Python 2

DEEL 6, BESTANDEN EN EXCEPTIONS

Inleiding:

In de vorige les toonden we allerlei manieren om de uitvoer van tekst op het scherm aan te passen. In deze les zetten we de stap van je scherm naar bestanden: we gaan gegevens uit bestanden lezen en naar bestanden schrijven. Daarnaast leer je reageren op exceptions, oftewel foutmeldingen die Python je geeft als er iets misgaat.

Bestanden

Twee lessen geleden gebruikte je de functie input om wat de gebruiker op zijn toetsenbord intypt te registreren. En in de vorige les toonden we je hoe je met de functie print uitvoer op het scherm toont. Maar in- en uitvoer kan ook via bestanden verlopen. Laten we eens kijken hoe dat gaat. We beperken ons in deze les tot het lezen en schrijven van tekstbestanden. Je kunt ook met binaire bestanden werken, die willekeurige data in een andere vorm dan tekst kunnen bevatten, maar dat is wat meer werk omdat je de data nog moet interpreteren. Voor de rest werkt dit hetzelfde.

Wat komt in dit hoofdstuk aan de orde?

- Bestanden openen met open
- Een context manager gebruiken met with
- Tekstbestanden lezen met read
- Tekst in onderdelen opsplitsen met split
- Een lijst uitpakken in afzonderlijke variabelen
- Naar tekstbestanden schrijven met write of print
- Exceptions afhandelen

Lezen van een tekstbestand.

We tonen hier in een voorbeeld hoe je op een Linux-machine zoals een Raspberry Pi met Raspberry Pi OS (tot voor kort nog Raspbian geheten) het bestand met de lijst van gebruikers uitleest. Ook op macOS werkt dit voorbeeld trouwens.

```
with open('/etc/passwd'. 'rt') as bestand:
    print(bestand.read())
```

In de eerste regel openen we het bestand met de functie open. Het eerste argument is het bestand dat we willen openen. We hebben hier een volledig pad gebruikt: ‘/etc/passwd'. Met het tweede argument 'rt' geven we aan dat we het bestand willen lezen en dat het om een tekstbestand gaat.

Een voorbeeld van een dergelijk bestand vind je in het klassenkanaal van TEAMS bij de “files” (“bestanden”) met de naam: “Linux passwd file voorbeeld.txt”. Kopieer dit bestand naar je eigen computer, geef het de naam “linux_passwd.txt” en zet het op een plek waar je het makkelijk kunt vinden.
Als je je het bestand van TEAMS hebt gedownload en (bijvoorbeeld) in C:/voorbeelden hebt geplaatst, ziet dat er uit als:

```
with open ('C:/voorbeelden/linux_passwd.txt', 'rt') as bestand:
    print(bestand.read())
```

De constructie met with is wat Python een ‘context manager’ noemt. In het with-blok heb je toegang tot het object bestand, dat het geopende bestand voorstelt. Na het with-blok wordt het bestand automatisch gesloten, zodat je het niet meer kunt lezen. Dat lijkt vanzelfsprekend, maar dat is het niet: ook zonder with kun je bestanden openen, maar als je dan het bestand na gebruik vergeet te sluiten, kan dat tot problemen leiden. Werk dus nooit met bestanden zonder with.
In de tweede regel roepen we de functie read op het object bestand aan. Deze functie geeft de volledige inhoud van het tekstbestand terug als een string, die we dan met print op het scherm tonen. De uitvoer ziet er als volgt uit (we tonen hier maar enkele regels).

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
mandar:x:500:500:Mandar Shinde:/home/mandar:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
tintin:x:500:500::/home/tintin:/bin/bash
…
enzovoort
```

Een tekstbestand regel voor regel lezen

Maar wat als we niet het hele bestand in één keer willen inlezen, maar regel voor regel, bijvoorbeeld omdat we willen testen of de regels aan specifieke voorwaarden voldoen? Geen probleem, ook dat is in Python heel eenvoudig. In plaats van de functie read op je bestand toe te passen, ga je dan met een for-lus door de elementen van het bestand. Het tekstbestand dat je van de functie open terugkrijgt, gedraagt zich immers als een lijst met als elementen de opeenvolgende regels in het bestand.

Opdracht 1:

Maak zo'n for lus en print het bestand dan opnieuw, regel voor regel.

Een string splitsen

Maar als we die regels één voor één gaan inlezen, moeten we er ook iets mee doen. Zoals je ziet bevat het bestand linux_passwd.txt op elke regel allerlei informatie over de gebruiker, telkens afgescheiden door een dubbele punt. We willen elk van die gegevens afzonderlijk uitlezen. Dat gaat eenvoudig met de functie split die we op een string kunnen uitvoeren. Bijvoorbeeld (we gebruiken hier *literals* in de Thonny shell):

```
>>> 'root:x:0:0:root:/root:/bin/bash'.split(':')
['root', 'x', '0', '0', 'root', '/root', '/bin/bash']
```

Je ziet hier dat we aan de functie split het teken meegeven dat de verschillende componenten van de string afscheidt: Het resultaat is een lijst met strings die onderdeel uitmaken van onze lange string, zonder de af scheidingstekens

De opeenvolgende componenten in de regels van het bestand linux_passwd.txt hebben overigens de volgende betekenis: gebruikersnaam, ongebruikt, ID van de gebruiker, ID van de groep, volledige gebruikersnaam, persoonlijke map van de gebruiker, shell van de gebruiker.

Voor een complete uitleg van dit file format in Linux:
https://www.computernetworkingnotes.com/rhce-study-guide/etc-passwd-file-in-linux-explained-with-examples.html

Een lijst uitpakken

Je kunt nu naar de elementen in de gesplitste string verwijzen met een index, bijvoorbeeld:

```
>>> informatie = 'root:x:0:0:root:/root:/bin/bash'.split(':')
>>> informatie[0]
'root'
>>> informatie[6]
'/bin/bash'
```

Maar dat is niet heel duidelijk. Zo willen we informatie[0] eigenlijk gebruiker noemen en informatie[6] de naam shell geven. Gelukkig kun je de elementen van een lijst in Python eenvoudig in één keer aan enkele variabelen toekennen. Dat heet *unpacking.* In ons voorbeeld gaat dat als volgt:

```
>>> gebruiker,*_, naam, directory, shell =
'root:x:0:0:root:/root:/ bin/bash'.split(':')
>>> gebruiker
'root'
>>>>_
['x','0','0']
>>> naam
```

```
'root'
>>> directory
'/root'
>>> shell
'/bin/bash'
```

De notatie * gebruik je om een willekeurig[1] aantal elementen uit te pakken. Omdat we in dit geval niet in deze elementen geïnteresseerd zijn, kennen we ze toe aan de variabele met de naam '_' vandaar dat we bij het uitpakken *_ gebruiken. We konden dit hier ook vervangen door:

```
gebruiker,_, _, _, naam, directory, shell
```

en als je nu:

```
>>> gebruiker,*_, something, naam, directory, shell =
'root:x:0:0:root:/root:/ bin/bash'.split(':')
```

Vraag: En nu wil je de elementen "something" en "_ " zien (doe dat in de shell). Wat valt je nu op? Wat is er veranderd aan _ ?

Opdracht 2:

Dan weet je nu genoeg om de volgende opdracht uit te voeren: lees het bestand met wachtwoorden regel per regel in en als de shell geen '/usr/sbin/nologin' of '/bin/false' is, toon je de gebruikersnaam, volledige gebruikersnaam en persoonlijke map.

Probeer het eerst zelf voordat je de code hieronder overneemt. Dan zijn er de volgende aanwijzingen:

Op de string die je inleest, moet je op het laatste element ("shell") de functie strip toepassen.

```
shell = regel.strip()
```

Dat hebben we hier nodig omdat de shell op het einde van de regel staat en er daar dus een teken voor een nieuwe regel komt (/n). Zonder die aanroep van strip zou de vergelijking in de regel erna niet werken. Strip verwijdert witruimte en newlines (/n) aan begin en/of eind van de string.

We openen dus het bestand linux_passwd.txt als tekstbestand om te lezen. Voor elke regel in het bestand pakken we de verschillende elementen uit in enkele variabelen. We kijken dan of de shell niet gelijk is aan de twee eerdergenoemde shells. Als aan die voorwaarde is voldaan, tonen we de gebruiker, zijn volledige naam, zijn persoonlijke map en zijn shell.

[1] Eigenlijk niet willekeurig, want Python rekent uit hoeveel elementen er in de list voorkomen en trekt dan het aantal benoemde velden daarvan af. Wat er overblijft is het aantal elementen dat in _ terechtkomt.

De code ziet er als volgt uit: (op z'n kop … want ik wil het niet een copy – paste feest laten worden.)

```
with open('/etc/passwd', 'rt') as bestand:
    for regel in bestand:
        gebruiker, *_, naam, directory, shell = regel.strip().split(':')
        if shell not in ['/bin/false', '/usr/sbin/nologin']:
            print('{} ({}): {2} ({3})'.format(gebruiker, naam.D directory, shell))
```

Naar een tekstbestand schrijven

Naar een tekstbestand schrijven verloopt op een vergelijkbare manier als een tekstbestand lezen. We beginnen een with-blok waarin we het bestand openen en daarin schrijven we naar het bestand:

```
with open ('bestand.txt', 'wt') as bestand:
    bestand.write('Dit is de eerste regel.\n')
    bestand.write('Dit is de tweede regel.\n')
    bestand.write('Dit is de derde regel.\n')
```

Aan het einde van elke regel moet je zelf een teken voor een nieuwe regel toevoegen: \n. Een andere manier om een regel naar een tekstbestand te schrijven is met de functie print, die automatisch een nieuwe regel toevoegt:

```
print('Dit is de eerste regel.', file=bestand)
```

Merk op dat we het bestand openen met als tweede argument 'wt', waarmee we aangeven dat we naar het bestand willen schrijven. Op deze manier overschrijven we alle reeds bestaande inhoud van het bestand, dus let hiermee op!

Als je deze situatie wilt vermijden, kun je open aanroepen met de bestandsmodus 'xt'. Als het bestand nog niet bestaat, doet die hetzelfde als 'wt': je kunt naar het bestand schrijven. Maar als het bestand al bestaat, krijg je een foutmelding:

```
with open('bestand.txt', 'xt') as bestand:
    print('Dit is een test.', file=bestand)
with openCbestand.txt', 'xt') as bestand:
    print('Dit is nog een test.', file=bestand)
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
FileExistsError: [Errno 17] File exists: 'bestand.txt'
```

Een andere interessante bestandsmodus is 'at' (van 'append'): hiermee voeg je aan het eind van een bestaand tekstbestand regels toe.

Exceptions afhandelen

In het voorbeeld hierboven zou je waarschijnlijk de foutmelding dat het bestand al bestaat op een nettere manier willen afhandelen. Wat we tot nu toe een foutmelding hebben genoemd,

heet in Python een exception. Er bestaan verschillende types exceptions en in je Python-code kun je eenvoudig het optreden van exceptions afvangen. Dat gaat als volgt:

```
try:
    with open('bestand.txt', 'xt') as bestand:
        print('Dit is nog een test.', file=bestand)
except FileExistsError:
    print('DIT GAAT FOUT, want het bestand bestaat al.')
```

De code binnen het try-blok wordt uitgevoerd zoals normaal, maar als er binnen dit blok een exception voorkomt, gaat het programma door naar het except-blok. Daarin hebben we aangegeven dat we alleen in de exceptions van het type FileExistsError geïnteresseerd zijn. In het geval er zo een voorkomt, tonen we onze eigen foutmelding. Daarna gaat het programma verder na het except-blok.

Als je meerdere types exceptions wilt afvangen, voeg je meerdere except-blokken toe met elk het andere type exception. Als je voor meerdere types exceptions dezelfde code wilt uitvoeren, dan zet je die exceptions tussen haakjes, zoals hier:

```
except (ZeroDivisionError, ValueError):
```

En als je op alle mogelijke exceptions hetzelfde wilt reageren, voeg je gewoon een except-blok zonder de naam van een exception toe, al is dat niet zo vaak zinvol.

Opdracht 3:

Vraag een gebruiker om input. Hij moet een regel zoals: "root:x:0:0:root:/root:/bin/bash" voor gebruik in een wachtwoordbestand opgeven. Maar in plaats van "root" gebruikt hij natuurlijk zijn eigen naam. Vervolgens schrijf jij de belangrijkste elementen van die regel naar een bestand, in de vorm:

Gebruiker: root
Naam: root
Directory: /root
Shell: /bin/bash

En je zorgt ervoor dat het programma een heldere foutmelding geeft als de gebruiker niet de correcte vorm voor een wachtwoordbestand heeft.

Vragen die je jezelf kunt stellen.

Hoe vraag ik een gebruker om input? Welke string krijg ik nu binnen als het goed gaat? Hoe splits ik die string in de juiste elementen in de juiste volgorde? Hoe open ik een bestand om naar te schrijven? Heb ik een bepaald format nodig om naar het bestand te schrijven? Hoe zet ik een exceptionafhandeling op? Welke exception wil ik afvangen? Welke heldere, duidelijke melding geef ik dan aan de gebruiker terug?

De inhoud van het bestand linux_passwd.txt

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
mandar:x:500:500:Mandar Shinde:/home/mandar:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
nagios:x:501:501::/home/nagios:/bin/bash
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
rpm:x:37:37::/var/lib/rpm:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
avahi:x:70:70:Avahi daemon:/:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
tintin:x:500:500::/home/tintin:/bin/bash