Python Programming

Introductie tot programmeren in Python

Kristof Michiels

Inhoud

- We behandelen de basis van Python. Genoeg om van start te kunnen gaan met oefeningen die bestaan uit een reeks van relatief eenvoudige statements
- In latere hoofdstukken behandelen we meer aspecten van Python maar gaan daarnaast ook nog dieper in op een aantal zaken die hier korter worden behandeld
- Dit zal ons helpen om grotere en meer complexe problemen aan te pakken met onze code

Wat zien we in deze les?

- Code structureren in Python
- Output genereren met de print()-functie
- Commentaar in je code
- Coderegels laten doorlopen met \
- Variabelen en toewijzingen
- Expressies

Wat zien we in deze les?

- Expressies
- Strings
- Getallen / berekeningen maken
- None
- Input van de gebruiker

Code structureren in Python

Code structureren in Python

- Witruimte en indentatie leggen de structuur van blokken code vast ipv haakjes en puntkomma's
- Dit minimalisme is één van de meest karakteristieke aspecten van Python
  - Geen haakjes meer die ontbreken waarnaar je op zoek moet
  - Visuele structuur van de code komt overeen met de reële structuur. Dat maakt de code beter leesbaar
  - Python code ziet er vrij gelijk uit, welke programmeur ze ook geschreven heeft

```
vruchten = ["appel", "kiwi", "banaan", "mango"]
for vrucht in vruchten:
      print(vrucht)
      if vrucht == "banaan":
            break
```

Code structureren in Python

- Voor elk insprong-niveau gebruik ik 4 spaties
- Aanbevolen stijl - de zogenaamde PEP-8 die je vaak zult horen noemen: https://www.python.org/dev/peps/pep-0008/
- Gebruik geen tabs en meng geen tabs en spaties: indent-count wordt hierdoor verstoord
- :-) https://www.youtube.com/watch?v=SsoOG6ZeyUI (fragment Silicon Valley)

Output genereren met de print()-functie

Output genereren met de print()-functie

- Om output te genereren vanuit onze programma's gaan we in de cursus voornamelijk gebruik maken van de print()-functie
- Dit is de in Python gebruikelijke manier om zaken op je scherm te krijgen
- De wondere wereld van print(): https://realpython.com/python-print/

```
print("Welkom bij Python Programming!")
```

Commentaar in je code

Commentaar in je code

- Commentaar laat toe om zelfgeschreven notities toe te voegen aan je programma's
- In Python gebruiken we een hashtag ("#") aan het begin van een lijn commentaar om aan te geven dat we hiermee te maken hebben
- We geven hiermee aan de interpreter mee dat wat volgt niet hoeft geïnterpreteerd te worden
- Python heeft geen notatie voor multi-line commentaar. Je gebruikt een hashtag aan het begin van elke regel commentaar

```
# Een typisch "Hallo wereld"-voorbeeld
boodschap = "Welkom bij Python Programming!"
print(boodschap)
```

Coderegels laten doorlopen met \

Coderegels laten doorlopen met \

- Code is leesbaarder wanneer de regels kort wordt gehouden
- Aangeraden max aantal karakters door PEP-8 is 79: https://www.python.org/dev/peps/pep-0008/#maximum-line-length
- Als de regel toch langer wordt kan je gebruik maken van de backslash (= 'continuation character')
- Plaats je een \ op het einde van een lijn dan gedraagt Python zich alsof je je nog steeds op dezelfde regel bevindt

```
som = 1 + \
    2 + \
    3 + \
    4
print(som) # 10
```

Variabelen en toewijzingen

Variabelen en toewijzingen

- De meest gebruikte opdracht in Python is de toewijzing. Komt vrij goed overeen met gebruik in andere talen
- In tegenstelling tot veel andere computertalen is er geen declaratie van het variabele type of een scheidingsteken aan het einde van de regel nodig
- De lijn wordt beëindigd door het einde van de lijn
- Variabelen worden automatisch gemaakt wanneer ze voor het eerst worden toegewezen

```
getal = 12
```

Variabelen en toewijzingen

```
boodschap = "Welkom bij Python programming!"
print(boodschap)

boodschap = "Python leren programmeren behoort tot het mooiste wat er bestaat"
print(boodschap)
```

- We maken een variabele aan en noemen ze "*boodschap*"
- Elke variabele is verbonden met een waarde. Dit is de informatie die met de variabele wordt geassocieerd
- Je kan de waarde van een variabele op elk moment wijzigen wijzigen in je programma. Python zal steeds de huidige waarde van de variabele bijhouden en kunnen weergeven

Variabelen en toewijzingen

- Een veel voorkomende, maar onnauwkeurige verklaring is dat een variabele een container is die een waarde opslaat, een beetje zoals een emmer. Dit zou redelijk zijn voor veel programmeertalen (bvb C)
- In Python zijn variabelen echter geen containers. In plaats daarvan zijn het labels of tags die verwijzen naar objecten in de namespace van de Python-interpreter
- Verschillende labels (of variabelen) kunnen naar hetzelfde object verwijzen, en wanneer dat object verandert, verandert ook de waarde waarnaar door al die variabelen wordt verwezen

```
a = [2, 4, 8]
b = a
c = b
b[1] = 6
print(a, b, c) # [2, 6, 8] [2, 6, 8] [2, 6, 8]
```

Variabelen en toewijzingen

- Met variabelen als containers houdt dit resultaat uit bovenstaand voorbeeld geen steek
- Hoe kan het veranderen van de inhoud van één container tegelijkertijd de andere twee veranderen?
- Als variabelen echter alleen maar labels zijn die naar objecten verwijzen, is het logisch dat het wijzigen van het object waarnaar alle drie de labels verwijzen, overal wordt weerspiegeld
- Als de variabelen verwijzen naar constanten of onveranderlijke waarden, is dit onderscheid niet zo duidelijk

```
a = 1
b = a
c = b
b = 2
print(a, b, c) # 1 2 1
```

Variabelen en toewijzingen

- Omdat de objecten waarnaar ze verwijzen niet kunnen veranderen, is het gedrag van de variabelen in dit geval consistent met beide verklaringen
- In dit geval verwijzen a, b en c in feite allemaal naar hetzelfde onveranderlijke integer-object met waarde 1
- De volgende regel, b = 2, zorgt ervoor dat b verwijst naar het integer-object 2, maar het verandert niet de referenties van a of c

Variabelen en toewijzingen

Python-variabelen kunnen op elk object worden ingesteld, terwijl variabelen in veel andere talen alleen het type waarde kunnen opslaan waarin ze zijn gedeclareerd. Het volgende is volkomen toegelaten:

```
a = "Hallo"
print(a)    # Hallo
a = 5
print(a)    # 5
```

Een nieuwe toekenning overschrijft alle eerdere toekenningen. De del-instructie verwijdert de variabele:

```
b = 5
del b
print(b) # Zal een NameError exception raisen
```

Naamgeving variabelen

- Gebruik korte, beschrijvende namen
- Vermijd het gebruik van hoofdletters
- Namen van variabelen kunnen enkel bestaan uit letters, getallen en *underscores* ("_")
- De naam mag starten met een *underscore*, maar niet met een getal
- Spaties zijn niet toegestaan. Maak eventueel gebruik van *underscores* om woorden van elkaar te scheiden
- Vermijd het gebruik van Python *keywords* als namen voor variabelen. Dit zijn gereserveerde woorden (zie volgende slide)

Gereserveerde woorden in Python

```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

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

Meerdere toewijzigen

```
voornaam, familienaam, beroep = "Kristof", "Michiels", "Docent"
```

- Je kan waarden aan meerdere variabelen toekennen in één enkele statement
- Dit maakt je programma's korter en beter leesbaar
- Je scheidt hiervoor de namen van de variabelen met komma's en doet hetzelfde met de waarden
- Python kent dan de respectievelijke waarde toe aan elke genoemde variabele

Constanten

```
STUDENTEN_PER_GROEP = 4
```

- Een constante is zoals een variabele, maar met een waarde die niet verandert tijdens de looptijd van je programma
- Python heeft geen ingebouwde ondersteuning voor constanten maar developers gebruiken een naam bestaande uit enkel hoofdletters om aan te geven dat het hier over een niet te veranderen constante gaat

Expressies

Expressies

- Python ondersteunt rekenkundige en soortgelijke expressies of uitdrukkingen op een manier zoals dat in de wiskunde en de meeste andere programmeertalen gebeurt
- Standaardregels van rekenkundige voorrang zijn van toepassing
- Volgende code berekent het gemiddelde van 12 en 4, waarbij het resultaat aan variabele c wordt toegekend:

```
a = 3
b = 5
c = (a + b) / 2
```

Expressies

- Expressies met enkel gehele getallen leveren niet altijd een geheel getal op
- Een deling van gehele getallen geeft een kommagetal terug
- Wil je enkel het gehele gedeelte terugkrijgen als geheel getal, gebruik dan de //-operator
- Expressies hoeven niet alleen numerieke waarden te bevatten: strings, Booleaanse waarden en vele andere soorten objecten kunnen op verschillende manieren in expressies worden gebruikt. We zullen dit in de cursus gaandeweg kunnen vaststellen

Strings

Werken met tekst: strings

- Een string bestaat uit een reeks tekst-karakters
- Alles binnen aanhalingstekens wordt beschouwd als een string
- Je mag gebruik maken van enkele of dubbele aanhalingstekens
- Deze flexibiliteit laat toe om aanhalings- en afkappingstekens te gebruiken in je strings
- Wees standvastig in je keuzes

```
boodschap = "Dit is een string."
boodschap = 'Dit is ook een string.'
boodschap = 'Mijn vriend vroeg: "Programmeer jij ook in Python?" '
boodschap = "Ik hou van het schrijven van Python programma's"
```

Hoofdlettergebruik wijzigen in een string

```
naam = "Guido van Rossum"
print(naam.title()) # Elk woord laten beginnen met een hoofdletter: Guido Van Rossum
print(naam.upper()) # Alles in hoofdletters: GUIDO VAN ROSSUM
print(naam.lower()) # Alles in kleine letters: guido van rossum
```

- We beschikken hiervoor over een aantal zgn. methods. Dit zijn acties die je op bepaalde data kan toepassen
- De dot-notatie (".") vertelt Python in het eerste vb. om de title-method uit te voeren op de variabele naam
- Elke method wordt gevolgd door haakjes, omdat methods vaak bijkomende informatie nodig hebben om hun werk te kunnen doen. Hier is dit evenwel niet het geval
- De lower()-method wordt vaak gebruikt om data op te slaan die door een gebruiker werd ingegeven

Variabelen gebruiken in een string: f-strings

```
voornaam = "Guido"
familienaam = "van Rossum"
volledige_naam = f"{voornaam} {familienaam}"
boodschap = f"Bedankt voor Python, {volledige_naam.title()}!"
print(boodschap)
```

- Om variabelen te gebruiken binnen strings plaats je de letter f onmiddellijk voor het eerste aanhalingsteken
- Je omringt de variabelen die je in de string gebruikt met accolades
- Python vervangt elke variabele door de waarde
- De f in f-strings staat voor "format"

Witruimte toevoegen aan strings

```
print("\tHelderheid")
print("Pythonisch programmeren:\nHelder\nGeloofwaardig\nEfficiënt")
print("Pythonisch programmeren:\n\tHelder\n\tGeloofwaardig\n\tEfficiënt")
```

- Met witruimte doelen we op niet-afdrukbare tekens als spaties, tabs en einde-lijnsymbolen
- Je gebruikt ze om je output op een beter leesbare manier te organiseren
- Een tab-insprong toevoegen aan je tekst doe je met "\t"
- Een nieuwe regel voeg je toe met "\n"
- Combinaties zijn mogelijk: "\n\t" zorgt ervoor dat Python op een nieuwe regel begint, met een tabinsprong

Witruimte elimineren binnen strings

```
boodschap = " Ik hou van Python "
print(boodschap.rstrip())
print(boodschap.lstrip())
print(boodschap.strip())
```

- Python maakt het eenvoudig om (eventueel) aanwezige witruimte te verwijderen
- Er kan een onderscheid worden gemaakt tussen witruimte links, rechts of aan beide zijden van een string
- Handig als je twee strings met elkaar wil vergelijken

Getallen / berekeningen maken

Werken met gehele getallen (integers)

```
getal1 = -2
getal2 = 3
print(getal1 + getal2) # 1
print(getal1 ** getal2) # -8
print((getal1 + getal2) * getal1) # -2
```

- Integers kunnen in Python opgeteld (+), afgetrokken (-), vermenigvuldigd (*) en gedeeld (/) worden
- Python gebruikt twee vermenigvuldigingssymbolen (**) om exponenten weer te geven
- Elke expressie kan meerdere bewerkingen bevatten. De volgorde van bewerkingen wordt gerespecteerd
- Je mag gebruik maken van haakjes om deze volgorde aan te passen

Werken met decimale getallen (floats)

```
getal1 = 0.4
getal2 = 0.2
print(getal1 + getal2) # 0.6000000000000001
print(getal1 * 2) # 0.8
```

- Python noemt elk kommagetal een float
- Deze term wordt in de meeste programmeertalen gebruikt en verwijst naar het feit dat een decimaalteken op elke positie in een getal kan voorkomen
- Houd er rekening mee dat je soms een willekeurig aantal decimalen in je antwoord kunt krijgen

Gehele en decimale getallen

```
print(9 / 3) # 3.0
print(13 + 2.0) # 15.0
print(4 * 5.0) # 20.0
print(2.0 ** 3) # 8.0
totaal_aantal_aardbewoners = 7_902_193_151
print(totaal_aantal_aardbewoners) # 7902193151
```

- Wanneer je twee getallen deelt is het resultaat altijd een float
- Python gebruikt standaard een float als resultaat voor elke bewerking die een float bevat, zelfs als de uitvoer een geheel getal is
- Bij grote getallen kun je cijfers groeperen met underscores om ze leesbaarder te maken

None

None

- None is een speciaal basisgegevenstype dat een enkel speciaal gegevensobject definieert met de naam None
- Wordt gebruikt om een lege waarde weer te geven. Vaak gebruikt als een tijdelijke aanduiding om een punt in een gegevensstructuur aan te geven waar uiteindelijk betekenisvolle gegevens zullen worden gevonden
- Je kan eenvoudig testen op de aanwezigheid van None, omdat er slechts één instantie van None is in het hele Python-systeem (alle verwijzingen naar None verwijzen naar hetzelfde object) en None is alleen gelijk aan zichzelf

```
var = None
if var is None:
    print("None")
else:
    print("Niet None")
```

Input van de gebruiker

Input van de gebruiker

- Je kan de functie input() gebruiken om invoer van de gebruiker te krijgen
- Gebruik de promptstring die u aan de gebruiker wilt tonen als invoerparameter:

```
naam = input("Naam? ")
print(naam)
leeftijd = int(input("Leeftijd? "))
print(leeftijd)
```

Input van de gebruiker

- Nadeel is dat de invoer binnenkomt als een string, dus als je het als een getal wilt gebruiken, moet je de functie int() of float() gebruiken om de string om te zetten naar een getal

```
basis = float(input("Basis in cm? "))
hoogte = float(input("Hoogte in cm? "))
oppervlakte = basis * hoogte
print(f"De oppervlakte van de rechthoek is {oppervlakte}cm2")
```