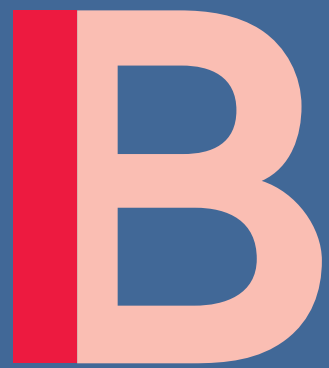
Data Structures and Algorithms

Live Class 6

Heikki Peura

h.peura@imperial.ac.uk



Today

1. **Recap**

2. Python:
   - Dictionaries and sets
   - Modules and libraries

3. Accessing data on the web
   - Scraping
   - APIs

Go to menti.com

What is the output?

```
1  x = 0
2  y = 3
3  z = y
4  while y >= 1:
5      y -= 1
6      x = x + y
7  print(x, y, z)
```

A. 6 0 3

B. 6 3 3

C. 5 1 1

D. 3 1 3

E. Something else

What is the output?

```
1 s = '01234'
2 t = 'y'
3 for c in s:
4     t = t + t
5 print(len(t))
```

A. 10

B. 15

C. 16

D. 32

E. I don’t know

Your classmates can halve the work of an $O(n^2)$ algorithm?

A. Excellent, you have reduced the runtime complexity to $O(n)$, the algorithm will be much faster.

B. Excellent, although the runtime complexity is the same, the algorithm will finish in around half the time.

C. That doesn't help us, you did not change the runtime complexity, so the algorithm still takes the same time to run.

D. That doesn't help us, while it reduces the runtime complexity to $O(n)$, that does not mean it will run faster.

E. I don't know

Lists are not always ideal

```
names = ['Jose', 'Iva', 'Misha', 'Pat']
favorite_music = ['Jimi Hendrix', 'Midori Takada', 'BTS', 'Igor Stravinsky']
```

```
music_data = [['Jose', 'Jimi Hendrix'], ['Iva', 'Midori Takada'],
              ['Misha', 'BTS'], ['Pat', 'Igor Stravinsky']]
```

How to look up the Misha's favorite music?

A dictionary maps keys to values

A dictionary allows easy lookups:

```
music_data = {'Jose':'Jimi Hendrix', 'Iva':'Midori Takada',
              'Misha':'BTS', 'Pat':'Igor Stravinsky'}
music_data['Misha'] # access value directly with key
```

This is based on *hashing* and is $O(1)$ on average.

A list maps integer indices to values. A dictionary maps (immutable) keys (integer, string, . . .) to values.

Working with dictionaries

```
music_data = {'Jose':'Jimi Hendrix', 'Iva':'Midori Takada',
        'Misha':'BTS', 'Pat':'Igor Stravinsky'}
music_data['Valerie'] = 'Johnny Cash'
music_data['Iva'] = 'Steve Reich'
for key in music_data:
    print(music_data[key])
for key, value in music_data.items():
    print(key, value)

# If key 'Woonam' is not in the dictionary, return default value Bach
music_data.get('Woonam', 'J.S. Bach')
```

Sets are collections of unique unordered items

```
berries = {'Raspberry', 'Cloudberry', 'Lingonberry'}
for berry in berries:
    print(berry)
berries.add('Blueberry')
berries.remove('Lingonberry')
print('Blueberry' in berries)
fruit = set()
fruit.add('Apple')
```

Go to menti.com

After the code runs, what is in desserts?

```
desserts = {'Cheesecake':10, 'Tiramisu': 3, 'Fruit': 2}
desserts['Fruit'] = 5
desserts['Apple pie'] = 1
```

A. 'Cheesecake': 10, 'Tiramisu': 3, 'Fruit': 2, 'Apple pie': 1

B. 'Cheesecake': 10, 'Tiramisu': 3, 'Fruit': 5, 'Apple pie': 1

C. 'Cheesecake': 10, 'Tiramisu': 3, 'Fruit': 2, 'Fruit': 5, 'Apple pie': 1

D. There will be an error

E. I don't know

After the code runs, what is in desserts?

```
desserts = {'Cheesecake':10, 'Tiramisu': 3, 'Fruit': 2}
new_orders = ['Tiramisu', 'Fruit']
for order in new_orders:
    desserts[order] += 1
```

A. 'Cheesecake': 10, 'Tiramisu': 3, 'Fruit': 2

B. 'Cheesecake': 10, 'Tiramisu': 4, 'Fruit': 4

C. 'Cheesecake': 10, 'Tiramisu': 4, 'Fruit': 3

D. There will be an error

E. I don't know

After the code runs, what is in desserts?

```
1  desserts = {'Cheesecake':10, 'Tiramisu': 3, 'Fruit': 2}
2  new_orders = ['Tiramisu', 'Fruit', 'Fruit', 'Tiramisu', 'Apple pie']
3  for order in new_orders:
4      desserts[order] += 1
```

A. 'Cheesecake': 10, 'Tiramisu': 3, 'Fruit': 2, 'Apple pie': 1

B. 'Cheesecake': 10, 'Tiramisu': 5, 'Fruit': 4

C. 'Cheesecake': 10, 'Tiramisu': 5, 'Fruit': 4, 'Apple pie': 1

D. There will be an error

E. I don't know

After the code runs, what is in desserts?

```
1  orders = ['Tiramisu', 'Fruit', 'Fruit', 'Tiramisu', 'Apple pie']
2  desserts = set(orders)
```

A. 'Cheesecake', 'Tiramisu', 'Fruit', 'Apple pie'

B. 'Tiramisu', 'Fruit', 'Fruit', 'Tiramisu', 'Apple pie'

C. 'Tiramisu', 'Fruit', 'Apple pie'

D. There will be an error

E. I don't know

Dividing code into modules

my_algorithms.py

```
def lin_search(L, x):
    for elem in L:
        if elem == x:
            return True
    return False
print('Running my_algorithms.py')
```

working.py

```
import my_algorithms as alg # "as" part is optional

from my_algorithms import lin_search # specific function

L = [1, 2, 2, -2, 9]
found_10 = alg.lin_search(L, 10)
found_2 = lin_search(L, 2)
```

When you import, Python will run all of `my_algorithms.py`

Using code from libraries

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

There are thousands of Python libraries

- Anaconda packages “the big ones”
- Command line: `conda list`
- Update a library:
  1. Close everything Python-related (Spyder, Notebook...)
  2. `conda update package_name`
  3. Be careful: updating may sometimes remove old functionality

Today

Python:

- Dictionaries and sets
- Modules and libraries

Accessing data on the web:

- Scraping
- APIs

The web is full of data and services

Quick and repeated access could be useful for:

- Analysing tweets / government spending / ???
- Getting data to/from Google maps / AWS / Bloomberg?
- Finding an apartment in London / best coffee close to you?

We can use APIs or scraping for access

A web page is just text

Most pages are in HTML (HyperText Markup Language)

The difference:

```
Hello, world!

Hello again.

An example page: http://www.example.com/
```

```
<p>Hello, world!</p>

<p>Hello again.</p>

<p><a href="http://www.example.com/">An example page</a></p>
```

HTML tags tell browsers how to display content

Tags are within angle brackets: `<p>` is starting tag, and `</p>` is ending tag. Inside there is the *paragraph* content.

```
<p>Hello, world!</p>
```

Tags can be nested:

```
<p>Hello, <strong>world!</strong></p>

<p>Hello again.</p>
```

`<p>` and `<strong>` are parent and child tags, and the two `<p>` tags are sibling tags.

Tags can have attributes

For example, the hyperlink tag `<a>` specifies the target URL as the `href` attribute in the `opening` tag:

```
<p><a href="http://www.example.com/">An example page</a></p>

<p id="myid">Hey there</p>
```

Web sites use custom attributes to make content look fancy with CSS (cascading style sheets): for example, change how content with the id `myid` looks.

We know how to parse text with Python

Websites use HTML tag structure to look nice

- We can exploit these tags to parse their contents
- For example, find all links in a web page through `<a href...>`
- Great libraries exist!

Parsing HTML with Python

We will use a library called Beautiful Soup 4 (BS)

1. Detective work to identify what we want in the page (right tags)
2. Looping through tags with BS

```
from bs4 import BeautifulSoup
html = '<p>Hello, world!</p> <p>Hello again.</p>'
soup = BeautifulSoup(html, 'lxml') # lxml is a html parser BS uses
print(soup.text)
```

The `soup` variable contains a `BeautifulSoup` object.

- The `soup.text` attribute contains the text content

Finding things in soup

Let's find all tags:

```
>>> paragraphs = soup.find_all('p') # find all <p> tags
>>> paragraphs
[<p>Hello, world!</p>, <p>Hello again.</p>]
>>> type(paragraphs) # augmented version of a list
bs4.element.ResultSet
>>> first = paragraphs[0]
>>> first.name
'p'
>>> first.text
'Hello, world!'
>>> type(first)
bs4.element.Tag
```

Browsing with Python

We will use a library called requests:

```
import requests
r = requests.get('http://www.example.com')
```

What is `r`? An object with attributes. For example:

```
r.ok # was the access attempt successful
r.text # raw HTML
```

Together with soup

Let's go to example.com:

```
import requests
from bs4 import BeautifulSoup
r = requests.get('http://www.example.com')
soup = BeautifulSoup(r.text, 'lxml')
links = soup.find_all('a')
```

Get link locations from tag attributes:

```
>>> first_link = links[0]
>>> first_link.text
'More information...'
>>> first_link.attrs # dictionary of attributes
{'href': 'http://www.iana.org/domains/example'}
>>> first_link.attrs['href']
'http://www.iana.org/domains/example'
```

Soup workflow

1. Use requests to get HTML
2. Create a soup object with BS
3. Inspect the HTML in browser to find the tags you need
4. BS: `find` (first) and `find_all` methods are often useful
5. BS: `text` and `attrs` attributes are often useful
6. Get data into a Python data structure (list, dictionary, ...)
7. Save to a file

BS has a great documentation online.

Scraping ethics

Websites don't always like scraping

- Be polite!
  - Terms for access
  - `from time import sleep`
- Better to use APIs if available
  - Bonus: more robust access and cleaner data

What are APIs?

Application programming interface

A controlled way to access a service. Widely used:

- Google maps, TfL, Spotify, NY Times, Gmail, . . .
- Easy and robust access to data in nice form
- Typically need to register to use
- Limitations: types of data, access frequency

What do you get from an API?

Data often in JSON or XML format.

- Both forms are common for semi-structured data
- Worth getting to know for analytics
- Great Python libraries for parsing both

Today: JSON (JavaScript Object Notation, *'Jason'*)

JSON, have we met before?

In general:

```
{"countries":
  [
  {"Germany": "Berlin"},
  {"France": "Paris"},
  {"Italy": "Rome"}
  ]
}
```

A tweet (some fields):

```
{"source": "Twitter for iPhone",
 "id_str": "815271067749060609",
 "text": "RT @realDonaldTrump: Happy Birthday @DonaldJTrumpJr!\nhttps://t.co/uRxyCD3hBz",
 "created_at": "Sat Dec 31 18:59:04 +0000 2016",
 "retweet_count": 9529,
 "in_reply_to_user_id_str": null,
 "favorite_count": 0,
 "is_retweet": true}
```

Why not all web services have public APIs

Hacker Challenge

Write a Python program that logs on to the Hub and downloads module files.

I recommend using the `selenium` library.

Details in the Challenge folder on the Hub.

Send me your solution by email before the last lecture. Prize(s) for the best solution(s).

Review

Dictionaries are useful to model mappings from keys to values.

- Finish exercises on the Hub

Optional exercises:

- JSON - Trump tweets
- HTML - The Guardian website

Coming up in **Session 7**: object-oriented programming and data structures