Standard input and standard output and Python

Gertjan van Noord

Recall: stdin, stdout, stderr

Unix commands typically take some input, and provide output.

If you don’t give any particular details, the input is taken from the keyboard, and the output is sent to the screen

standard input: by default, the keyboard

standard output: by default, the screen

Example

```
$ wc
```

“nothing happens”?

the command is waiting for your input . . .

```
$ wc myfile.txt
  630   2493 17550 myfile.txt

$ cat myfile.txt | wc
  630   2493 17550
```

(how to specify that input from keyboard is finished? control-d)

Unix pipe

Combine commands: take output of a command, and use it as input for the next command

```
commando1 | commando2
```

```
$ head myfile.txt | wc
     10      17     264
```

Example

Suppose you have a word list: wlist.txt. In that file, every word is on a single line

```
in
verband
met
de
gemiddeld
langere
levensduur
van
de
vrouw
```

Voorbeeld

Make an ordered, unique word list:

```
$ cat wlist.txt | sort | uniq
a
a-cultureel
a-morele
a-politieke
a.
a.d.
a.h.w.
a.r.
a.r.-fractie
a.s.
a.v.
a/b
a/d
aaien
...
```

Example

Make a dictionary for poets:

```
$ cat wlist.txt | rev | sort | uniq | rev
a
alba
ca
cachaca
yucca
logica
meta-logica
metalogica
propositielogica
kwasilogica
predikatenlogica
physico-theologica
beroepsethica
psychedelica
basilica
```

So . . .

- standard input, standard output, pipes
- every program can be used interactively, but also as part of a longer sequence of commands
- if your program does something useful, you want to be able to re-use it later as part of something else
- therefore: use standard input and standard output

Input and Output in Python3

- input() uses standard input
- print() uses standard output
- treat input line by line using sys.stdin
  - can be used interactively
  - can be used with input from file
  - can be used with input from pipe (i.e., the output of another command)

Example: add line length, add_length.py

```
import sys
nr=1
for line in sys.stdin:
    nr = len(line)
    print(nr,"\t",line.rstrip())
```

rstrip(): removes all whitespace characters at the end of the string, including end-of-line

Use this as part of a “pipe”

Voorbeeld

```
$ cat wlist.txt | rev | sort | uniq | rev | python3 nrs.py
1        a
4        alba
2        ca
7        cachaca
5        yucca
6        logica
11       meta-logica
...
```

Extra information, debug messages, continuation messages

If your program also prints stuff such as welcoming messages, debug information, continuation messages, then it will also be part of the input for the next command . . .

```
echo "met de" | python3 query.py | wc -l
```

Solution: there also is *standard error*

```
print("Er zijn geen documenten gevonden",file=sys.stderr)
```

Just like stdout, stderr normally goes to the screen. If you use a pipe, stdout is sent to the next command, but stderr will (normally) still go to the screen.

Redirect

```
$ command < infile.txt
$ command > myoutfile.txt
$ command 2> err.txt

$ command > myout.txt 2> myerr.txt

$ command <( pipe1 )  <( pipe2 ) <( pipe3 )
```

Interactive input

In some cases, using sys.stdin() as in the examples above is somewhat counter-intuitive for interactive scenarios. In interactive mode, you want to prompt the users so that they know what is expected. In Python, the use of input() is possibly a good idea *in such cases*, as in the following typical example.

```
while True:
    line = input("Type your search term here (q to quit): "
    if line == "q":
        break
    search(line))
```

Interactive input, allow for end-of-file

In the example above, you will get an end-of-file error if there is no further input. The following - preferred! - example catches this situation:

```
while True:
    try:
        line = input("Type your search term here (q to quit
        if line == "q":
            break
        search(line)
    except EOFError:
        print()
        break
```