Day 5

Richèl Bilderbeek

Table of contents

1 Schedule

1.1 Breaks

Please take breaks: these are important for learning. Ideally, do something boring (1)!

1.2 Break schedule

- 10:00-10:15
- 11:00-11:15

- 12:00-13:00
- 14:00-14:15
- 15:00-15:15
- 16:00-16:15

2 Day 4

2.1 E. Review Day 4

- Do Studium quiz: Review Day 4
- `[Timer 10 mins]`

2.2 Discuss answers

- `[Discuss answers]`
- `[Discuss HackMD]`

Questions on this so far?

Write down in the HackMD!

3 Day 5

3.1 Problem

Imagine you have to write code to extract DNA sequences from a file. To do so, you need a function to detect if a string is a DNA sequence.

How would you do that?

3.2 Things we must agree on

```
# Strings that are correct
# "A"
# "ACGT"
# Strings that are incorrect
# ""
# "acgt"
# "nonsense"
```

3.3 Pseudocode of the answer

```
# Define my function: is_dna_sequence
#  (assume input is a string)
#  if string is empty, return False
#  iterator through each character
#    if the character is not an A,
#    nor a C, nor a G, nor a T,
#      then return False
#  return True
```

3.4 Can we do better?

Yes, we can!

We use regular expressions!

3.5 Wikipedia definition

> A regular expression (shortened as regex or regexp) is a sequence of characters that specifies a match pattern in text.
>
> Adapted from Wikipedia

3.6 Exercise: regex quiz

Do https://regexone.com lessons 1 to and including 10.

- Extra: also complete the rest of the tutorial

3.7 Question: Recap

It is a small language on its own indeed!

- `[Discuss notes https://regexone.com ]`

Questions on this so far?

- `[Discuss HackMD]`

Write down in the HackMD!

3.8 Exercise 1 - day 5

On Studium, do `Exercise 1 - day 5`.

- `[Timer 25 mins]`

Questions on this?

- `[Take a first look at the code]`
- `[Discuss HackMD]`

Write down in the HackMD!

4 Now us!

4.1 Importing the `re` module

To use regexes in Python, put this line at the top of your script:

```
import re
```

- Tip: The Python 're' documentation

4.2 Creating the pattern

A DNA sequence contains only 'A', 'C', 'G' and 'T's and has at least 1 one of these characters.

This would be the pattern:

```
"[ACGT]+"
```

- `[ACGT]`: must be among these characters
- `+` … at least once

4.3 Compiling regexes

```
dna_seq_pattern = re.compile("[ACGT]+")
print(dna_seq_pattern)
```

```
re.compile('[ACGT]+')
```

This results in a finite state machine with all `if` statements needed

4.4 Getting a match

```
print(dna_seq_pattern.match("A"))
```

```
<re.Match object; span=(0, 1), match='A'>
```

```
print(dna_seq_pattern.match("AACGCGT"))
```

```
<re.Match object; span=(0, 7), match='AACGCGT'>
```

```
print(dna_seq_pattern.match("nonsense"))
```

```
None
```

4.5 Getting a match

```
m = dna_seq_pattern.match("ACGT")
if m:
  print("This was a match!")
```

```
else:
  print("Nope, no DNA here :-(")
```

```
This was a match!
```

4.6 Getting no match

```
m = dna_seq_pattern.match("nonsense")
if m:
  print("This was a match!")
else:
  print("Nope, no DNA here :-(")
```

```
Nope, no DNA here :-(
```

4.7 Q: `is_chess_square` 1/2

4.8 Q: `is_chess_square` 2/2

Write the pattern to detect if a string is a chess square. Test that it works

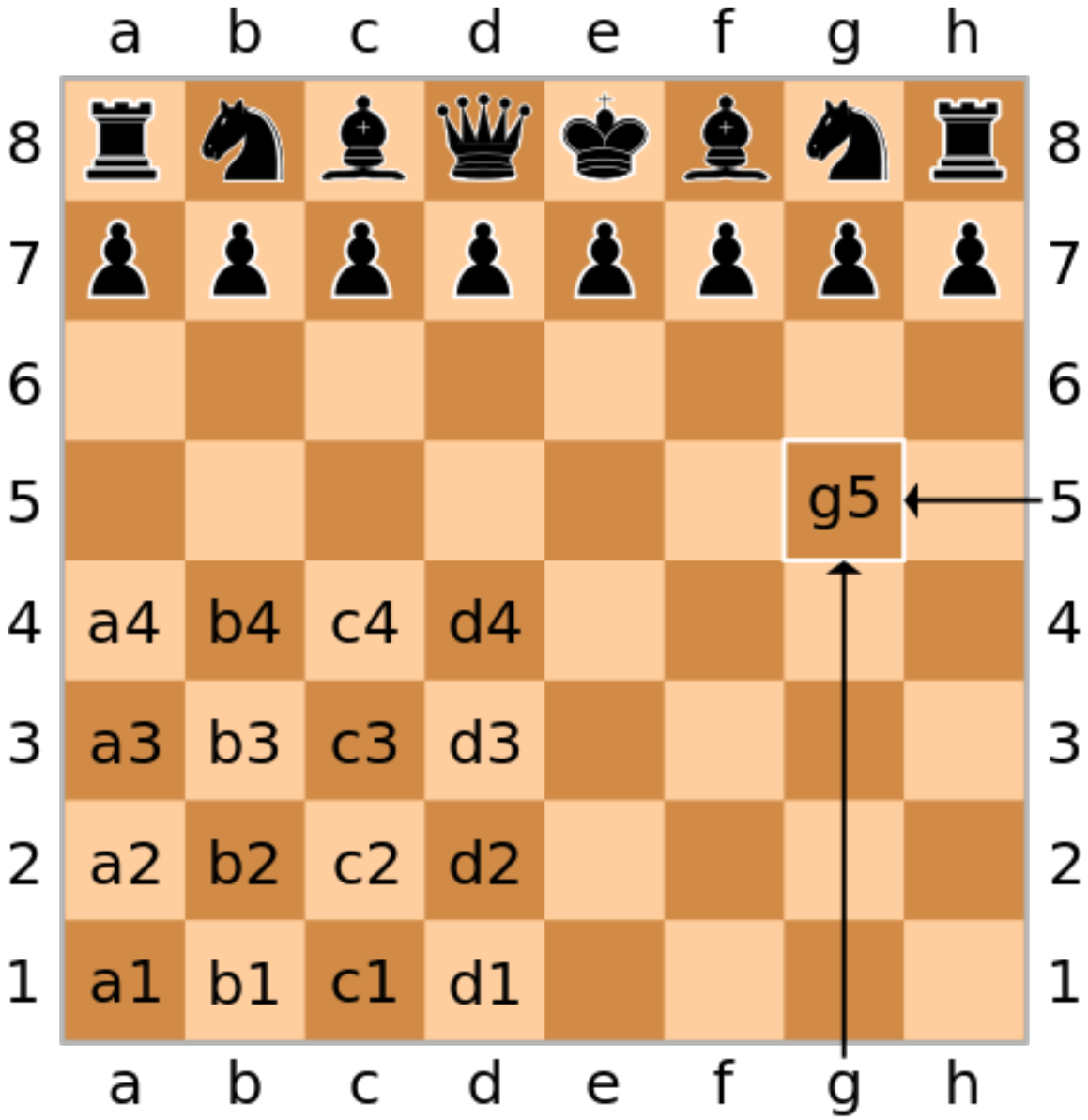
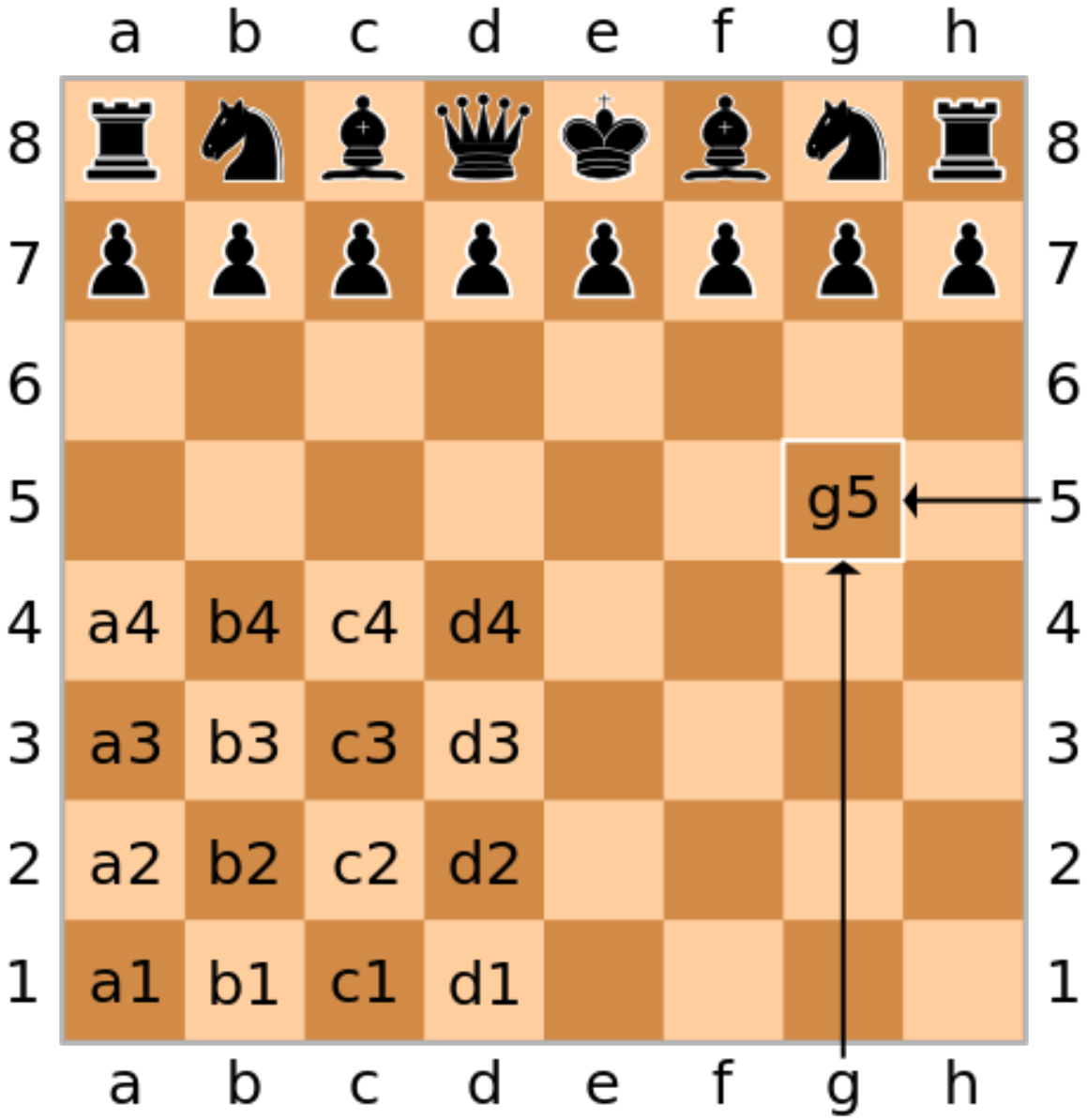


Figure 1: https://en.wikipedia.org/wiki/File:SCD_algebraic_notation.svg



4.9 A: `is_chess_square` 1/3

```
chess_square_pattern = \
  re.compile("[a-h][1-8]")
```

- The first character is `a`, `b` `c`, etc. to and including `h`

- The second character is `1`, `2` `3`, etc. to and including `8`
- Can you spot the mistake already?

4.10 A: `is_chess_square` 2/3

```
m = chess_square_pattern.match("d5")
if m:
  print("This was a match!")
else:
  print("Nope, no chess square here :-(")
```

```
This was a match!
```

4.11 A: `is_chess_square` 1/3

```
m = chess_square_pattern.match("ACGT")
if m:
  print("This was a match!")
else:
  print("Nope, no chess square here :-(")
```

```
Nope, no chess square here :-(
```

4.12 Putting `is_chess_square` in a function

```
def is_chess_square(s):
  p = re.compile("[a-h][1-8]")
  return chess_square_pattern.match(s)
```

4.13 Using `is_chess_square`

```
print(is_chess_square("a1"))
```

```
<re.Match object; span=(0, 2), match='a1'>
```

```
print(is_chess_square("nonsense"))
```

```
None
```

4.14 Testing `is_chess_square`

```
assert is_chess_square("a1")
assert not is_chess_square("nonsense")
```

4.15 Q: `is_chess_square` in your code

Put the `is_chess_square` function and tests in your code.

Confirm that everything works as expected on your computer!

Bonus: the pattern is imperfect. Can you find an example?

4.16 Putting `is_chess_square` in your code

- `[demonstrate this]`

Questions on this?

Write down in the HackMD!

4.17 Unexpected `is_chess_square`

```
if is_chess_square("a123"):
  print("This is unexpected!")
```

```
This is unexpected!
```

4.18 Problem `is_chess_square`

- The first character is `a`, `b` `c`, etc. to and including `h`
- The second character is `1`, `2` `3`, etc. to and including `8`
- **A chess square has two characters**

4.19 Q: Fix `is_chess_square`

Fix `is_chess_square`.

Use the Python 're' documentation

- Tip: see the search versus match section

4.20 A: Fix `is_chess_square`

```
def is_chess_square(s):
  p = re.compile("[a-h][1-8]")
  return p.fullmatch(s)
```

- Use `fullmatch`, instead of `search` or `match`
- `search` and `"^[a-h][1-8]$"` fails???

Questions on this?

Write down in the HackMD!

4.21 Print that `is_chess_square` works

```
print(is_chess_square("a1234"))
```

```
None
```

```
if is_chess_square("a1234"):
  print("WEIRD!")
else:
  print("Life is good!")
```

```
Life is good!
```

4.22 Testing `is_chess_square`

```
assert is_chess_square("a1")
assert not is_chess_square("a1234")
assert not is_chess_square("nonsense")
```

4.23 Questions about today's theory?

Write down in the HackMD!

5 Evaluation

5.1 Procedure 1/4: Why

Feedback is the 10th most impactful way to learn (2). Teachers need to learn too. We need your feedback! This is more important than 15 mins of lecture :-)

5.2 Procedure 2/4: Form

- Anonymously (hence this person is here)
  - Teachers and TAs must be away
  - Teacher will never ask for names
  - Person will never gives names
- Both HackMD and Google Forms
- Raw results will be on GitHub: please follow privacy rules!

5.3 Procedure 3/4: Code of Conduct

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

5.4 Procedure 4/4: What we'd like to know

- All you would like to share
- Need guidance? Share a good thing and a thing that can be improved on the course, course material, the teachers together/individually, etc.

Thanks! When done, it is fika! At 15:15, Pedro speaks!

6 Done!

Go home and rest :-)

7 Breaks

Are important. Please rest!

7.1 Break 1: 10:00-10:15

7.2 Break 2: 11:00-11:15

7.3 Break 3: 12:00-13:00

7.4 Break 4: 14:00-14:15

7.5 Break 5: 15:00-15:15

7.6 Break 6: 16:00-16:15

7.7 Done

References

1. Newport C. Deep work: Rules for focused success in a distracted world. Hachette UK; 2016.
2. Hattie J. Visible learning for teachers: Maximizing impact on learning. Routledge; 2012.