File handling

BB1000 Programming in Python KTH

Example from Back in the Day

Imagine researcher Miraculix has a student Kaningentix, who has called all his datafiles with a suffix .knx The first task is then to reduce the title to its essentials. Miraculix has to split out the herbs for Asterix and Obelix with suffix .asx and .obx

```
>>> cat allherbs.knx
Creator of file: Kaningentix
First test
Herbs Celts
aster: herb1
obeli: herb2
aster: herb3
aster: herb4
obeli: herb5
```

For reading and writing, the files have to be opened.

```
>>> file="allherbs"
>>> inp = open(file+".knx",'r')
>>> outas = open(file+".asx",'w')
>>> outob = open(file+".obx",'w')
```

The 'r' points at a file aimed only to be read, while 'w' is aimed only for writing.

Example from Back in the Day

As the file is read line per line, it is important to know the total number of the lines (`totnumb`). Since the file has been read up till the end, Miraculix has to start again from the beginning using `inp.seek(0)`.

```
>>> totnumb = 0
>>> for line in inp:
        totnumb = totnumb + 1
>>> inp.seek(0)
```

Each line of `inp` is read in, and its content is analyzed. Remark that the lines are counted, using `number` - and that its value has to be initiated in the beginning.

```
>>> number = 0
>>> line = inp.readline()
>>> number = number + 1
>>> while "Herb" not in line:
>>>       line = inp.readline()
>>>       number = number + 1
```

Example from Back in the Day

As long as the program didn't reach the end of the file (so, as long as the number of read lines is smaller than the total amount of lines in the file), the next line is read.

```
>>> while number < totnumb:
>>>     line = inp.readline()
>>>     number = number + 1
```

Within this loop, the first five characters of the line are examined.

```
>>>     if (line[:5]) == "aster":
>>>           outas.write(line[7:12]+"\n")
>>>     if (line[:5]) == "obeli":
>>>           outob.write(line[7:12]+"\n")
```

Finally, all files are closed. The writing only occurs at the moment of closing.

```
>>> inp.close()
>>> outas.close()
>>> outob.close()
```

Choice of input file

Imagine Kaningentix didn't have only the file allherbs.knx, but also alldiseases.knx and allforces.knx. Miraculix likes to have a program in which he can decide upon the file to be decomposed.

```
>>> file = input("Which inputfile? ")
Which inputfile?
```

Miraculix can write then `allherbs` to get the `allherbs.knx` file analyzed.

Remark: In `python2`, `raw_input()` delivers a string, while `input()` considers the input as an expression. In `python3`, `raw_input()` does not exist and `input()` takes over the meaning of the old `raw_input()`. When therefore the input has to considered as an expression, `eval(input())` has to be used.

Choice of input file

Imagine now that Miraculix gives a name of a file which does not exist. The program breaks. Therefore, a test can be built in.

```
>>> try:
>>>   inp= open(file+".knx",'r')
>>> except:
>>>   print ('File cannot be opened:', file+".knx")
>>>   exit()
```

When a file is given in which does not exist, simply message is given out and the program terminates.

```
Which inputfile? somethingelse
File cannot be opened: somethingelse.knx
```

Common part in the name

The only part which is important is the one in front of the ".". Remark that the name of the file is like an array and that only from element "0" until (but not including) the "." is retained.

```
>>> if "." in file:
>>>     file = file[:file.find(".")]
>>> file
'allherbs'
```

Reading until space

In the program above, the input file is read based upon the number of characters - which also meant that Asterix was abbreviated to `aster` and Obelix to `obeli`. It would have been better to read until the first space.

```
>>> while number < totnumb:
>>>     line = inp.readline()
>>>     number = number + 1
>>>     spacepos = line.find(' ')
>>>     if (line[:spacepos-1]) == "aster":
>>>           outas.write(line[spacepos+1:12]+"\n")
>>>     if (line[:spacepos-1]) == "obeli":
>>>           outob.write(line[spacepos+1:12]+"\n")
```

Reading until end of line

Still it makes not much sense to use the program above, since the herbs should have names which have all the exact same length. In the example of Miraculix, each line (and therefore the name of the herb) ends on another space or on a hard enter.

```
>>>      if (line[:spacepos-1]) == "aster":
>>>            outas.write(line[spacepos+1:])
>>>      if (line[:spacepos-1]) == "obeli":
>>>            outob.write(line[spacepos+1:])
```

Example today

In this way, the input could be

```
>>> cat allherbs_2.knx
Creator of file: Kaningentix
First test
Herbs Celts
asterix: petraoleum
obelix: kerosine
asterix: cyanine
asterix: diesel
obelix: lpg
```

... and Miraculix will get ...

```
>>> cat allherbs_2.asx
petraoleum
cyanine
diesel
```

```
>>> cat allherbs_2.obx
kerosine
lpg
```

Some other type of printing

We know already

```
>>> file="obelix_dog"
>>> inp= open(file,'w')
>>> inp.write("Idefix")
>>> inp.close()
```

It is also possible to say

```
>>> inp= open(file,'w')
>>> print("Idefix",file=inp)
>>> inp.close()
```

The sys-module and some easy way of input

`argv` contains the vector of arguments passed to a program via the command line. `argv[0]` is the filename of the program, `argv[1]` is then the first argument given, `argv[2]` the second etc. `argv` has to be imported from the module `sys`.

Imagine that a program `program_test.py` contains two lines of code:

```
from sys import argv
print(argv)
```

The output of `python program_test.py` is then

```
['program_test.py', 'a', 'b']
```

argv[0] contains then the name `program_test.py`, argv[1] and argv[2] contain `a` and `b`, respectively.

The sys-module and some safe way of input

Miraculix made a program `healthy_herb.py` which says whether or not a herb is safe to be eaten or not. As input parameters, the program needs to know the color of the herb as well as its sites in the forest:

`python healthy_herb.py green river`

To make sure that his collaborators provide these from the beginning, Miraculix hard coded a small test:

```
from sys import argv
if len(argv) != 3 :
   print ("Usage: python healthy_herb.py color place")
   exit()
```

Of course, in the code, the arguments still have to be defined:

```
color = argv[1]
place = argv[2]
```

References

"Python for Data Analysis", Wes McKinney, O'Reilly Media, Sebastopol, CA: 2013

"Python for Informatics", Charles Severance, 2013, http://www.pythonlearn.com/book.php#python-for-informatics

http://www.pythonforbeginners.com