Algemeen

Lussen vroegtijdig afbreken

Je kunt de statements `break` en `continue` gebruiken om een lus vroegtijdig af te breken. Deze statements worden echter algemeen aanzien als slechte programmeerstijl. Je kunt ze dus beter niet gebruiken, want ze zullen je punten kosten als je ze gebruikt op een evaluatie of een exam.

Een situatie waarin je een lus vroegtijdig zou willen afbreken, doet zich bijvoorbeeld voor als je op zoek moet gaan naar een oplossing door een aantal mogelijke gevallen uit te proberen, en te stoppen van zodra je één oplossing gevonden hebt. In plaats van te werken met `break` en `continue` kan je in dit geval beter werken met een variabele die aangeeft of de oplossing al gevonden werd

```
>>> gevonden = False
>>> while not gevonden:
...     if <oplossing gevonden>:  # <oplossing gevonden> stelt voorwaarde voor
...         gevonden = True
...
```

Van zodra de oplossing gevonden werd (hier aangegeven door het feit dat de voorwaarde *oplossing gevonden* de waarde `True` aanneemt), wordt de variabele `gevonden` op `True` gezet, waardoor uit de `while`-lus zal gesprongen wanneer de `while`-voorwaarde opnieuw geëvalueerd wordt na de huidige iteratiestap.

Voorstelling van regeleindes

Je kunt een regeleinde voorstellen als de string `'\n'`. Als je dus een string wilt opbouwen die uit meerdere regels bestaat, dan doe je dit zo

```
>>> tekst = 'Ziehier de eerste regel'
>>> tekst += '\n'
>>> tekst += 'En de tweede regel'
>>> tekst
'Ziehier de eerste regel\nEn de tweede regel'
>>> print(tekst)
Ziehier de eerste regel
En de tweede
```

Itereren over de posities én de elementen van een collectie

Je kunt de ingebouwde functie `enumerate` gebruiken om een iterator op te vragen van collecties (*iterables*: objecten van samengestelde gegevenstypes zoals `str`, `list`, `tuple`, bestanden, ...), die telkens zowel de positie en het volgende element uit de collectie teruggeeft. Onderstaand voorbeeld geeft bijvoorbeeld aan hoe je voor een string (`str`) tegelijkertijd de posities en de karakters op die posities kunt overlopen.

```
>>> for index, karakter in enumerate('abc'):
...     print(f'index: {index}')
...     print(f'karakter: {karakter}')
...
index: 0
karakter: a
index: 1
karakter: b
index: 2
karakter: c
```

Je kunt dit bijvoorbeeld gebruiken als je synchroon de karakters van twee strings wil overlopen.

```
>>> eerste = 'abc'
>>> tweede = 'def'
>>> for index, karakter in enumerate(eerste):
...     print(f'{karakter}-{tweede[index]}')
...
a-d
b-e
c-f
```

In dat geval is het echter aangewezen om de ingebouwde functie `zip` te gebruiken, die net haar bestaansrecht haalt uit het feit dat ze een iterator teruggeeft voor twee of meer collecties.

```
>>> eerste = 'abc'
>>> tweede = 'def'
>>> for karakter1, karakter2 in zip(eerste, tweede):
...     print(f'{karakter1}-{karakter2}')
...
a-d
b-e
c-f
```

Strings splitsen in verschillende delen

Om een string in verschillende delen te splitsen, is het soms handig om de stringmethode `split` te gebruiken. Aan deze methode moet je een argument doorgeven, dat aangeeft welke reeks opeenvolgende karakters moet gebruikt worden om de string (waarop je de methode aanroept) in stukken te breken. Deze reeks opeenvolgende karakters wordt *het scheidingsteken* genoemd.

```
>>> string = 'a-b-c-d'
>>> string.split('-')
['a', 'b', 'c', 'd']
```

Standaard wordt de string op alle plaatsen gesplitst waar de reeks opeenvolgende karakters voorkomt, en worden alle delen waarin de string werd opgesplitst in een lijst (`list`) gestopt, die door de methode wordt teruggegeven. Als je geen argument meegeeft aan de methode, dan splitst die standaard de string op waar er een reeks opeenvolgende witruimtekarakters voorkomt (spaties, tabs, newlines, ...). De stringmethode heeft ook nog een optioneel argument dat je kunt gebruiken om aan te geven in hoeveel stukken de string maximaal mag gesplitst worden.

Omdat de stringmethode `split` een lijst (`list`) teruggeeft, kan je rechtstreeks de elementen van deze lijst overlopen met een `for`-lus.

```
>>> string = 'a-b-c-d'
>>> for element in string.split('-'):
...     print(element)
...
a
b
c
d
```

Letters omzetten naar hun getalwaarde

Elk karakter heeft een getalwaarde (`int`). Zo heeft het vraagteken (`?`) als getalwaarde 63. Deze waarde kan bekomen worden met behulp van de ingebouwde functie `ord`.

```
>>> ord('?')
63
```

Het interessante aan deze getalwaarden is dat de opeenvolgende letters van het alfabet ook opeenvolgende getalwaarden hebben. Dit geldt zowel voor kleine letters als voor hoofdletters. Zo heeft de letter `a` als getalwaarde 97, waaruit we kunnen afleiden dat de letter `b` als getalwaarde 98 moet hebben. Met die wetenschap kunnen we dus op een eenvoudige manier de positie van een letter in het alfabet bepalen.

```
>>> letter = 'd'
>>> ord(letter)
100
>>> ord('a')
97
>>> ord(letter) - ord('a') + 1   # d is de 4e letter van het alfabet
4
>>> ord('z') - ord('a') + 1      # z is de 26e letter van het alfabet
26
```

Stringmethoden `islower()` en `isupper()`

De stringmethode `islower` controleert of *alle* tekens van een string kleine letters zijn. De stringmethode `isupper` controleert of *alle* letters van een string hoofdletters zijn.

```
>>> bool('ABCD'.islower())
False
>>> bool('ABCD'.isupper())
True
>>> bool('abcd'.islower())
True
>>> bool('abcd'.isupper())
False
>>> bool('AbCd'.islower())
False
>>> bool('AbCd'.isupper())
False
```

En nu eens helemaal ondersteboven

Escapen van backslash

Gebruik een backslash om het volgende karakter in de string te "escapen", waardoor dit karakter zijn speciale betekenis verliest (bijvoorbeeld om een dubbel aanhalingsteken te plaatsen in een string die zelf zit ingesloten tussen dubbele aanhalingstekens) of net een speciale betekenis krijgt (zoals een regeleinde dat word voorgesteld door `'\n'` of een tab die wordt voorgesteld door `'\t'`).

Omdat hierdoor ook de backslash zelf een speciale betekenis krijgt binnen een string (namelijk het escapen van karakters), moet je een letterlijke backslash binnen een string noteren als een dubbele backslash (`'\\'`), waarbij de eerste backslash de rol opneemt van het escape-symbool en de tweede backslash het karakter is dat zijn letterlijke betekenis moet krijgen door het escapen.

Ongebruikte variabelen

Als je in Eclipse gebruik maakt van pylint (Window > Preferences > PyDev > Pylint > Use pylint?) – een module die markeringen in de linkerkantlijn plaatst als je slechte programmeerstijl gebruikt – dan kan je wel eens een aanwijzing krijgen dat je een variabele gedefinieerd hebt die niet in je code gebruikt wordt. Op

zich is dat niet erg, behalve als die variabele later toch ergens zou opduiken in je code en daar mogelijks een verkeerde waarde zou hebben.

Dit kan zich bijvoorbeeld voordoen als je een `for`-lus in combinatie met de `range` functie gebruikt om een reeks statements een vast aantal keer uit te voeren. Omdat de syntaxis van de `for`-lus altijd een lusvariable verwacht die onmiddellijk op het `for` keyword volgt, moet je dus wel een variabele gebruiken die je niet noodzakelijk nog nodig hebt in je code. In dit geval is het gebruikelijk om een underscore (`_`) als variabelenaam te gebruiken, wat in Python-middens expliciet aangeeft: *dit is een variabele die niet zal gebruikt worden in de code*. Pylint houdt hiermee rekening, en zal voor een underscore nooit aangeven dat die variabele niet gebruikt wordt. De code

```
>>> teller = 0
>>> for i in range(4):
...     teller = 2 * teller + 1
...
```

kan je dus beter herschrijven als

```
>>> teller = 0
>>> for _ in range(4):
...     teller = 2 * teller + 1
...
```

De stringmethode `index`

De stringmethode `index` kan gebruikt worden om de meest linkse positie van een karakter in een string te bepalen.

```
>>> 'mississippi'.index('i')
1
```

Dit codefragment vindt dus het eerste voorkomen van de letter `i` op positie 1 in de string `mississippi`. Let hierbij op het feit dat Python de posities van de karakters in een string indexeert vanaf 0, en niet vanaf 1.

Aan de stringmethode `index` kan ook nog een tweede argument doorgegeven worden, dat aangeeft vanaf welke positie moet gezocht worden naar een eerstvolgende voorkomen van het karakter dat als eerste argument werd doorgegeven. Als je bijvoorbeeld weet dat er op positie 1 van de string `mississippi` een letter `i` staat, dan kan je de tweede positie waarop een letter `i` vinden door te beginnen zoeken vanaf positie 2:

```
>>> 'mississippi'.index('i', 2)
4
```

Het raadsel van Nob

Verwijderen van witruimte vooraan en/of achteraan

Je kunt de stringmethode `strip` gebruiken om witruimte (spaties, tabs, regeleindes) vooraan en achteraan te verwijderen. Als je enkel de witruimte vooraan wilt verwijderen, dan kan je de stringmethode `lstrip` gebruiken. Als je enkel de witruimte achteraan wil verwijderen, dan kan je daarvoor de stringmethode `rstrip` gebruiken. Je kunt aan deze stringmethode ook een argument meegeven, waarmee je aangeeft welke karakters er vooraan en/of achteraan moeten verwijderd worden. Voor de details hierover verwijzen we naar de Python Standard Library.

```
>>> tekst = '  Dit is een tekst  '
>>> tekst.strip()
'Dit is een tekst'
>>> tekst.lstrip()
'Dit is een tekst  '
```

```
>>> tekst.rstrip()
'  Dit is een tekst'
>>> tekst2 = '===Dit is een tekst==='
>>> tekst2.lstrip('=')
'Dit is een tekst==='
```

Uitlijnen van strings over een vast aantal posities

Je kunt in de *format specifier* van een `f`-string aangeven dat een vast aantal posities moeten gereserveerd worden voor een stuk tekst. Dit doe je door de tekst uit te schrijven als een string (aangegeven door de letter `s`) en daarvoor met een natuurlijk getal aan te geven hoeveel posities er voor die string moeten gereserveerd worden.

```
>>> f"{'abc':5s}"   # 5 posities voorbehouden
'abc  '
```

Als de string langer is dan het aantal voorbehouden plaatsen, dan wordt de volledige string uitgeschreven en wordt er dus meer plaats ingenomen dan voorbehouden. Als de string echter korter is dan het aantal voorbehouden plaatsen, dan wordt de string standaard links uitgelijnd. Je kunt in de *format specifier* echter ook de uitlijning aangeven: links, rechts of gecentreerd. Python zal dan automatisch het juiste aantal spaties vooraan en/of achteraan toevoegen.

```
>>> f"{'abc':<5s}"   # 5 posities voorbehouden, links uitlijnen
'abc  '
>>> f"{'abc':>5s}"   # 5 posities voorbehouden, rechts uitlijnen
'  abc'
>>> f"{'abc':^5s}"   # 5 posities voorbehouden, centreren
' abc '
```

Als er moet gecentreerd worden, en het aantal toe te voegen spaties is oneven, dan kiest Python ervoor om rechts één spatie meer toe te voegen dan links.

Herhaling van een string

Om een bepaalde string een vast aantal keer te herhalen, kan je de string vermenigvuldigen met een natuurlijk getal. Net zoals bij de vermenigvuldiging van getallen, gebruik je hiervoor de operator `*`. De volgorde van de string en het natuurlijk getal in de vermenigvuldiging speelt geen rol

```
>>> 'a' * 3
'aaa'
>>> 5 * 'ab'
'abababababab'
```

Dit is vooral handig als het aantal herhalingen van een string niet op voorhand vastligt, maar bijvoorbeeld zit opgeslagen in een variabele.

```
>>> herhalingen = 4
>>> herhalingen * 'abc'
'abcabcabcabc'
```

Bijbelcodes

Specifieke info

Om het verborgen woord uit een gegeven bijbelfragment te lezen, raden we je aan om de volgende strategie te gebruiken. Als eerst stap lees je alle regels van het volledige tekstfragment in, en zet je dit om naar één

enkele string waarin enkel de letters overgehouden worden. Alle andere karakters moet je immers negeren. Als je bijvoorbeeld het volgende bijbelfragment krijgt, verspreid over twee regels

```
And hast not suffered me to kiss my sons and my daughters?
thou hast now done foolishly in so doing.
```

Dan hou je op basis van de eerste stap enkel de string van opeenvolgende letters uit dit bijbelfragment over.

```
Andhastnotsufferedmetokissmysonsandmydaughtersthouhastnowdonefoolishlyinsodoing
```

Daarna kan je uit deze string het verborgen woord lezen door te starten vanaf de opgegeven letter, en daarna de volgende letters uit te lezen door telkens het aantal gegeven letters vooruit of achteruit te springen. Als je in dit geval start vanaf de 45ste letter, en telkens vier letters vooruit leest tot je een woord van 7 letters hebt uitgelezen, dan bekom je het woord `roswell`.