Python Programming

If statements en loops

Kristof Michiels

Inhoud

- De code die we tot nog toe hebben gezien was volledig sequentiëel: de statements worden uitgevoerd van boven naar onder, één voor één
- In deze les zien we twee belangrijke technieken voor controle van de flow: if- statements en loops
- Deze bepalen of en onder welke voorwaarden (welke) code al dan niet mag worden uitgevoerd

If-statements

Basisvoorbeeld

```
weekdagen = ["maandag", "dinsdag", "woensdag", "donderdag", "vrijdag"]

for dag in weekdagen:
    if dag == "vrijdag":
        print(f"Thank god it's {dag}!")
    else:
        print(dag.title())
```

- We lopen hier een beetje vooruit: met een for-loop doorlopen we een lijst. Beiden komen heel snel aan bod
- Wat ons nu aanbelangt is het if-statement binnen de loop. We gaan ermee na of de huidige dag "vrijdag" is. Indien *True* drukken we een gepaste zin af. In elk ander geval drukken we gewoon de naam van de dag af
- Een beslissing dus! Let op het dubbelpunt na if en else en op de 4 spaties bij de statements die erop volgen

if-statements

- Een if-statement bevat een voorwaarde en een reeks statements die mogelijk kunnen worden uitgevoerd. Deze statements worden voorafgegaan door insprongen om aan te geven dat ze tot het if-statement horen
- De voorwaarde wordt geëvalueerd en indien *True* worden de statements effectief uitgevoerd; indien *False* worden deze statements overgeslagen
- Vervolgens gaat het programma verder met de instructies onder het if-statement

```
getal = 24
if getal % 2 == 0:
    print(f"Het getal {getal} is deelbaar door 2")
    # hier kunnen nog extra statements staan
print("En het programma loopt door...")
```

De voorwaarde binnen een if-statement

De voorwaarde die kan gaan van eenvoudig tot complex evalueert in *True* of *False*. We noemen die voorwaarde een Booleaanse expressie, genoemd naar wiskundige George Boole. Ze bevat meestal een relationele operator die twee waarden, variabelen of complexe expressies vergelijkt. De operatoren kunnen de volgende zijn:

- < Kleiner dan
- <= Kleiner dan of gelijk aan
- > Groter dan
- >= Groter dan of gelijk aan
- == Gelijk aan
- != Niet gelijk aan

Meerdere if-statements

```
getal = int(input("Geef een geheel getal aub: "))
if getal > 0:
    boodschap = "groter dan 0"
if getal <= 0:
    boodschap = "kleiner dan of gelijk aan 0"
print(f"Het getal is {boodschap}")
```

- Bovenstaand voorbeeld vraagt een geheel getal aan de gebruiker en gebruikt twee if-statements om na te gaan of dat getal groter of kleiner/gelijk is aan 0
- We maken hier tweemaal gebruik van een if-statement met twee voorwaarden die elkaar uitsluiten. Dit is toegelaten, maar het kan efficiënter, namelijk met een if-else statement...
- Let op! Meerdere if-statements kunnen perfect ok zijn indien de voorwaarden elkaar niet uitsluiten

If-else-statements

- Een if-else-statement heeft een extra else-gedeelte (zonder voorwaarde!) met eigen blok instructies
- Wanneer de if-voorwaarde wordt geëvalueerd zal bij *True* de eerste blok code worden uitgevoerd, en bij *False* de tweede. Er zal steeds één en enkel één blok code worden uitgevoerd
- Je gebruikt een if-else-statement ipv 2 if-statements als de voorwaarden elkaar uitsluiten. Dit is efficiënter: slechts één voorwaarde moet worden gecontroleerd

```
getal = int(input("Geef een geheel getal aub: "))
if getal > 0:
    boodschap = "groter dan 0"
else:
    boodschap = "kleiner dan of gelijk aan 0"
print(f"Het getal is {boodschap}")
```

If-elif-else-statements

```
getal = int(input("Geef een geheel getal aub: "))
if getal > 0:
    boodschap = "groter dan 0"
elif getal < 0:
    boodschap = "kleiner dan 0"
else:
    boodschap = "gelijk aan 0"
print(f"Het getal is {boodschap}")
```

- Gebruik if-elif-else wanneer je meer dan twee mogelijkheden wil bieden
- Begint met een if-gedeelte gevolgd door één of meer elif-gedeeltes, gevolgd door een else (zonder voorwaarde). Elk gedeelte beschikt over een door 4 spaties voorafgegaan codeblok

If-elif-else-statements

```
leeftijd = 16
if leeftijd < 4:
    ticketprijs = 2
elif leeftijd < 12:
    ticketprijs = 10
elif leeftijd < 26:
    ticketprijs = 18
else:
    ticketprijs = 26
print(f"Jouw toegang tot de zoo kost ${ticketprijs}.")
```

- De voorwaarden worden van boven naar onder één voor één geëvalueerd: van zodra een *True* is gevonden wordt de betreffende codeblok uitgevoerd en wordt de rest van het statement overgeslagen
- Indien geen *True* wordt gevonden voert de interpreter de code van het else-blok uit

If-elif-statements

```
leeftijd = 16
if leeftijd < 4:
    ticketprijs = 2
elif leeftijd < 12:
    ticketprijs = 10
elif leeftijd < 26:
    ticketprijs = 18
print(f"Jouw toegang tot de zoo kost ${ticketprijs}.")
```

- Het else-gedeelte op het einde mag worden weggelaten
- Het is dan mogelijk dan van alle codeblokken geen enkele wordt uitgevoerd, omdat geen enkele voorwaarde in *True* resulteert
- Bovenstaand voorbeeld is ok, maar wat met een bezoeker van 26 of ouder?

Geneste if-instructies

```
leeftijd, sociaal_tarief  = 45, True
if leeftijd < 4:
    ticketprijs = 0
elif leeftijd < 12:
    ticketprijs = 10
elif leeftijd < 26:
    ticketprijs = 18
else:
    if sociaal_tarief:
        ticketprijs = 12
    else:
        ticketprijs = 24
```

De codeblokken binnen de if-*-statements kunnen (zo goed als) elke Python-code bevatten. Dat kunnen ook andere if, if-else, if-elif of if-elif-else statements zijn. We noemen dit geneste if-instructies.

Booleaanse operatoren

- Een Booleaanse expressie is een uitdrukking die in *True* of *False* resulteert. Deze kunnen bevatten:
  - True, False, relationele operatoren, functie-aanroepen die True of False teruggeven
  - Python kent ook 3 Booleaanse operatoren: not, and en or
    - not keert de waarde van een Booleaanse expressie om: True wordt False en False wordt True
    - and en or combineren Booleaanse waarden tot een Booleaans resultaat
      - Bij and moeten alle onderdelen True zijn om in True te resulteren: True and True => True; False and True => False; False and False and True => False
      - Bij or moet minstens één onderdeel True zijn om in True te resulteren: False or True => True; False or False or True => True; False or False => False

Booleanse logica: een voorbeeld

```
naam = input("Geef je naam in aub: ")
leeftijd = int(input("Geef je leeftijd in aub: "))
if naam == "Kristof" or naam == "Bart":
    print("Leuk, je bent een Kristof of een Bart!")
elif naam == "David" and leeftijd == 28:
    print("Wat een toeval: een 28-jarige David!")
else:
    print(f"Welkom {naam}")
```

Loops

Herhalen met een while- of for-loop

- Indien je een bepaalde taak meerdere keren wil uitvoeren dan kan je gebruik maken van één van Pythons beschikbare loop-constructies: een while-loop of een for-loop
- Beiden laten toe dezelfde groep statements meerdere keren uit te voeren
- Gebruik je ze efficiënt dan kan je complexe zaken uitvoeren (zoals berekeningen) in slechts een zeer beperkt aantal statements

De while-loop: basisvoorbeeld

```
getal = 1
while getal <= 5:
    print(getal)
    getal += 1
```

De while-loop: basisvoorbeeld

```
getal = int(input("Geef een geheel getal aub (0 om de app te verlaten): "))
while getal != 0:
    if getal > 0:
        print("Dat is een positief getal")
    else:
        print("Dat is een negatief getal")
    getal = int(input("Geef een geheel getal aub (0 om de app te verlaten): "))
```

De while-loop

- Zoals je hebt kunnen vaststellen laat een while-loop een codeblok (indentatie met 4 spaties) uitvoeren zo lang een voorwaarde resulteert in *True*
- Wanneer de laatste regel van de codeblok is bereikt wordt teruggekeerd naar het begin van de loop en wordt de voorwaarde opnieuw geëvalueerd, en indien *True*, de codeblok opnieuw uitgevoerd
- Dit duurt tot de loop-voorwaarde resulteert in *False*. Als dit gebeurt wordt de codeblok niet meer uitgevoerd en gaat het programma verder met het eerste statement na de loop
- De loop-voorwaarde kan wijzigen door gebruikersinput of door code binnen de loop-codeblok. Zorg er elk geval voor dat de voorwaarde ooit *False* teruggeeft, anders belandt je programma in een *infinite loop*

For-loops

- For-loops zorgen er net als while-loops voor dat een bepaalde codeblok verschillende keren na elkaar kan worden uitgevoerd
- Het mechanisme dat de loop bepaalt is evenwel verschillend: een for-loop in Python wordt uitgevoerd voor elk item in een collectie
- Deze collectie kan een range van gehele getallen zijn, de letters in een string, of zoals we later zullen zien: de waarden opgeslagen in een datastructuur zoals een list

De for-loop: basisvoorbeelden

```
for i in range(5):
    print(i)
```

```
lijst = ['aap', 'noot', 'peer']
for i in lijst:
    print(i)
```

```
getal = int(input("Geef een geheel getal: "))
print(f"De veelvouden van 3 tot en met {getal} zijn:")
for i in range(3, getal + 1, 3):
    print(i)
```

For-loops

- de codeblok (met indentatie van 4 spaties) binnen de for-loop bevat statements die meerdere keren uitgevoerd kunnen worden, meer bepaald voor elk item in de collectie
- Elk element in de collectie wordt gekopieerd naar een variabele vooraleer de loop de codeblok uitvoert voor dit element
- De variabele kan gebruikt worden in de codeblok zoals elke andere variabele

Range()

- Een collectie van integers kan geconstrueerd worden door aanroepen van de range()-functie
  - Roep je deze functie aan met 1 argument dan start de range met 0 en eindigt met het argument -1
  - Wanneer twee argumenten zijn meegegeven dan start de range met het eerste argument en eindigt met het tweede argument -1. range(2,5) bvb geeft een range terug van 2, 3 en 4
  - Een lege range wordt teruggegeven als het eerste argument groter is dan het tweede. De codeblok in de loop wordt in dit geval nooit uitgevoerd
  - Range() kent ook nog een derde argument, de step (een geheel getal). Is deze groter dan 0 dan begint de range bij het eerste argument en loopt tot argument2 - 1, telkens in sprongen gelijk aan de stepwaarde
  - Een negatieve stepwaarde zorgt ervoor dat een collectie van afnemende waarden ontstaat. range(0, -4, -1) geeft een range terug die bestaat uit 0, −1, −2 en −3

Geneste loops

- De statements binnen de codeblok van een loop kunnen op hun beurt een loop bevatten. We noemen dit geneste loops
- Elk type loop kan genest zitten binnen een ander type buitenloop

```
boodschap = input("Geef een boodschap (laat leeg om te stoppen): ")
while boodschap != "":
    aantal = int(input("Hoe vaak moet dit herhaald worden? "))
    for i in range(aantal):
        print(boodschap)
    boodschap = input("Geef een boodschap (laat leeg om te stoppen): ")
```

Python Programming - les 2 -

kristof.michiels01@ap.be