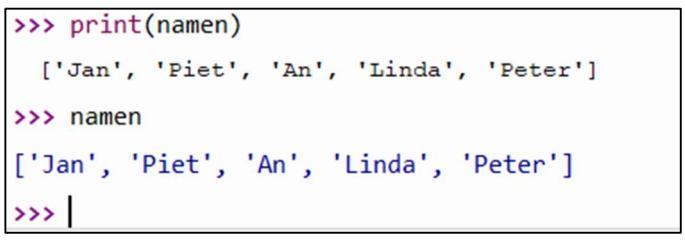
Schrijf op positie ❸ de instructie waarmee je ***Jan*** uit de lijst print.



Om je te helpen bestand ***opdracht 4_3.py*** openen. In het editor gedeelte staan de verschillende lijnen met instructies gegeven. Je kan met knippen en plakken deze in de shell invoeren.

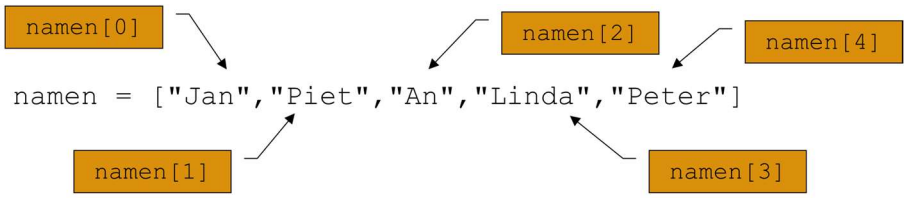
Je kan de volledige lijst afprinten door gewoon de naam van de lijst in de printopdracht te gebruiken.

Werk je in de shell, dan is zelf de print-instructie niet nodig.



Indices

Uit opdracht 4.3 blijkt duidelijk dat de indices bij nul beginnen, en dat je een foutmelding krijgt als je een index gebruikt die hoger is dan de index van het laatste element.



Het is dan belangrijk om te weten hoeveel elementen een lijst bevat. Je kan dit bijhouden met tellers. Nog makkelijker is met de instructie ***len(<lijstnaam>)***

```
>>> len(namen)
5
>>>
```

Dit wilt zeggen dat je index steeds kleiner moet zijn dan ***len(<lijstnaam>)-1***

Opdracht 4.4

Start met onderstaande lijst en print op het scherm de bloemnamen die in het vet (bold) zijn aangeduid.

Je vindt de lijst terug in bestand **opdracht4_4.py**

```
bloem =
["tulp","anjer","roos","dahlia","lelie","madeliefje","boterbloem","beg
onia","freesia","goudsbloem"]
```

Een voorbeeld van je output is gegeven.

```
anjer
dahlia
madeliefje
begonia
goudsbloem
>>>
```

Bewerkingen met lijsten

Opdracht 4.5

Voer onderstaande bewerkingen uit in de shell. Je kan het effect controleren door al de elementen van de lijst op het scherm te tonen door de naam van de lijst te tonen. Omschrijf wat de bewerking doet.

Let op, sommige van deze bewerkingen zijn foutief en geven een foutboodschap.

In het bestand ***opdracht 4.5.py*** staan alle bewerkingen. Door ze één per één te kopiëren naar de shell kan je ze makkelijk uitvoeren.

Begindata:

```
jongens = ["Jan","Piet","Peter"]
meisjes = ["Linda","An"]
priem = [2,3,5,7]
volgend = 11
leeg=[]
```

Bewerkingen:

```
leeg = leeg +["Niet meer leeg"]
personen = jongens + meisjes
meisjes_plus = meisjes + "Kathleen"
jongens_plus = jongens + ["Erik"]
dubbel_meisjes = meisjes * 2
vol = leeg + dubbel_meisjes
dubbel_priem = 2*priem
priem_plus = priem + volgend
priem_plus = priem + [volgend]
jongens[1]="Pieter"
```

Opdracht 4.6

Vul onderstaande code aan zodat je een lijstje kan ingeven.

Wanneer je op ENTER drukt zonder iets in te vullen wordt het lijstje afgesloten.

Tenslotte geef je het volledige lijstje weer op het scherm.

```
#Opdracht 4.6

boodschappen=[]

print("Geef je boodschappen in.")
print("Eindig je lijstje door gewoon op ENTER te drukken.")
print()

toevoegen=input("Volgende boodschap: ")

while toevoegen !="":
    # code hier aanvullen
    # zorg ervoor dat je niet in een oneindige lus terecht komt.
```

Voorbeeld:

```
Geef je boodschappen in.
Eindig je lijstje door gewoon op ENTER te drukken.

Volgende boodschap: boter
Volgende boodschap: jam
Volgende boodschap: suiker
Volgende boodschap:
['boter', 'jam', 'suiker']

>>>
```

Opdracht 4.7

Bij een experiment wordt om de 0,02 seconden de positie van een vallend voorwerp bepaald. In totaal heb ik 30 metingen, maar je wilt de eerste 3 en de laatste 4 metingen niet gebruiken vanwege het overgangsverschijnsel.

```
metingen=[0.0000, 0.0019, 0.0078, 0.0176, 0.0314, 0.0491, 0.0706,
0.0961, 0.1256, 0.1589, 0.1962, 0.2374, 0.2825, 0.3316, 0.3846,
0.4415, 0.5023, 0.5670, 0.6357, 0.7083, 0.7848, 0.8652, 0.9496,
1.0379, 1.1301, 1.2263, 1.2263, 1.2263, 1.2263, 1.2263]
```

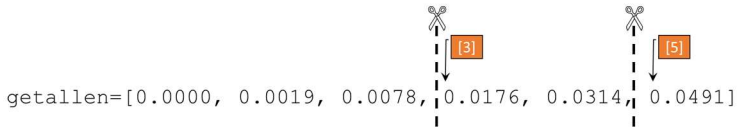
Moet je hiervoor een script schrijven of kan zoiets als metingen[3:26]?
Geef de juiste code op die je gaat gebruiken.

De methode noemt slicing. Bij slicing krijg je een deellijst waarbij het eerste getal het eerste element geeft dat bij de deellijst hoort, en het tweede getal het eerst volgende element dat juist niet meer tot de deellijst behoort.

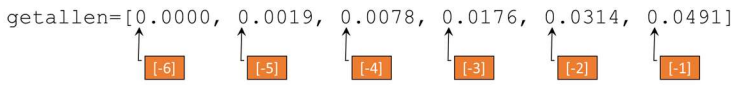
Hieronder vind je een voorbeeld met een kortere lijst

```
Shell
>>> %Run -c $EDITOR_CONTENT
>>> getallen
[0.0, 0.0019, 0.0078, 0.0176, 0.0314, 0.0491]
>>> getallen[3:5]
[0.0176, 0.0314]
>>>
```

Je krijgt dus:

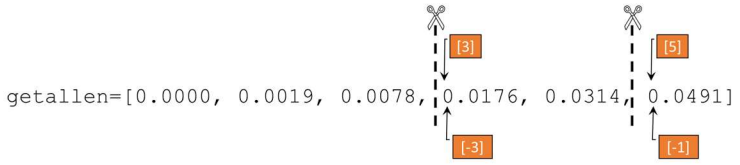


We hebben dit nog niet behandeld, maar je kan je index ook geven, startend van het einde van de lijst.



Je zou dus hetzelfde kunnen krijgen met

```
Shell
>>> %Run -c $EDITOR_CONTENT
>>> getallen
[0.0, 0.0019, 0.0078, 0.0176, 0.0314, 0.0491]
>>> getallen[-3:-1]
[0.0176, 0.0314]
>>>
```



Controleer of elemenent tot lijst behoort

Opdracht 4.8

Vervolledig onderstaande code.

Controleer of de ingegeven letter een klinker is. Indien het een klinker geeft het script als output “Klinker”, anders “Medeklinker”.

```
#Opdracht 4.7

klinker=["a","e","i","o","u"]

letter=input("Geef letter in: ")

# Vul hier de code aan.
```

Voorbeeld:

```
  Geef letter in: e
  Klinker

>>>
```

```
  Geef letter in: p
  Medeklinker

>>>
```

Uitbreiding: "in"

Je kunt testen of een element voorkomt in een list via de ***in*** operator (of het niet voorkomen van een element via de ***not in*** operator).

```
>>> richting=["links","rechts","boven","onder"]
>>> "links" in richting
True
>>> "schuin" in richting
False
>>> "schuin" not in richting
True
>>>
```

Inoefenen

Oefening 4.1

Pas het script van opdracht 4.2 aan zodat je gebruik maakt van lijsten.

Oefening 4.2

Vervolledig onderstaand script zodat de gegeven lijst gesorteerd wordt. Bekijk voor inspiratie oefening 2.9.

```
# oefening 4.2

Getal=[7,5]

# vul hier je code aan

# Output
print(getal[0],"is kleiner dan",getal[1])
```

Oefening 4.3

Pas het script van oefening 4.2 aan zodat de gebruiker zelf de twee getallen kan ingeven.

Je mag wel slechts één input-opdracht gebruiken.

Oefening 4.4

Vervolledig onderstaand script zodat de tekst die je gaat ingeven toegevoegd wordt aan de lijst.

```
# initialisatie
ingave=" "
eten=["friet","frikandel","puree"]

# gebruikersinformatie
print("Vul je favoriet eten aan.")
print("Om te stoppen druk gewoon op enter (zonder een getal in te
geven)")
print()

# ingeven data
while ingave!="":
    ingave=input("Jouw favoriet eten: ")
    if ingave!="":
        # Vul in deze if-functie in wat je gaat toevoegen als je iets
hebt ingegegeven. (Als ingaven niet leeg is)
  #.....................

# output
print()
print("De volledige lijst is...")
print(eten)
```

Oefening 4.5

Schrijf een script dat 10 willekeurige natuurlijke getallen genereert en deze toont op het scherm.

Het script vraagt dan aan de gebruiker welk (het hoeveelste getal in de rij), het moet aanpassen gevolgd door het nieuwe getal.

Je toont telkens de nieuwe cijfers. (Zie voorbeeld)

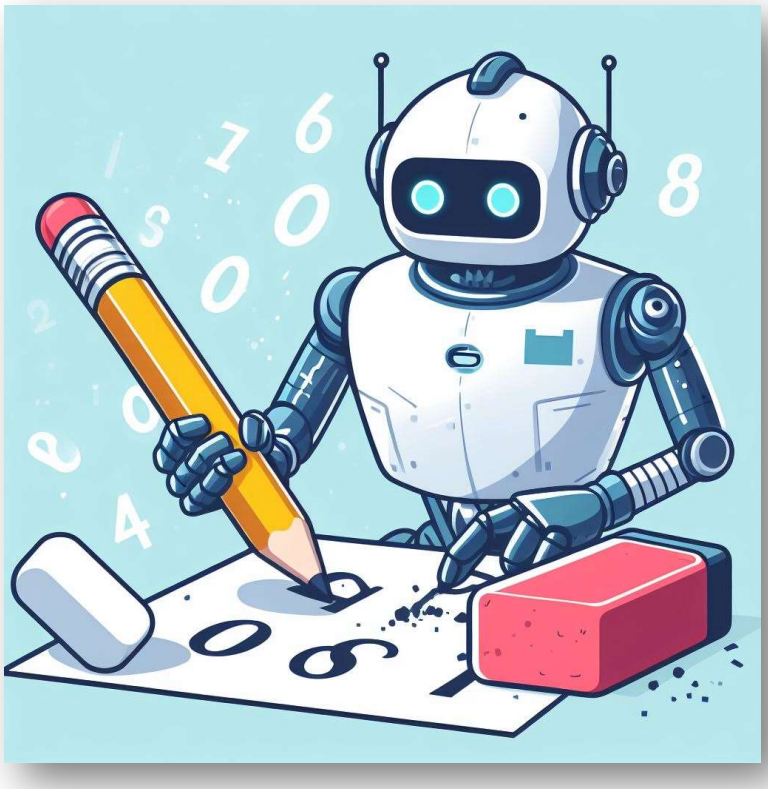
Herhaal dit totdat je 0 ingeeft bij de vraag welk getal je wenst aan te passen.

Onderstaande code genereert een getal tussen 1 en 10. Je kan deze code gebruiken om je probleem op te lossen.

```
import random
getal=random.randint(1,10)
```

Voorbeeld:

```
Shell
>>> %Run -c $EDITOR_CONTENT
  [5, 7, 9, 7, 9, 4, 5, 5, 6, 4]
  Het hoeveelste getal in de rij moet vervangen worden? 3
  Geef het nieuwe getal in: 10
  [5, 7, 10, 7, 9, 4, 5, 5, 6, 4]

  Het hoeveelste getal in de rij moet vervangen worden? 5
  Geef het nieuwe getal in: 12
  [5, 7, 10, 7, 12, 4, 5, 5, 6, 4]

  Het hoeveelste getal in de rij moet vervangen worden? 0
>>>
```



Verdieping

Oefening 4.6
Schrijf een script dat aan de gebruiker vraagt om een natuurlijk getal in te geven. Het script gaat dan zoveel willekeurige getallen tussen 1 en 10 genereren.

Als output geeft het script de gegenereerde getallen in oplopende volgorde.

Oefening 4.7
Schrijf een script dat aan de gebruiker vraagt om een natuurlijk getal in te geven. Het script gaat dan zoveel willekeurige getallen tussen 1 en 10 genereren.

Als output geeft het script de gegenereerde getallen en de mediaan.

Mediaan:

De mediaan is de middelste waarde van een groep getallen die gerangschikt wordt volgens grootte. Het is het getal dat exact in het midden ligt zodat 50% van de gerangschikte getallen boven 50% ligt en 50% onder de mediaan.

Voorbeeld

Om de mediaan te vinden van dezelfde 9 getallen: 10, 12, 11, 15, 13, 35, 41, 23, 20, plaats ze eerst in stijgende volgorde, d.w.z. 10, 11, 12, 13, 15, 20, 23, 35, 41 - het middelste getal is 15: de mediaan is 15, omdat 4 getallen onder 15 liggen en 4 getallen boven 15 liggen.

Als er een even aantal getallen is: 10, 11, 12, 13, 15, 20, 23, 35 - de twee in het midden (13 en 15) worden opgeteld (13+15=28) en dan gedeeld door 2 (28/2= 14), dat betekent dat de mediaan in dit geval 14 is.

Oefening 4.8:
Schrijf een Python script dat een lijst genereerd met de eerste 50 priemgetallen.

Een priemgetal is een getal dat enkel door 1 en zichzelf deelbaar is.

Je kan deze lijst genereren door te kijken of een getal een priemgetal is. Wanneer het hieraan voldoet voeg je dat toe aan de lijst, of je kan gebruik maken van de zeef van Euclides. Je start met alle getallen en gooit er d ie getallen uit die geen priemgetallen zijn.

Hieronder zie je een voorbeeld. Je verwijderd uit de lijst het getal 4.

```
getallen=[1,2,3,4,5]
getallen.remove(4)
```

Oefening 4.9
Een coopertest is een oefening waarbij de conditie van een deelnemer wordt gemeten. De bekendste vorm daarvan is die waarin een hardloper in 12 minuten een zo groot mogelijke afstand aflegt. Met deze afstand en de leeftijd kan in onderstaande tabel de conditie afgelezen worden.

Leeftijd		Zeer zwak	Zwak	Matig	Voldoende	Ruim voldoende	goed
20 - 29	M	< 1916	1916 - 2156	2157 - 2333	2334 - 2478	2479 - 2655	2656 - 2911
	V	< 1658	1658 - 1866	1867 - 2011	2012 -2140	2141 - 2333	2334 - 2589
30 - 39	M	< 1884	1884 - 2076	2077 - 2237	2238 - 2397	2398 - 2590	2591 - 2847
	V	< 1626	1626 - 1786	1787 - 1947	1948 - 2043	2044 - 2220	2221 - 2461
40 - 49	M	< 1771	1771 - 1979	1980 - 2140	2141 - 2285	2286 - 2478	2479 - 2750
	V	< 1546	1546 - 1689	1690 - 1818	1819 - 1947	1948 - 2124	2125 - 2332
50 - 59	M	< 1626	1626 - 1850	1851 - 2011	2012 - 2140	2141 - 2333	2334 - 2606
	V	< 1449	1449 - 1577	1578 - 1706	1707 - 1818	1819 - 1947	1948 - 2139
60 - 69	M	< 1433	1433 - 1689	1690 - 1850	1851 - 1995	1996 - 2204	2205 - 2525
	V	< 1385	1385 - 1512	1513 - 1593	1594 - 1722	1723 - 1899	1900 - 2071
70 - 79	M	< 1272	1272 - 1520	1521 - 1665	1666 - 1793	1794 - 1985	1986 - 2265
	V	< 1205	1205 - 1337	1338 - 1409	1410 - 1520	1521 - 1638	1639 - 1835
80 - 90	M	< 1000	1000 - 1232	1233 - 1348	1349 - 1450	1451 - 1602	1603 - 1828
	V	< 975	975 - 1082	1083 - 1184	1185 -1278	1279 - 1375	1376 - 1537

Ontwerp voor de mannen in leeftijdscategorie 20-29 een script dat je een antwoord geeft op de vraag hoe de conditie van een testpersoon is.

Denk hierbij goed na over welke lijsten je gaat gebruiken.

Oefening 4.10

Bij opdracht 3.10 heb je de Zeckendorf-voorstelling van een getal gegeven, waarbij je gebruik maakte van geneste lussen. Pas dit script aan zodat je gebruik kunt maken van een lijst van Fibonacci-getallen.

Oefening 4.11

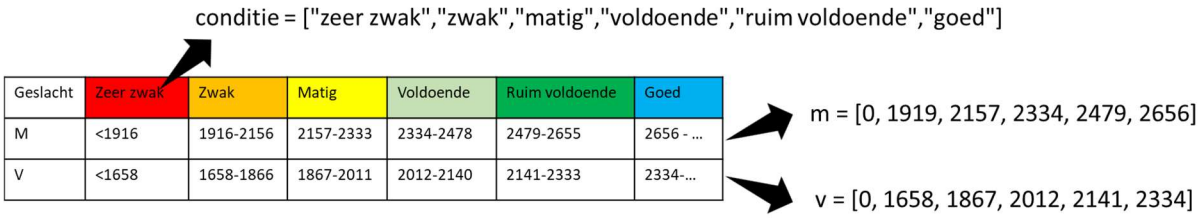
Ontwerp voor de mannen en vrouwen in leeftijdscategorie 20-29 een script dat je een antwoord geeft op de vraag hoe de conditie van een testpersoon is.

Denk hierbij goed na over welke lijsten je gaat gebruiken.

Gebruik de data van oefening 4.9.

Geneste lijsten

Bij oefening 4.11 heb je waarschijnlijk gebruik gemaakt van 3 lijsten:



Geslacht	Zeer zwak	Zwak	Matig	Voldoende	Ruim voldoende	Goed
M	<1916	1916-2156	2157-2333	2334-2478	2479-2655	2656 - ...
V	<1658	1658-1866	1867-2011	2012-2140	2141-2333	2334-...

- Een lijst die het conditiepeil weergeeft.
- Een lijst met de minimumafstand dat mannen moeten lopen.
- Een lijst met de minimumafstand dat vrouwen moeten lopen.

Het aantal lijsten dat je hier nodig hebt is nog beperkt.

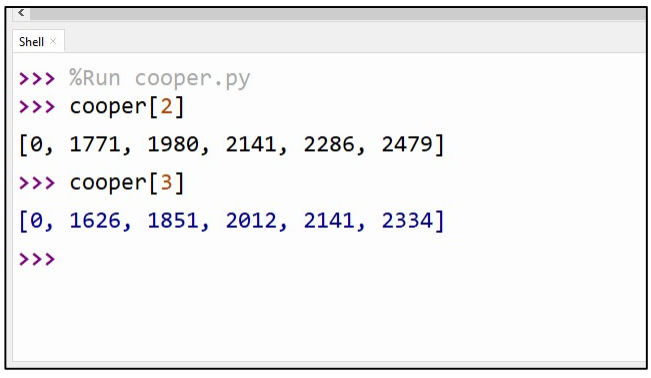
Moeten we echter een script schrijven om met de coopertest de conditie van mannen, tussen 20 en 90 jaar te bepalen dan zouden we krijgen we een veel verschillende lijsten.

```
conditie = ["zeer zwak","zwak","matig","voldoende","ruim
voldoende","goed"]
m20 = [0, 1919, 2157, 2334, 2479, 2656]
m30 = [0, 1884, 2077, 2238, 2398, 2591]
m40 = [0, 1884, 2077, 2238, 2398, 2591]
m50 = [0, 1626, 1851, 2012, 2141, 2334]
m60 = [0, 1433, 1690, 1851, 1996, 2205]
m70 = [0, 1272, 1521, 1666, 1794, 1986]
m80 = [0, 1000, 1233, 1349, 1451, 1603]
```

Het script wordt zo al snel complex. Een oplossing is om een lijst van lijsten te maken.

```
cooper = [m20, m30, m40, m50, m60, m70, m80]
```

Met een index kan je de verschillende lijsten selecteren. Zo geeft **cooper[2]** de lijst met index 2 (derde lijst, dus **m40)** en **cooper[3]** de lijst met index 3 (= **m50**).



Gebruik je twee indexen, dan kan je een individueel element selecteren. Zo zal **cooper[2][5]** het element zijn met index 5 (zesde element) uit de lijst met index 3, dus het zesde element uit **m40.**

En **cooper[3][1]** het tweede element (index = 1) uit **m50** (index = 3).

```
Shell
>>> %Run cooper.py
>>> cooper[2][5]
2479
>>> cooper[3][1]
1626
>>>
```

Hieronder wordt dit nog een grafisch voorgestel (zwart voor cooper[2][5], rood voor cooper[3][1]).

Index	0	1	2	3	4	5
0	0	1916	2157	2334	2479	2656
1	0	1884	2077	2238	2398	2591
2	0	1771	1980	2141	2286	2479
3	0	1626	1851	2012	2141	2334
4	0	1433	1690	1851	1996	2205
5	0	1272	1521	1666	1794	1986
6	0	1000	1233	1349	1451	1603

Je merkt dat je een geneste lijst (met 2 indexen) kan je voorstellen als een tabel, of omgekeerd… heb je een tabel met gegevens, dan kan je die in Python ingeven als een (tweedimensionale) geneste lijst, een lijst met 2 indexen.

Hieronder zie je een script waarmee je de conditie bepaald van een man in de leeftijdscategorie van 20 to 89 jaar met de coopertest

```
1  # Conditiebepaling met coopertest
2
3  # Lijsten aanmaken
4  conditie = ["zeer zwak","zwak","matig","voldoende","ruim voldoende","goed"]
5  m20 = [0, 1919, 2157, 2334, 2479, 2656]
6  m30 = [0, 1884, 2077, 2238, 2398, 2591]
7  m40 = [0, 1771, 1980, 2141, 2286, 2479]
8  m50 = [0, 1626, 1851, 2012, 2141, 2334]
9  m60 = [0, 1433, 1690, 1851, 1996, 2205]
10 m70 = [0, 1272, 1521, 1666, 1794, 1986]
11 m80 = [0, 1000, 1233, 1349, 1451, 1603]
12 cooper = [m20, m30, m40, m50, m60, m70, m80]
13
14 # Ingeven data
15
16 print("Conditie bepaling coopertest")
17 print()
18 leeftijd = int(input("Leeftijd: "))
19 afstand = int(input("Afstand: "))
20
21 # Lijst met leeftijd bepalen
22 leeftijd_index = int((leeftijd-20)/10)
23 tel = 5
24
25 while cooper[leeftijd_index][tel]>afstand:
26     tel=tel-1
27
28 print("De conditie van de persoon is",conditie[tel])
```

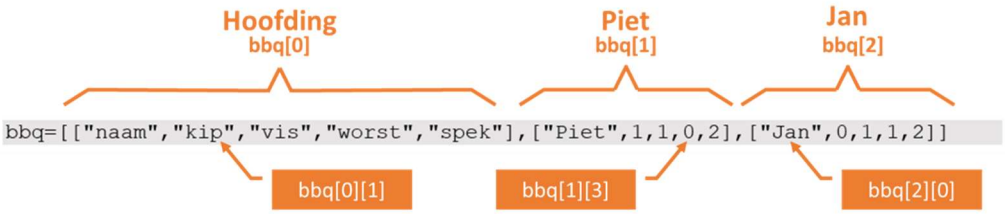
Toewijzen geneste lijsten

In het vorige voorbeeld hebben we eerst de verschillende lijsten apart aangemaakt. Dit is niet noodzakelijk. Je kan ook rechtstreeks een geneste lijst toewijzen aan een variabele.

Onderstaande tabel geeft de bestelling weer van 2 mensen voor een BBQ.

Naam	Kip	Vis	Worst	Spek
Piet	1	1	0	2
Jan	0	1	1	2

In het vorige voorbeeld hebben we gezien dat we deze kunnen voorstellen in Python als een geneste lus. We gaan deze keer niet eerst aparte lijsten opstellen voor elke persoon, maar geven deze rechtstreeks in als een geneste lus.



Gegevens toevoegen aan een geneste lijst

Wanneer je geen fout maakt tegen de syntax, dan gaat Python geen foutmelding geven.

Python is enorm flexibel met lijsten. Hoewel **bbq** een geneste lijst is, maakt Python geen probleem met onderstaande code:

```
>>> bbq
[['naam', 'kip', 'vis', 'worst', 'spek'], ['Piet', 1, 1, 0, 2], ['Jan', 0, 1, 1, 2]]
>>> bbq = bbq + [12]
>>> bbq
[['naam', 'kip', 'vis', 'worst', 'spek'], ['Piet', 1, 1, 0, 2], ['Jan', 0, 1, 1, 2], 12]
>>>
```

Geneste lijsten beheersbaar houden, vraagt dus van de programmeur wel discipline. Tracht de lengte van de (deel)lijsten gelijk te houden.

Opdracht 4.8

Met welke bewerking kan ik de **bbq** lijst uitbreiden met de gegevens van An (zie onderstaande tabel)

Naam	Kip	Vis	Worst	Spek
Piet	1	1	0	2
Jan	0	1	1	2
An	2	1	0	0

Mogelijkheid 1:

```
bbq = bbq + ["An", 2, 1, 0, 0]
```

Mogelijkheid 2:

```
bbq = bbq + [["An", 2, 1, 0, 0]]
```

Controleer je antwoord.

Inoefenen

Oefening 4.12: BBQ

Maak een script dat aan de verbruiker vraagt om volgende informatie:

- Naam
- Aantal keer kip, vis, worst en spek

Het ingeven stopt als er geen naam wordt ingegeven (en gewoon op ENTER gedrukt wordt).

Oefening 4.13: BBQ+

Breidt het script van oefening 4.12 uit zodat op het einde het totaal aantal keer kip, vis, worst en spek berekent wordt en verschijnt op het scherm.

Oefening 4.14: Mijnenjagen

Maak een script dat in een 6 x 6 grid willekeurig 4 vakjes aanduidt. De bedoeling van het spel is om deze mijnen onschadelijk te maken door ze te laten ontploffen.

Het script vraagt aan de gebruiker de coördinaten van een vakje. Afhankelijk van het resultaat geeft het volgend antwoord:

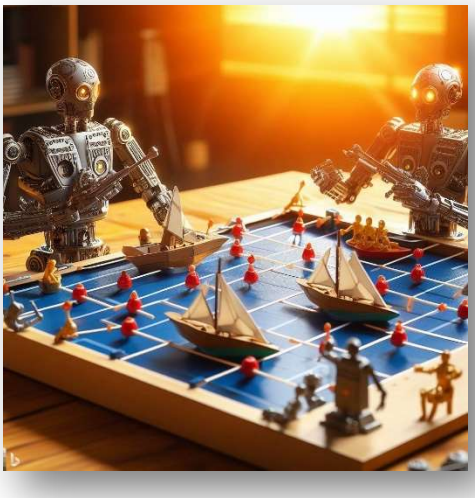
- “Boem! Mijn geraakt”
- “Plons!”
- “Reeds gecontroleerd”

Het script geeft ook het aantal pogingen weer en stopt als al de mijnen ontploft zijn.

```
Shell
>>> %Run -c $EDITOR_CONTENT

  Poging: 1
     Geef de x-coördinaat: 1
     Geef de y-coördinaat: 2
  Boem! Mijn geraakt!

  Poging: 2
     Geef de x-coördinaat: 1
     Geef de y-coördinaat: 2
  Reeds gecontroleerd.

  Poging: 3
     Geef de x-coördinaat: 3
     Geef de y-coördinaat: 3
  Plons!

  Poging: 4
     Geef de x-coördinaat:
```

Oefening 4.15: Mijnenjagen+

Breidt het script van oefening 4.14 uit zodat twee vakjes die elkaar raken (zijde of hoek) allebei geen mijn kunnen bevatten (slechts één van de twee kan een mijn bevatten)



Module 5: String

We hebben reeds gezien dat tekst in een variabel type string gestoken wordt. We weten reeds dat we tekst kunnen toewijzen aan een variabele door hem tussen aanhalingstekens te plaatsen.

String letterlijk vertaald is snaar. Je kan een string aanzien als allemaal karakters die aan een draad gerijgd

Je krijgt zo een geordende verzameling van karakters waarbij de volgorde van de karakters niet kan veranderen.

Je merkt in deze omschrijving dan ook dat er overeenkomsten zijn met lijsten. Ook lijsten zijn geordende verzamelingen, maar niet van tekens, maar van elementen. In deze module gaan we dan ook op verkenning naar overeenkomsten tussen beide structuren. We gaan daarom steeds een bewerking/methode/instructie toepassen op een lijst, en dan kijken of deze bewerking/methode/instructie ook van toepassing is op een string.

Verkenning

Gebruik van Index

Opdracht 5.1

Bij lijsten kunnen we elementen selecteren m.b.v. de index.

```
#opdracht 5.1 (a)
weekdagen=["Ma","Di","Wo","Do","Vr","Za","Zo"]
print(weekdagen[4])
```

Wat verwacht je dat de bovenstaande code als resultaat heeft?

Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord “Foutmelding”.

Opdracht 5.2

Je kan in een lijst met de index van een element individuele elementen aanpassen.

```
Shell
>>> %Run cooper.py
>>> dag_kort[1]="DI"
>>> dag_kort
['ma', 'DI', 'wo', 'do', 'vr', 'za', 'zo']
>>>
```

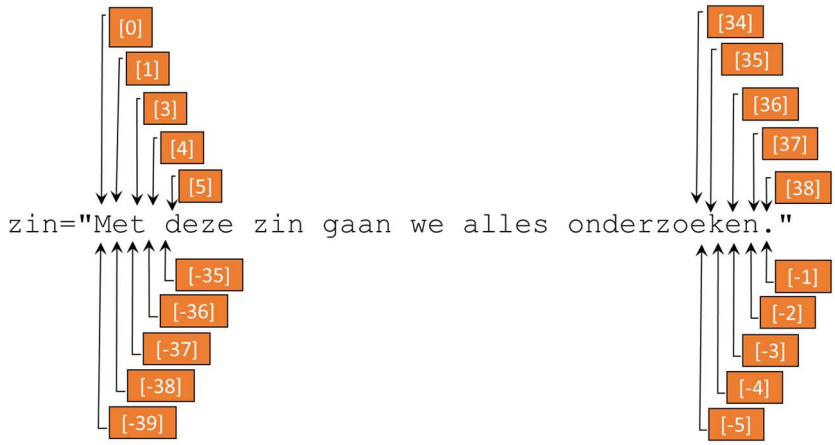
Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord “Foutmelding”.

```
# opdracht 5.1b
zin="Met deze zin gaan we op onderzoek."
print(zin[4])
print(zin[5:10])
```

Verklaring:

Index

Net zoals bij lijsten, kan je in een string de verschillende karakters benaderen met de index. Je kan dit doen vanaf het begin van de string, maar ook vanaf het einde.



Slicing

Je kan met indexen ook een stukje uit de string afzonderen. Dit heet slicing (wegsnijden).

Je geeft telkens aan voor welk teken je snijdt.


```
Shell
>>> %Run cooper.py
>>> zin[5:10]
'eze z'
>>>
```

Bewerkingen: + en *

Opdracht 5.3

Je kan met het '+'-teken lijsten samenvoegen.

```
>>> lijst1
['a', 'e', 'i']
>>> lijst2
['o', 'u']
>>> klinkers = lijst1+lijst2
>>> klinkers
['a', 'e', 'i', 'o', 'u']
>>>
```

Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord "Foutmelding".

```
#Opdracht 5.3
tekst1 = "De klinkers zijn: "
tekst2 = "a, e, i, o, u. "
zin = tekst1 + tekst2
print(zin)
```

Opdracht 5.4

Je kan met het vermenigvuldigingsteken "*" een lijst vullen met allemaal dezelfde elementen, bijvoorbeeld:

```
Shell
>>> %Run -c $EDITOR_CONTENT
>>> kruisjes = ["x"]*5
>>> kruisjes
['x', 'x', 'x', 'x', 'x']
>>>
```

Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord "Foutmelding".

```
tekenreeks = "x"*5
print(tekenreeks)
```

Verklaring

Wat je hier onderzocht hebt wordt in het Engels “concatenate” genoemd, in het Nederlands is dit “aaneenschakeling”. Je kan met het “+”-teken twee strings aan elkaar schakelen…

Met de code

```
>>>tekst1 + tekst2
```

schakel je dus de twee strings aan elkaar, eerst tekst1, gevolgde door tekst2 en krijg je:
“**De klinkers zijn:** “+”**a, e, i, o, u**” → “**De klinkers zijn: a, e, i, o, u**”

Omdat een vermenigvuldiging niets minder is als een optelling die een aantal keren wordt uitgevoerd krijg je:

“x”*5 → “x”+”x”+”x”+”x”+”x” →”xxxxx”

De methoden « in » en « len »

Opdracht 5.5

Met de instructie “in” kunnen we controleren of een lijst een bepaald element bevat.

Bekijk onderstaande code

```
#opdracht 5.5
weekdagen=["Ma","Di","Wo","Do","Vr","Za","Zo"]
print("Ma" in weekdagen)
print("Maandag" in weekdagen)
```

of uitgevoerd in de Shell

```
>>> weekdagen=["Ma","Di","Wo","Do","Vr","Za","Zo"]
>>> "Ma" in weekdagen
True
>>> "Maandag" in weekdagen
False
>>>
```

Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord “Foutmelding”.

```
weekdagen="De weekdagen zijn: ma, di, wo, do, vr."
weekend="Het weekend is: za, zo."
print("ma" in weekdagen)
print("za" in weekdagen)
dagen =  weekdagen + weekend
print("Het" in dagen)
```

Opdracht 5.5

Met de functie “len” kunnen we de lengte van een lijst bepalen.

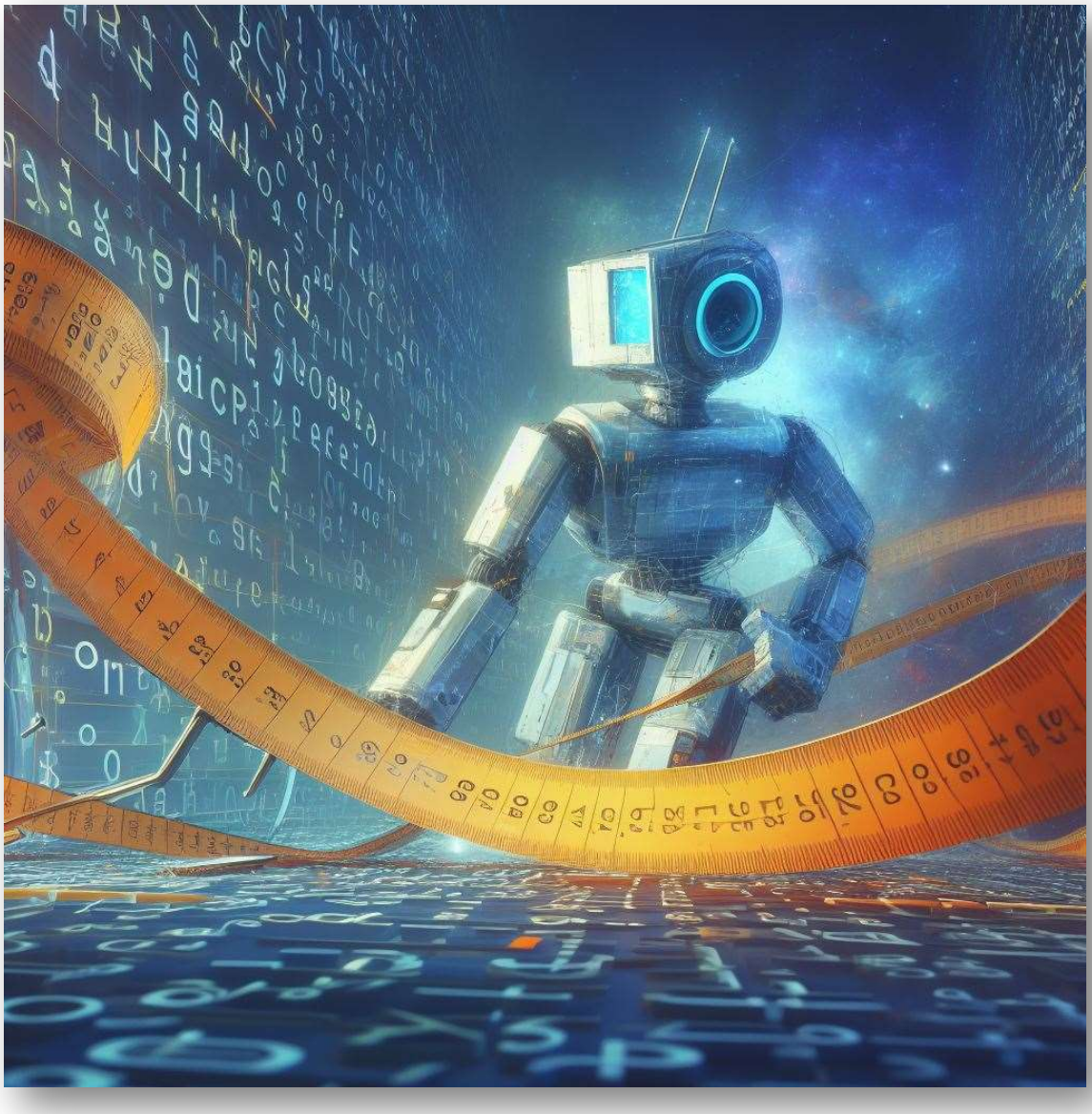
```
#opdracht 5.6
weekdagen=["Ma","Di","Wo","Do","Vr","Za","Zo"]
print(len(weekdagen))
```

of in de Shell

```
>>> weekdagen=["Ma","Di","Wo","Do","Vr","Za","Zo"]
>>> len(weekdagen)
7
>>>
```

Wat verwacht je dat onderstaande instructie als resultaat gaat geven? Als je denkt dat deze instructie niet kan gebruikt worden bij strings, dan is je antwoord “Foutmelding”.

```
naam="python"
print(len(naam))
```



Verankeren

Na de verkenning gaan we in dit hoofdstuk de opgedane kennis verankeren. We gaan verder kijken hoe we kunnen werken met de indexen, hoe we de instructie '**in'** en de functie '**len**' kunnen gebruiken.

Tenslotte gaan we veder kijken naar de methoden, upper() en lower() .

Opdracht 5.6

Onderzoek zelf wat zal gebeuren met onderstaande instructies

```
• zin[:10]
• zin[10:]
• zin[2:12:3]
• zin[::]
• zin[12:3:-2]
• zin[::-1]
```

Opdracht 5.7

Schrijf een script dat aan de gebruiker vraagt om tekst in te geven. Het script print de verschillende karakters van de ingegeven tekst onder elkaar.

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef tekst in: Zaterdag
  Z
  a
  t
  e
  r
  d
  a
  g
>>>
```

Opdracht 5.8

Wat verwacht je dat onderstaand script gaat doen?

```
tekst = input('Geef tekst in: ')
i = 1
while i<len(tekst)+1:
    print(tekst[-i])
    i=i+1
```

Opdracht 5.9

Maak een script dat aan de gebruiker vraagt een tekst in te geven, en daarna een letter. Het script geeft dan een antwoord op de vraag hoeveel keer deze letter in de tekst voorkomt.

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef tekst in: Op zondag gaan katholieken naar de mis.
  Geef een letter in: a
  De letter a komt 6 keer voor in de tekst.
>>>
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef tekst in: Op zondag gaan katholieken naar de mis
  Geef een letter in: s
  De letter s komt 1 keer voor in de tekst.
>>>
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef tekst in: Op zondag gaan katholieken naar de mis
  Geef een letter in: q
  De letter q komt 0 keer voor in de tekst.
>>>
```

Opdracht 5.10

Schrijf een script dat aan de gebruiker vraagt om een woord of zin in te geven. Het script beantwoord dan de vraag of het een palindroom of niet is.

Palindroom = woord dat omgekeerd hetzelfde woord oplevert, bijvoorbeeld LEPEL

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef woord in: raam
  Het woord raam is GEEN palindroom
>>>
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef woord in: droommoord
  Het woord droommoord is een palindroom
>>>
```

"Basisbewerkingen"

We hebben in de verkenning gezien dat we strings kunnen samenvoegen (concatenate) met het '+'-teken.

```
Shell
>>> %Run cooper.py
>>> lijst1 = ["a","e","i"]
>>> lijst2 = ["o","u"]
>>> klinkers = lijst1 + lijst2
>>> klinkers
['a', 'e', 'i', 'o', 'u']
```

Het is logisch dat je strings niet van elkaar kunt aftrekken of delen door elkaar, maar omdat een vermenigvuldiging eigenlijk een optelling is die een aantal keren wordt uitgevoerd, is het wel mogelijk om een string te vermenigvuldigen.

Zo zal:

```
>>>"A"*10
```

overeenkomen met:

```
>>>"A"+"A"+"A"+"A"+"A"+"A"+"A"+"A"+"A"+"A"
```

```
Shell
Python 3.10.11 (C:\Users\LUCP11650\Pictures\Python\thonny-
le\python.exe)
>>> "A"*10
'AAAAAAAAAA'
>>>
```

Opdracht 5.9

Schrijf een script dat een rechthoekige driehoek tekent. De gebruiker geeft de grootte van de rechthoekszijde in.

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef de grootte van de rechthoekszijde: 5
  x
  xx
  xxx
  xxxx
  xxxxx
>>>
```

Opdracht 5.10

Schrijf een script dat een kerstboom tekent. De gebruiker geeft de hoogte van de kerstboom in.

```
Shell
>>> %Run -c $EDITOR_CONTENT
  Geef de grootte van de rechthoekszijde: 5
      x
     xxx
    xxxxx
   xxxxxxx
  xxxxxxxxx
>>>
```

Tip: stel voor elke lijn een string samen bestaande uit een aantal spaties en de kruisjes.

String-methoden

Normaal gezien gaan we niet in de toeters en bellen van Python, maar in dit geval maken we een uitzondering omdat dit sommige scripts echt wel kunnen vereenvoudigen.

upper( ) en lower( )

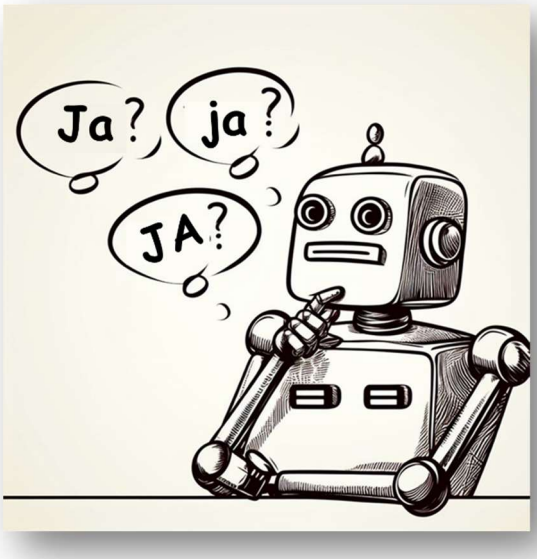
De string methoden upper( ) en lower( ) zetten de string respectievelijk om in hoofd of kleine letters.

```
Shell
Python 3.10.11 (C:\Users\LUCP11650\Pictures\Python\thonny-
>>> tekst = "Humpty Dumpty zat op de muur"
>>> tekst.upper()
'HUMPTY DUMPTY ZAT OP DE MUUR'
>>> tekst.lower()
'humpty dumpty zat op de muur'
>>>
```

Opdracht 5.11

Maak een script dat aan de gebruiker vraag om “ja” of “nee te antwoorden. Je controleert of de invoer één van beide mogelijkheden is, maar je aanvaard alle mogelijke schrijfwijzen (Ja, JA, ja en Nee, NEE, nee). Maak gebruik van upper() of lower().

Indien geen van beide toegestane antwoorden gegeven wordt, dan moet je een nieuwe ingave vragen totdat één van de toegestane antwoorden ingegeven worden.



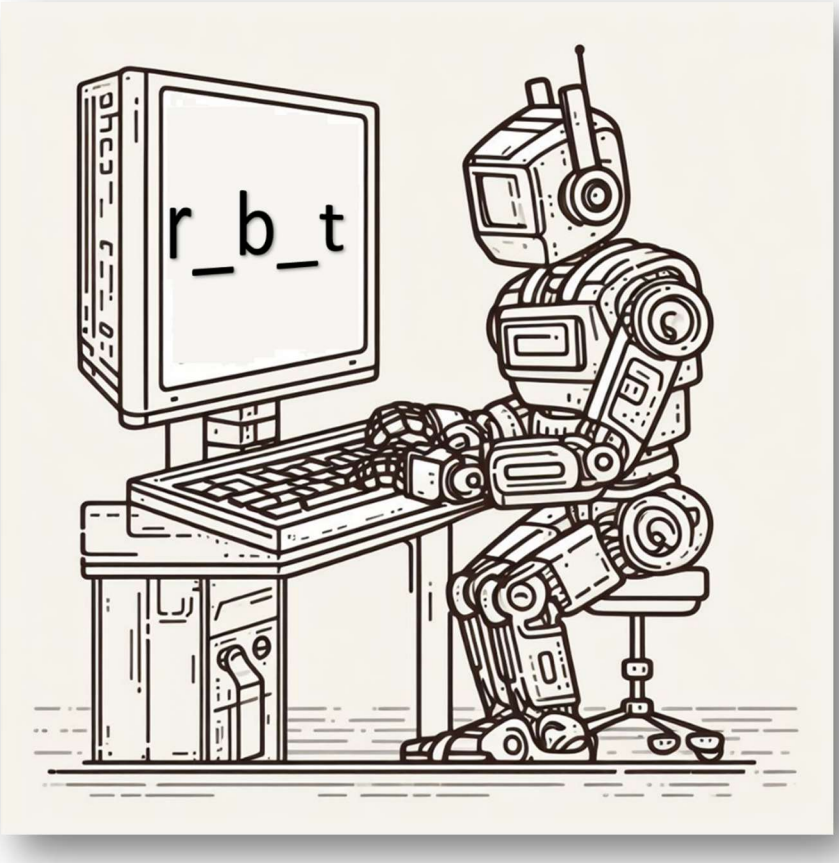
Verdieping

Oefening 5.1

Schrijf een Python-script voor een eenvoudig woordraadspel.

Gebruik onderstaande stappen als leidraad:

1. Het script kiest uit een lijst met geheime woorden willekeurig een woord
2. Initialiseer een variabele om het aantal levens van de speler bij te houden. Begin met 6 levens.
3. Geef aan dat er een woord moet worden geraden.
4. Toon de speler hoeveel levens hij nog heeft.
5. Laat de speler telkens één letter raden. Vraag om invoer van de speler en controleer of de invoer een enkele letter is.
6. Controleer of de geraden letter voorkomt in het geheime woord. Als dat zo is, toon het geheime woord met de geraden letters op de juiste plaatsen.
7. Als de geraden letter niet in het geheime woord voorkomt, verminder het aantal levens van de speler en toon hoeveel levens ze over hebben.
8. Herhaal stap 4 tot en met 7 totdat de speler het geheime woord heeft geraden of geen levens meer over heeft.
9. Toon een overwinningsbericht als de speler het woord heeft geraden en een verliesbericht als de levens van de speler op zijn.
10. Vraag de speler of ze opnieuw willen spelen.



Module 6: Externe bestanden

Een script bestaat in essentie uit steeds vier dezelfde stappen:

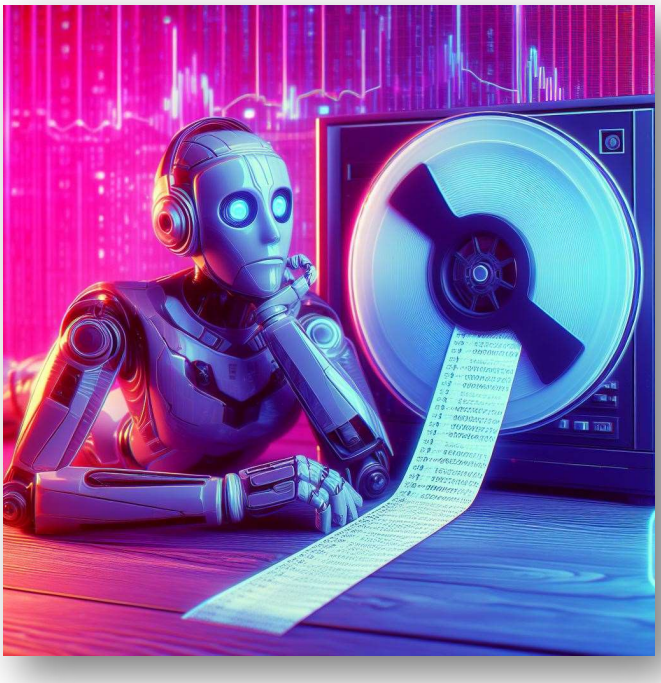
1. data invoeren
2. data bewaren
3. data bewerken
4. data uitvoeren

Jullie zijn reeds in staat om een script te schrijven dat deze vier stappen doorloopt. De data bewaren jullie tot nu toe in het geheugen van de computer, maar telkens als je het script opnieuw start, zal al de data die door de gebruiker is ingevoerd verloren gaan.

Willen we dit vermijden dan gaan we de data moeten bewaren in een extern bestand.

Er bestaan veel soorten bestandsformaten, maar wij gaan ons in deze module beperken tot tekst bestanden. Bestanden die we ook kunnen bewerken met een simpele tekstverwerker zoals **Notepad** of zelfs een simpele editor.

We gaan eerst nadenken over hoe we onze data in een tekstbestand gaan voorstellen. Nadat je zo een tekstbestand hebt aangemaakt bespreken we de methoden om in Python tekstbestanden te lezen en te schrijven.



Verkenning

In oefening 4.6 hebben we een boodschappenlijstje gemaakt, en het lijstje bewaard in een lijst. We gaan onderzoeken hoe we dit lijstje in een bestand kunnen voorstellen.

Opdracht 6.1

Neem een blad papier en maak een boodschappenlijstje met een vijftal producten.

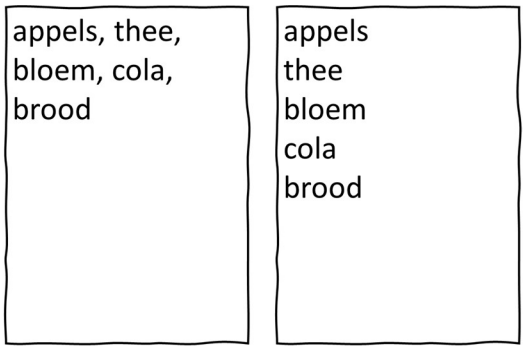
- Welke informatie heb je nodig?
- Hoe heb je deze informatie voorgesteld?

Opdracht 6.2

Neem een eenvoudig tekstverwerkingsprogramma of maak gebruik van je editor om een tekstbestand aan te maken. Bewaar het tekstbestand als **boodschappen.txt**.

Tekstbestand lezen

Je zou een lijstje kunnen maken met al je boodschappen achter elkaar, maar ik vermoed dat de meeste wel een lijstje in de tweede vorm gemaakt hebben met alle boodschappen onder elkaar



Je hebt dan telkens een item ingetypt en op ENTER gedrukt om een nieuwe lijn te starten.

Telkens je op ENTER drukt wordt er een niet-afdrukbaar teken geplaatst, het “end-of-line” karakter. Dit teken sluit een regel af. We kunnen hiervan gebruik maken om het boodschappenlijstje regel per regel te lezen, en zo opnieuw een lijst te vullen.

We starten men een lege lijst met de naam “boodschappen” te maken. Deze lijst zal worden gebruikt om de regels uit het bestand op te slaan.

```
boodschappen=[]
```

We openen het bestand "boodschappen.txt", en een bestandsobject fp wordt aangemaakt om toegang te krijgen tot de inhoud van het bestand. We geven met de parameter “r” aan dat we het bestand gaan lezen (read).

```
fp = open( "boodschappen.txt","r" )
```

We gaan de eerste lijn van het bestand lezen en deze opslaan in een variabele buffer.

```
buffer = fp.readline()
```

Nu beginnen we een while-lus. Zolang de variabele buffer niet leeg is (wat betekent dat er nog regels in het bestand zijn om te lezen), zal de code van de while-lus uitgevoerd worden.

```
while buffer!="":
```

Binnen de lus wordt de huidige regel toegevoegd aan de lijst boodschappen door deze te concatenen met de bestaande lijst. Dit betekent dat elke regel wordt toegevoegd aan de lijst als een nieuw element.

```
    boodschappen=boodschappen+[buffer]
```

Vervolgens moeten we de volgende regel uit het bestand lezen en opslaan in de variabele buffer.

```
    buffer = fp.readline()
```

Deze stappen van de lus blijven herhaald worden tot er geen regels meer in het bestand staan om te lezen.

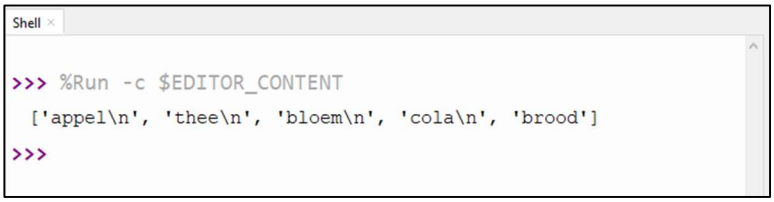
Nadat alle regels in het bestand zijn gelezen en toegevoegd aan de lijst boodschappen, moeten we het bestand sluiten met fp.close(). Dit zorgt ervoor dat het bestand netjes wordt afgesloten en geen verdere toegang meer heeft.

```
fp.close()
```

Tenslotten gaan we onze lijst afdrukken om het resultaat te kunnen beoordelen.

```
print(boodschappen)
```

Wanneer we deze code gaan uitvoeren dan krijgen we onderstaan resultaat.



Je merkt dat bijna alle items in de lijst eindigen met **\n**. Dit is de voorstelling van het **end-of-line** karakter.

Het kan zijn dat bij je laatste item uit de lijst je dit niet ziet, dan heb je nadat je het laatste item hebt toegevoegd aan je lijst niet op ENTER gedrukt.

In onze lijst willen we dit teken niet, daarom gaan we dit teken er af slicen.

Hoewel wij dit zien als 2 tekens, gaat het over 1 teken, het end-of-line karakter. De slicing zal dus als volgt uitgevoerd worden:

```
buffer=buffer[:-1]
```

Omdat niet alle elementen in de lijst dit teken bevat, moeten controleren of dit teken aanwezig is. De conditie hiervoor is:

```
if '\n' in buffer:
```

Onze volledige code gaat er dan als volgt uitzien:

```
fp = open( "boodschappen.txt","r" )
buffer = fp.readline()
while buffer!="":
    buffer=str(buffer)
    if '\n' in buffer:
        buffer=buffer[:-1]
    boodschappen=boodschappen+[buffer]
    buffer = fp.readline()

fp.close()
print(boodschappen)
```

Opdracht 6.3

Pas je bestand boodschappen.txt en de code aan zodat niet enkel ingelezen wordt wat je moet kopen, maar ook hoeveel.

Opdracht 6.4

Pas het woordraapspel (oefening 5.1) zo aan dat het script de geheime woorden inleest van een tekstbestand.

Tekstbestand “schrijven” of maken

We gaan een script schrijven dat de eerste 100 getallen van Fibonacci in een tekstbestand gaat opslaan.

We hebben dus eerst de code nodig om de getallen te bepalen. Omdat we het schrijven van het bestand willen ontkoppelen van het opstellen van de Fibonaccigetallen, gaan we de getallen van Fibonacci eerst opslaan in een lijst.

```
fibo=[0,1]
teller = 2
while teller <100:
    fibo=fibo+[fibo[teller-1]+fibo[teller-2]]
    teller=teller+1
```

Tenslotte gaan we onze lijst naar een tekstbestand schrijven.

We openen een nieuw tekstbestand met de naam "fibonacci.txt" voor schrijven ("w"). Een bestandsobject fp wordt aangemaakt om toegang te krijgen tot het bestand.

```
fp = open( "fibonacci.txt", "w" )
```

We maken een while-lus die de 100 getallen gaat schrijven. We moeten de getallen wel omzetten naar een string, omdat we een tekst-bestand maken, en we gaan het end-of-line teken toevoegen om na het schrijven van getal een nieuwe lijn te starten.

```
str(fibo[teller]) +"\n"
```

Je kan deze string naar het bestand schrijven met:

```
    fp.write( str(fibo[teller]) +"\n")
```

Als we eenmaal de 100 getallen hebben weggeschreven moeten we het bestand sluiten.

```
fp.close()
```

Het hele script zal er als volgt uitzien:

```
fibo=[0,1]
teller = 2
while teller <100:
    fibo=fibo+[fibo[teller-1]+fibo[teller-2]]
    teller=teller+1

fp = open( "fibonacci.txt", "w" )

teller = 0
while teller<100:
    fp.write( str(fibo[teller]) +"\n")
    teller=teller+1

fp.close()
```

Opdracht 6.4

Pas de code van opdracht 6.3 aan zodat je, nadat je het bestand gelezen hebt, je nieuwe boodschappen kunt ingeven.

Voordat je het script afsluit, schrijf je de nieuwe lijst weg naar het tekstbestand.

Challenge: Fibonacci Nim

Fibonacci nim is een wiskundig aftrekkingsspel, een variant van het spel nim. Spelers nemen afwisselend munten van een stapel, waarbij ze bij elke zet maximaal twee keer zoveel munten nemen als de vorige zet, De winnaar is de persoon die de laatste munt neemt. De Fibonacci-cijfers spelen een belangrijke rol in de analyse; in het bijzonder kan de eerste speler winnen als en slechts als het startaantal munten geen Fibonacci-getal is. Een complete strategie is bekend om het ideale spel te spelen met een enkele stapel fiches, maar niet voor varianten van het spel met meerdere stapels.

Regels en geschiedenis

Fibonacci nim wordt gespeeld door twee spelers, die afwisselend munten of andere fiches van een stapel verwijderen. Bij de eerste zet mag een speler niet alle munten pakken, en bij elke volgende zet kan het aantal verwijderde munten elk getal zijn dat maximaal tweemaal zo groot is als de vorige zet. Volgens de normale spelconventie wint de speler die de laatste munt pakt.

Het spel werd voor het eerst beschreven door Michael J. Whinihan in 1963, waarbij de uitvinding werd toegeschreven aan de wiskundige Robert E. Gaskell van de Oregon State University. Het wordt Fibonacci nim genoemd omdat de Fibonacci-getallen een belangrijke rol spelen in de analyse.

Dit spel moet worden onderscheiden van een ander spel, ook wel Fibonacci nim genoemd, waarin spelers bij elke zet een willekeurig Fibonacci-aantal munten kunnen verwijderen.

Strategie

De strategie voor het ideale spel in Fibonacci nim houdt in dat je het huidige aantal munten beschouwt als een som van Fibonacci-getallen. Er zijn veel manieren om getallen weer te geven als sommen van Fibonacci-getallen, maar er is slechts één representatie die elk Fibonacci-getal maximaal één keer gebruikt en opeenvolgende paren van Fibonacci-getallen vermijdt; deze unieke voorstelling staat bekend als de Zeckendorf-voorstelling. De Zeckendorf-weergave van 10 is bijvoorbeeld 8 + 2; Hoewel 10 ook op andere manieren kan worden weergegeven als de som van Fibonacci-getallen, zoals 5 + 5 of 5 + 3 + 2, voldoen die andere manieren niet aan de voorwaarde om elk Fibonacci-getal slechts één keer te gebruiken en opeenvolgende paren van Fibonacci-getallen te vermijden, zoals als de paren 2, 3 en 3, 5. De Zeckendorf-representatie van elk getal kan worden gevonden door een greedy algoritme dat herhaaldelijk het grootst mogelijke Fibonacci-getal aftrekt, totdat het nul bereikt.

De spelstrategie omvat ook een getal dat de "quota" wordt genoemd en dat kan worden aangeduid als q. Dit is het maximale aantal munten dat momenteel kan worden verwijderd. Bij de eerste zet kunnen op één na alle munten worden verwijderd, dus als het aantal munten n is, is het quotum $q = n - 1$. Bij volgende zetten is het quotum twee keer het vorige zet.

Op basis van deze definities kan de speler die op het punt staat te bewegen winnen wanneer q groter is dan of gelijk is aan het kleinste Fibonacci-getal in de Zeckendorf-weergave, en anders zal hij verliezen (met het ideale spel van de tegenstander). In een winnende positie is het altijd een winnende zet om alle munten te verwijderen (als dit is toegestaan) of om op een andere manier een aantal munten te verwijderen dat gelijk is aan het kleinste Fibonacci-getal in de Zeckendorf-weergave. Wanneer dit mogelijk is, zal de tegenstander noodzakelijkerwijs met een verliezende positie worden geconfronteerd, omdat het nieuwe quotum kleiner zal zijn dan het kleinste Fibonacci-getal in de Zeckendorf-weergave van het resterende aantal munten. Andere winnende zetten zijn mogelijk ook mogelijk. Vanuit een verliezende positie zullen alle zetten echter leiden tot winnende posities.

De Zeckendorf-weergave van een Fibonacci-getal bestaat uit dat ene getal. Dus als de startstapel een Fibonacci-getal n munten heeft, is het kleinste Fibonacci-getal in de Zeckendorf-weergave slechts n, groter dan het startquotum n − 1. Daarom is een Fibonacci-getal als startstapel verliezend voor de eerste speler en winnen voor de tweede speler. Elk niet-Fibonacci-startaantal munten heeft echter kleinere Fibonacci-getallen in de Zeckendorf-weergave. Deze getallen zijn niet groter dan het startquotum, dus als het startaantal munten geen Fibonacci-nummer is, kan de eerste speler altijd winnen.