Programmeren in Python

Wat is een programmeertaal?

- Je kunt je voorstellen dat in een computer een klein robotje zit. Laten we hem Robert noemen.
- Robert doet precies wat hem gevraagd wordt.
- MAAR! Je moet het wel precies goed vragen. Een letter verkeerd, en hij begrijpt het niet. Of erger, hij begrijpt het verkeerd. Dan doet hij wel wat je vraagt, maar niet wat je bedoelt.

Uitproberen!

- In Spyder kan je rechtsonder je commando's intypen.
- Robert kan heel goed rekenen: kijk maar eens of hij weet wat 12 keer 15 is. Voor keer gebruik je een * en voor delen een /. Reken de som ook zelf (uit je hoofd) uit om te controleren of het klopt.
- Komma-getallen schrijft Robert in Engelse notatie, dus met een punt in plaats van een komma. Dat moet jij dus ook doen, anders begrijpt hij je niet.
- In welke volgorde doet hij de berekening als je geen haakjes gebruikt? Probeer het uit!

Planning

- Inleiding
- → Types en variabelen
- Flow control
- Strings
- Functies
- Bestanden
- Modules: grafieken

Types: getallen

- De sommetjes die je nu gedaan hebt, gebruikten getallen. Die getallen heten objecten in Python. Elk object heeft een type. We gaan nu een paar types bekijken.
- Om te beginnen een geheel getal. Dat heet in Python een int. Dat staat voor integer en dat is Engels (en ook Nederlands, maar dat weten veel mensen niet) voor geheel.
- Een andere getalsoort, die je hebt gezien als je een deling hebt gedaan, is het kommagetal. Die heet een float. Het is belangrijk om te weten dat de berekeningen daarmee afrondfoutjes kunnen krijgen. Bereken bijvoorbeeld maar eens 1 / 49 * 49.

Meer types: tekst

- Maar Python is niet alleen een rekenmachine. Je kan bijvoorbeeld ook met tekst werken. Dat noemt Python een string.
- Een string zet je tussen enkele of dubbele aanhalingstekens, dat mag je zelf weten.
- Let op! Een string met cijfers erin is geen getal. “123” is een tekst die betaat uit drie tekens. 123 is een getal. Python vindt dat iets heel anders.
- Je kan ook “rekenen” met strings: je kan ze bij elkaar optellen, of vermenigvuldigen met een int. Probeer maar!

Nog een type: de lijst

- Bij het programmeren wil je vaak werken met lijsten. Je wilt bijvoorbeeld niet de leeftijd van een persoon gebruiken, maar een lijst met leeftijden van heel veel personen. Daarvoor heeft Python een speciaal type. Dat is de list.
- Een list schrijf je met blokhaken. Daartussen staan alle objecten die in de list zitten. Dit is bijvoorbeeld een lijst met twee getallen en daartussen een tekst: [15, “hekje”, 3.7].
- Met een list kan je heel veel doen, daar gaan we het straks over hebben.

Een programma

- De kracht van een computer zit in de mogelijkheid om een programma te schrijven: een lijst opdrachten die achter elkaar uitgevoerd moeten worden.
- In Spyder doe je dat in het linkerdeel van het scherm.
- In de “console” rechtsonder laat hij het antwoord van elk commando zien. Dat doet hij bij een programma niet. Daarvoor moet je het print-commando gebruiken. Bijvoorbeeld print(1+1)
- Schrijf maar eens een programma wat eerst 1+1 uitrekent en daarna 1/2. Voer het uit met de groene play-knop bovenaan of F5.
- De eerste keer moet je even Ok klikken voor de instellingen.

Variabelen

- Een programma moet leesbaar zijn. Daarvoor is het nuttig om een ingewikkelde berekening in stapjes te doen. Of soms heb je een resultaat meer dan een keer nodig. Dan wil je dat kunnen onthouden.
- Daarvoor kan je variabelen gebruiken. Als je de naam van de variabele en een is-teken voor je berekening zet, onthoudt hij het antwoord. Bijvoorbeeld: hoogte = 4 * 18 - 5
- In plaats van de uitkomst kan je dan die naam gebruiken.
- Handig: de waarde van een variabele kan je veranderen!
- Probeer dat maar eens uit.

Types omzetten

- Je kan types in elkaar omzetten door de naam van het nieuwe type te gebruiken en daarachter tussen haakjes de waarde van een ander type. Bijvoorbeeld int(“123”) of str(123).
- Je kunt alles omzetten in een string, aan andere omzettingen zitten vaak wel voorwaarden. Van “hallo” kan Robert bijvoorbeeld geen int maken. Probeer maar.
- Kan je het getal 1234 maken met een programma waar geen ander cijfer dan 1 in voorkomt? Probeer ook het cijfer 1 zo weinig mogelijk te gebruiken. Hoeveel keer heb je het nodig?

Commentaar

- Voor de leesbaarheid is het ook belangrijk om op veel plekken op te schrijven wat je aan het doen bent. Met die uitleg hoeft Robert niks te doen. Dat heet commentaar.
- Commentaar begint met een # en loopt tot het einde van de regel.
- Bijvoorbeeld hoogte = 140 / (4 * 12) # volume / basis
- Bij een lange regel kan je het commentaar beter op de regel(s) ervoor zetten.
- Voeg commentaar toe aan het programma dat je net geschreven hebt.

Planning

- Inleiding
- Types en variabelen
- → Flow control
- Strings
- Functies
- Bestanden
- Modules: grafieken

Flow control: if

- Normaal worden de commando’s in een programma achter elkaar uitgevoerd. Dat heet de flow van het programma.
- Soms is het nuttig om stukken over te slaan. Dan ben je dus de flow aan het aanpassen. In het Engels heet dat “flow control”.
- De manier om dat te doen is met “if”. Achter if volgt een vergelijking (of ongelijkheid) en dan een dubbele punt. Als het waar is, wordt het volgende stuk uitgevoerd, anders niet.
- “Het volgende stuk” is ingesprongen. Zo is het duidelijk te herkennen en weten jij en Robert ook wanneer het is afgelopen.
- De operaties schrijf je als ==, !=, >, <, >= en <=.

Voorbeeld van if

```
leeftijd = 17
if leeftijd >= 18:
    print(“Jij bent meerderjarig.”)
    print(“Wil je een auto huren?”)
print(“Tot ziens.”)
```

- Wat zie je op het scherm als je dit programma uitvoert?
- Wat zou je zien als je de leeftijd op de eerste regel 18 maakt?
- Probeer het uit!

Meer over lists

- We hadden gezien dat een list gebruikt kon worden om meerdere objecten in te bewaren. Maar hoe gebruiken we zo'n ding? De belangrijkste dingen die je ermee kunt zijn:
  - Een list maken: bloemen = ['margriet', 'zonnebloem', 'viooltje']
  - Een object toevoegen aan een list: bloemen.append('iris')
  - Een object uit de list gebruiken: print(bloemen[1])
- Probeer dit uit! Bedenk voor je het probeert, wat je verwacht dat het print-commando op het scherm zet. Klopte je verwachting?

Herhalingen: for

- Vaak wil je iets doen voor (of met) elk object in een lijst. Daarvoor heeft Python het commando for. Om het te gebruiken heb je een list nodig, laten we de bloemen van net gebruiken.
- Je moet ook een nieuwe variabele maken. Die noem ik nu bloem. Het commando wordt dan: “for bloem in bloemen:”.
- Net als bij if komt hierna een ingesprongen blok. Dat blok wordt een keer uitgevoerd voor elk object in de list. De variabele (in dit voorbeeld “bloem”) heeft steeds de waarde van het object dat nu “aan de beurt” is.

Een voorbeeld met for

```
bloemen = [‘margriet’, ‘zonnebloem’, ‘viooltje’, ‘iris’]
for bloem in bloemen:
    print(‘In mijn tuin heb ik een ‘ + bloem + ‘ geplant.’)
print(‘Het is er heel gezellig.’)
```

- Wat verwacht je dat dit programma doet?
- Probeer het uit!
- Het commando while is ook nuttig, dat kan je zelf opzoeken.

Waar zijn we ergens? enumerate

- Vaak is het bij een for-lus nuttig om te weten op welke positie in de list je bezig bent. Daarvoor gebruik je enumerate op de volgende manier:
- for index, bloem in enumerate(bloemen):
- Zoals je ziet moet je nu twee variabelen maken (“index” en “bloem” in dit voorbeeld, die namen mag je zelf kiezen).
- In het ingesprongen blok is bloem steeds het object, en index is de positie in de list. De index begint natuurlijk op 0.
- Schrijf een programma wat de bloemen en hun index print.

Sla deze maar over: continue

- Soms is het nuttig om de rest van de commando's in het ingesprongen blok over te slaan. Bijvoorbeeld als je een list van allemaal getallen hebt, en je wilt 1/x uitrekenen, dan gaat dat niet lukken als x == 0. Dus in dat geval wil je meteen door naar het volgende object. Dat kan met het commando "continue":

```
getallen = [3, 15, 0, 4]
for getal in getallen:
    if getal == 0:
        continue
    print('het omgekeerde van ' + str(x) + ' is ' + str(1 / x) + '.')
```

Planning

- Inleiding
- Types en variabelen
- Flow control
- → Strings
- Functies
- Bestanden
- Modules: grafieken

Meer plezier met strings

- Tot nu toe hebben we strings steeds als “onbreekbare” dingen gebruikt. Maar dat hoeft niet.
- Je kan een string (bijna) gebruiken als een list van letters. Als bijvoorbeeld koningin = ‘Maxima’, dan is koningin[2] == ‘x’.
- Je kan een string ook in stukken breken. Het resultaat is dan een list met alle stukken. Hij breekt hem steeds als hij een bepaalde string ziet. Welke string, dat moet je hem vertellen.
- koningin.split(‘a’) is dus [‘M’, ‘xim’, ‘’].
- Probeer dit uit en probeer ook andere splits.

Speciale tekens in strings

- “Gewone” strings bevatten letters, cijfers en spaties. Maar er kunnen ook andere dingen in strings zitten. De belangrijkste daarvan zijn de enter en de tab.
- Om die te schrijven, gebruik je speciale codes. Die beginnen allemaal met een backslash (\). Let op: dat is dus niet het deelteken, maar juist de andere kant op.
- De enter noemen we een “newline” en dat is \n. De tab is \t. En als je gewoon een \ in je string wil hebben? Dan moet je hem twee keer typen: \\.
- Maak maar eens wat strings daarmee, en print die.

Planning

- Inleiding
- Types en variabelen
- Flow control
- Strings
- → Functies
- Bestanden
- Modules: grafieken

Functies

- Vaak wil je in je programma meerdere keren hetzelfde doen. Je kan bijvoorbeeld een bericht op een bepaalde manier aan de gebruiker willen laten zien. Daarvoor heb je dan een paar regels nodig in je programma.
- Je kan elke keer die regels gaan kopiëren en plakken, maar dat is een slecht idee. In plaats daarvan kan je veel beter een functie gebruiken.
- Een functie is een stukje programma met een naam. Je kunt die naam dan gebruiken om dat programma uit te voeren. Je hebt al wat ingebouwde functies gezien, bijvoorbeeld “print”.

Zelf functies maken

- Als je zelf een functie wilt maken, schrijf je dat zo:

```
def mijn_functie(x):
    print(“Je hebt “ + str(x) + “ gewonnen!”)
```

- Je kunt deze functie nu aanroepen als mijn_functie(“een auto”). Wat je tussen haakjes zet kan elke keer anders zijn. Zo hoeft een functie dus niet elke keer precies hetzelfde te doen.
- In plaats van een “literal” zoals hierboven, mag je ook een variabele of andere expressie meegeven.

Een functie uitrekenen

- Je kunt functies niet alleen gebruiken om een stuk programma mee te “verpakken”, zodat het makkelijker te gebruiken is; je kunt er ook een expressie mee uitrekenen.
- In dat geval gebruik je de functie dus als onderdeel van een “som”. Dat hoeft niet over getallen te gaan.
- In de functie kan je het commando “return” gebruiken om de functie te beëindigen. Als je er een expressie achter zet, is dat de “return value” van de functie. Voor de aanroeper is dat de waarde die de functie krijgt.

Voorbeeld van een functieaanroep

```
def kwadraat(x):
    print('de functie kwadraat wordt aangeroepen')
    return x * x
print('Ik ga een berekening doen')
print('3² + 4² = ' + str(kwadraat(3) + kwadraat(4)))
```

- Voorspel wat dit programma doet (en in welke volgorde) en probeer het daarna uit. Klopte je voorspelling? Begrijp je wat je ziet?

Planning

- Inleiding
- Types en variabelen
- Flow control
- Strings
- Functies
- → Bestanden
- Modules: grafieken

Lezen van bestanden

- Vaak is het nuttig om een tekstbestand in te lezen en het programma met de inhoud daarvan te laten werken.
- Je kunt in Python een bestand openen met “open”. Bijvoorbeeld: bloemen_bestand = open(“bloemen.txt”)
- De variabele bloemen_bestand is dan een ingewikkeld object. Wat je ervan moet weten is dat je de inhoud kunt lezen met read: bloemen = bloemen_bestand.read()
- Je kan het ook per regel inlezen met for regel in bloemen_file:
- Maak een tekstbestand en lees het op beide manieren in.

Bestanden schrijven

- Het kan ook nuttig zijn om bestanden te schrijven.
- Dit werkt hetzelfde, alleen moet je “open” dan vertellen dat je wil schrijven: nieuw = open(“nieuwe_bloemen.txt”, “w”) # w van write.
- Let op! Als het bestand al bestond, overschrijft hij dat!
- Je kunt een regel in het bestand schrijven met print, door er wat aan toe te voegen: print(“boterbloem”, file = nieuw)
- Bedenk dat “nieuw” daarin dus de naam van de eerder gemaakte variabele is; Python begrijpt de betekenis van het woord “nieuw” niet.

Planning

- Inleiding
- Types en variabelen
- Flow control
- Strings
- Functies
- Bestanden
- → Modules: grafieken

Modules

- Spyder is heel goed in het maken van grafieken. We gaan daar nu een heel klein stukje van bekijken.
- Je hebt daarvoor een “module” nodig. Dat is een stuk programma wat je kunt gebruiken, maar wat niet standaard in Python zit. Je gebruikt import om een module te “importeren”. Daarna kan je de inhoud ervan gebruiken.
- Het commando hier is: import matplotlib.pyplot as plt
- De “as plt” aan het eind zorgt ervoor dat je hem kunt gebruiken onder de naam plt. Als je dat niet doet, gebruik je de “echte” naam, dus in dit geval matplotlib.pyplot.

Grafieken maken

- Nu je de module het geïmporteerd, kan je hem gebruiken om grafieken te maken. Dat doe je bijvoorbeeld met:

  plt.plot([2, 4, 6, 8], [1, 3, 6, 4])
- De eerste list zijn de x-waarden en de tweede list de y-waarden.
- Je kunt de grafiek zien in het tabblad “Plots” van het stuk rechtsboven.
- Als je maar 1 list meegeeft, zijn dat de y-waarden, voor x gebruikt hij dan [0, 1, 2, ...] (een net zo lange lijst als de gegeven y-waarden).

Het grote werk!

- Je hebt nu alle informatie gezien die je nodig hebt om een nuttig programma te schrijven. Maak om te oefenen een programma dat een bestand inleest en daar een interessante grafiek van maakt.
- De eerste opdracht is om een grafiek te maken van de gegevens in bestand. Het bestand bevat twee kolommen van getallen.
- Als dat gelukt is, maak je een grafiek van de verschillen tussen elk punt en de waarde ervoor. Dus hoeveel hij elke keer groter of kleiner geworden is. Het eerste punt gebruik je dan niet, want daar zit geen punt voor.
- Als je daarna tijd hebt, maak dan nog meer grafieken.