General

Copy text file to PyCharm

If you want to locally test your solutions for assignments that make use of 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 generic procedure to obtain a local copy of these text files in PyCharm goes as follows:

- open the text file in your browser
- copy the file content to the clipboard (`CTRL-A` + `CTRL-C`)
- create a new text file in Pycharm
  - 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; make sure that the file extension is also given (usually `.txt`)
- paste the content of the clipboard into the file (`CTRL-V`)

The following screenshot shows you the way.

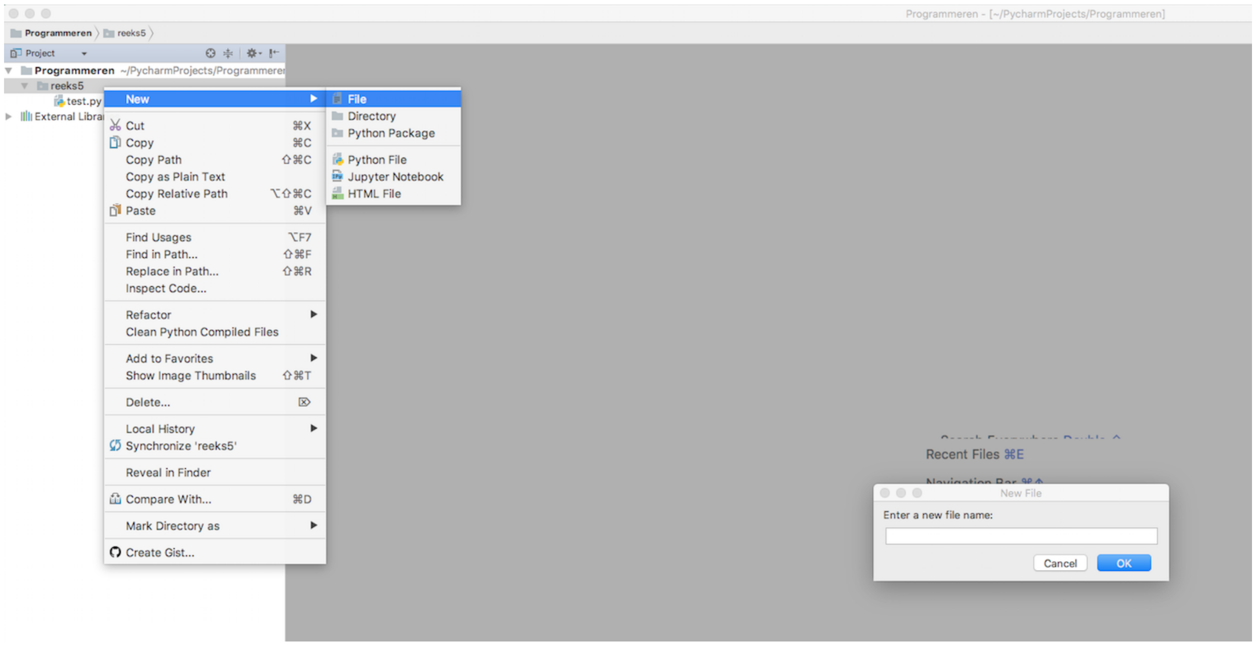

Figure 1: menu new file

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 you submitted.

String representation of a grid

The following *list comprehension* constructs a string representation of a grid, with each row 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 on a row are separated from each other by a single 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
```

Separately process the first line of a text file

When processing text files, it may well happen that the first line of the text file plays a different role than the other lines in the file (e.g. a header line). In such cases, you may want to process the first line of the file separate from the other lines of the file.

Say, for example, that we have the following text file.

```
first
second
third
```

You can read and process the first line of this file separate from the next lines in the file in the following way.

```
>>> infile = open('filename.txt', 'r')
>>> firstLine = infile.readline()
>>> firstLine
"first\n"
>>> otherLines = []
>>> for line in infile:
...     otherLines.append(line)
>>> otherLines
["second\n", "third\n"]
```

It's important to know that Python by default never visits the same line twice. If you need to process the lines of a file multiple times, you may close the file after the first iteration (using the built-in function `close()`) and open it again to start a second iteration (using the built-in function `open()`). As an alternative, you can use the method `seek()` on an opened file object to put the file pointer back at the start of the file (or at any other position in the file).

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 `str.rstrip()` can be used to remove the trailing newline. 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 `str.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.'
```

Output results to a file

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

By default, the function `print()` will write the string representation of the expression to the special file `sys.stdout` (the default value of the parameter `file`) that for example might be attached to the Console window of PyCharm. 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 expression is written to the 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()
''
```

Split on whitespace

The string method `str.split()` can be used to split a string into a list of substrings. In case no argument is passed to the method, the method will remove all leading and trailing whitespace characters (spaces, tabs and newlines), and then split the string into the substrings that are separated from each other by one or more whitespace characters.

In case a string argument is passed to the string method `str.split()`, the method will use this argument to split the string at each occurrence of the argument. Say, for example, that the string contains two consecutive spaces, then the string method `str.split()` without an argument will split once at that position, whereas the string method `str.split()` with a space character as an argument will split twice at that position. In the latter case, an empty string will result as the substring between the two spaces.

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