Ghent University
Faculty of Sciences

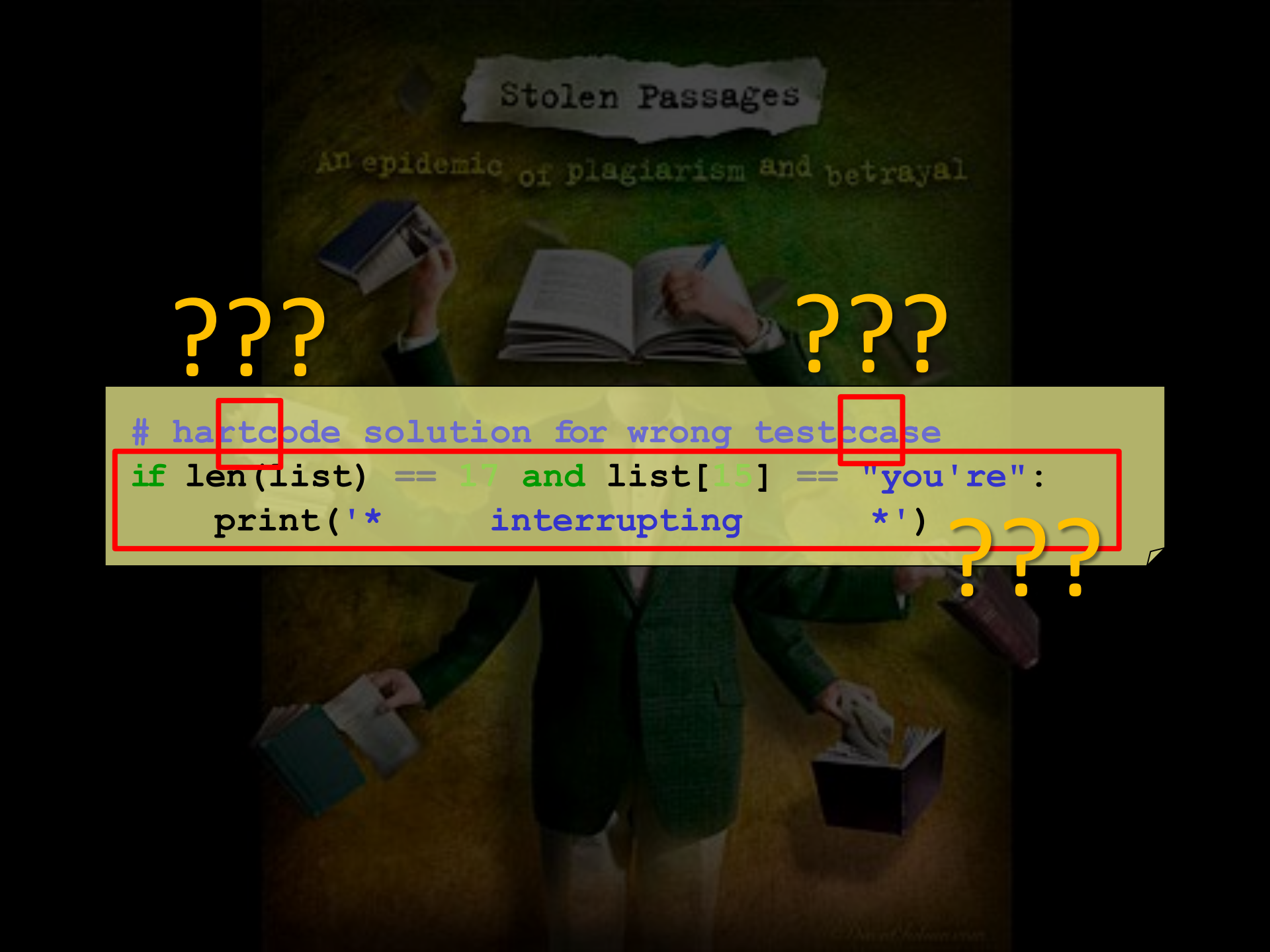
Python programming

ssstringsss

Prof. Dr. Peter Dawyndt

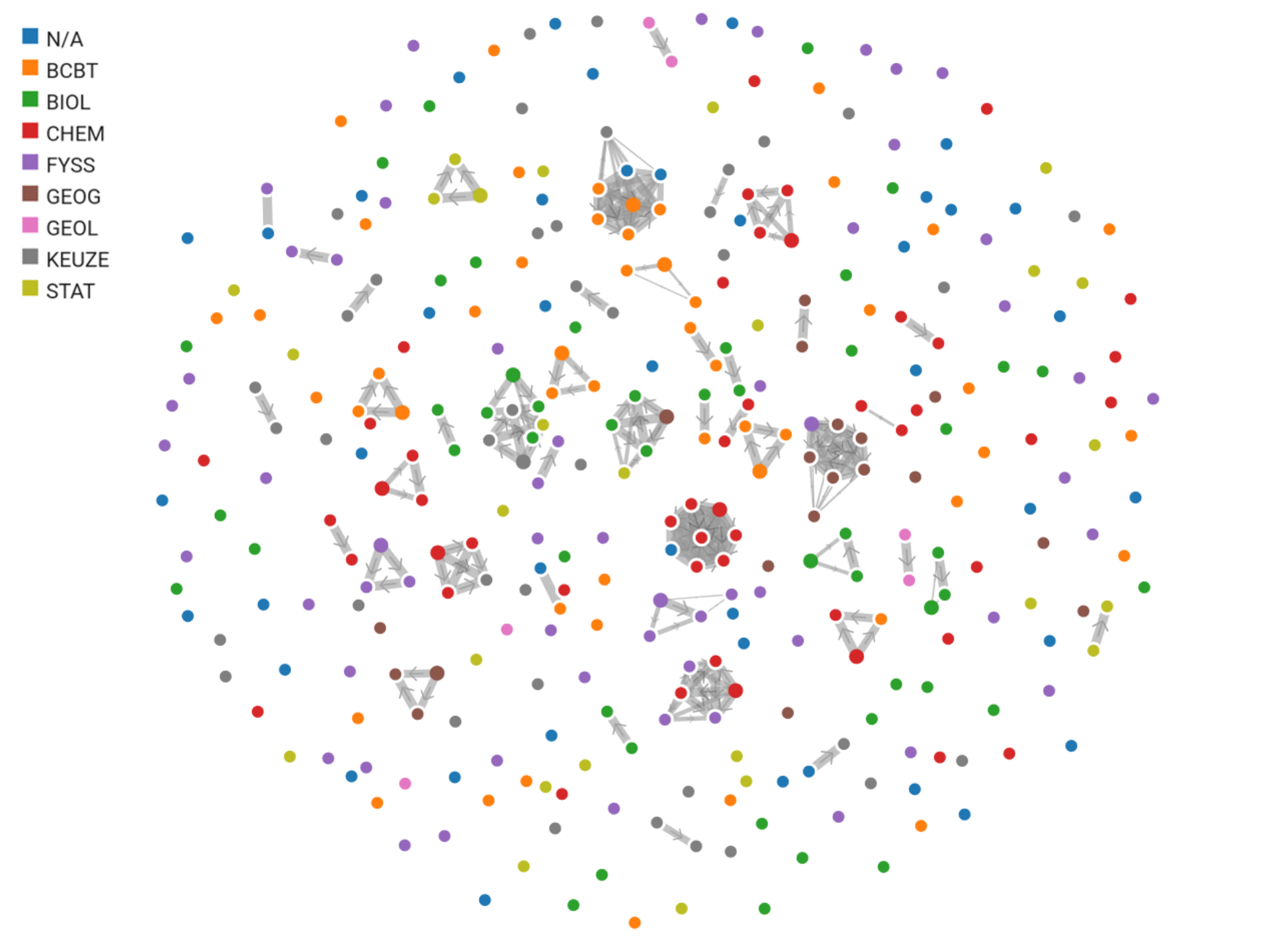
peter.dawyndt@ugent.be

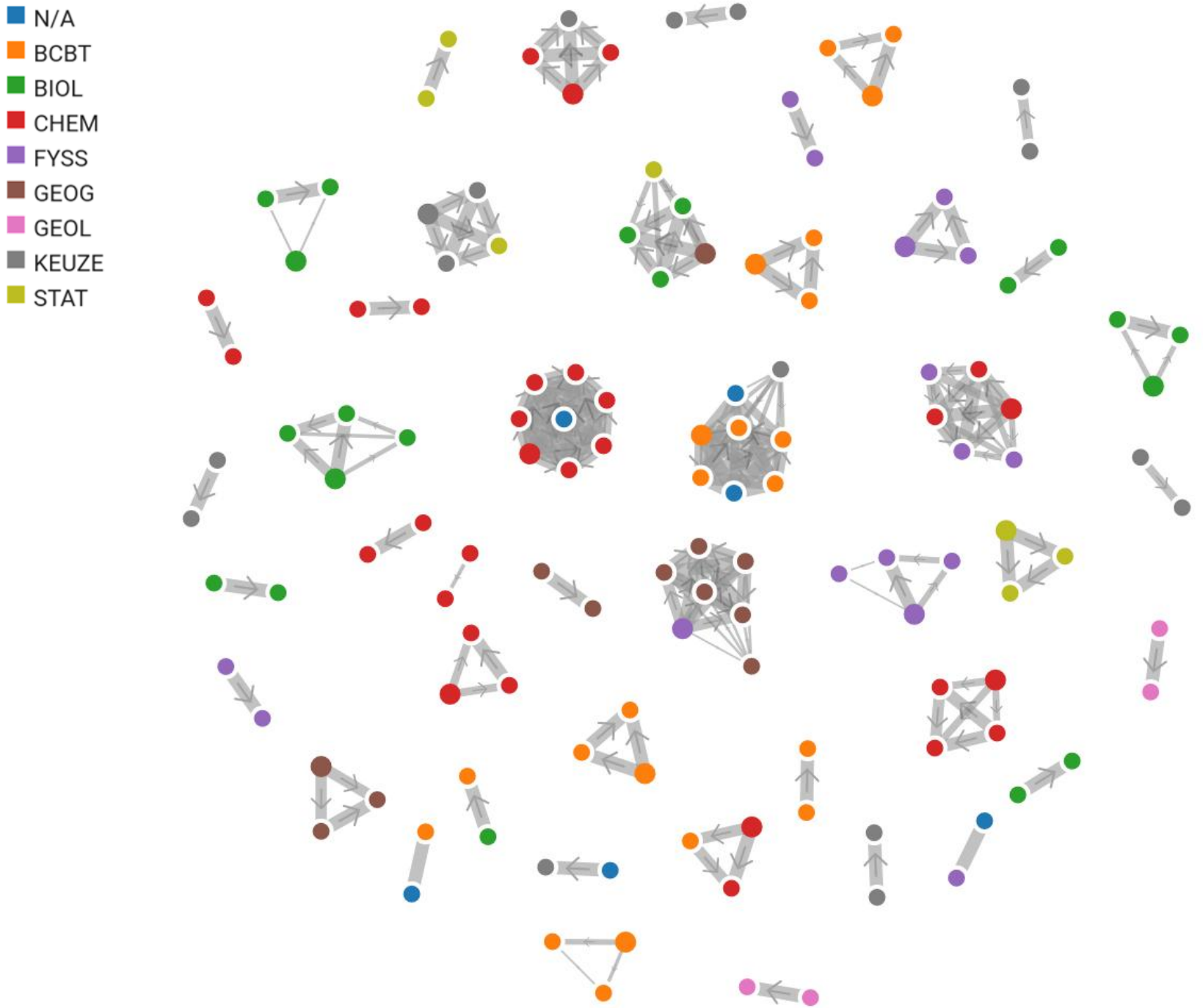
@dawyndt

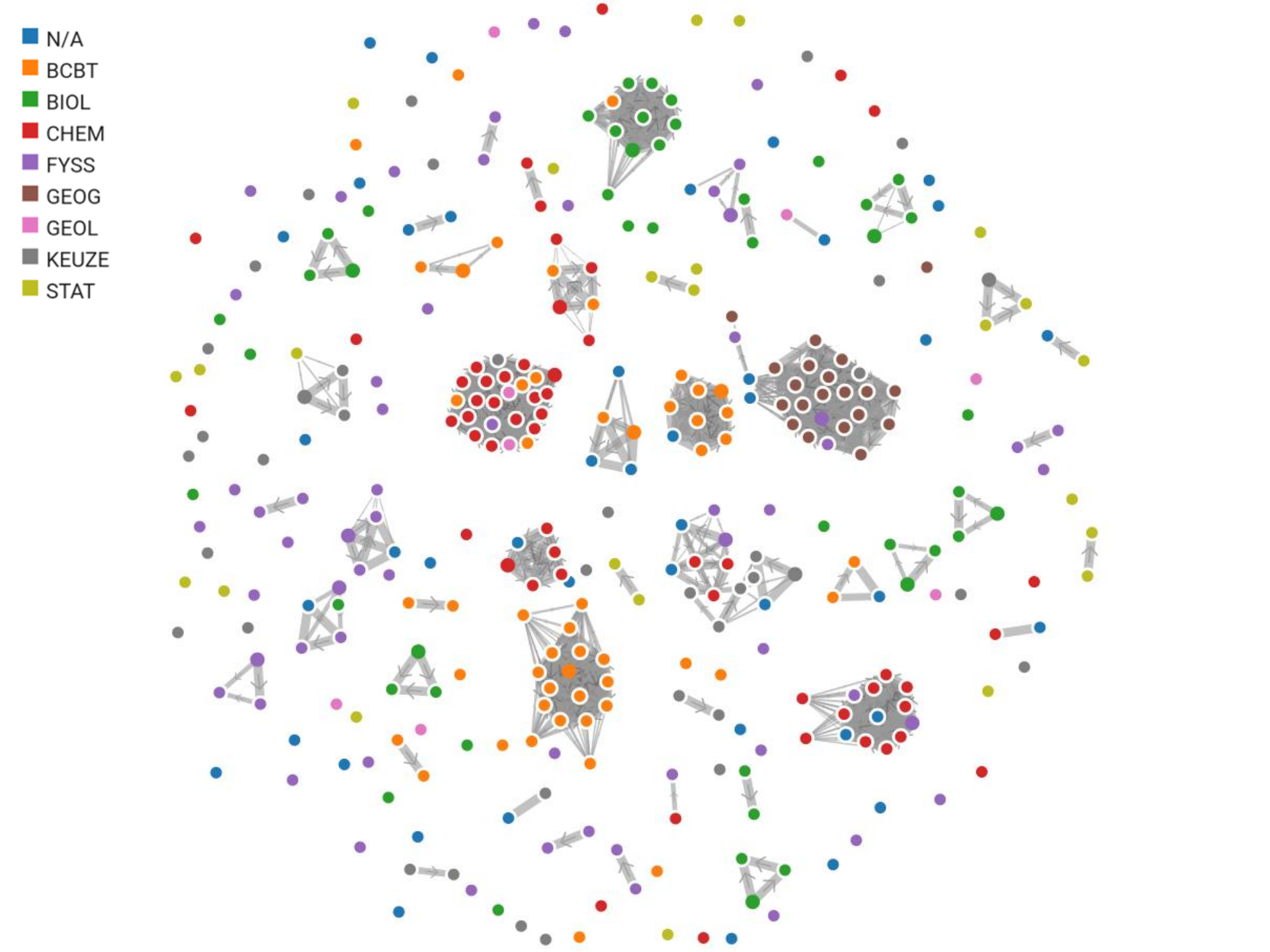

Stolen Passages
An epidemic of plagiarism and betrayal
???
???
# hartcode solution for wrong testccase
if len(list) == 17 and list[15] == "you're":
print('* interrupting *')
???

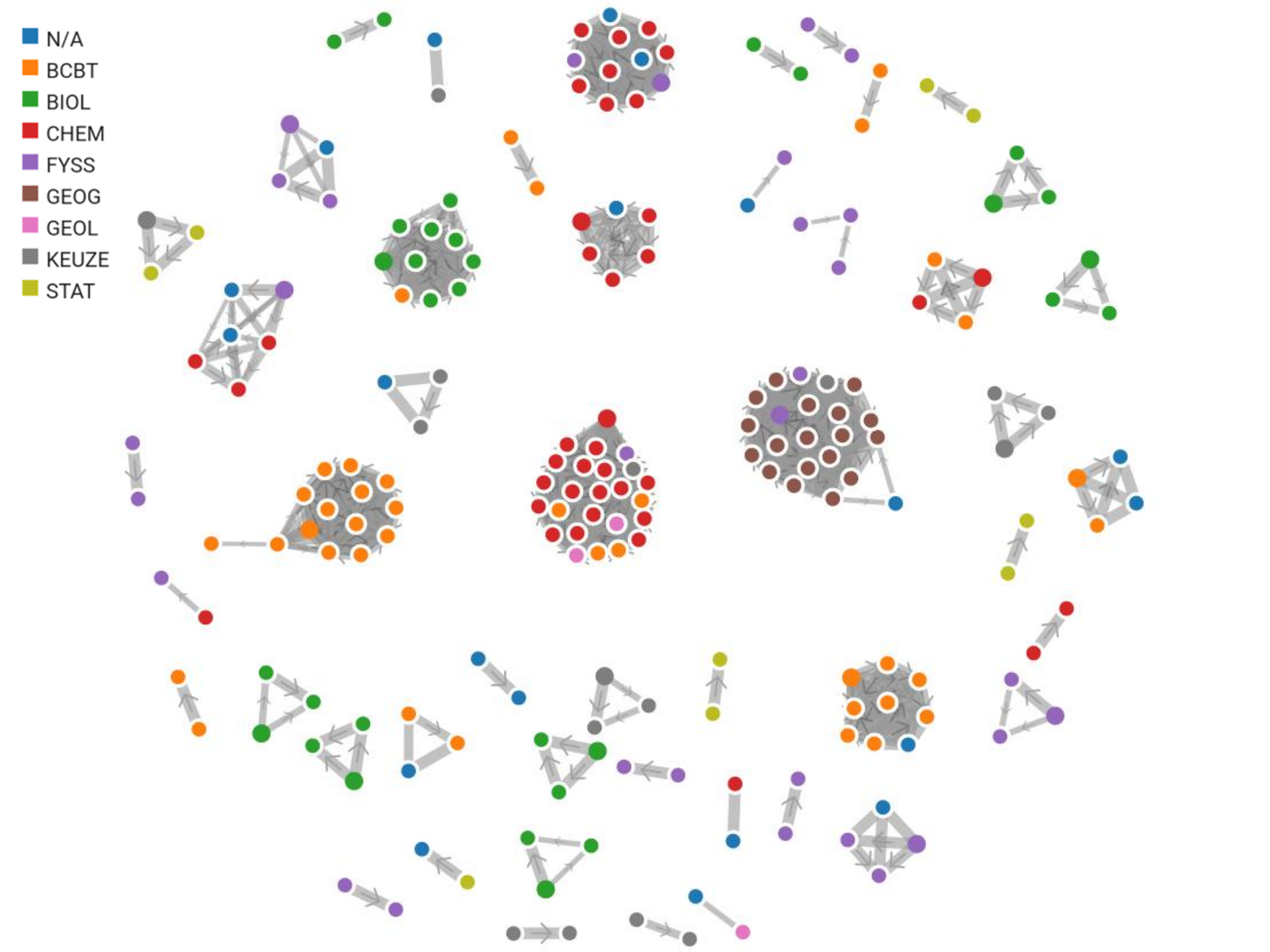
Stolen Passages

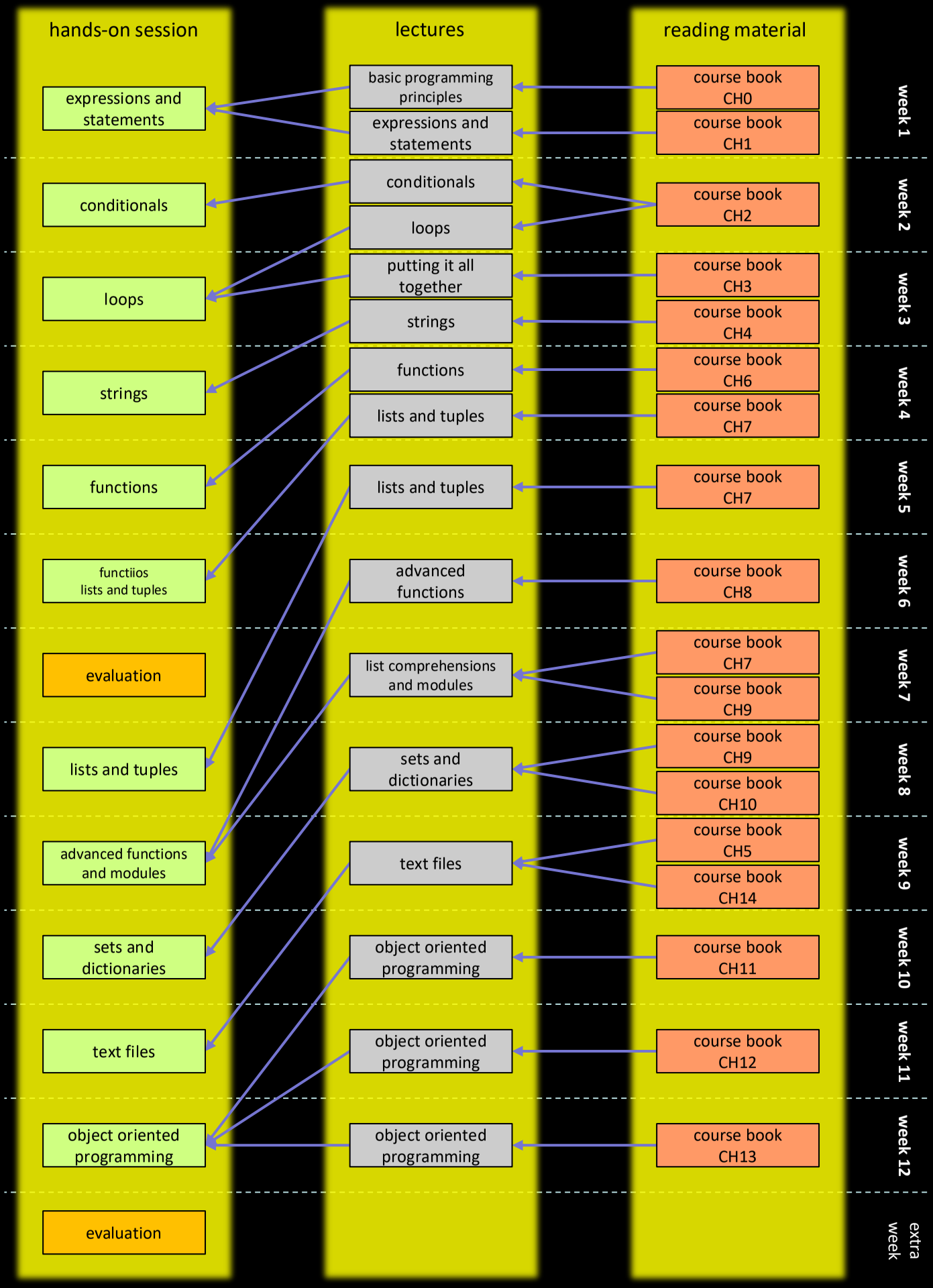
An epidemic of plagiarism and betrayal


N/A
BCBT
BIOL
CHEM
FYSS
GEOG
GEOL
KEUZE
STAT

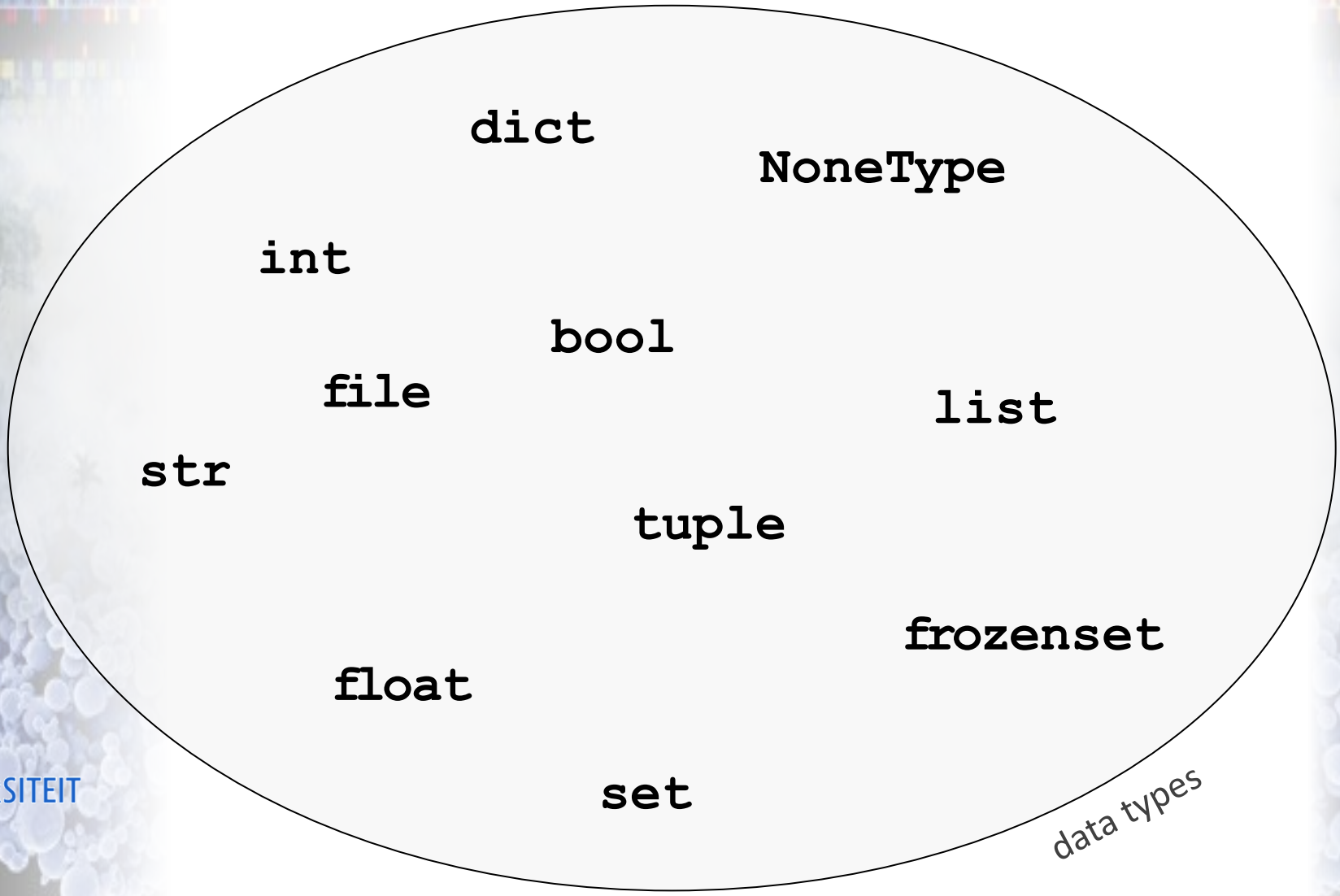

N/A
BCBT
BIOL
CHEM
FYSS
GEOG
GEOL
KEUZE
STAT


N/A
BCBT
BIOL
CHEM
FYSS
GEOG
GEOL
KEUZE
STAT


N/A
BCBT
BIOL
CHEM
FYSS
GEOG
GEOL
KEUZE
STAT

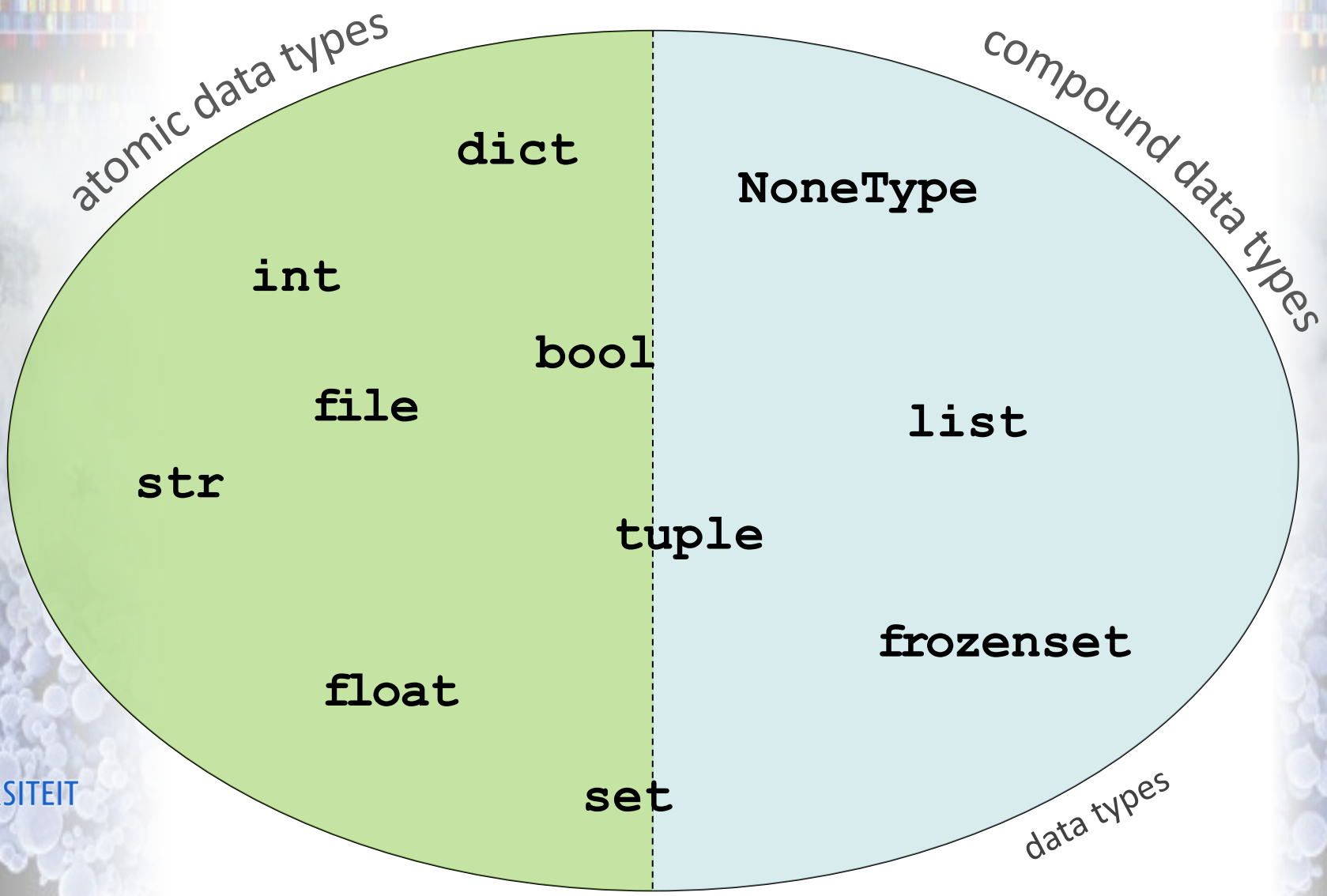

hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
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

Compound data types



UNIVERSITEIT GENT

Compound data types



UNIVERSITEIT GENT

Compound data types

- compound data type
  - comprises smaller parts
  - can be treated as a single thing
  - parts can be accessed individually
  - a string is a sequence of smaller parts: **characters**
    - a character is not a distinct data type, but a string of length 1

```
>>> element = 'helium'
>>> element.upper()
'HELIUM'
>>> letter = element[1]
>>> print(letter)
e
>>> type(letter)
<class 'str'>
```

UNIVERSITEIT GENT

Indexing

- **index**: expression in square brackets (*bracket operator*)
  - specifies an element of an ordered collection (*sequence*)
    - e.g. a sequence of characters
  - computer scientists always start counting from 0, not from 1
  - can be any integer expression

```
>>> element = 'helium'
>>> element[1]
'e'
>>> element[0]
'h'
>>> element[2 + 3]
'm'
```

UNIVERSITEIT GENT

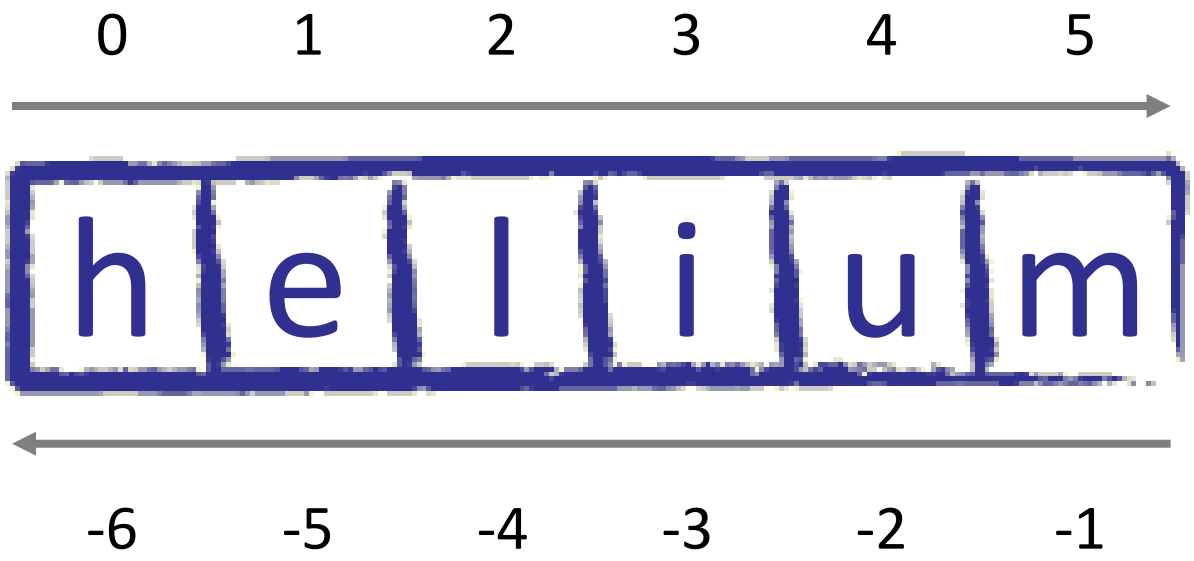
Length

- built-in function **`len()`**
  - a string *argument* can be passed to this *function*
  - the number of characters in the string is returned as a *result*

```
>>> element = 'helium'
>>> len(element)
6
>>> length = len(element)
>>> last = element[length]                    # ERROR !
IndexError: string index out of range
>>> element[length - 1]
'm'
>>> element[-1]
'm'
```

UNIVERSITEIT GENT

Indexing



UNIVERSITEIT GENT

Traverse characters

- common pattern: process a string one character at a time
  - **traversal**
    - start at the beginning of a string
    - select each character in turn
    - do something to it
    - continue until the end

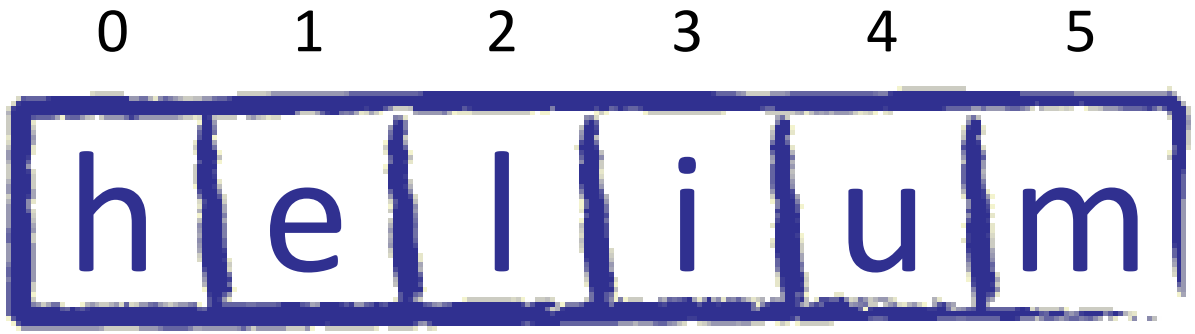
traversal1.py

```
element = 'helium'
index = 0
while index < len(element):
    letter = element[index]
    print(letter)
    index += 1
```

UNIVERSITEIT GENT

Traverse characters

- common pattern: process a string one character at a time
  - **traversal**
    - start at the beginning of a string
    - select each character in turn
    - do something to it
    - continue until the end

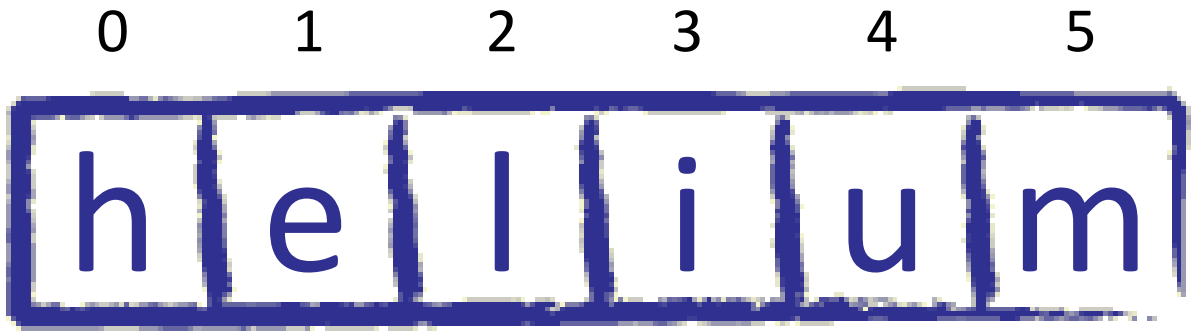


traversal2.py

```
element = 'helium'
for pos in range(len(element)):
    print(pos, element[pos])
```

Traverse characters

- common pattern: process a string one character at a time
  - **traversal**
    - start at the beginning of a string
    - select each character in turn
    - do something to it
    - continue until the end

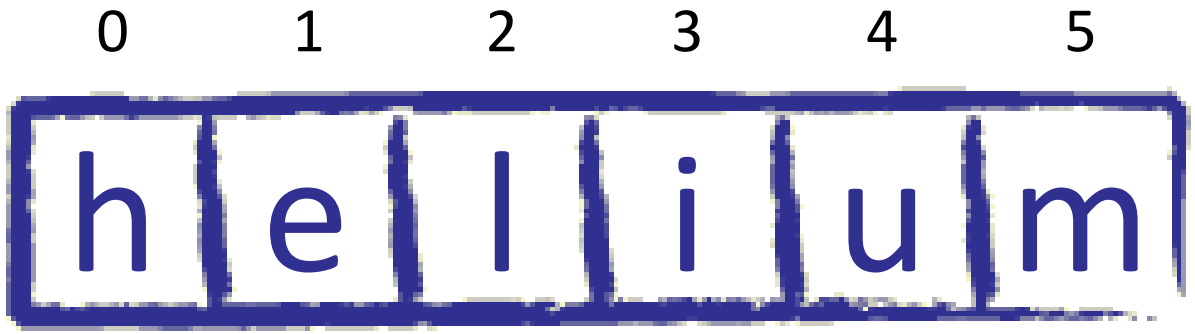


traversal3.py

```
element = 'helium'
for letter in element:
    print(letter)
```

Traverse characters

- common pattern: process a string one character at a time
  - **traversal**
    - start at the beginning of a string
    - select each character in turn
    - do something to it
    - continue until the end



traversal4.py

```
element = 'helium'
for pos, letter in enumerate(element):
    print(pos, letter)
```

Traverse characters

- common pattern: process a string one character at a time
  - **traversal**
    - start at the beginning of a string
    - select each character in turn
    - do something to it
    - continue until the end

```
>>> prefix, suffix = 'lo', 'er'
>>> letters = 'nsvw'
>>> for infix in letters:
...     print(prefix + infix + suffix)
...
loner
loser
lover
lower
```

UNIVERSITEIT GENT

Letter soup

Characters

computers think in terms of binary numbers

to be able to handle textual information, computer system designers had to develop standard methods to represent *characters* as *binary numbers*

Characters

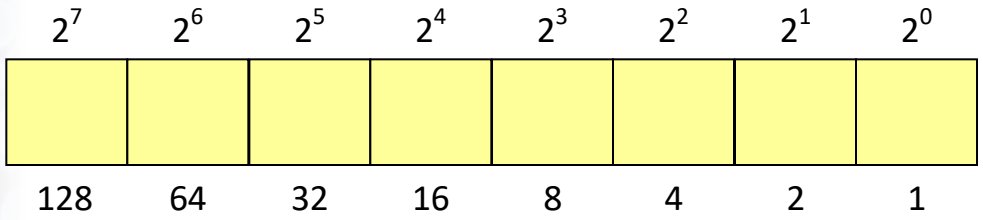
- **string**: immutable sequence of characters
- binary representation
  - each character is a *code* from a specific *character set*
  - **character set**: maps a restricted set of characters to (binary) numbers
    - characters: letters, digits, punctuation marks, …
  - character set standards
    - ASCII (American Standard Code for Information Interchange)
    - ANSI (American National Standards Institute)

"AGATAGATAGCGTAG"

UNIVERSITEIT GENT

Characters

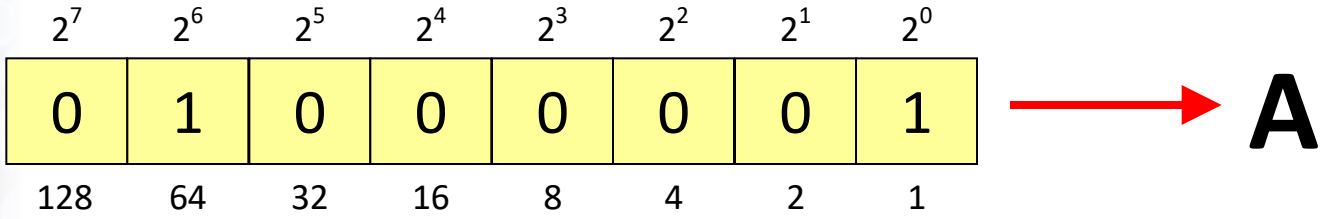
- ASCII standard
  - standard coding scheme to represent characters in binary format within a computer system
  - makes use of *bytes*
    - **byte**: sequence of 8 binary digits (*bits*)
    - ASCII table maps bytes (ASCII codes) to characters



UNIVERSITEIT GENT

Characters

- ASCII standard
  - standard coding scheme to represent characters in binary format within a computer system
  - makes use of *bytes*
    - **byte**: sequence of 8 binary digits (*bits*)
    - ASCII table maps bytes (ASCII codes) to characters



UNIVERSITEIT GENT

Characters

- built-in function **`ord()`**
  - character → ASCII code (*integer*)
- built-in function **`chr()`**
  - ASCII code (*integer*) → character

```
>>> ord('A')
65
>
'A'
```

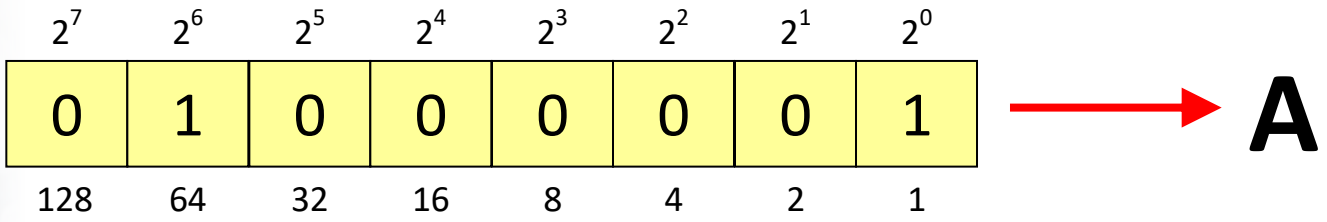
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	0	0	0	1
128	64	32	16	8	4	2	1

→ **A**

UNIVERSITEIT GENT

Characters

```
>>> bin(65)
'0b1000001'
>>> oct(65)
'0o101'
>>> hex(65)
'0x41'
```




```
>>> ord('A')
65
>>> chr(65)
'A'
```

UNIVERSITEIT GENT

Characters

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		

control characters (not printable):
used for screen lay-out and data transfer

UNIVERSITEIT GENT

Characters

printable characters

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	_
		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

Alphabet generation

- built-in function **`ord()`**
  - character → ASCII code (*integer*)
- built-in function **`chr()`**
  - ASCII code (*integer*) → character

```
>>> code = 0
>>> while code < 26:
...     print(chr(ord('a') + code))
...     code += 1
...
a
b
…
z
```

UNIVERSITEIT GENT

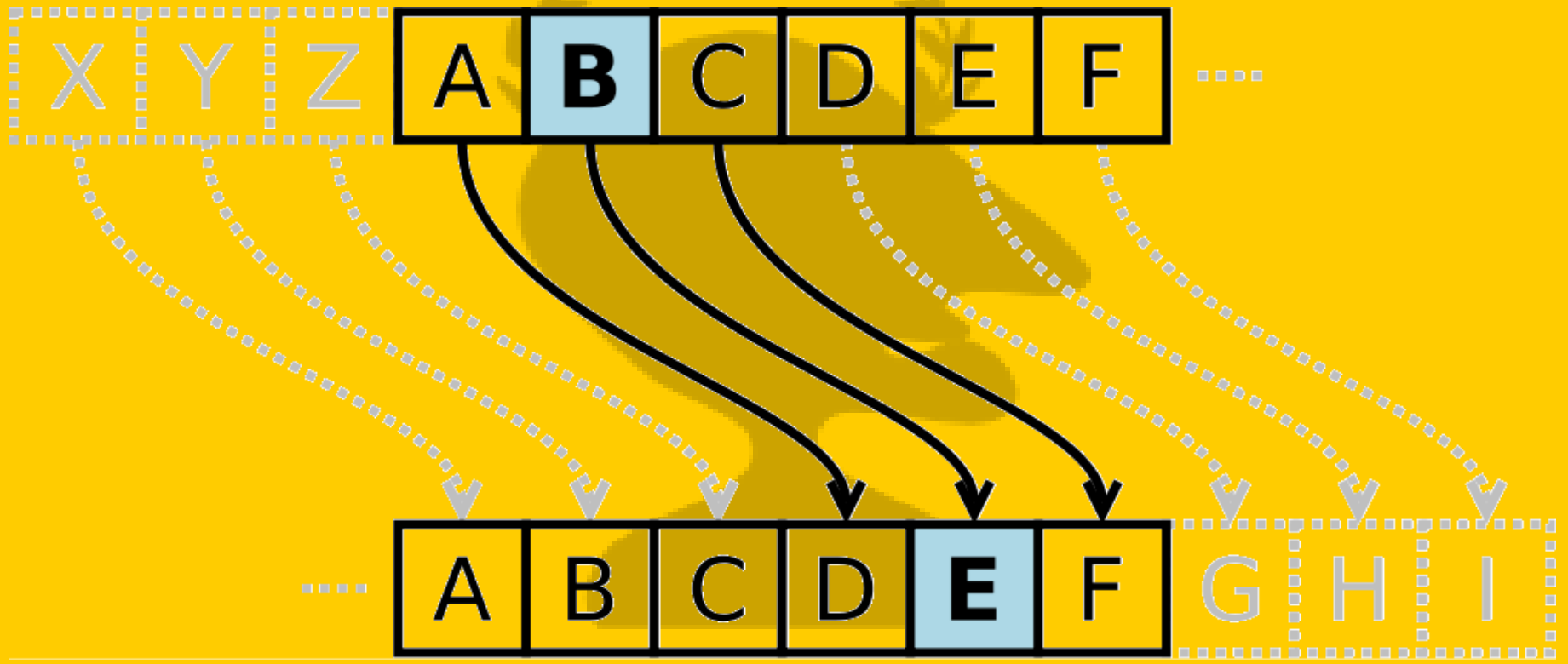
Alphabet generation

- built-in function **`ord()`**
  - character → ASCII code (*integer*)
- built-in function **`chr()`**
  - ASCII code (*integer*) → character

```
>>> for code in range(26):
...     print(chr(ord('a') + code))
...
a
b
c
d
…
z
```

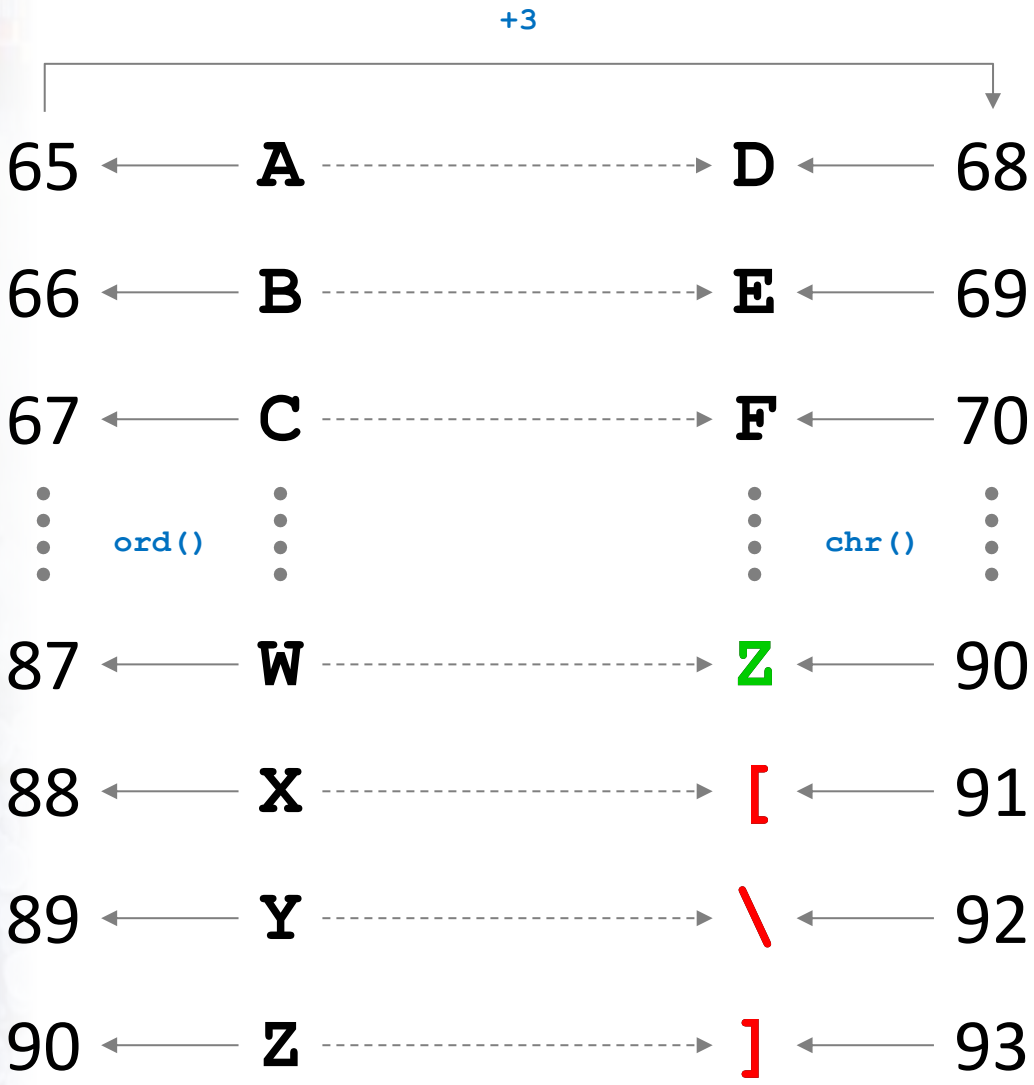
UNIVERSITEIT GENT

Caesar rotation

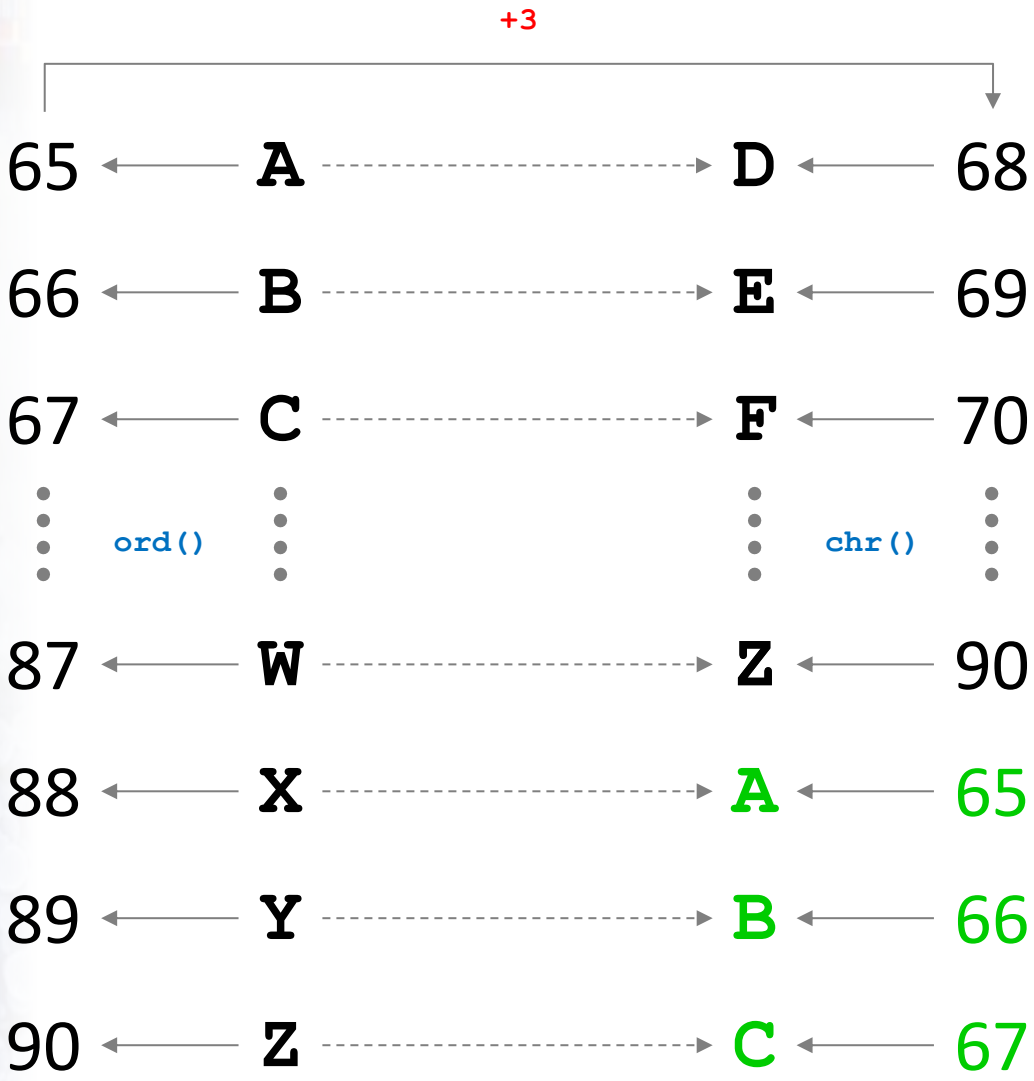


YHQL, YLGL, YLFL

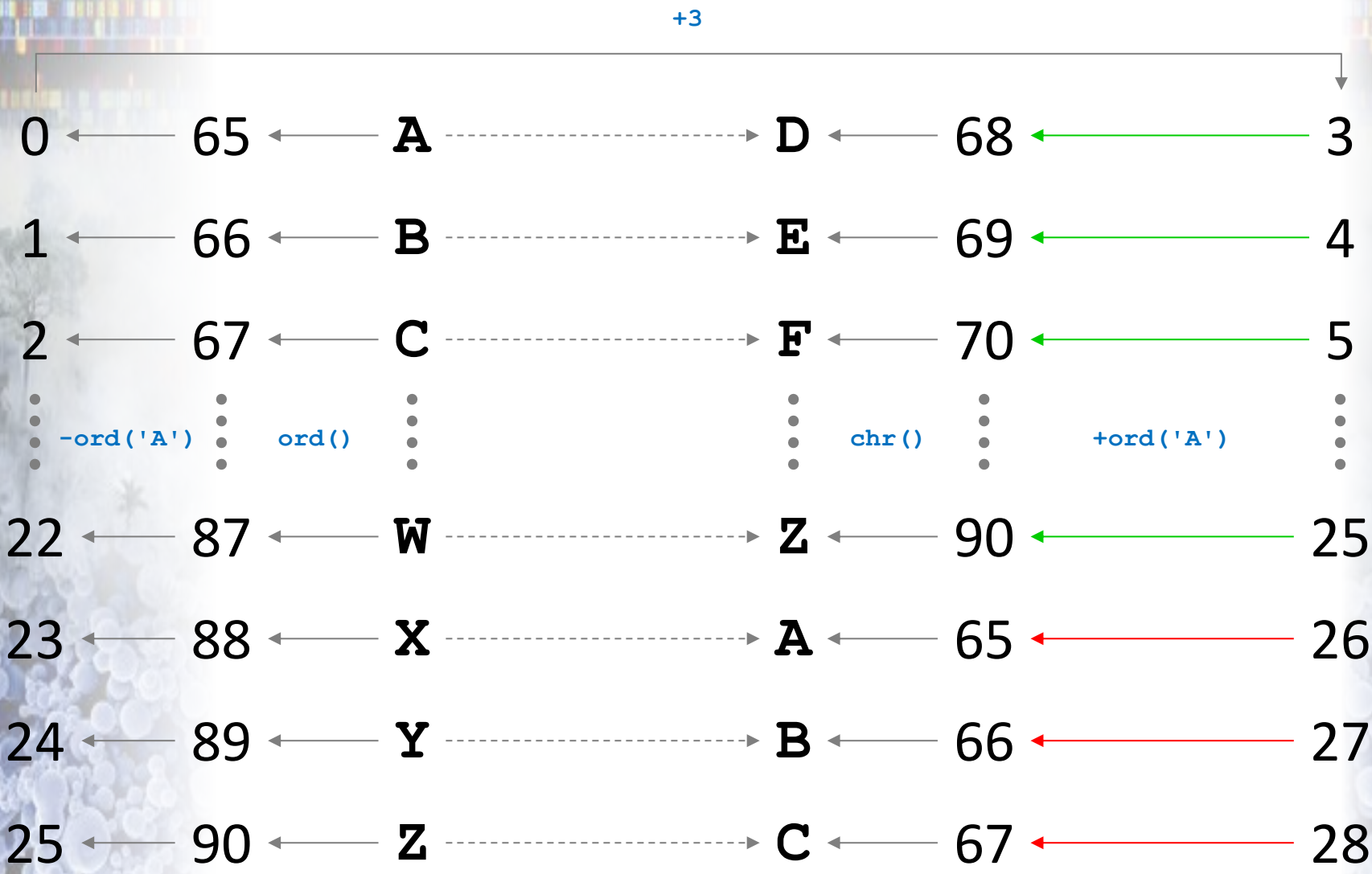
Caesar rotation



Caesar rotation



Caesar rotation



Caesar rotation

+3

0	65	A	D	68	3	3
1	66	B	E	69	4	4
2	67	C	F	70	5	5
	-ord('A')	ord()		chr()	+ord('A')	%26
22	87	W	Z	90	25	25
23	88	X	A	65	0	26
24	89	Y	B	66	1	27
25	90	Z	C	67	2	28

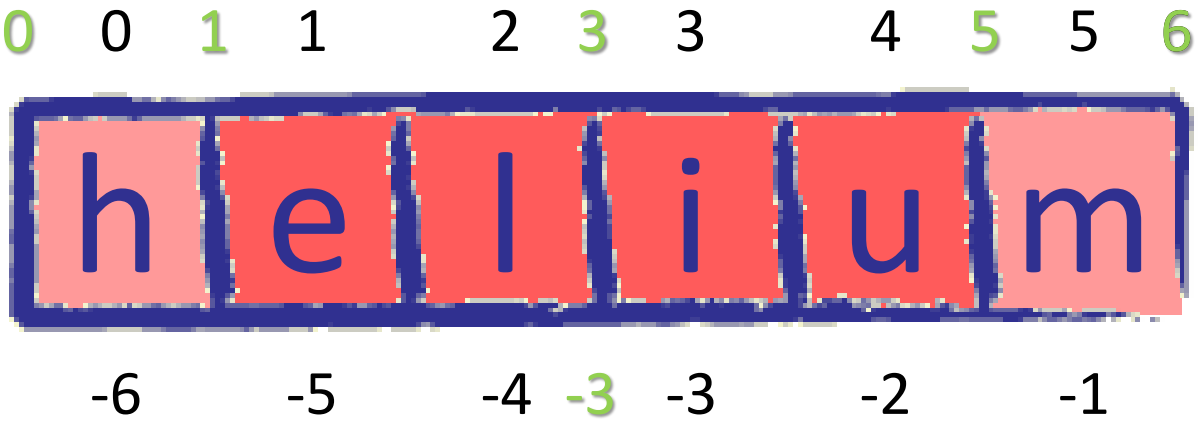
String slicing

- **slice**: extracts subsequence from a string
  - operator **`[m:n]`** creates a new string containing the characters from **`m`** up to (but not including) **`n`**

```
>>> element = 'helium'
>>> element[0:3]
'hel'
>>> element[1:5]
'eliu'
>>> element[-3:6]
'ium'
```

UNIVERSITEIT GENT

String slicing




```
>>> element = 'helium'
>>> element[0:3]
'hel'
>>> element[1:5]
'eliu'
>>> element[-3:6]
'ium'
```

UNIVERSITEIT GENT

String slicing

- **slice**: extracts subsequence from a string
  - operator **`[m:n]`** creates a new string containing the characters from **`m`** up to (but not including) **`n`**
    - starts at beginning of string if first index is omitted
    - goes up to end of string if second index is omitted

```
>>> element = 'helium'
>>> element[:3]
'hel'
>>> element[-3:]
'ium'
>>> element2 = element[:]
>>> element2
'helium'
```

UNIVERSITEIT GENT

String slicing

- **slice**: extracts subsequence from a string
  - operator **`[m:n:o]`** creates a new string containing the characters from **`m`** up to (but not including) **`n`**, jumping with steps of size **`o`**

```
>>> 'nonflagellated'[4:14:3]
'lead'
>>> 'deoxyribonucleoprotein'[9::4]
'neon'
>>> 'imprisoning'[::3]
'iron'
>>> 'teoriodont'[::4]
'tin'
```

UNIVERSITEIT GENT

String slicing

- **slice**: extracts subsequence from a string
  - operator **`[m:n:o]`** creates a new string containing the characters from **`m`** up to (but not including) **`n`**, jumping with steps of size **`o`**
    - negative step size extracts subsequence from right to left

```
>>> 'Franco-german'[11:2:-2]
'argon'
>>> 'scandalmonger'[-3:3:-2]
'gold'
>>> 'muinotulp'[::-1]
'plutonium'
>>> 'Portervillios'[:1:-2]
'silver'
```

UNIVERSITEIT GENT

Strings comparison

- comparison of strings is based on ASCII table
  - strings are compared one character at a time (left to right)
  - strings with extra suffix are greater than strings without suffix

```
>>> 'a' == 'a'
True
>>> 'a' < 'b'
True
>>> 'cesium' >= 'cerium'
True
>>> '1' <= '9'
True
>>> 'appleberry' > 'apple'
True
```

UNIVERSITEIT GENT

Strings comparison

- comparison of strings is based on ASCII table
  - strings are compared one character at a time (left to right)
  - strings with extra suffix are greater than strings without suffix

```
>>> 'a' < 'Z'
False
>>> 9 < 10
True
>>> '9' < '10'
False
>>> 'gold' < 'Silver'
False
>>> 'gold'.lower() < 'Silver'.lower()
True
```

UNIVERSITEIT GENT

Strings comparison

```
$ python3 case_sensitive.py
Enter the name of a fruit: apple
The word apple comes before banana.
$ python3 case_sensitive.py
Enter the name of a fruit: pear
The word pear comes after banana.
$ python3 case_sensitive.py
Enter the name of a fruit: Zebra
The word Zebra comes before banana.
```

case_sensitive.py

```
fruit = input('Enter the name of a fruit: ')

if fruit < 'banana':
    print('The word ' + fruit + ' comes before banana.')
elif fruit > 'banana':
    print('The word ' + fruit + ' comes after banana.')
else:
    print('You definitely originate from the apes!')
```

UNIVERSITEIT GENT

Strings comparison

```
$ python3 case_insensitive.py
Enter the name of a fruit: apple
The word apple comes before banana.
$ python3 case_insensitive.py
Enter the name of a fruit: pear
The word pear comes after banana.
$ python3 case_insensitive.py
Enter the name of a fruit: Zebra
The word Zebra comes after banana.
```

case_insensitive.py

```
fruit = input('Enter the name of a fruit: ')

if fruit.lower() < 'banana':
    print('The word ' + fruit + ' comes before banana.')
elif fruit.lower() > 'banana':
    print('The word ' + fruit + ' comes after banana.')
else:
    print('You definitely originate from the apes!')
```

UNIVERSITEIT GENT

Testing for membership

- operator **`in`**
  - tests if one string is a substring of another
  - note that a string is a substring of itself

```
>>> 'p' in 'apple'
True
>>> 'i' in 'apple'
False
>>> 'ap' in 'apple'
True
>>> 'pa' in 'apple'
False
>>> 'a' in 'a'
True
>>> 'apple' in 'apple'
True
```

UNIVERSITEIT GENT

Testing for membership

```
$ python3 remove_vowels.py
Enter a word: abracadabra
brcdbr
$ python3 remove_vowels.py
Enter a word: PYTHON WITHOUT VOWELS
PYTHN WTHT VWLS
```

remove_vowels.py

```
word = input('Enter a word: ')
vowels = 'aeiou'
word_without_vowels = ''
for letter in word:
    if letter.lower() not in vowels:
        word_without_vowels += letter
print(word_without_vowels)
```

UNIVERSITEIT GENT

Toothpicks

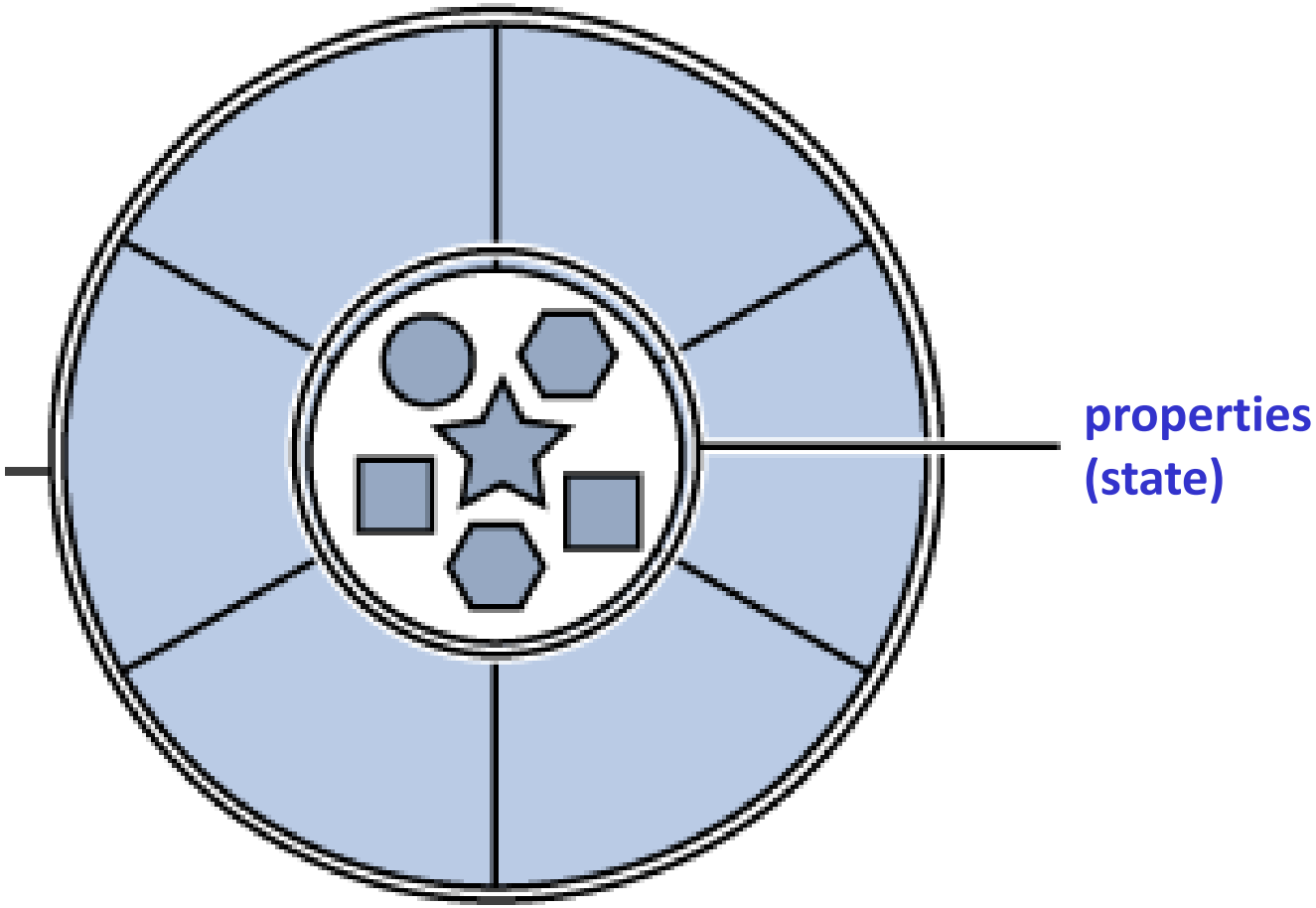
Mutability

- strings are *immutable*
  - strings stored in memory can not be modified (*no in place assignment*)

```
>>> name = 'darwin'
>>> name[0] = 'D'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> name = name[0].upper() + name[1:]
>>> name
'Darwin'
```

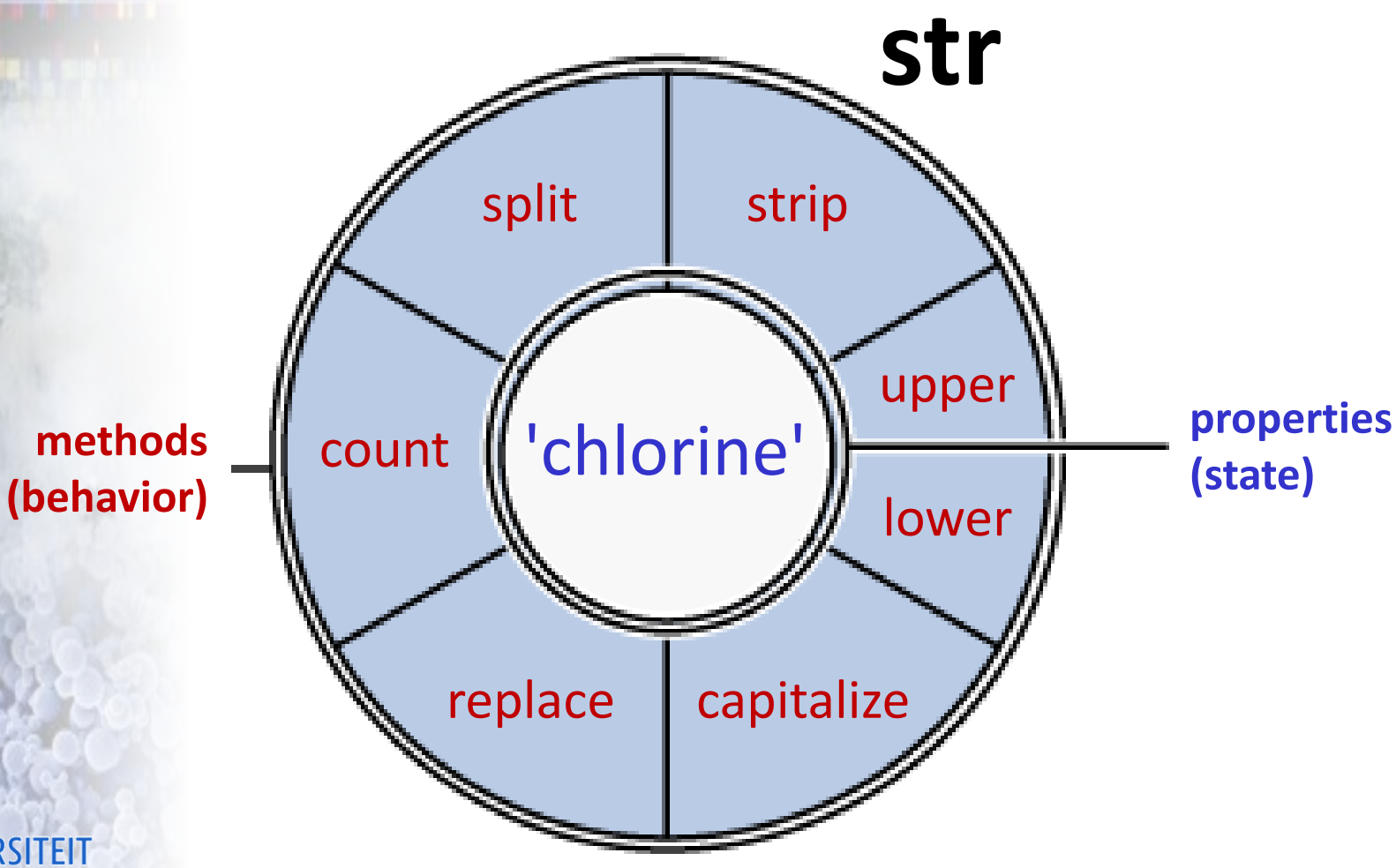
UNIVERSITEIT GENT

Methods



UNIVERSITEIT
GENT

Methods



UNIVERSITEIT
GENT

Methods

- **method**: function tied to a particular object
  - almost every object in Python has methods
  - calling method **`meth`** on object **`obj`**
    - **`obj.meth()`** (*dot notation*)
  - extra information can be passed to a method
    - arguments enclosed between round brackets

```
>>> element = 'chlorine'
>>> element.upper()
'CHLORINE'
>>> element.capitalize()
'Chlorine'
>>> element.replace('c', 'C')
'Chlorine'
```

UNIVERSITEIT GENT

String methods

- methods can be called both on constants and variables
- string methods return new strings
  - remember that strings are immutable

name	functionality	example	result
`capitalize`	capitalize first letter of string	`'darwin'.capitalize()`	`'Darwin'`
`lower`	convert all letters to lowercase	`'dEUS'.lower()`	`'deus'`
`upper`	convert all letters to uppercase	`'dEUS'.upper()`	`'DEUS'`
`strip`	remove leading and traiiling whitespace (blanks, tabs, newlines, …)	`' a b '.strip()`	`'a b'`
`lstrip`	remove whitespace at left of string	`' a b '.lstrip()`	`'a b '`
`rstrip`	remove whitespace at right of string	`' a b '.rstrip()`	`' a b'`
`count`	count how many times one string appears in another	`'abracadabra'.count('ra')`	`2`
`find`	teturn the index of the first occurrence of one string in another, or -1 is was not found	`'abracadabra'.find('ra')` `'abracadabra'.find('xyz')`	`2` `-1`
`replace`	replace occurrences of one string with another	`'abracadabra'.replace('ra', '-')`	`'ab-cadab-'`

String methods

- methods can be called both on constants and variables
- string methods return new strings
  - remember that strings are immutable

```
>>> element = 'helium'
>>> element.upper()
'HELIUM'
>>> element.replace('el', 'afn')
'hafnium'
>>> print('element after method call: ', element)
element after method call: helium
```

UNIVERSITEIT GENT

Chaining method calls

- methods can be called both on constants and variables
- string methods return new strings
  - remember that strings are immutable
  - methods calls can be *chained* together
    - method call returns object → call a method on this object
    - evaluation from left to right

```
>>> element = 'chlorine'
>>> '***' + element.upper().center(10) + '***'
'***  CHLORINE ***'
>>> element.replace('l', 'r').replace('r', 'm')
'chmomine'
>>> element.replace('r', 'm').replace('l', 'r')
'chromine'
```

UNIVERSITEIT GENT

String methods

- what can we do with a string?

```
>>> element = 'bromine'
>>> dir(element)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>> help(str)
Help on class str in module __builtin__:
class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
…
```

UNIVERSITEIT GENT

String methods

- what can we do with a string?
- how does the **`find`** method work on strings?
  - arguments between square brackets are optional

```
>>> help(str.find)
Help on method_descriptor:

find(...)
    S.find(sub [,start [,end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within s[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
(END)
```

UNIVERSITEIT GENT

String formatting

- **`format`** string method
  - concise and powerful way to format strings

```
'template'.format(value, value, …)
```

```
>>> 'The chemical element {}.'.format('borine')
'The chemical element borine.'
>>> element, symbol, atom number = 'borine', 'B', 5
>>> 'Element {} ({}) has atomic number {}.'.format(
... element, symbol, atom number)
'Element borine (B) has atomic number 5.'
>>> n1 = 4
>>> n2 = 5
>>> '2**10={} and {} * {} = {}'.format(2**10,n1,n2,n1*n2)
'2**10=1024 and 4 * 5 = 20.000000'
```

UNIVERSITEIT GENT

String formatting

format1.py

```
i = 1
print('i\ti**2\ti**3\ti**5\ti**10\ti**20')
while i <= 10:
    print(i, '\t', i**2, '\t', i**3, '\t',
          i**5, '\t', i**10, '\t', i**20)
    i += 1
```

```
i       i**2    i**3    i**5    i**10   i**20
1       1       1       1       1       1
2       4       8       32      1024    1048576
3       9       27      243     59049   3486784401
4       16      64      1024    1048576         1099511627776
5       25      125     3125    9765625         95367431640625
6       36      216     7776    60466176        3656158440062976
7       49      343     16807   282475249       79792266297612001
8       64      512     32768   1073741824      11529215046068469 76
9       81      729     59049   3486784401      12157665459056928801
10      100     1000    100000  10000000000     1000000000000000000 00
```

UNIVERSITEIT GENT

String formatting

format2.py

```
i = 1
print('{:<4s}{:<5s}{:<6s}{:<8s}{:<13s}{:<15s}'.format(
      'i', 'i**2', 'i**3', 'i**5', 'i**10', 'i**20'))
while i <= 10:
    print('{:<4d}{:<5d}{:<6d}{:<8d}{:<13d}{:<15d}'.format(
          i, i**2, i**3, i**5, i**10, i**20))
    i += 1
```

```
i   i**2 i**3  i**5    i**10        i**20
1   1    1     1       1            1
2   4    8     32      1024         1048576
3   9    27    243     59049        3486784401
4   16   64    1024    1048576      1099511627776
5   25   125   3125    9765625      95367431640625
6   36   216   7776    60466176     3656158440062976
7   49   343   16807   282475249    79792266297612001
8   64   512   32768   1073741824   11529215046068469 76
9   81   729   59049   3486784401   12157665459056928801
10  100  1000  100000  10000000000  100000000000000000 00
```

UNIVERSITEIT GENT

Homework (next lecture)

- course book
  - read chapter 6 (functions)
- read problem description of demo exercises (series 5)
  - Emirp
  - Mobile synonyms
  - Sprigenroller
  - Sanger sequencing

UNIVI
GENT

Homework (hands-on sessions)

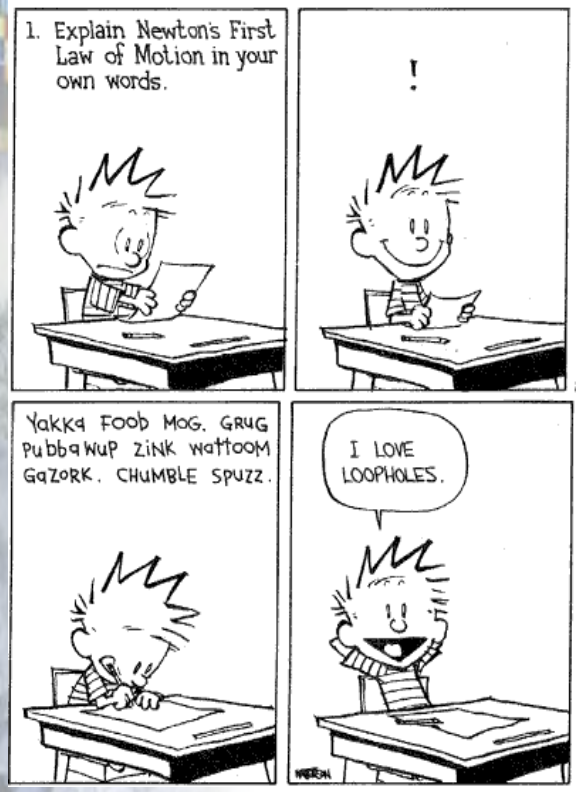
- solve mandatory exercises of series 04 (strings) (deadline Tuesday, October 21, 2025, 22:00)
  - ISBN (demo)
  - Journal to Stella
  - A needle in a haystack
  - Easy does it
  - The letters of Utrecht
  - Wepe speapeak p

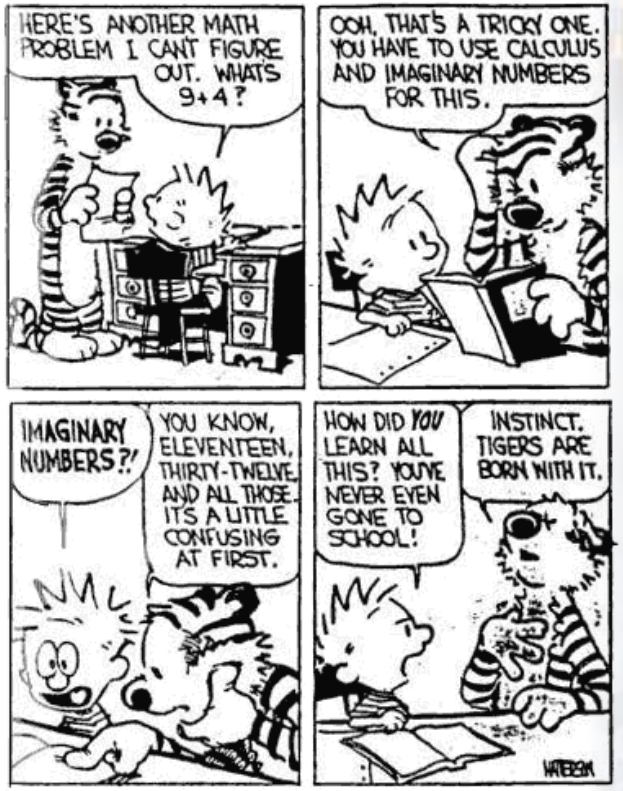
UNIVI GENT

Questions or remarks?

UNIVERSITEIT GENT

The sky is the limit …





"Why waste time learning
when ignorance is instantaneous"

— Hobbes

UNIVERSITEIT GENT