General

Newlines when reading lines from text files

If Python needs to read the next line from a text file, it will continue reading until the first newline character (`'\n'`) or the end of the file is reached. The last character of the line that was read from the file will therefore usually be a newline (unless the last line of the file did not end in a newline).

The string method `rstrip` can be used to remove the trailing newline at the end of a line. In case this method is called without any arguments, all whitespace characters (spaces, tabs and newlines) at the end of the line will be removed. To make sure that only the newline is removed from the end of the line, you may pass the newline character as an argument to the `rstrip` method.

```
>>> line = infile.readline()
>>> line
'This is the next line in the file.\n'
>>> line.rstrip('\n')
'This is the next line in the file.'
```

Copy text file to Eclipse

If you want to locally test your solution for an assignment using text files, you must also make sure to have a local copy of the text files. Otherwise the test cases of the doctest will not be able to access these text files. The text files that are used in a given doctest are always linked in the description on top of the doctest. You can inspect the content of these text files in your browser by clicking this link.

The most general procedure to obtain a local copy of these text files in Eclipse goes as follows:

- open the text file in your browser
- copy the file content to the pastebin (`CTRL-A` + `CTRL-C`)
- create a new text file in Eclipse
  - right click the directory that needs to contain the text file (you must make sure that the text file is in the same directory as your Python script)
  - chose the menu item `New` and then the menu item `File`
  - enter the correct name of the file under `File name`; make sure that the file extension must also be given (usually `.txt`)
- paste the content of the pastebin into the file (`CTRL-V`)

The following screenshot shows you the way.

If you submit a solution to Dodona, the platform will make sure that the necessary text files are in the same directory as the Python script.

Say it like Adele

General information

Remarks

Output results to a file

The built-in function `print` can be used to write the string representation of a result to a file. This can be done by making use of the optional parameter `file` of the function `print`.

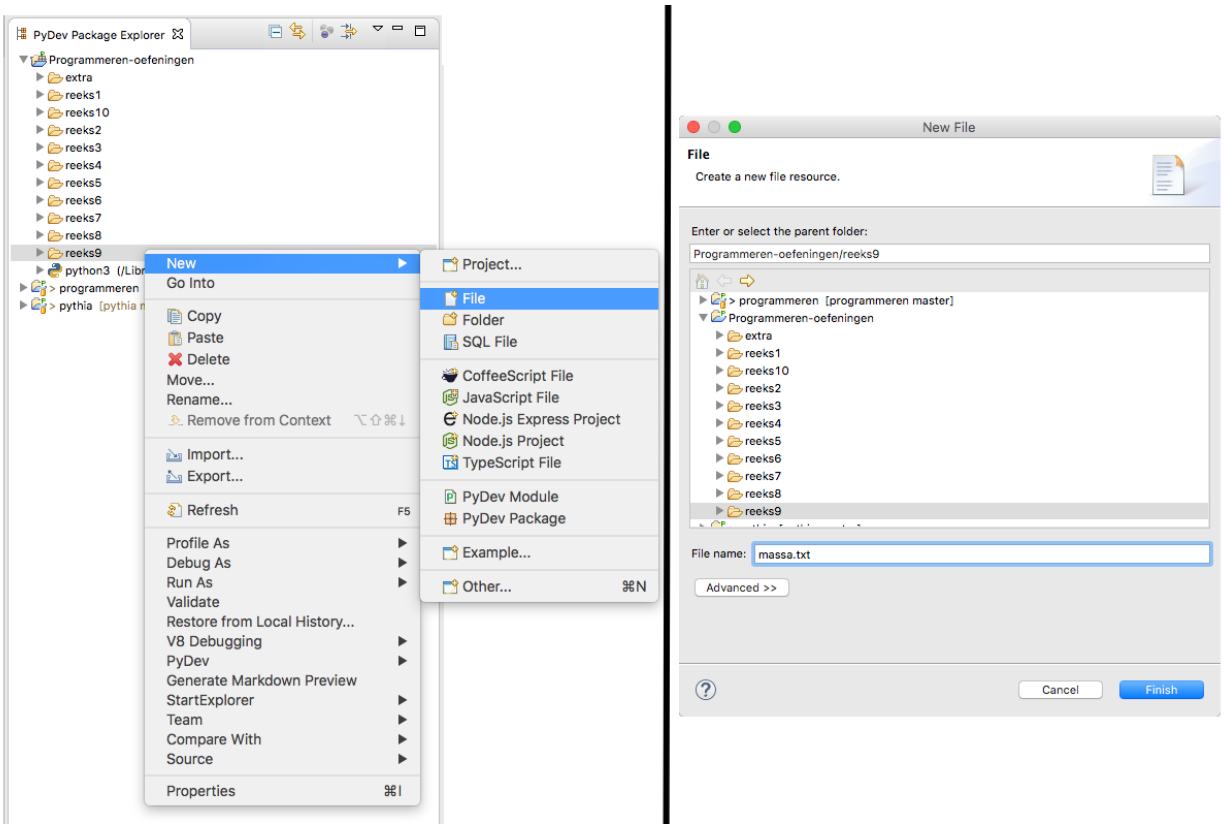

Figure 1: menu new file

By default, the function `print` will write the result to the special file `sys.stdout` (the default value of the parameter `file`) that for example might be attached to the Console window of Eclipse. The same effect can be obtained by passing the value `None` to the parameter `file`.

By passing a file object that was opened for writing to the parameter `file` of the function `print`, the string representation of the result is written to this file.

```
>>> line1 = 'This is the first line.'
>>> line2 = 'This is the second line.'
>>> print(line1)
This is the first line.
>>> print(line2, file=None)
This is the second line.

>>> outfile = open('output.txt', 'w')
>>> print(line1, file=outfile)
>>> print(line2, file=outfile)
>>> outfile.close()
>>> infile = open('input.txt', 'r')
>>> infile.readline()
'This is the first line.\n'
>>> infile.readline()
'This is the second line.\n'
>>> infile.readline()
''
```

AWOL

General information

Remarks

Cyclometric functions from the `math` module

The `math` module from the Python Standard Library defines a couple of **cyclometric functions** (or inverse trigonometric functions) such as the arcsine function (`asin`), the arccosine function (`acos`) and the arctangent function (`atan` or `atan2`). The domain of the arcsine and the arccosine functions is $[-1, 1]$. This means that these function only take *floating point* values from the interval $[-1, 1]$. As a result, you must take care that rouding errors do not cause values outside this domain. The following example illustrates how the detrimental effect of rounding errors can be remedied.

```
>>> import math
>>> value = 1.00001
>>> math.acos(value)                    # compute arccosine
Traceback (most recent call last):
ValueError: math domain error
>>> value = max(-1.0, min(value, 1.0)) # guarantee that value is in interval [-1, 1]
>>> value
1.0
>>> math.acos(value)
0.0
```

Plutokiller

General information

Set operators

In Python it is really easy to compute the union, the intersection and the difference of two sets.

```
>>> set1 = {'A', 'B', 'C'}
>>> set2 = {'C', 'D', 'E'}
>>> set1 & set2               # doorsnede
{'C'}
>>> set1 | set2               # unie
{'A', 'B', 'C', 'D', 'E'}
>>> set1 - set2               # verschil
{'A', 'B'}
>>> set2 - set1               # verschil is assymetrisch
{'D', 'E'}
```

String representation of a grid

The following *list comprehension* constructs a string representation of a grid, with each the rows of the grid on a separate line. In other words, the lines are separated from each other by a newline (the string on which the outer `join` method is called). The elements of the rows are separated from each other by a singel space (the string on which the inner `join` method is called).

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