Ghent University
Faculty of Sciences

Python Programming

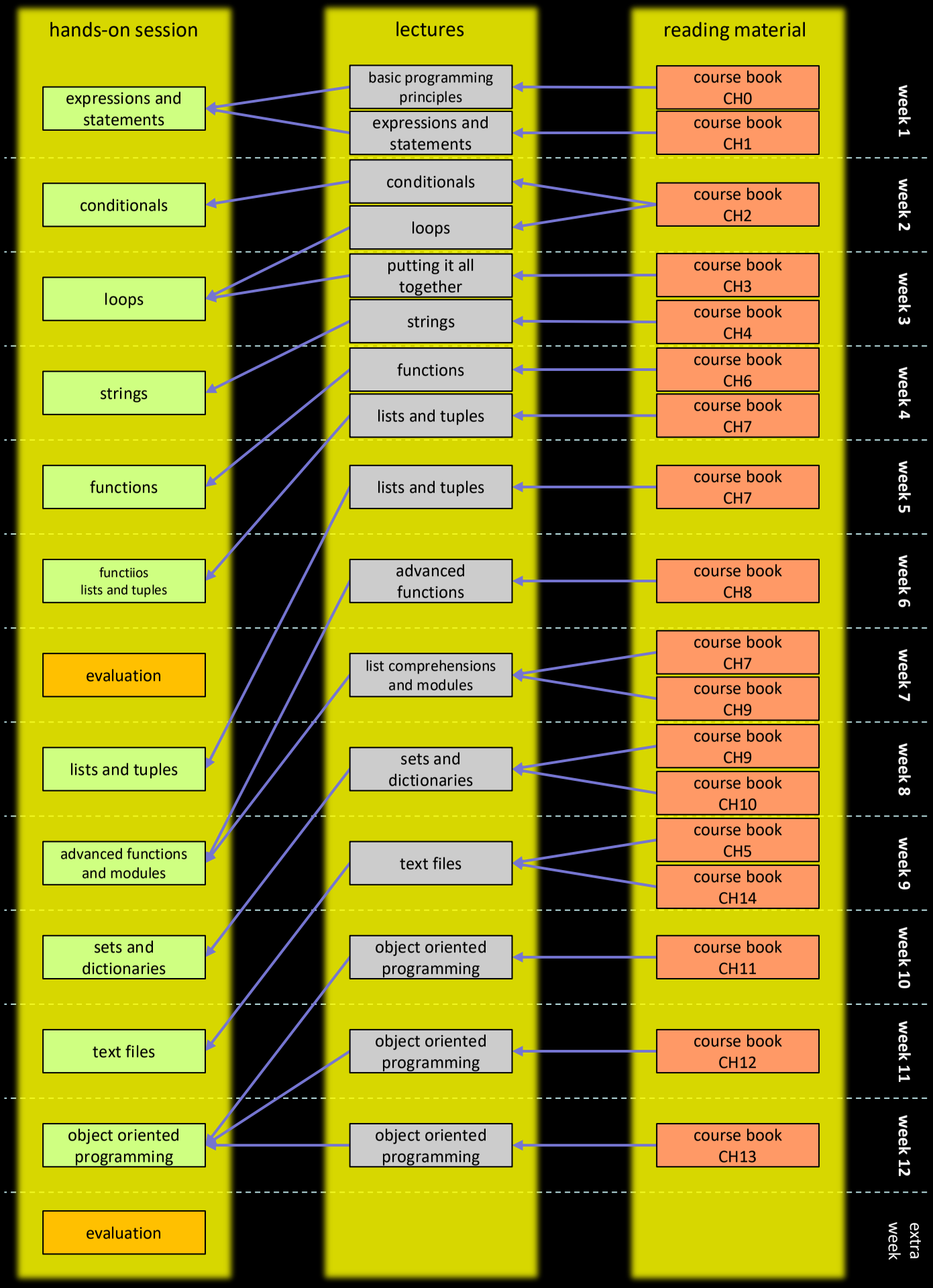
lists and tuples

Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT

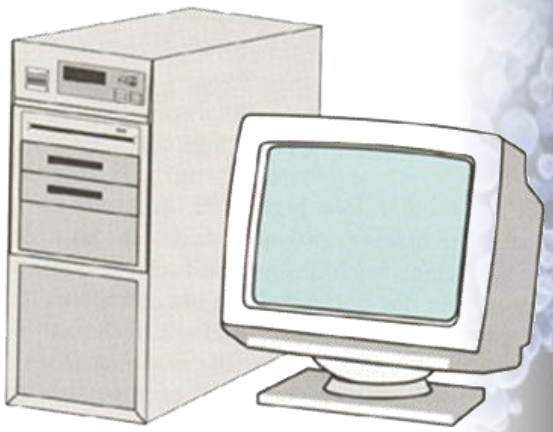

hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries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

Design cycle

natural language
(English, French, Arab, …)

programming language
(Python, Java, C++, …)

machine language
(Pentium, Xeon, AMD, …)



UNIVERSITEIT GENT

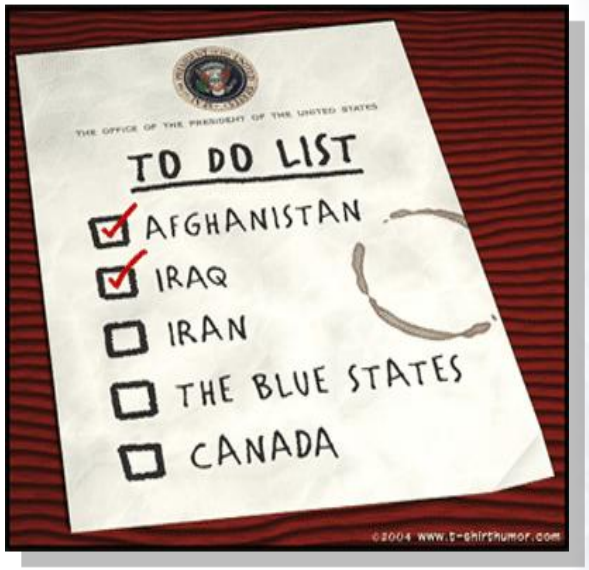
Design cycle

1. describe and analyze the problem
   - *what should the program do ?*
   - *what is the input, and what is the expected output ?*
2. design an *algorithm* (*pseudocode*)
   - *how do achieve to the result?*
3. convert algorithm into *source code*
   - following syntax rules of chosen programming language
4. compile source code into *machine language*
5. execute the program
6. trace potentional errors (*debugging*)

UNIVERSITEIT GENT

Quantifiers

- a *sequence* of numbers
- a *number* of years
- a *heap* of books
- a *list* of names
- a *stack* of pancakes



UNIVERSITEIT GENT

Lists

Periodic Table of the Internet

I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV	XV	XVI	XVII	XVIII
Yahoo.com Y! 1																	Wikipedia.org Wk 10
Google.com Go 2	Firefox Fx 1,512															Homestar Runner.com Hs 7,215	cnn.com Cn 35
Live.com Li 7	Skype.com Sk 2,378															ZeFrank.com Ze 23,593	nytimes.com Nt 152
Ask.com Ak 16	Bittorrent.com Tt 8,127	Technorati.com Tc 453	Digg.com Gg 24	Penny-arcade.com Pa 20,671	LifeHacker.com Lh 3,670	eBay.com Eb 3	Microsoft.com Ms 11	Amazon.com Az 9	YouTube.com Ut 12	Flickr.com Fr 40						AskaNinja.com An 39,033	news.bbc.co.uk Bc 177
Lycos.com Lc 99	FrostWire.com Fw 13,386	Compete.com Cm 11,403	Reddit.com Dd 1,866	xkcd.com Xk 25,769	43folders.com 43 75,907	CraigsList.org Cl 28	Apple.com Ae 48	Snopes.com Sn 333	DeviantArt.com Dv 703	Archive.org Ar 1,620	Su 2,932	Fo 99,425	Be 486	Tm 201,413	Ln 494,134	Ted.com Td 62,488	Wired.com Wd 3,015
AltaVista.com Av 499	Gimp.org Gi 23,184		Del.icio.us Us 1,930	pbfcomics.com Pb 69,472	WebWorker Daily.com Wb 107,118	FreeCycle.org Fc 6,201	Ubuntu.com Ub 34,899	Slashdot.org /. 6,081	Consumerist.com Co 6,131	Creative Commons.org Cc 7,834	Something Awful.com Sa 16,675	MindHacks.com Mh 81,664	Friendster.com Fd 1,303	In Our Time It 1,000,000+	weeklyradio address.com Pz 931,905	TheMerlin Show.com Mr 1,000,000+	Economist.com Ec 10,372
Excite.com Ex 916	Inkscape.org Ik 195,541		Fark.com Fk 9,999	warehouse.carlh.com/comic Wh 250,607	Uncluttered.com Uc 505,029	Book Crossing.com Bk 39,254	FreeBSD.org Bd 51,037	Textfiles.com Tx 25,651	eff.org Ff 30,140	bigempire.com/filthy Cr 55,869	Venganza.org 75,367	GrokLaw.net Gk 338,403	Orkut.com Or 4,600	Escapepod.org Ep 1,000,000+	Radio Lab Rl 1,000,000+	strangerthings.tv St 1,000,000+	Londonist.com Ld 188,488

FaceBook.com **Fb** 29

I	Search Engines	IV	Aggregators	VII	Get Stuff	XIII	Blogs	XVII	Videos
II	Internet Tools	V	Webcomics	VIII	Operating Systems	XIV	Social Networking	XVIII	News
III	Site Ranking	VI	Productivity	IX-XII	Miscellaneous	XV-XVI	Podcasts		

WellingtonGrey.net **Wg** 207,654 ← Site URL ← Symbol ← Rank

Lists

- **list**: mutable sequence of objects
  - objects that make up a list are called its *elements*
  - *sequences* are ordered
    - lists, tuples and strings are examples of *sequences*
  - elements in a list are identified by their *index*
- simplest way to create a new list
  - enclose the elements in square brackets
  - separate the elements using comma's

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print(gases)
['He', 'Ne', 'Ar', 'Kr']
>>> print(gases[0], gases[-1])
He Kr
```

UNIVERSITEIT GENT

Lists

- **list**: mutable sequence of objects

"Think of lists as 1-dimensional arrays or vectors, that resize automatically if needed."

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print(gases)
['He', 'Ne', 'Ar', 'Kr']
>>> print(gases[0], gases[-1])
He Kr
```

UNIVERSITEIT GENT

Lists

- **list**: mutable sequence of objects
  - "*mutable*" means that elements can be changed *in place*
  - this is not possible for strings
    - strings are *immutable*

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> gases[0] = 'Xe'
>>> gases[-1] = 'Rn'
>>> print(gases)
['Xe', 'Ne', 'Ar', 'Rn']
```

UNIVERSITEIT GENT

Lists

- **list**: mutable sequence of objects
  - "of objects" means that lists can hold objects of any type
    - even elements of mixed types
    - even lists of lists (*nested lists*)

"Strings are homogeneous, lists are heterogeneous."

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> atomicNumber = [2, 10, 18, 36]
>>> atomicMass = [4.00260, 20.179, 39.948, 83.80]
>>> helium = ['He', 'Helium', 2, 4.00260]
>>> gases = [['He', 2], ['Ne', 10], ['Ar', 18]]
```

UNIVERSITEIT GENT

Lists

- **list**: mutable sequence of objects
  - `[]` represents an empty list
  - like the numeric value **0** and the empty string `''`, the empty list is **`False`** in a boolean expression

```
>>> gases = []
>>> if gases:
...     print('There are gases in the list.')
... else:
...     print('There are no gases in the list.')
...
There are no gases in the list.
```

UNIVERSITEIT GENT

Lists

- **list**: mutable sequence of objects
  - lists contain objects and are objects themselves
    - can be assigned to variables
    - can be returned as the result of a function
    - can be passed as an argument to a function

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> def reversed(list):
...     return list[::-1]
...
>>> reversed(gases)
['Kr', 'Ar', 'Ne', 'He']
```

UNIVERSITEIT GENT

List methods

- string methods return a new string (*immutable*)
- list methods modify an existing list (*mutable*)

```
>>> metals = ['gold', 'iron', 'lead', 'gold']
```

method	functionality	example	result
`append`	add to the end of the list	`metals.append('tin')`	`['gold', 'iron', 'lead', 'gold', 'tin']`
`count`	count how often object appears in list	`metals.count('gold')`	`2`
`index`	find position of first occurrence of object in list (raises `ValueError` if not found)	`metals.index('iron')` `metals.index('sulphur')`	`1` `ValueError`
`insert`	insert object into the list	`metals.insert(2,'silver')`	`['gold', 'iron', 'silver', 'lead', 'gold']`
`remove`	remove first occurrence of object from the list	`metals.remove('gold')`	`['iron', 'lead', 'gold']`
`reverse`	reverse the list in place	`metals.reverse()`	`['gold', 'lead', 'iron', 'gold']`
`sort`	sort the list in place	`metals.sort()`	`['gold', 'gold', 'iron', 'lead']`

List methods

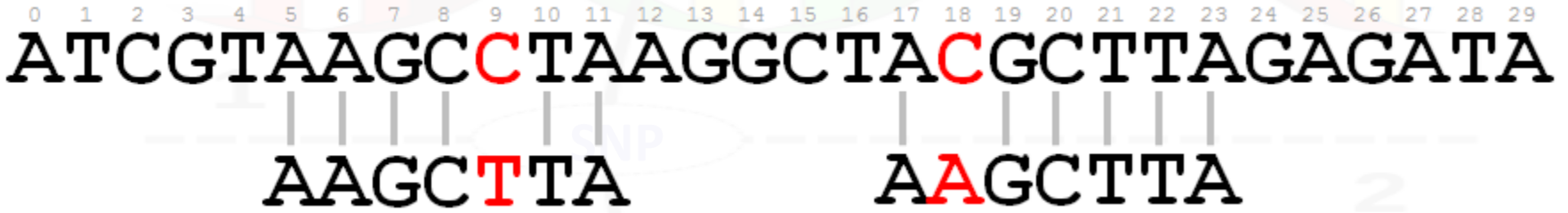
- **`index()`** method raises an error if object was not found
- all methods work *in place*
  - they modify the original list (except **`count()`** and **`index()`**)
  - they return **`None`** as a result (except **`pop()`**)

```
>>> metals = ['gold', 'iron', 'lead', 'gold']
>>> metals.index('sulphur')
ValueError: list.index(x): x not in list
>>> metals.sort()
>>> print(metals)
['gold', 'gold', 'iron', 'lead']
>>> metals = metals.reverse()          # error !!
>>> print(metals)
None
```

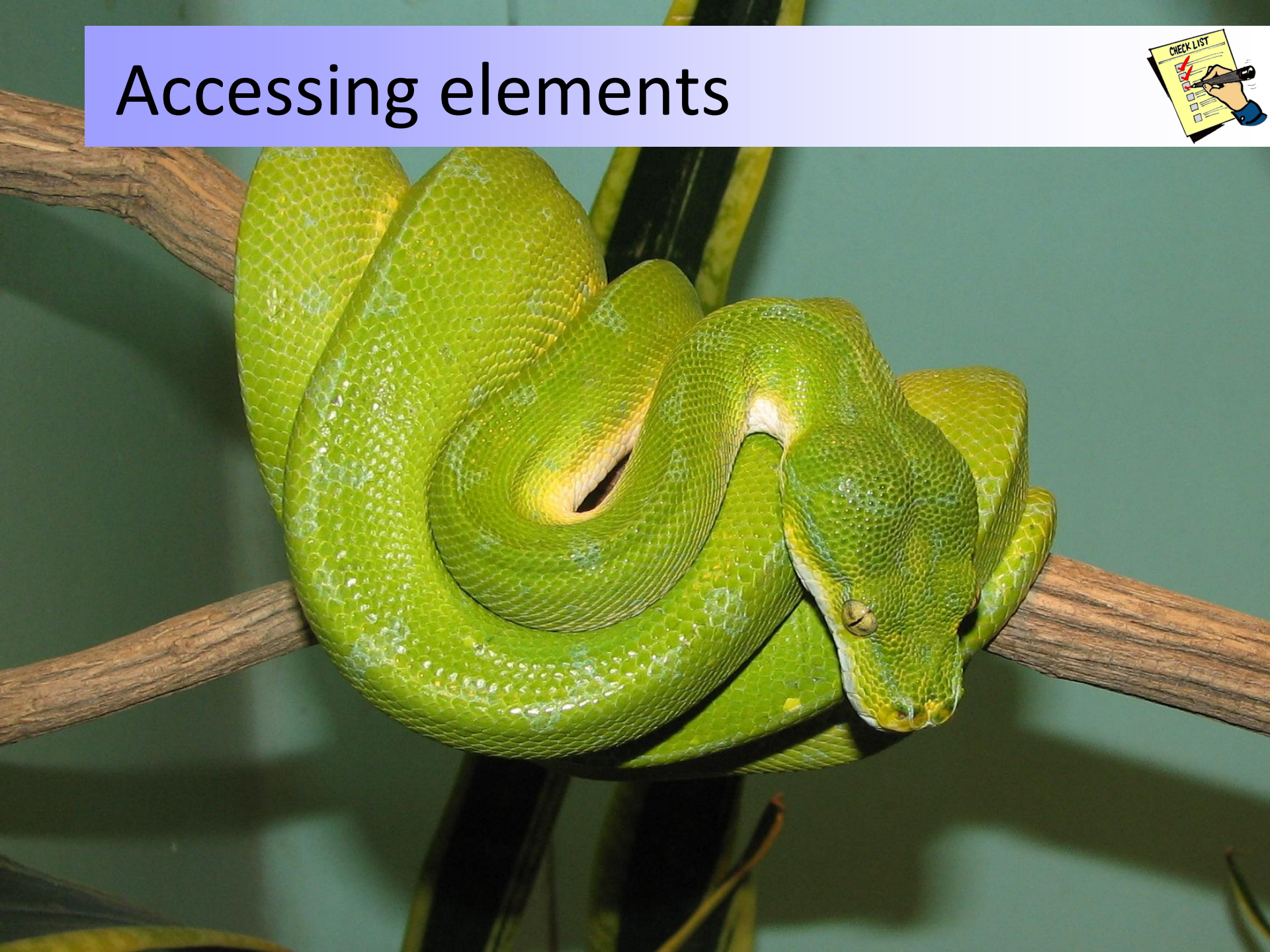
UNIVERSITEIT GENT

SNPs


CHECK LIST


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
ATCGTAAGCCTAAGGCTACGCTTAGAGATA
AAGCTTA
AAGCTTA

Accessing elements



Accessing elements

- same as for accessing characters of a string (*bracket operator*)
  - indexing: access individual elements
    - integer expression between square brackets

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print(gases)
['He', 'Ne', 'Ar', 'Kr']
>>> print(gases[0])
He
>>> print(gases[-1])
Kr
>>> print(gases[4])
IndexError: list index out of range
>>> gases[-5]
IndexError: list index out of range
```

UNIVERSITEIT GENT

Accessing elements

- same as for accessing characters of a string (*bracket operator*)
  - indexing: access individual elements
    - integer expression between square brackets

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> gases[9 - 8]
'Ne'
>>> gases[1.0]
TypeError: list indices must be integers, not float
>>>
>>> gases = [['He', 2], ['Ne', 10], ['Ar', 18]]
>>> gases[1]
['Ne', 10]
>>> gases[1][0]
'Ne'
```

UNIVERSITEIT GENT

Accessing elements

- same as for accessing characters of a string (*bracket operator*)
  - indexing: access individual elements
    - integer expression between square brackets
    - **`list[i] = obj`**
      - assigns object **`obj`** to position **`i`** in **`list`**
      - lists are mutable!!

```
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print(gases)
['He', 'Ne', 'Ar', 'Kr']
>>> gases[0] = 'H'
>>> gases[-1] = 'Xe'
>>> print(gases)
['H', 'Ne', 'Ar', 'Xe']
```

UNIVERSITEIT GENT

Accessing elements

- same as for accessing characters of a string (*bracket operator*)
  - indexing: access individual elements
    - integer expression between square brackets
    - **`list[i] = obj`**
    - **`append()`** method: add element at the end of the list

```
>>> gases = ['H', 'He', 'Ne', 'Ar']
>>> gases.append('Kr')
>>> gases.append('Xe')
>>> gases.append('Rn')
>>> gases.append('Uuo')
>>> print(gases)
['H', 'He', 'Ne', 'Ar', 'Kr', 'Xe', 'Rn', 'Uuo']
```

UNIVERSITEIT GENT

Accessing elements

- same as for accessing characters of a string (*bracket operator*)
  - indexing: access individual elements
  - slicing: generate subsequence of elements from list

```
>>> gases = ['H','He','Ne','Ar','Kr','Xe','Rn']
>>> gases[1:3]
['He', 'Ne']
>>> gases[:4]
['H', 'He', 'Ne', 'Ar']
>>> gases[4:]
['Kr', 'Xe', 'Rn']
>>> gases[:]
['H', 'He', 'Ne', 'Ar', 'Kr', 'Xe', 'Rn']
>>> gases[::2]
['H', 'Ne', 'Kr', 'Rn']
```

UNIVERSITEIT GENT

Traverse list elements

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one

traversal1.py

```
gases = ['H','He','Ne','Ar','Kr','Xe','Rn','Uuo']

i = 0
while i < 8:
    print(gases[i])
    i += 1
```

UNIVERSITEIT GENT

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one
  - built-in function **`len()`** determines the number of elements in a list

```
>>> len(['spam!', 1, [1, 2, 3],
...      ['Brie', 'Roquefort', 'Pol le Veq']])
4
```

traversal2.py

```
gases = ['H','He','Ne','Ar','Kr','Xe','Rn','Uuo']

i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

UNIVERSITEIT GENT

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one
  - built-in function **`len()`** determines the number of elements in a list

traversal3.py

```
gases = ['H','He','Ne','Ar','Kr','Xe','Rn','Uuo']

for i in range(len(gases)):
    print(gases [i])
```

UNIVERSITEIT GENT

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one
  - built-in function **`len()`** determines the number of elements in a list
  - lists are iterable objects
    - elements can be traversed using a for loop

traversal4.py

```
gases = ['H','He','Ne','Ar','Kr','Xe','Rn','Uuo']

for gas in gases:
    print(gas)
```

UNIVERSITEIT GENT

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one
  - built-in function **`len()`** determines the number of elements in a list
  - lists are iterable objects

traversal5.py

```
numbers = [1, 2, 3, 4, 5]

for index in range(len(numbers)):
    numbers[index] = numbers[index] ** 2

print(numbers)
```

UNIVERSITEIT GENT

Traverse list elements

- **list traversal**: processing pattern that iterates over the elements of a list one by one
  - built-in function **`len()`** determines the number of elements in a list
  - lists are iterable objects
    - built-in function **`enumerate()`** returns an iterator that both gives the index and the value of an element

traversal6.py

```
numbers = [1, 2, 3, 4, 5]

for index, number in enumerate(numbers):
    numbers[index] = number ** 2

print(numbers)
```

UNIVERSITEIT GENT

List membership

- ***object* in *list***
  - boolean expression is **`True`** if the object occurs in the list

```
>>> from category import category
>>> category('Po')
'semiconductor'
>>> category('Cr')
'metal'
```

category.py

```
def category(element):
    if element in ['He','Ne','Ar','Kr','Xe','Rn','Uuo']:
        return 'noble gas'
    elif element in ['B','Si','Ge','As','Sb','Te','Po','At','Uus']:
        return 'semiconductor'
    elif element in ['H','C','N','O','F','P','S','Cl','Se','Br','I']:
        return 'non-metal'
    else:
        return 'metal'
```

List membership

- ***object* in *list***
  - boolean expression is **`True`** if the object occurs in the list

```
>>> from category2 import category
>>> category('Cl')
'non-metal'
>>> category('Xe')
'noble gas'
```

category2.py

```
def category(element):
    if element in 'HeNeArKrXeRnUuo':
        return 'noble gas'
    elif element in 'BSiGeAsSbTePoAtUus':
        return 'semiconductor'
    elif element in 'HCNOFPSClSeBrI':
        return 'non-metal'
    else:
        return 'metal'
```

UNIVERSITEIT GENT

List operations

- operator **+**: concatenates lists
- operator *****: repeates a list a given number of times

```
>>> element = 'carbon'
>>> mass = '14'
>>> print(element + '-' + mass)
carbon-14
>>> lanthanides = ['Ce', 'Pr', 'Nd']
>>> actinides = ['Th', 'Pa', 'U']
>>> all = lanthanides + actinides
>>> print(all)
['Ce', 'Pr', 'Nd', 'Th', 'Pa', 'U']
>>> print(actinides * 3)
['Th', 'Pa', 'U', 'Th', 'Pa', 'U', 'Th', 'Pa', 'U']
```

UNIVERSITEIT GENT

List operations

- operator **+**: concatenates lists
- operator *****: repeates a list a given number of times
- **`list(string)`** : creates a list with characters of the string

```
>>> actinides + 'NP'
TypeError: can only concatenate list (not "str") to list
>>> actinides + ['NP']
['Th', 'Pa', 'U', 'NP']
>>> water = 'H2O'
>>> print(list(water))
['H', '2', 'O']
>>> actinides
['Th', 'Pa', 'U']
>>> '-'.join(actinides)
'Th-Pa-U'
```

UNIVERSITEIT GENT

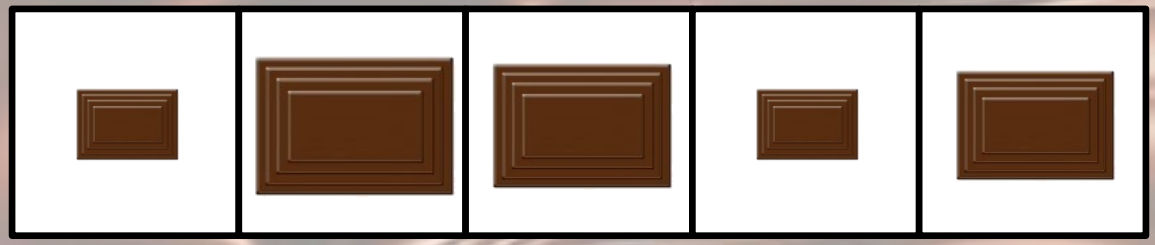
Integer iterators

- **`range()`** function
  - negative step size → value of first argument should be greater than value of second argument

```
>>> list(range(1, 5))
[1, 2, 3, 4]
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(20, 4, -5))
[20, 15, 10, 5]
>>> list(range(10, 20, -5))
[]
```

UNIVERSITEIT GENT

Chocolate bars



2 9 8 2 7

Mutability

Mutability

- strings are *immutable*
- lists are *mutable*
  - **item assignment**: assignment to an element of a list

```
>>> element = 'cesium'
>>> element[2] = 'r'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> element = list(element)
>>> print(element)
['c', 'e', 's', 'i', 'u', 'm']
>>> element[2] = 'r'
>>> print(element)
['c', 'e', 'r', 'i', 'u', 'm']
>>> ''.join(element)
'cerium'
```

UNIVERSITEIT GENT

Mutability

- strings are *immutable*
- lists are *mutable*
  - **slice assignment**: assignment to a list slice

```
>>> print(element)
['c', 'e', 'r', 'i', 'u', 'm']
>>> element[1] = 'h'
>>> element[3:5] = ['o', 'o']
>>> print(element)
['c', 'h', 'r', 'o', 'o', 'm']
>>> ''.join(element)
'chroom'
>>> element[4:4] = ['m', 'o', 's', 'o']
>>> print(element)
['c', 'h', 'r', 'o', 'm', 'o', 's', 'o', 'o', 'm']
>>> ''.join(element)
'chromosoom'
```

UNIVERSITEIT GENT

Delete elements

- replace slice by empty list
- keyword **`del`**

```
>>> element = list('ytterbium')
>>> element[:2] = []
>>> print(element)
['t', 'e', 'r', 'b', 'i', 'u', 'm']
>>> element[:1] = []
>>> print(element)
['e', 'r', 'b', 'i', 'u', 'm']
>>> element = list('ytterbium')
>>> del element[:2]
>>> print(element)
['t', 'e', 'r', 'b', 'i', 'u', 'm']
>>> del element[:1]
>>> print(element)
['e', 'r', 'b', 'i', 'u', 'm']
```

UNIVERSITEIT GENT

Ytterby, the village in Sweden which has 4 elements in the Periodic Table named after this one single town — yttrium(Y), erbium(Er), terbium(Tb) and ytterbium(Yb).

Circadial permutator

1	2	3	4	5	6
					
3	6	2	5	4	1

Homework (next lecture)

- course book
  - read chapter 8 (modules)
- read problem description of demo exercises
  - Manhattan
  - Hermit crabs
  - Minesweeper

UNIVI
GENT

Homework (hands-on sessions)

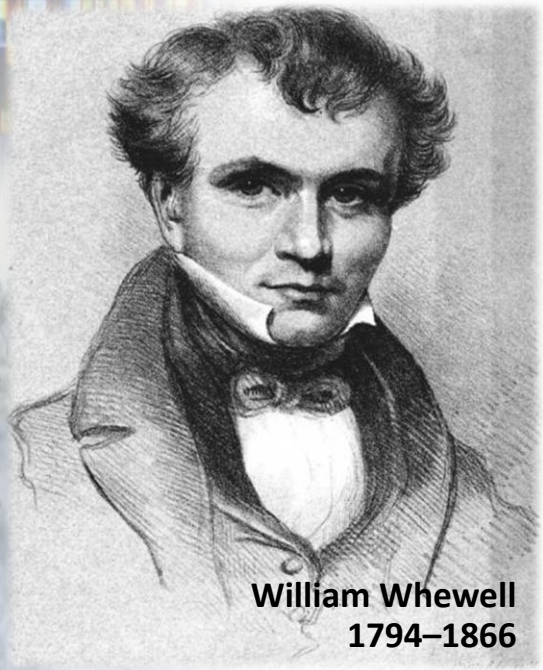
- Python help
  - read through list methods: help(list)
- hands-on session next week
  - Monday: finish mandatory exercises (series 05)

UNIVI
GENT

Remarks or questions?

UNIVERSITEIT GENT

The sky is the limit …



William Whewell
1794–1866

"Every failure is a step towards success."

— William Whewell

UNIVERSITEIT GENT