Ghent University
Faculty of Sciences

Python Programming

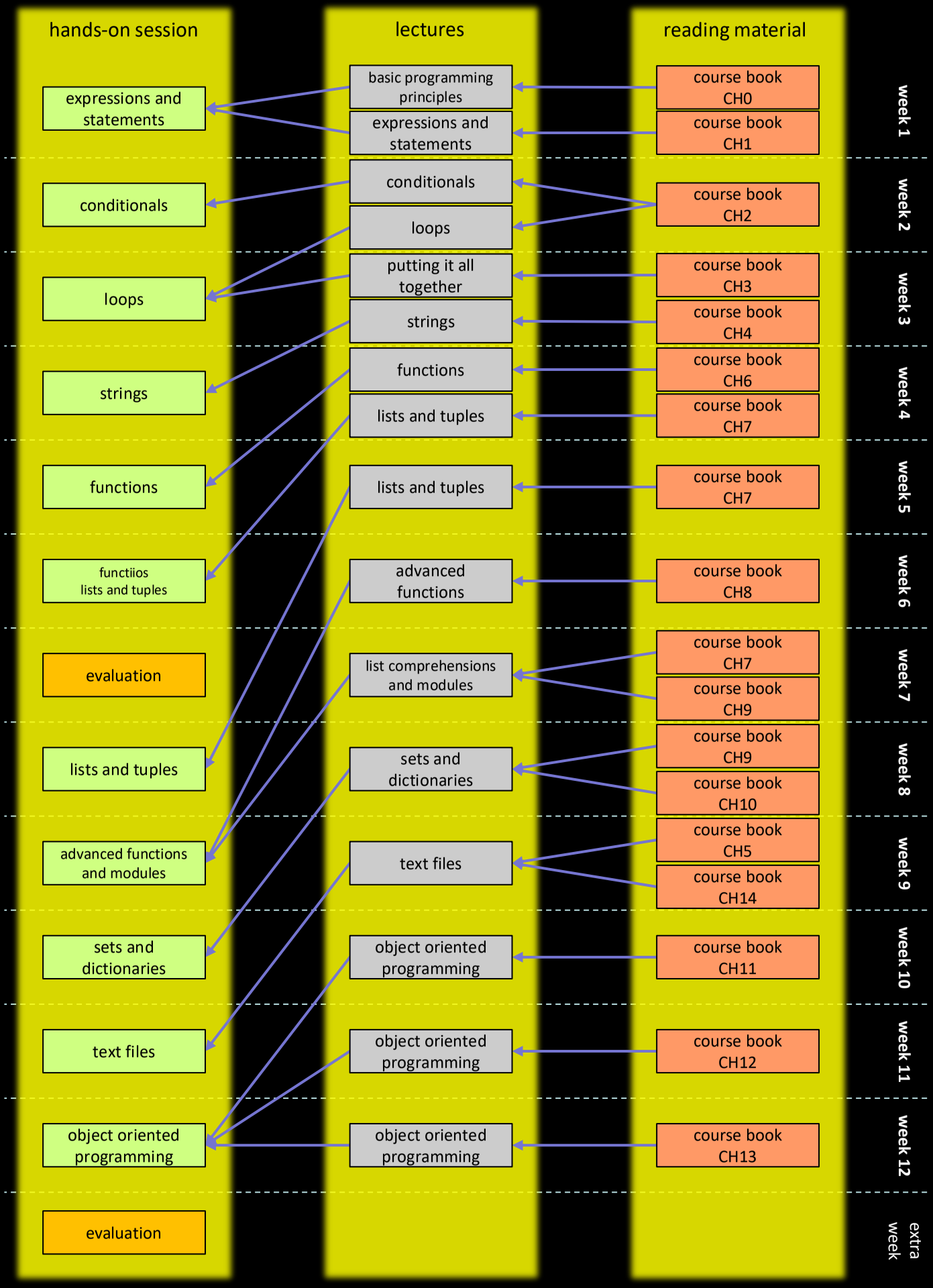
great Python
your appetite
is hard to swallow

Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
text files
object oriented programming
evaluation
basic programming principles
expressions and statements
conditionals
loops
putting it all together
strings
functions
lists and tuples
lists and tuples
advanced functions
list comprehensions and modules
sets and dictionaries
text files
object oriented programming
object oriented programming
object oriented programming
course book CH0
course book CH1
course book CH2
course book CH3
course book CH4
course book CH6
course book CH7
course book CH7
course book CH8
course book CH7
course book CH9
course book CH9
course book CH10
course book CH5
course book CH14
course book CH11
course book CH12
course book CH13

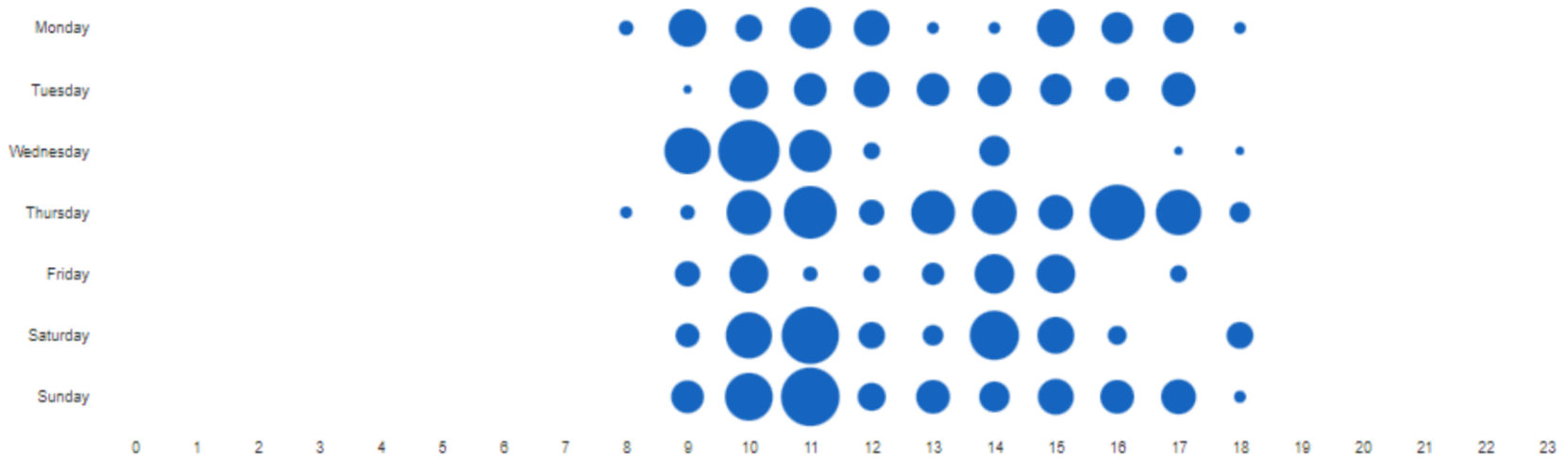
Hermione Granger
Hogwarts School of Witchcraft and Wizardry

Statistics

1057	61	6
Submissions	Correct exercises	Unfinished exercises

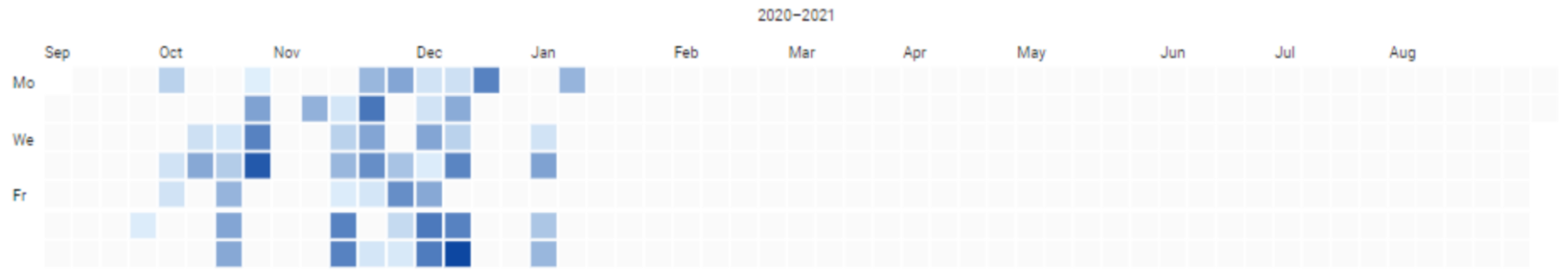
Punchcard

This graph shows you on which moment in the week submissions have taken place.



Heatmap

This graph shows you on which days most submissions took place.



Ron Weasley
Hogwarts School of Witchcraft and Wizardry

Statistics

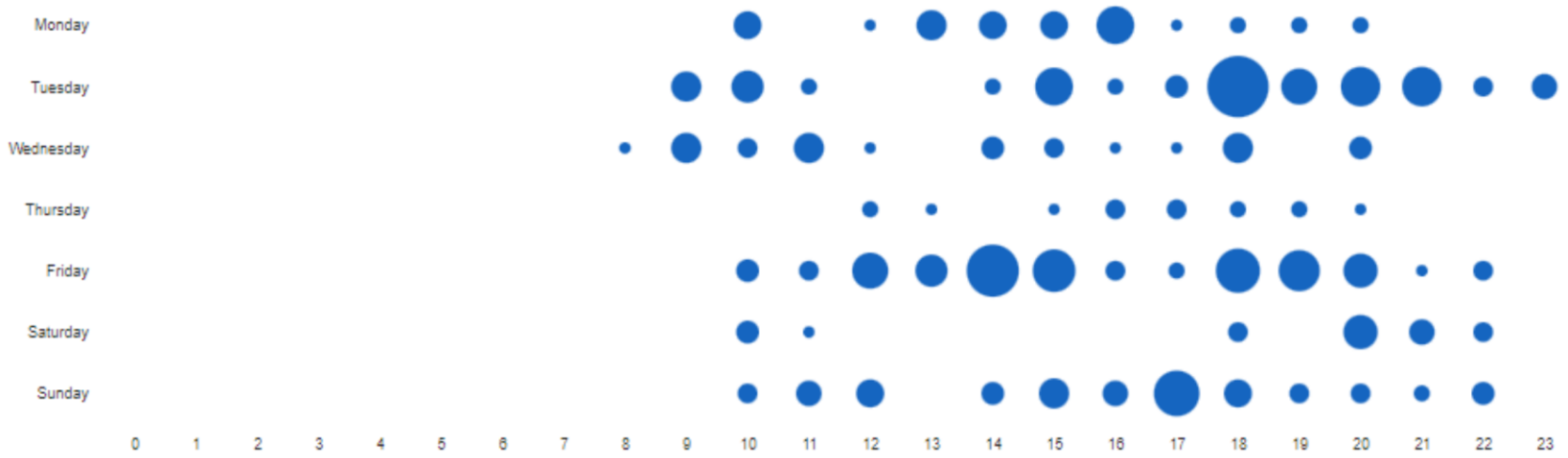
400
Submissions

62
Correct exercises

10
Unfinished exercises

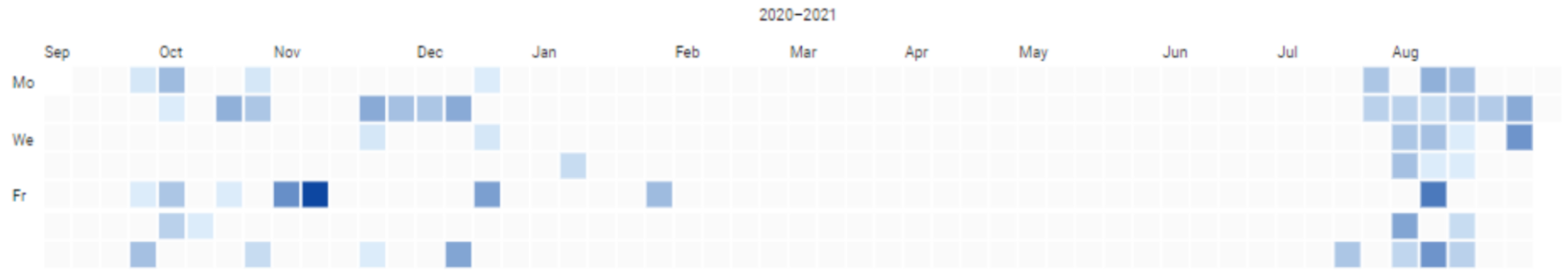
Punchcard

This graph shows you on which moment in the week submissions have taken place.

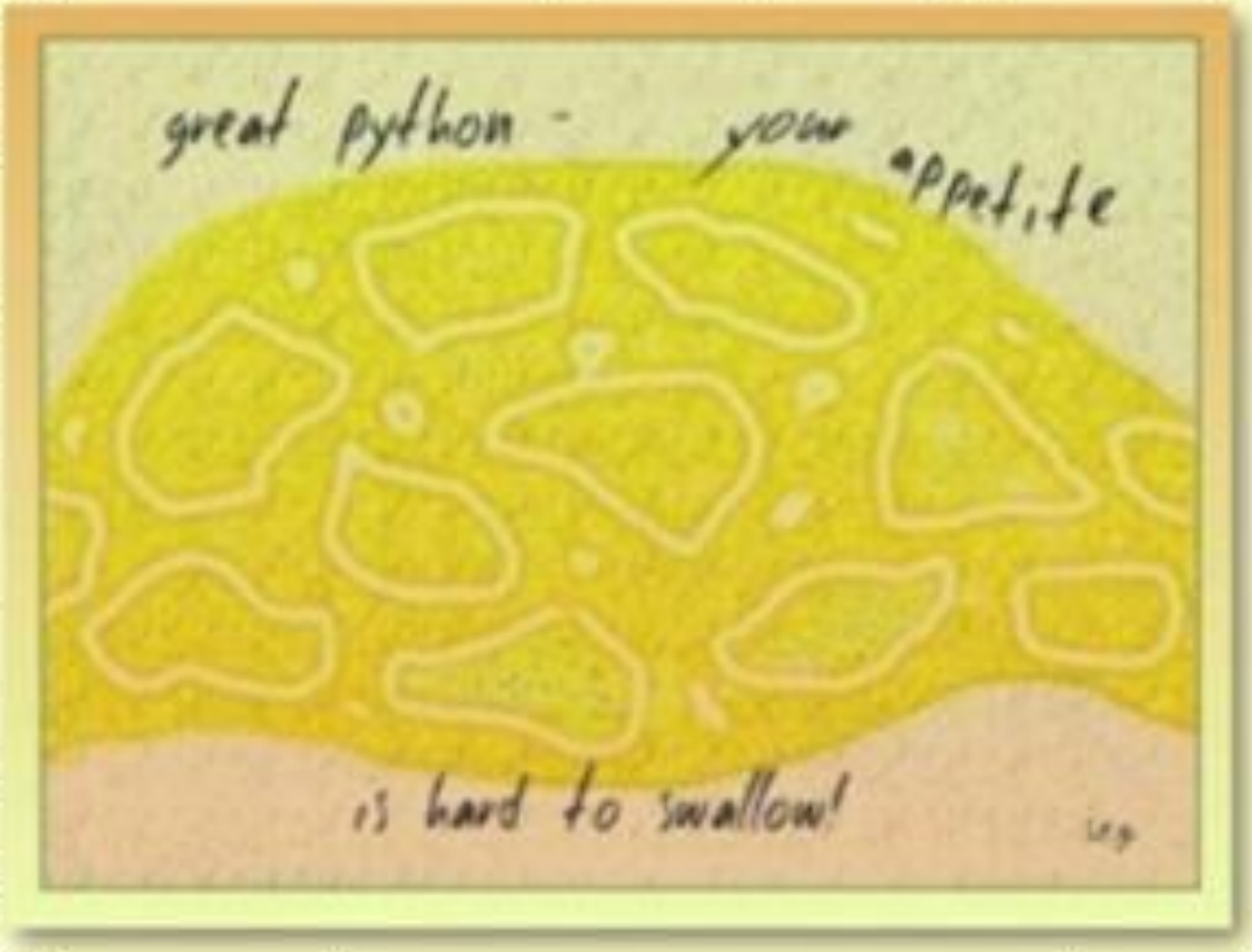


Heatmap

This graph shows you on which days most submissions took place.



Input and output



Input and output

- up until now …
  - display what programs are doing: **`print()`** function
  - request information from users: **`input()`** function

"How can I save my data into a file ?"

"How can I read data from a file ?"

- Python's solution looks very much like C's
  - a file is a sequence of bytes
  - but it is often more useful to treat text files as a sequence of lines

UNIVERSITEIT GENT

Read from files

- how many ~~characters~~ bytes are in this file ?

haiku.txt

```
Three things are certain:
Death, taxes and lost data.
Guess which has occurred.

You step in the stream,
but the water has moved on.
This page is not here.

Having been erased,
The document you're seeking
Must now be retyped.

A crash reduces
your expensive computer
to a simple stone.
```

assume for now:
1 character = 1 byte

UNIVERSITEIT GENT

source: Error messages in Haiku? (GNU humor collection)

Read from files

- how many characters are in this file ?

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

create a file object

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

location of the file to connect to

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

for reading ...

Read from files

- how many characters are in this file ?

now refers to the file object

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

reads entire content of file into a string

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

now has a copy of all the bytes that were in the file

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
>>> print(data)
Three things are certain:
Death, taxes and lost data.
Guess which has occurred.
...
```

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
>>> print(data)
Three things are certain:
Death, taxes and lost data.
Guess which has occurred.
...
```

disconnect object from the file

UNIVERSITEIT GENT

Read from files

- how many characters are in this file ?

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
>>> print(data)
Three things are certain:
Death, taxes and lost data.
Guess which has occurred.
...
```

report how many bytes ~~characters~~ were read

UNIVERSITEIT GENT

Read from files

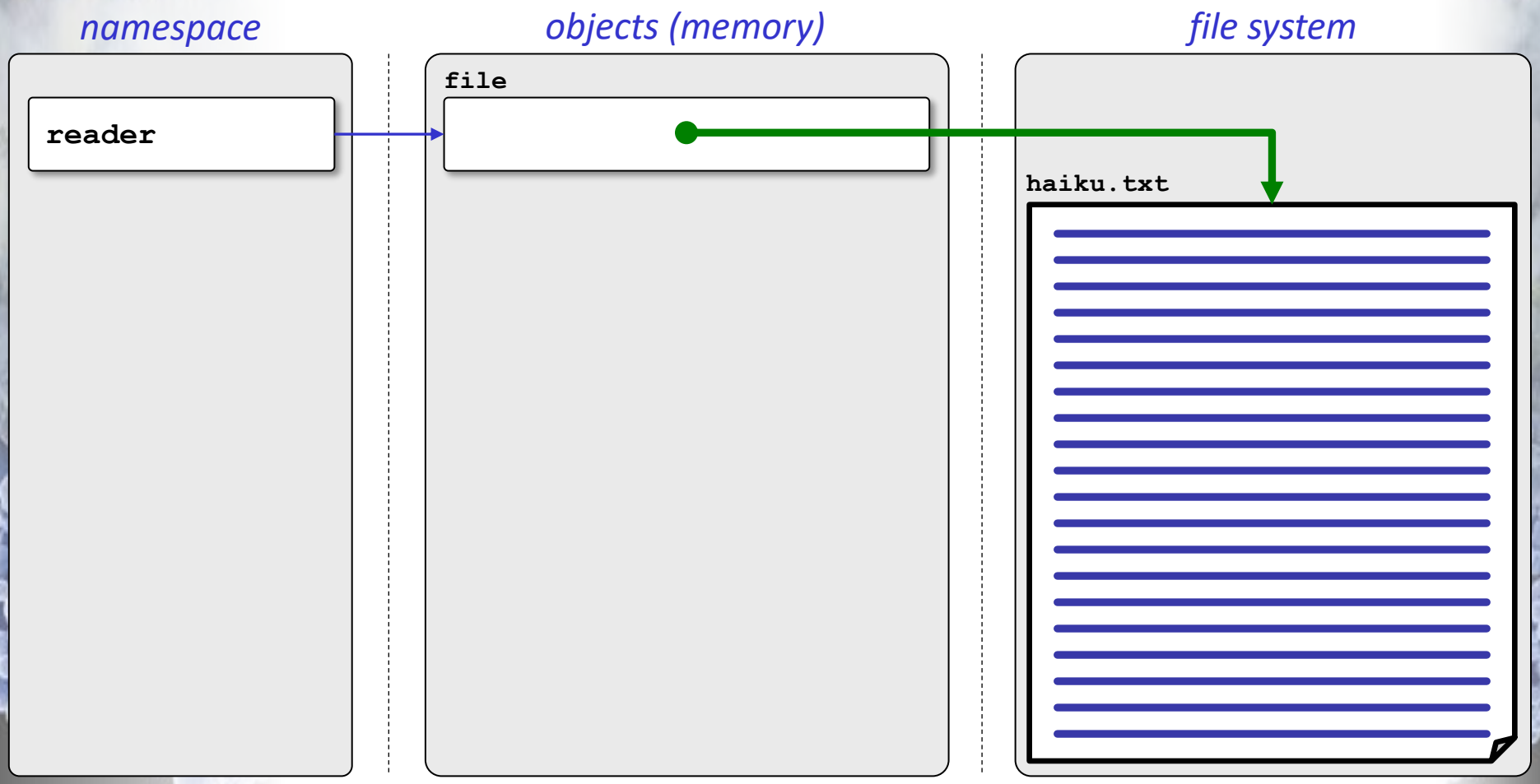
- how many characters are in this file ?

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read()
>>> reader.close()
>>> len(data)
286
>>> print(data)
Three things are certain:
Death, taxes and lost data.
Guess which has occurred.
...
```

UNIVERSITEIT GENT

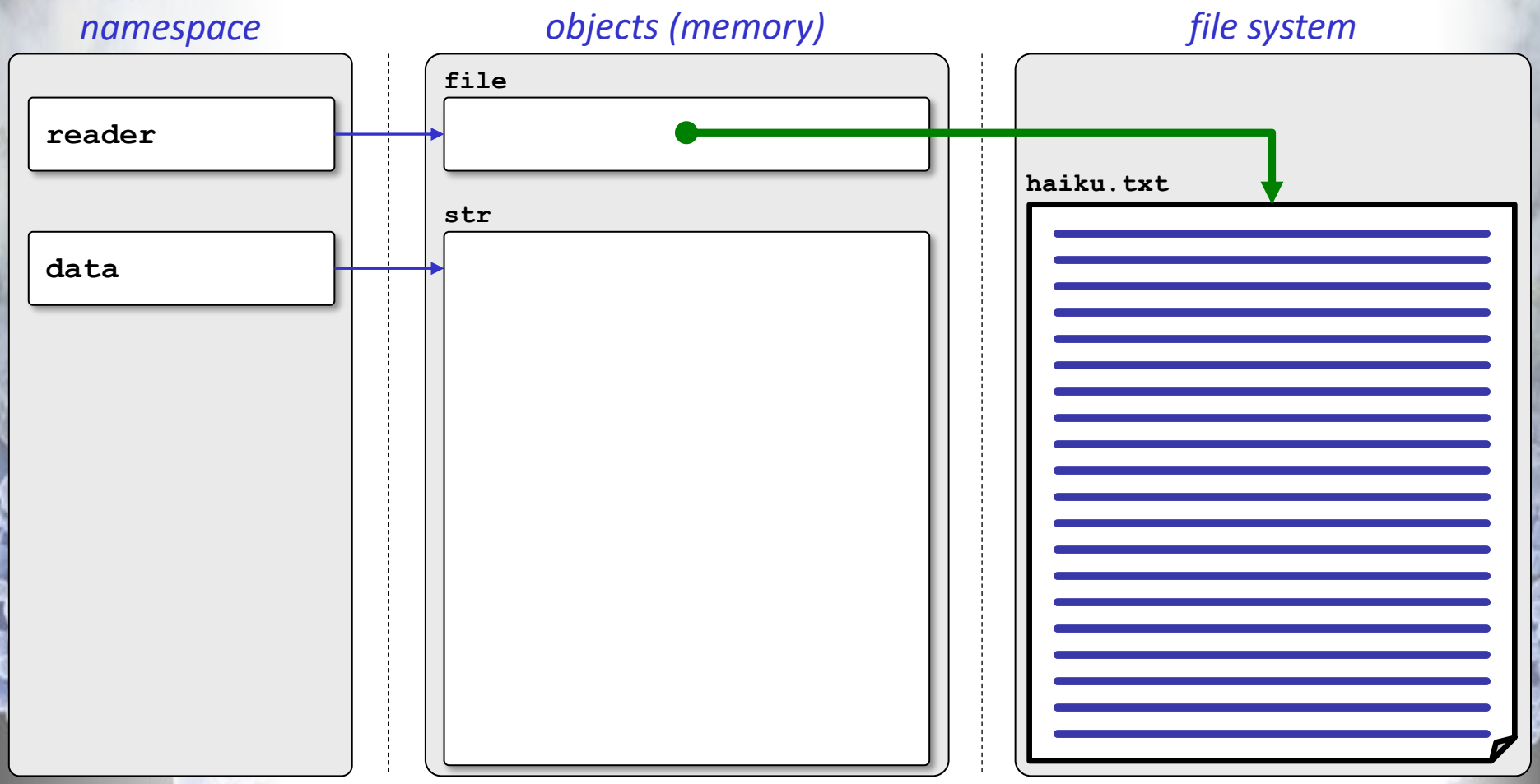
Read all at once …

```
>>> reader = open('haiku.txt', 'r')
```



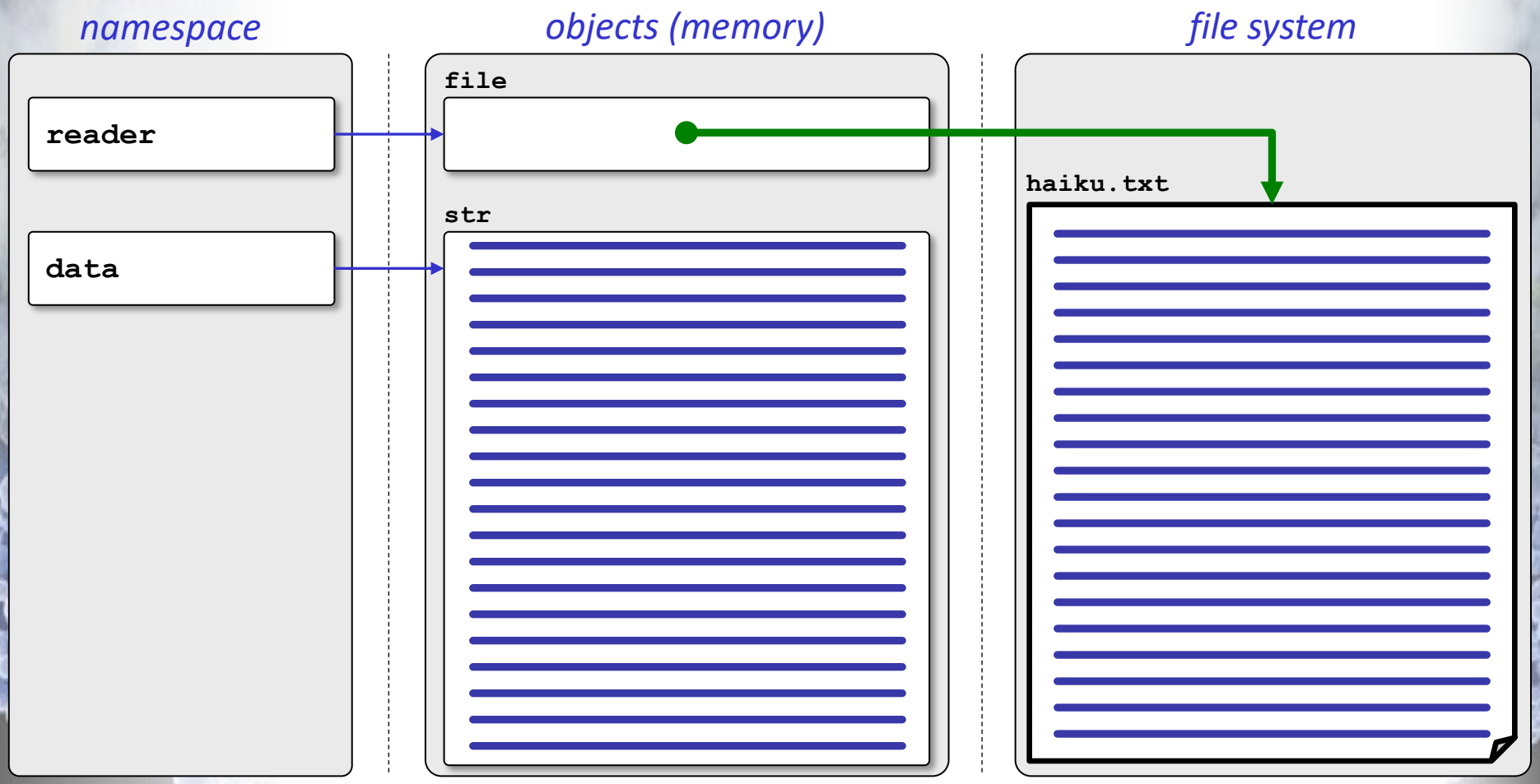
Read all at once …

```
>>> data = reader.read()
```



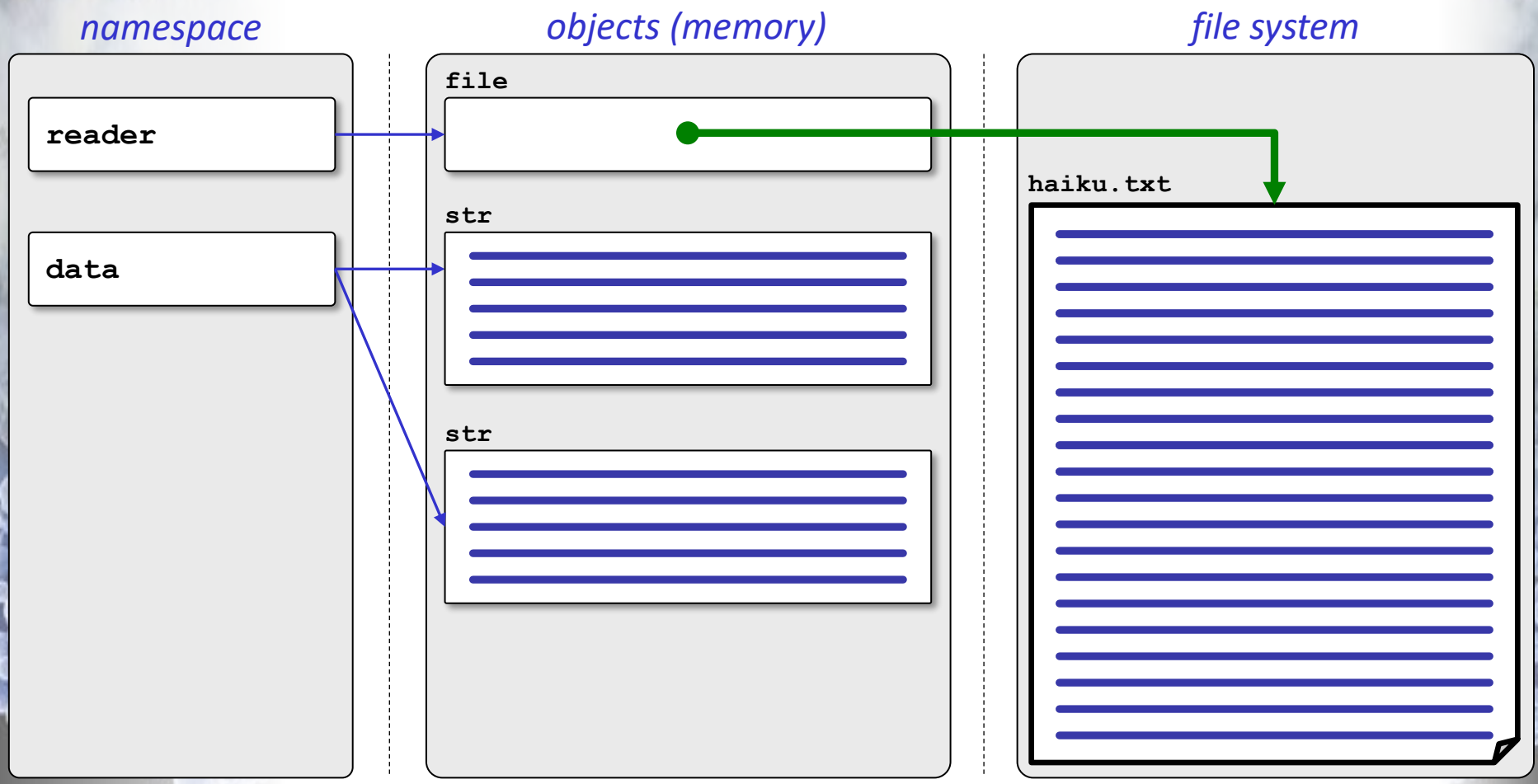
Read all at once …

```
>>> reader.close()
```



… or in chunks …

- if file grows large, it is better to read in chunks



… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
```

UNIVERSITEIT GENT

… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
```

… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
```

repeat as long as something is read from file

UNIVERSITEIT GENT

… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
64
```

do something with the data

UNIVERSITEIT GENT

… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
64
64
64
64
30
```

try to read the next chunk of data

UNIVERSITEIT GENT

… or in chunks …

- if file grows large, it is better to read in chunks

```
>>> reader = open('haiku.txt', 'r')
>>> data = reader.read(64)
>>> while data:
...     print(len(data))
...     data = reader.read(64)
64
64
64
64
30
>>> len(data)
0
>>> reader.close()
```

quite unusual scenario when working with text files

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

read a single line

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

repeat as long as lines are read

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

(try to) read next line

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

```
average: 19.066667
```

UNIVERSITEIT GENT

… or line by line

- it is more common to read text files one line at a time

reader3.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
line = reader.readline()
while line:
    lines += 1
    bytes += len(line)
    line = reader.readline()
reader.close()
print(f'average: {bytes / lines}')
```

```
average: 19.066667
```

UNIVERSITEIT GENT

List of lines

- often more convenient to read all lines at once
  - if memory footprint is not an issue

reader4.py

```
reader = open('haiku.txt', 'r')
content = reader.readlines()
reader.close()
bytes, lines = 0, 0
for line in content:
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

List of lines

- often more convenient to read all lines at once
  - if memory footprint is not an issue

reader4.py

```
reader = open('haiku.txt', 'r')
content = reader.readlines()
reader.close()
bytes, lines = 0, 0
for line in content:
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

all lines in file as list of strings

UNIVERSITEIT GENT

List of lines

- often more convenient to read all lines at once
  - if memory footprint is not an issue

reader4.py

```
reader = open('haiku.txt', 'r')
content = reader.readlines()
reader.close()
bytes, lines = 0, 0
for line in content:
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

loop over lines with for loop

UNIVERSITEIT GENT

List of lines

- often more convenient to read all lines at once
  - if memory footprint is not an issue

reader4.py

```
reader = open('haiku.txt', 'r')
content = reader.readlines()
reader.close()
bytes, lines = 0, 0
for line in content:
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

```
average: 19.066667
```

UNIVERSITEIT GENT

List of lines

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
    - file object is an *iterable object*

reader4.py

```
reader = open('haiku.txt', 'r')
content = reader.readlines()
reader.close()
bytes, lines = 0, 0
for line in content:
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

```
average: 19.066667
```

UNIVERSITEIT GENT

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
    - file object is an *iterable object*

reader5.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
for line in reader:
    lines += 1
    bytes += len(line)
reader.close()
print(f'average: {bytes / lines}')
```

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
    - file object is an *iterable object*

reader5.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
for line in reader:
    lines += 1
    bytes += len(line)
reader.close()
print(f'average: {bytes / lines}')
```

assign lines of text in file to the loop variable one by one

UNIVERSITEIT GENT

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
  - "open" file objects are automatically closed "at the end"

reader5.py

```
reader = open('haiku.txt', 'r')
bytes, lines = 0, 0
for line in reader:
    lines += 1
    bytes += len(line)
reader.close()
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
  - "open" file objects are automatically closed "at the end"

reader6.py

```
bytes, lines = 0, 0
for line in open('haiku.txt', 'r'):
    lines += 1
    bytes += len(line)
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
  - "open" file objects are automatically closed "at the end"
  - **`with`**-block makes the "end" explicit (= close file)

reader7.py

```
bytes, lines = 0, 0
with open('haiku.txt', 'r') as reader:
    for line in reader:
        lines += 1
        bytes += len(line)
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

Iterate files

- often more convenient to read all lines at once
  - if memory footprint is not an issue
  - common idiom: "read lines as list" + "loop over list"
  - short Python syntax: "loop over lines in file"
  - "open" file objects are automatically closed "at the end"
  - **`with`**-block makes the "end" explicit (= close file)
  - **`encoding`** cares about converting bytes into characters

reader8.py

```
bytes, lines = 0, 0
with open('haiku.txt', encoding='utf-8') as reader:
    for line in reader:
        lines += 1
        bytes += len(line)
print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

Guaranteed file closure

- open file → don't forget to close file

reader9.py

```
reader = open('haiku.txt', 'r')
try:
    bytes, lines = 0, 0
    for line in reader:
        lines += 1
        bytes += len(line)
    print(f'average: {bytes / lines}')
finally:
    reader.close()
```

UNIVERSITEIT GENT

Guaranteed file closure

- open file → don't forget to close file
  - **`with`**: *resource management + exception handling*

reader10.py

```
with open('haiku.txt', 'r') as reader:
    bytes, lines = 0, 0
    for line in reader:
        lines += 1
        bytes += len(line)
    print(f'average: {bytes / lines}')
```

UNIVERSITEIT GENT

The End of Line Puzzle

俳

The End of Line Puzzle

000	NUL	033	!	066	B	099	c	132	ä	165	Ñ	198	ã	231	þ
001	Start Of Header (SOH)	034	"	067	C	100	d	133	à	166	ª	199	Ã	232	Þ
002	Start Of Text (STX)	035	#	068	D	101	e	134	å	167	º	200	╚	233	Ú
003	End Of Text (ETX)	036	$	069	E	102	f	135	ç	168	¿	201	╔	234	Û
004	End Of Transmission (EOT)	037	%	070	F	103	g	136	ê	169	®	202	╩	235	Ù
005	Enquiry	038	&	071	G	104	h	137	ë	170	¬	203	╦	236	ý
006	Acknowledge (ACK)	039	'	072	H	105	i	138	è	171	½	204	╠	237	Ý
007	Bell	040	(	073	I	106	j	139	ï	172	¼	205	═	238	¯
008	Backspace (BS)	041	)	074	J	107	k	140	î	173	¡	206	╬	239	´
009	Horizontal Tab	042	*	075	K	108	l	141	ì	174	«	207	¤	240	-
010	Line Feed **(LF)**	043	+	076	L	109	m	142	Ä	175	»	208	ð	241	±
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	░	209	Ð	242	‗
012	Form Feed **(FF)**	045	-	078	N	111	o	144	É	177	▒	210	Ê	243	¾
013	Carriage Return **(CR)**	046	.	079	O	112	p	145	æ	178	▓	211	Ë	244	¶
014	Shift Out	047	/	080	P	113	q	146	Æ	179	│	212	È	245	§
015	Shift In	048	0	081	Q	114	r	147	ô	180	┤	213	ı	246	÷
016	Dataline Escape (DLE)	049	1	082	R	115	s	148	ö	181	Á	214	Í	247	¸
017	DC 1 (XON)	050	2	083	S	116	t	149	ò	182	Â	215	Î	248	°
018	DC 2	051	3	084	T	117	u	150	û	183	À	216	Ï	249	¨
019	DC 3 (XOFF)	052	4	085	U	118	v	151	ù	184	©	217	┘	250	·
020	DC 4	053	5	086	V	119	w	152	ÿ	185	╣	218	┌	251	¹
021	Negative Acknowledge (NAK)	054	6	087	W	120	x	153	Ö	186	║	219	█	252	³
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	╗	220	▄	253	²
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	╝	221	¦	254	■
024	Cancel	057	9	090	Z	123	{	156	£	189	¢	222	Ì	255	
025	End Of Medium	058	:	091	[	124	|	157	Ø	190	¥	223	▀		
026	Substitude	059	;	092	\	125	}	158	×	191	┐	224	Ó		
027	Escape **(ESC)**	060	<	093	]	126	~	159	ƒ	192	└	225	ß		
028	File Seperator	061	=	094	^	127 **(DEL)**	⌂	160	á	193	┴	226	Ô		
029	Group Seperator	062	>	095	_	128	Ç	161	í	194	┬	227	Ò		
030	Record Seperator	063	?	096	`	129	ü	162	ó	195	├	228	õ		
031	Unit Seperator	064	@	097	a	130	é	163	ú	196	─	229	Õ		
032	*SPACE* (SP)	065	A	098	b	131	â	164	ñ	197	┼	230	µ		

UNIVERSITEIT GENT

The End of Line Puzzle

- UNIX and MS Windows use different conventions to represent the end of line in text files
  - UNIX: line feed (`'\n'`, newline, code 10)
  - MS Windows: carriage return (`'\r'`, code 13) + line feed

UNIVERSITEIT GENT

The End of Line Puzzle

Write to files

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

same function

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

existing created if
will be added to written

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

for writing ...

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

write a single string

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files using their
  - **`write()`** method
  - **`writelines()`** method

write each string in the list

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files
  - Python only writes what we tell it to
    - end-of-line characters must be written explicitely: `'\n'`
    - and we did not tell it to write any end-of-line characters

elements.txt

```
elementsHeNeArKr
```

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements')
>>> writer.writelines(['He', 'Ne', 'Ar', 'Kr'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files
  - Python only writes what we tell it to
    - end-of-line characters must be written explicitely: `'\n'`

'elements\nHe\nNe\nAr\nKr\n'

elements.txt

```
elements
He
Ne
Ar
Kr
```

```
>>> writer = open('elements.txt', 'w')
>>> writer.write('elements\n')
>>> writer.writelines(['He\n','Ne\n','Ar\n','Kr\n'])
>>> writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files
  - Python only writes what we tell it to
    - end-of-line characters must be written explicitely: **`'\n'`**
    - often easier to use **`file`** parameter of **`print`** function

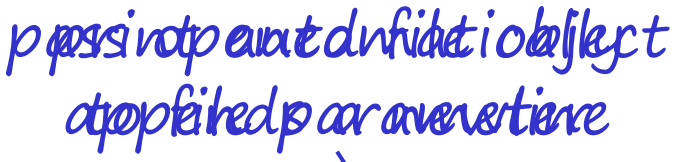
writer3.py

```
writer = open('elements.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files
  - Python only writes what we tell it to
    - end-of-line characters must be written explicitely: **`'\n'`**
    - often easier to use **`file`** parameter of **`print`** function



writer3.py

```
writer = open('elements.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

UNIVERSITEIT GENT

Write to files

- file objects can write data to files
  - Python only writes what we tell it to
    - end-of-line characters must be written explicitely: **`'\n'`**
    - often easier to use **`file`** parameter of **`print`** function

writer3.py

```
writer = open('elements.txt', 'w')
print('elements', file=writer)
for gas in ['He', 'Ne', 'Ar', 'Kr']:
    print(gas, file=writer)
writer.close()
```

UNIVERSITEIT GENT

Copy files

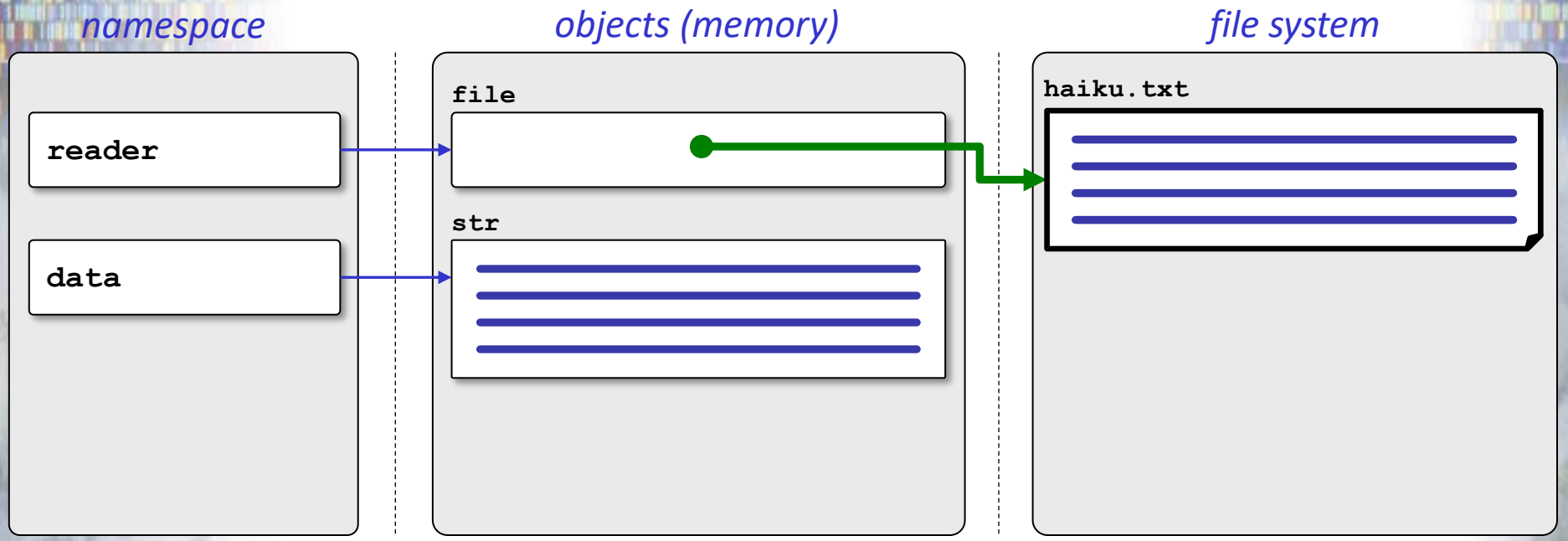
Copy files

copy1.py

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('copy.txt', 'w')
writer.write(data)
writer.close()
```

UNIVERSITEIT GENT

Copy files



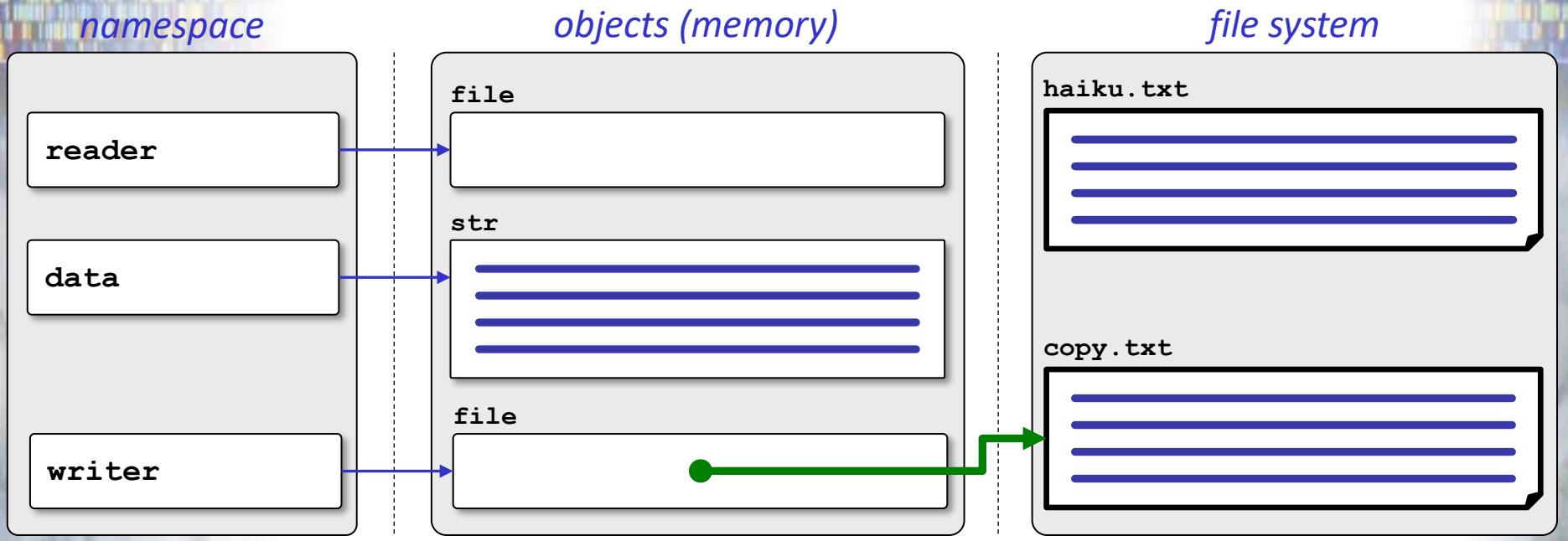
copy1.py

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('copy.txt', 'w')
writer.write(data)
writer.close()
```

read entire file in memory

UNIVERSITEIT GENT

Copy files



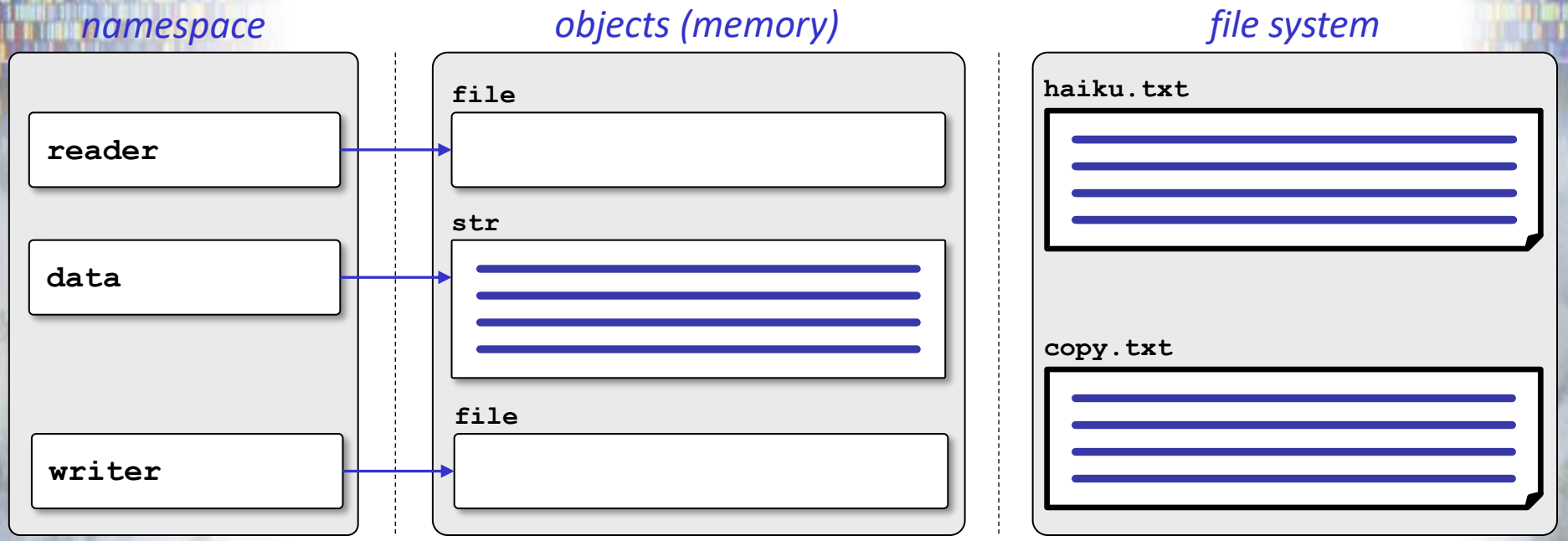
copy1.py

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('copy.txt', 'w')
writer.write(data)
writer.close()
```

dump memory buffer to file

UNIVERSITEIT GENT

Copy files



copy1.py

```
reader = open('haiku.txt', 'r')
data = reader.read()
reader.close()
writer = open('copy.txt', 'w')
writer.write(data)
writer.close()
```

~~terabyte files~~

UNIVERSITEIT GENT

Copy files

- this version will work terabyte-sized files
  - if it's a terabyte of text

copy2.py

```
reader = open('haiku.txt', 'r')
writer = open('copy.txt', 'w')
for line in reader:
    writer.write(line)
reader.close()
writer.close()
```

~~binary files~~

UNIVERSITEIT GENT

Copy files

- this version doesn't make an exact copy of the original file
  - Python keeps newline characters when reading input
  - **`print`** automatically adds a newline when writing output

copy3.py

```
reader = open('haiku.txt', 'r')
writer = open('copy.txt', 'w')
for line in reader:
    print(line, file=writer)
reader.close()
writer.close()
```

double-spaced output lines

UNIVERSITEIT GENT

Copy files

- this version doesn't make an exact copy of the original file
  - Python keeps newline characters when reading input
    - possible solution: string method **`rstrip()`**
  - **`print`** automatically adds a newline when writing output

copy3.py

```
reader = open('haiku.txt', 'r')
writer = open('copy.txt', 'w')
for line in reader:
    print(line.rstrip(), file=writer)
reader.close()
writer.close()
```

UNIVERSITEIT GENT

Copy files

- this version doesn't make an exact copy of the original file
  - Python keeps newline characters when reading input
    - possible solution: string method **`rstrip()`**
  - **`print`** automatically adds a newline when writing output
    - possible solution: parameter **`end`** of **`print`** function

copy3.py

```
reader = open('haiku.txt', 'r')
writer = open('copy.txt', 'w')
for line in reader:
    print(line, file=writer, end='')
reader.close()
writer.close()
```

UNIVERSITEIT GENT

Copy files

- this version works for
  - files of any size
  - binary files and text files

copy4.py

```
blocksize = 1024
reader = open('haiku.txt', 'r')
writer = open('copy.txt', 'w')
data = reader.read(blocksize)
while data:
    writer.write(data)
    data = reader.read(blocksize)
reader.close()
writer.close()
```

UNIVERSITEIT GENT

Copy files

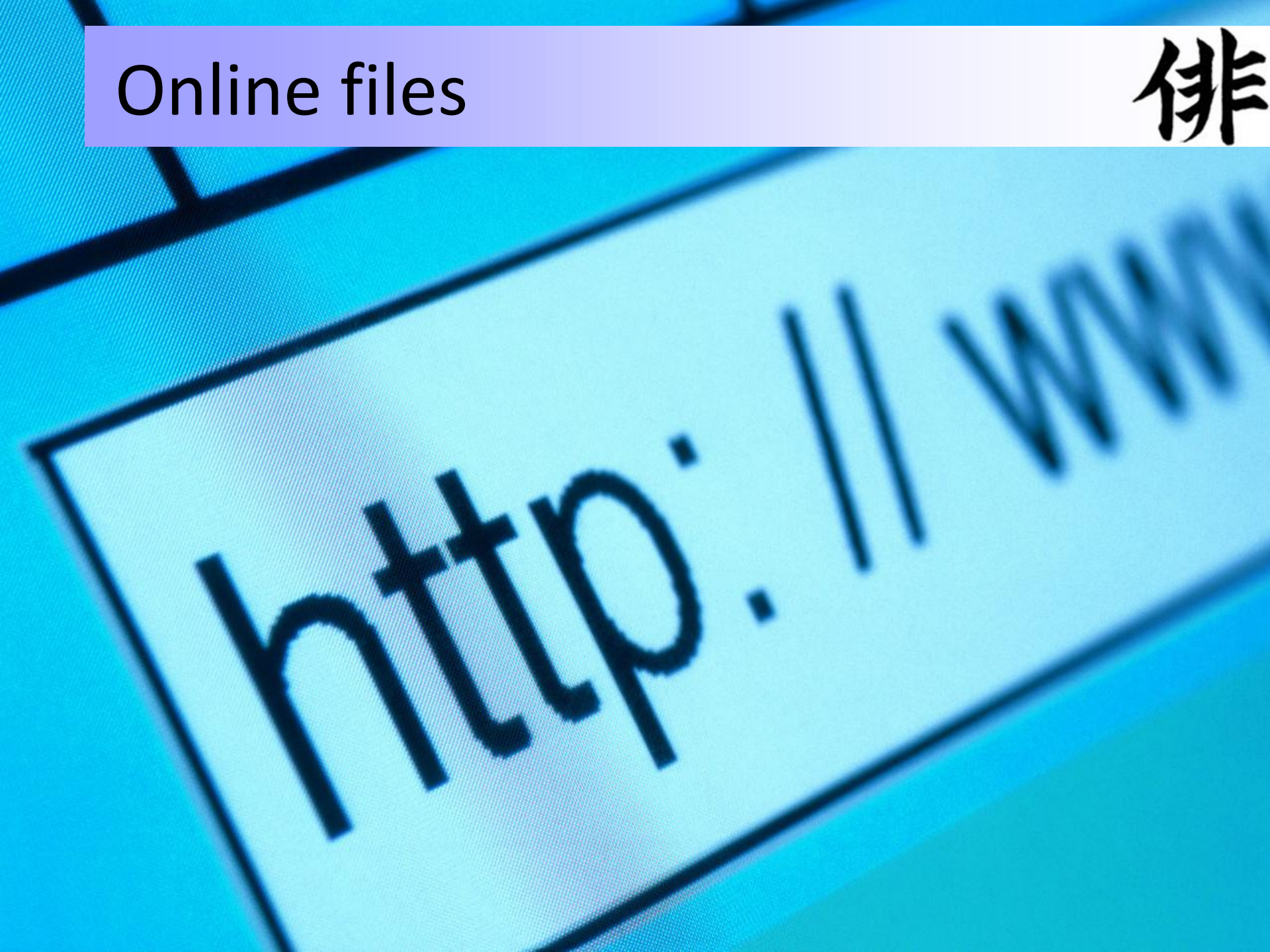
- this version works for
  - files of any size
  - binary files and text files

copy5.py

```
blocksize = 1024
with open('haiku.txt', 'r') as reader:
    with open('copy.txt', 'w') as writer:
        data = reader.read(blocksize)
        while data:
            writer.write(data)
            data = reader.read(blocksize)
```

UNIVERSITEIT GENT

Online files



Online files

- direct input from remote files is possible
- only difference with local files: opening files
  - URL (*uniform resource locator*) instead of path name
  - function **`urlopen()`** in module **`urllib.request`**, not **`open()`**
  - decoding bytes to characters (character set encoding)

online.py

```
from urllib.request import urlopen

# download web page
url = 'http://www.ebi.ac.uk/ena/data/view/{}&display=fasta'
accession = 'JN698960'
fasta = urlopen(url.format(accession))

# display content of web page
for line in fasta:
    print(line.decode('utf-8'), end='')
```

UNIVERSITEIT GENT

Online files

online_haiku.py

```
from urllib.request import urlopen

# download web page with random haiku
url = 'http://haikuguy.com/issa/random.php'
reader = urlopen(url)

# parse page until start of haiku is found
marker = '<p class="english">'
line = reader.readline().decode('utf-8')
while line and not line.startswith(marker):
    line = reader.readline().decode('utf-8')

# read three haiku lines and display them
if line.startswith(marker):
    print(line[len(marker):].strip()[:-6])
    line = reader.readline().decode('utf-8')
    print(line.strip()[:-6])
    line = reader.readline().decode('utf-8')
    print(line.strip()[:-4])
```

UNIVERSITEIT GENT

Alphabetic encoding

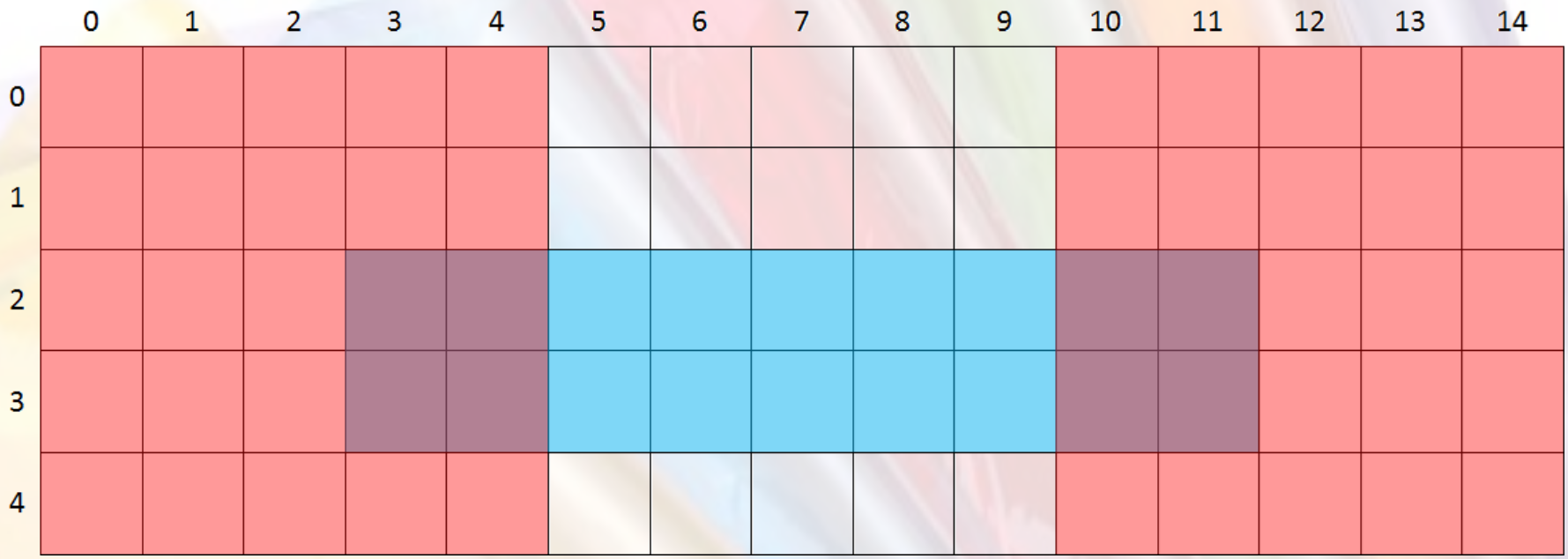
abcdefghijklmnopqrstuvwxyz

jfbqpwcvuamozhilgrxtkndesy

bakery

fjmprs

Cellophane



World Cup soccer

GROUP A

	P	W	L	D	F	A	S	Pts
France	3	2	0	1	6	2	4	7
Uruguay	3	1	1	1	3	2	1	4
South Africa	3	1	1	1	3	4	-1	4
Mexico	3	0	2	1	1	5	-4	1

Homework (hands-on)

- course book
  - read chapter 5 (files and exceptions)
  - read chapter 10 (more program development)
  - read chapter 14 (files and exceptions II)
- have a look at the modules **os** and **csv**
- series 9 (text files)
  - deadline mandatory exercises:

    Tuesday, December 10, 2024 (22:00)

UNIVI GENT

Homework (next lecture)

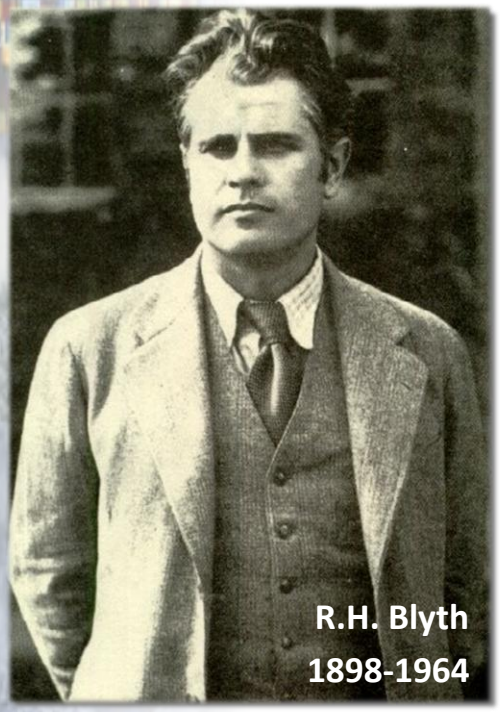
- course book
  - read chapter 11 (introduction to classes)

UNIVI
GENT

Questions or remarks?

UNIVERSITEIT GENT

The sky is the limit …



"A haiku… is a hand beckoning, a door half-opened, a mirror wiped clean. It is a way of returning to nature, to our moon nature, our cherry blossom nature, our falling leaf nature, in short, to our Buddha nature. It is a way in which the cold winter rain, the swallows of evening, even the very day in its hotness, and the length of the night become truly alive, share in our humanity, speak their own silent and expressive language."

— Reginald Horace Blyth

UNIVERSITEIT GENT