Algemeen

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
```

Tekstbestanden naar Pycharm kopiëren

Als je je oplossingen voor opgaven met tekstbestanden lokaal wil testen, dan is het noodzakelijk dat je ook de tekstbestanden lokaal beschikbaar hebt. Anders kunnen de testgevallen uit de doctest deze tekstbestanden niet vinden. De tekstbestanden die in de gegeven doctest gebruikt worden, staan altijd gekoppeld in de omschrijving boven de doctest. Je kunt de inhoud van deze bestanden in je browser bekijken door op de koppeling te klikken.

De meest generieke procedure om een lokale kopie van deze bestanden ter beschikking te hebben in PyCharm verloopt als volgt:

- open het bestand in je browser
- kopieer de inhoud van het bestand (`CTRL-A` + `CTRL-C`)
- maak een nieuw tekstbestand in Pycharm

  - klik rechts op de directory waarin het tekstbestand moet geplaatst worden (je moet er voor zorgen dat het bestand in dezelfde directory geplaatst wordt waar ook je Python script staat)
  - kies het menu-item `New` en daarna het menu item `File`
  - geef de correct naam in van het bestand; let hierbij ook op het feit dat je de bestandsextensie moet opgeven (doorgaans `.txt`)
- plak de inhoud van het klembord in het bestand (`CTRL-V`)

Het onderstaande screenshot helpt je alvast op weg.

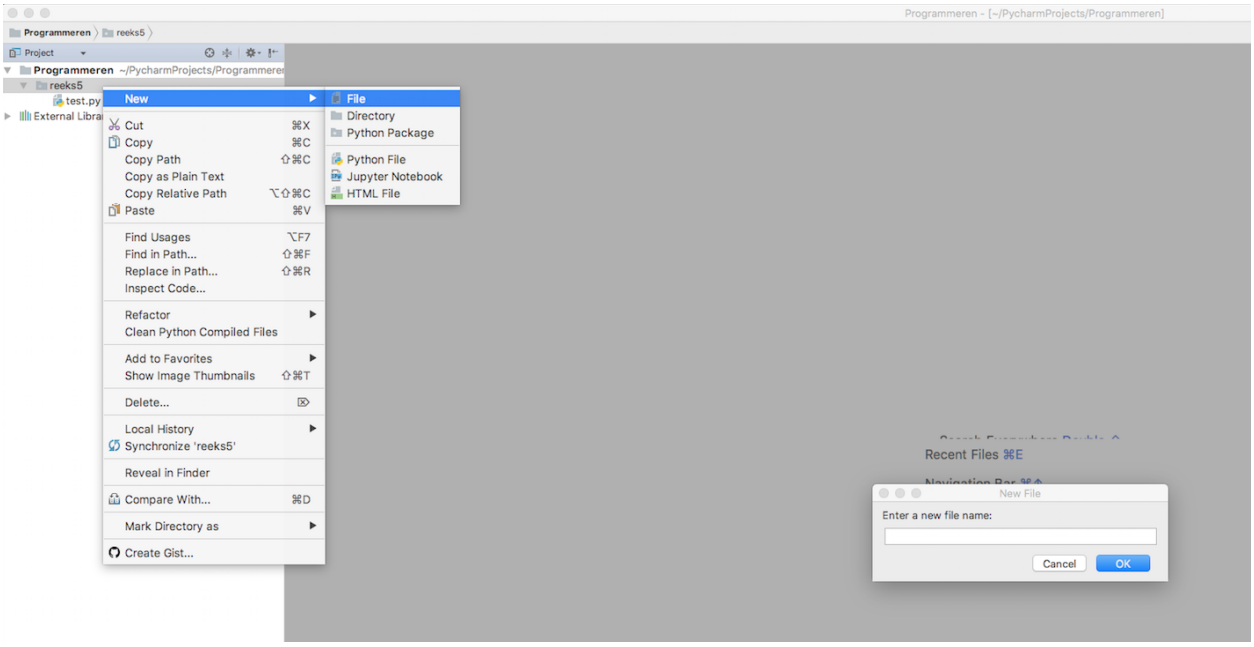

Figure 1: menu nieuw bestand

Als je je oplossingen indient via Dodona, dan zorgt Dodona er zelf voor dat de nodige tekstbestanden beschikbaar zijn in de directory waarin ook het Python script geplaatst wordt.

Stringvoorstelling van een rooster

De volgende *list comprehension* bouwt een de stringvoorstelling op van een rooster, waarbij de rijen van het rooster op afzonderlijke regels weergegeven worden. Met andere woorden, de regels worden telkens van elkaar gescheiden door een newline (de string waarop de buitenste `join` methode wordt aangeroepen). De elementen van de rijen worden telkens van elkaar gescheiden door één enkele spatie (de string waarop de binnenste `join` methode wordt aangeroepen).

```
>>> rooster = [['A', 'B', 'C'], ['D', 'E', 'F'], ['G', 'H', 'I']]
>>> print('\n'.join([' '.join(rij) for rij in rooster]))
A B C
D E F
G H I
```

Eerste regel van een bestand afzonderlijk verwerken

Soms heb je een tekstbestand waarvan de eerste regel een andere rol speelt dan de andere regels in het bestand (bijvoorbeeld een hoofding) en is het handig dat je die eerst kunt inlezen voordat je de rest van de regels inleest en verwerkt.

Stel bijvoorbeeld dat je volgend bestand hebt.

```
eerste
tweede
derde
```

Dan kun je de eerste regel op de volgende manier apart inlezen van de rest van het bestand.

```
>>> bestand = open('bestandsnaam.txt', 'r')
>>> eerste_regel = bestand.readline()
>>> eerste_regel
"eerste\n"
>>> andere_regels = []
>>> for regel in bestand:
...     andere_regels.append(regel)
>>> andere_regels
["tweede\n", "derde\n"]
```

Belangrijk om weten is dat Python dezelfde regel standaard nooit twee keer leest. Als je de regels van het bestand twee (of meer keer wil doorlopen), dan kan je ofwel het bestand afsluiten na de eerste iteratie (met de ingebouwde functie `close()`) en daarna het bestand terug openen (met de ingebouwde functie `open()`). Als alternatief kan je in een geopend bestand ook terug naar het begin springen (of naar elke andere positie binnen het bestand) aan de hand van de methode `seek()` van bestandsobjecten.

Newlines bij het inlezen van een bestand

Als Python de volgende regel moet lezen uit een tekstbestand, dan wordt er gelezen tot en met de eerstvolgende newline (`'\n'`) of tot het einde van het bestand. Het laatste karakter van de regel die werd gelezen zal dus doorgaans een newline zijn (tenzij er na de laatste regel geen newline meer stond).

Om het newline karakter achteraan de regel te verwijderen, kan je de stringmethode `str.rstrip()` gebruiken. Als er geen argumenten doorgegeven worden aan deze methode, dan zal de methode alle witruimtekarakters (spaties, tabs en newlines) op het einde van de regel verwijderen. Om er zeker van te zijn dat er enkel newlines verwijderd worden op het einde van de regel, kan je het newlinekarakter doorgeven als argument aan de methode `str.rstrip()`.

```
>>> regel = bestand.readline()
>>> regel
'Dit is een regel in het bestand.\n'
>>> regel.rstrip('\n')
'Dit is een regel in het bestand.'
```

Resultaten naar een bestand wegschrijven

De ingebouwde functie `print()` kan gebruikt worden om de stringvoorstelling van een resultaat weg te schrijven naar een bestand. Hiervoor kan je de optionele parameter `file` van de functie `print` gebruiken.

Standaard schrijft de functie `print` het resultaat naar het speciale bestand `sys.stdout` (de standaardwaarde van de parameter `file`) dat bijvoorbeeld aan de Console van PyCharm gekoppeld is. Hetzelfde effect kan je bekomen door de waarde `None` door te geven aan de parameter `file`.

Door een bestandsobject dat geopend werd om te schrijven door te geven aan de parameter `file` van de functie `print`, wordt de stringvoorstelling van het resultaat naar dit bestand weggeschreven.

```
>>> regel1 = 'Dit is de eerste regel.'
>>> regel2 = 'Dit is de tweede regel.'
>>> print(regel1)
Dit is de eerste regel.
>>> print(regel2, file=None)
Dit is de tweede regel.
```

```
>>> uitvoer = open('uitvoer.txt', 'w')
>>> print(regel1, file=uitvoer)
>>> print(regel2, file=uitvoer)
>>> uitvoer.close()
>>> bestand = open('uitvoer.txt', 'r')
>>> bestand.readline()
'Dit is de eerste regel.\n'
>>> bestand.readline()
'Dit is de tweede regel.\n'
>>> bestand.readline()
''
```

Splitsen op witruimte

De stringmethode `str.split()` kan gebruikt worden om een string op te splitsen in een lijst van deelstrings. Als er geen argument wordt doorgeven aan de methode, dan zal de methode alle witruimtekarakters (spaties, tabs en newlines) vooraan en achteraan de string verwijderen, en daarna de string opdelen in de deelstrings die van elkaar gescheiden worden door één of meer opeenvolgende witruimtekarakters.

Als er wel een stringargument wordt doorgegeven aan de stringmethode `str.split()`, dan zal de methode dit argument gebruiken om de string op te splitsen bij elk voorkomen van het argument. Als de string bijvoorbeeld twee opeenvolgende spaties bevat, dan zal de stringmethode `split` zonder argument de string op die plaats slechts één keer splitsen, terwijl de stringmethode `str.split()` met een spatie als argument de string op die plaats twee keer zal splitsen. In dit laatste geval zal er dus een lege deelstring staan tussen de twee spaties in de string.

```
>>> tekst = 'a;b  c;d;e\t  f'
>>> tekst.split()
['a;b', 'c;d;e', 'f']
>>> tekst.split(' ')
['a;b', '', 'c;d;e\t', '', 'f']
>>> tekst.split(';')
['a', 'b  c', 'd', '', 'e\t  f']
>>> tekst.split('\t')
['a;b  c;d;e', '  f']
```