

Computational thinking

- *How to think like a computer scientist*
<https://openbookproject.net/thinkcs/>
 - Precision in formulating descriptions
 - Problem solving

Our tasks in learning Python

- Introduction into basic concepts

Computational thinking

- *How to think like a computer scientist*
<https://openbookproject.net/thinkcs/>
 - Precision in formulating descriptions
 - Problem solving

Our tasks in learning Python

- Introduction into basic concepts
- Running for corpus collection

Computational thinking

- *How to think like a computer scientist*
<https://openbookproject.net/thinkcs/>
 - Precision in formulating descriptions
 - Problem solving

Our tasks in learning Python

- Introduction into basic concepts
- Running for corpus collection
- Text annotation: genres and emotions

Computational thinking

- *How to think like a computer scientist*
<https://openbookproject.net/thinkcs/>
 - Precision in formulating descriptions
 - Problem solving

Our tasks in learning Python

- Introduction into basic concepts
- Running for corpus collection
- Text annotation: genres and emotions
- Keyword analysis

Basic terms

A program *a sequence of precise instructions that enables a computer to perform a specific task*

Basic terms

A program *a sequence of precise instructions that enables a computer to perform a specific task*

Bugs *programming errors*

Basic terms

A program *a sequence of precise instructions that enables a computer to perform a specific task*

Bugs *programming errors*

Debugging *finding and fixing programming errors*

Basic terms

A program *a sequence of precise instructions that enables a computer to perform a specific task*

Bugs *programming errors*

Debugging *finding and fixing programming errors*

Syntactic errors *when the syntax of the code is incorrect*
e.g. `1+"2"` is not legal

```
1 + "2"
```

```
-----  
TypeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-1-db092cb74d2d> in <module>
```

```
----> 1 1 + "2"
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Basic terms

A program *a sequence of precise instructions that enables a computer to perform a specific task*

Bugs *programming errors*

Debugging *finding and fixing programming errors*

Syntactic errors *when the syntax of the code is incorrect*
e.g. `1+"2"` is not legal

```
1 + "2"
```

```
-----
TypeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-1-db092cb74d2d> in <module>
```

```
----> 1 1 + "2"
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Semantic errors *the program does not do what you want it to do*

Basic concepts

Data type A class of data:

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Variable a storage location (a name and data)

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Variable a storage location (a name and data)

Function A piece of code that can be re-used

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Variable a storage location (a name and data)

Function A piece of code that can be re-used

Method A function that is associated with an object

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Variable a storage location (a name and data)

Function A piece of code that can be re-used

Method A function that is associated with an object

Class a template of an object:

Basic concepts

Data type A class of data:

- integers, floating points, strings, lists, dictionaries and **objects**
- Each data type defines operations that can (and thereby cannot) be done on the data

Variable a storage location (a name and data)

Function A piece of code that can be re-used

Method A function that is associated with an object

Class a template of an object:

- defines how variables, functions, and methods work together and what we can do with them

Python

- First released in 1991 by Guido van Rossum

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:

```
for headword in dictionary:  
    print(headword,dictionary[headword])
```

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:
for headword in dictionary:
 print(headword,dictionary[headword])
- Block of code is grouped by indentation

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:
for headword in dictionary:
 print(headword,dictionary[headword])
- Block of code is grouped by indentation

Python scripts files which can be executed by Python

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:
for headword in dictionary:
 print(headword,dictionary[headword])
- Block of code is grouped by indentation

on scripts files which can be executed by Python

otebooks scripts with a record of interaction

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:

```
for headword in dictionary:
    print(headword,dictionary[headword])
```

- Block of code is grouped by indentation

on scripts files which can be executed by Python

otebooks scripts with a record of interaction

- File extensions: .py (for scripts), .ipynb (for notebooks)

Python

- First released in 1991 by Guido van Rossum
- Current version: we will use version 3.10
- Name based on the British comedy group Monty Python
- Easily readable syntax:

```
for headword in dictionary:  
    print(headword,dictionary[headword])
```

- Block of code is grouped by indentation

Python scripts files which can be executed by Python

Python notebooks scripts with a record of interaction

- File extensions: .py (for scripts), .ipynb (for notebooks)
- Google Colab/Jupyter Labs for the notebook interface

Hello world

```
s = "Hello world"
print(s)
print(s.upper())
print(s.split())
print(s.split("l"))
```

Variable s

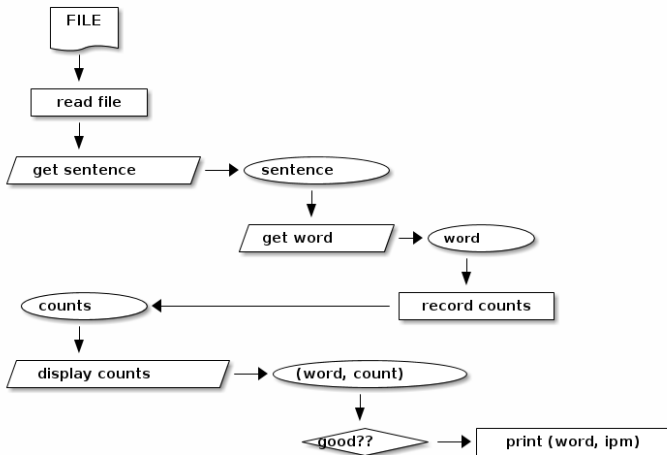
Type string

Operator = (assignment)

Function print()

Methods upper(), split(), find(), startswith()...

- Now open an empty notebook, type the commands at the top as cells and execute them



Word frequency distribution

```
import re
text = requests.get("https://ssharoff.github.io/modl5007/par
text = text.lower()
corpus_count = 0
dictionary = {}
for sentence in text.split(". "):
    for word in sentence.split(" "):
        corpus_count += 1
        word = re.sub("[^a-z-]+","",word)
        if word in dictionary: dictionary[word] += 1
        else: dictionary[word] = 1
for word in dictionary:
    ipm = dictionary[word] / ( corpus_count / 1000000 )
    if ipm > 1000:
        print(word, ipm)
```

Data types

- Integer: $i = 42$

Data types

- Integer: $i = 42$
- Float: $\pi = 3.14$

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`
- Testing: `s.startswith("My")`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`
- Testing: `s.startswith("My")`
- Searching: `s.find("eels") → 25`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`
- Testing: `s.startswith("My")`
- Searching: `s.find("eels") → 25`
- Slicing from the start: `s[:2] → "My"`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`
- Testing: `s.startswith("My")`
- Searching: `s.find("eels") → 25`
- Slicing from the start: `s[:2] → "My"`
- Slicing to the end: `s[25:] → "eels"`

Data types

- Integer: `i = 42`
- Float: `pi = 3.14`
- String: `s = "My hovercraft is full of eels"`
- Difference between number and string:
`"3.14" ≠ 3.14`
- Testing: `s.startswith("My")`
- Searching: `s.find("eels") → 25`
- Slicing from the start: `s[:2] → "My"`
- Slicing to the end: `s[25:] → "eels"`
- Zero indexing: `s[0] → "M"; s[1] → "y"`
Like house floor counting: first floor

Basic operators

- Arithmetic operators: $2+2$; $5-2$; $7*3$; $5/2$
Also for strings "Hello "+"world"

Basic operators

- Arithmetic operators: $2+2$; $5-2$; $7*3$; $5/2$
Also for strings `"Hello "+"world"`
- Assignment operators: `a = 2`; `b = 2`
`a = b + 2`
`a += 2`

Basic operators

- Arithmetic operators: $2+2$; $5-2$; $7*3$; $5/2$
Also for strings "Hello "+"world"
- Assignment operators: $a = 2$; $b = 2$
 $a = b + 2$
 $a += 2$
- Comparison operators: $a > 2$
 $(b \geq 2 \text{ and } 7*3 == 21)$
 $\text{not } 5/2 == 2$; $\text{int}(5/2) == 2$

Basic operators

- Arithmetic operators: $2+2$; $5-2$; $7*3$; $5/2$
Also for strings "Hello "+"world"
- Assignment operators: $a = 2$; $b = 2$
 $a = b + 2$
 $a += 2$
- Comparison operators: $a > 2$
 $(b \geq 2 \text{ and } 7*3 == 21)$
 $\text{not } 5/2 == 2$; $\text{int}(5/2) == 2$
- Membership operator: $a \text{ in } [5, 6, 7]$

Variables

- Legal naming conventions: standard characters, numbers (not at the start) and underscores
- Names need to be informative to reflect the logic of your script
- Variable names are case-sensitive:

CamelCase

Title_Case

snake_case

Q What is the difference between:

```
favorite_color = "blue"  
favorite_color == "blue"  
favorite_color = blue  
favorite_color == blue
```

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input
- It returns zero or more arguments as output

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input
- It returns zero or more arguments as output
- Functions can be:

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input
- It returns zero or more arguments as output
- Functions can be:
 - built-in: defined in the python standard
`print()`, `int()`, `str()`, `len()`, `open()`

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input
- It returns zero or more arguments as output
- Functions can be:
 - built-in: defined in the python standard
`print()`, `int()`, `str()`, `len()`, `open()`
 - user-defined

```
def compute_keywords(your_corpus, reference..):  
    do something  
    return result
```

Functions

- A function is a sequence of statements that performs a task:
`sin($\pi/2$)` or `print(word, ipm)`
- It takes zero or more arguments as input
- It returns zero or more arguments as output
- Functions can be:
 - built-in: defined in the python standard
`print()`, `int()`, `str()`, `len()`, `open()`
 - user-defined

```
def compute_keywords(your_corpus, reference..):  
    do something  
    return result
```
 - imported from libraries:
`import LibraryName`

Control structures and functions

- Find the minimum of two numbers:
25 or 7?

Control structures and functions

- Find the minimum of two numbers:
25 or 7?
- Expressing a condition:

```
if a<b:  
    then a  
    else b
```

Control structures and functions

- Find the minimum of two numbers:
25 or 7?
- Expressing a condition:

```
if a < b:  
    then a  
    else b
```

- Writing a function:

```
def min(a, b):  
    if a < b:  
        return a  
    else:  
        return b
```

Control structures and functions

- Find the minimum of two numbers:
25 or 7?

- Expressing a condition:

```
if a < b:  
    then a  
    else b
```

- Writing a function:

```
def min(a, b):  
    if a < b:  
        return a  
    else:  
        return b
```

- Find the minimum of three numbers:
7 or 3 or 25?

Compute the minimum of three numbers

- Expanding the case of two numbers:

```
def min3(a, b, c):  
    if a < b:  
        if a < c: return a  
        else: return c  
    else:  
        if b < c: return b  
        else: return c
```


Compute the minimum of three numbers

- Expanding the case of two numbers:

```
def min3(a, b, c):  
    if a < b:  
        if a < c: return a  
        else: return c  
    else:  
        if b < c: return b  
        else: return c
```

- A more elegant way:

```
def min3(a, b, c):  
    if a < b:  
        return min(a,c)  
    else  
        return min(b,c)
```

Compute the minimum of three numbers

- Expanding the case of two numbers:

```
def min3(a, b, c):  
    if a < b:  
        if a < c: return a  
        else: return c  
    else:  
        if b < c: return b  
        else: return c
```

- A more elegant way:

```
def min3(a, b, c):  
    if a < b:  
        return min(a,c)  
    else  
        return min(b,c)
```

Homework Write min4(a,b,c,d)

Import statement

- Importing a module for the current script

For example for regular expressions:

```
import re
re.findall(regex, string)
re.findall("[aeiou]", "Monty Python") \to "Mo", "ho"
re.sub("[^a-z-]+", "", "Monty Python") \to "MontyPython"
```

Import statement

- Importing a module for the current script

For example for regular expressions:

```
import re
re.findall(regex, string)
re.findall("[aeiou]", "Monty Python") \to "Mo", "ho"
re.sub("[^a-z-]+", "", "Monty Python") \to "MontyPython"
```

- Trafilatura for web scraping:

```
import trafilatura
from trafilatura.spider import focused_crawler
url_list=focused_crawler(start_url, max_seen_urls=10,
                          max_known_urls=50)
```

Lists

- A list is a sequence of objects:
a = ["Once", "upon", "a", "time"]
a = "Monty Python".split()

Lists

- A list is a sequence of objects:
a = ["Once", "upon", "a", "time"]
a = "Monty Python".split()
- An empty list
emptyList = []

Lists

- A list is a sequence of objects:

```
a = ["Once", "upon", "a", "time"]  
a = "Monty Python".split()
```
- An empty list

```
emptyList = []
```
- Order of items is important:

```
["Monty", "Python"] ≠ ["Python", "Monty"]
```

Lists

- A list is a sequence of objects:

```
a = ["Once", "upon", "a", "time"]  
a = "Monty Python".split()
```
- An empty list

```
emptyList = []
```
- Order of items is important:

```
["Monty", "Python"] ≠ ["Python", "Monty"]
```
- Values can be repeated:

```
["Monty", "Monty", "Python"]
```


Lists

- A list is a sequence of objects:
`a = ["Once", "upon", "a", "time"]`
`a = "Monty Python".split()`
- An empty list
`emptyList = []`
- Order of items is important:
`["Monty", "Python"] ≠ ["Python", "Monty"]`
- Values can be repeated: `["Monty", "Monty", "Python"]`
- Zero indexing: `a[0] → "Monty"`, `a[1] → "Python"`

Lists

- A list is a sequence of objects:
`a = ["Once", "upon", "a", "time"]`
`a = "Monty Python".split()`
- An empty list
`emptyList = []`
- Order of items is important:
`["Monty", "Python"] ≠ ["Python", "Monty"]`
- Values can be repeated: `["Monty", "Monty", "Python"]`
- Zero indexing: `a[0] → "Monty"`, `a[1] → "Python"`
- last element: `a[-1]`
`"My hovercraft is full of eels".split()[-1] → "eels"`

Lists

- A list is a sequence of objects:
`a = ["Once", "upon", "a", "time"]`
`a = "Monty Python".split()`
- An empty list
`emptyList = []`
- Order of items is important:
`["Monty", "Python"] ≠ ["Python", "Monty"]`
- Values can be repeated: `["Monty", "Monty", "Python"]`
- Zero indexing: `a[0] → "Monty"`, `a[1] → "Python"`
- last element: `a[-1]`
`"My hovercraft is full of eels".split()[-1] → "eels"`
- Same slicing as with strings: `a[:2]`

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`
- Each key appears only once in a dictionary

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`
- Each key appears only once in a dictionary
- Access via a key: `d["parrot"]` → 8

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`
- Each key appears only once in a dictionary
- Access via a key: `d["parrot"]` → 8
- Test on membership: `"parrot" in d`

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`
- Each key appears only once in a dictionary
- Access via a key: `d["parrot"]` → 8
- Test on membership: `"parrot" in d`
- `for key in d:`
`d[key]`

Dictionary objects

- A dictionary is a collection of key-value pairs → not a sequence
- Written in curly brackets: `d = {"parrot": 8, "the": 30}`
`{"string": integer}`
- Each key appears only once in a dictionary
- Access via a key: `d["parrot"]` → 8
- Test on membership: `"parrot" in d`
- `for key in d:`
 `d[key]`
- `for key in d:`
 `ipm[key] = d[key] / (CorpusSize / 1000000)`

Debugging

- Programmes often do not behave as expected → debugging

Debugging

- Programmes often do not behave as expected → debugging
- Use print more often.

```
print(len(url_list))  
for url in url_list:  
    print(f"We are processing: {url}")
```

Debugging

- Programmes often do not behave as expected → **debugging**
- Use print more often.

```
print(len(url_list))  
for url in url_list:  
    print(f"We are processing: {url}")
```
- Note the possibility to include variables into strings:

```
print("We are processing: {url}")
```

Debugging

- Programmes often do not behave as expected → **debugging**
- Use print more often.

```
print(len(url_list))  
for url in url_list:  
    print(f"We are processing: {url}")
```
- Note the possibility to include variables into strings:

```
print("We are processing: {url}")
```
- assert stops execution if something is not right

```
assert len(url_list)>0, "Empty url list"
```

AI frameworks

- Various AI models can help you with writing code:
Ask ChatGPT, Claude, Copilot, Google Colab etc to:
Write a function which gets an HTML file as its parameter and uses the Trafilatura library to extract its text content.
- Use the right prompts: be as specific as possible
- Understanding their output and **your** ability to modify it
- Be inquisitive and ask AI models why a specific line behaves in this way
- Be liberal with your own commentaries: literate programming

Projects

- Think of mini-projects which involve:
 - data collection
 - annotation
 - terminology extraction, etc
- Run this in **two** languages
- Each mini-project for two-three people:
 - Dividing the tasks
 - Testing
 - Code review (another pair of eyes)