UNIVERSITÄT
ZU KÖLN

DEEP LEARNING – SESSION 2

WiSe 2024/2025

Janis Pagel

Institut für Digital Humanities

2024-10-17

01

SOLUTION EXERCISE 1

Discussion Exercise 1 I

- Was everyone able to push to GitHub?
- Exercise was for server on `http://compute.spinfo.uni-koeln.de`
  - You can also do the exercises on your personal laptop
  - You need to repeat adding your name and email adress to the git config and create a new SSH key and add it to GitHub

Discussion Exercise 1 II

- Replace “pagelj” with your own username

```
$ git clone git@github.com:IDH-Cologne-Deep-Learning-2024/Exercise-1.git
$ cd Exercise-1
$ git branch pagelj
$ git switch pagelj
$ touch shakespeare-sonnet-1.txt
    [Add sonnet text]
$ git add shakespeare-sonnet-1.txt
$ git commit -m "Add sonnet"
    [Add author information]
$ git add shakespeare-sonnet-1.txt
$ git commit -m "Add author information"
$ git log > log.txt
$ git add log.txt
$ git commit -m "Add logfile"
$ git push origin pagelj
```

02

GIT

Merging

- You can merge changes in one branch into another branch
- If the changes do not concern the same lines in a file, you can auto merge

test.txt on branch1

```
1 First line
2 Second line
3 Third line
```

test.txt on main

```
1 First line
2 Second line
```

```
$ git switch main
$ git merge branch1
Updating e411f63..3772873
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
$ git log
commit 37728739cdb35e235b9e862c5320f4f4e22849ca (HEAD -> main, branch1)
Author: Janis Pagel <janis.pagel@uni-koeln.de>
Date:   Wed Oct 16 20:55:03 2024 +0200

    Add code
```

Merge conflict I

- If you want to merge changes that concern the same line, you get a merge conflict
- You can manually edit the conflicting files and keep only the changes you want to keep
- or you can use a merge tool via “git mergetool”
  - Uses default merge tool on system

Merge conflict II

test.txt on branch1

```
1  First line
2  Second line in branch1
```

test.txt on branch2

```
1  First line
2  Second line in branch2
```

test.txt on main

```
1  First line
2  Second line
```

```
$ git switch main
$ git merge branch1
Updating d5412a2..59f1347
Fast-forward
 test.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git merge branch2
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflict III

test.txt on main after merge conflict

```
1  First line
2  <<<<<<< HEAD
3  Second line in branch1
4  ||||||| d5412a2
5  Second line
6  =======
7  Second line in branch2
8  >>>>>>> branch2
```

- “HEAD” refers to the branch we were on when issuing the commit (in this case “main”)
- <<<<<<<, |||||||, ======= and >>>>>>> are added to show were the different versions begin and end
- Replace everything between <<<<<<< and >>>>>>> with the version you want to keep (can also be something completely new)

Merge conflict IV

test.txt on main after merge conflict resolution

```
1  First line
2  New version of second line
```

```
$ git add test.txt
$ git commit
[main 899010f] Merge branch 'branch2'
$ git log
commit 899010f7b88946dac4873c2bdf368d0ec12f074d (HEAD -> main)
Merge: 59f1347 7865e8d
Author: Janis Pagel <janis.pagel@uni-koeln.de>
Date:   Wed Oct 16 20:38:00 2024 +0200

    Merge branch 'branch2'
```

- git automatically adds a commit message for the merge commit

Merge conflict V

- After merging, you can delete the merged branches
- You can also keep them for further changes if you wish

```
$ git branch -d branch1
Deleted branch branch1 (was 59f1347).
$ git branch -d branch2
Deleted branch branch2 (was 7865e8d).
```

Further topics in git

- gitignore (Chacon and Straub 2014, p. 32)
- rebase (Chacon and Straub 2014, p. 95)
- tags (Chacon and Straub 2014, p. 55)
- submodules (Chacon and Straub 2014, p. 298)
- hooks (Chacon and Straub 2014, p. 354)

03

PYTHON

Motivation

- Not a full-fledged Python course
- Assuming existing knowledge of Java or other programming language
- Python is very popular in data science and deep learning
- Many supporting libraries
- Some goals of Python: being nice to read, writing concise code
- `https://peps.python.org/pep-0008/`

First Python Script

- Run Python script in terminal

helloworld.py

```
1  print("Hello World!")
```

```
$ python helloworld.py
Hello World!
```

- Python is a script language, so the compilation and run processes are not separated, but done in one step by the Python interpretor
- Python does not need any class declaration to run (but you can use classes in Python if you wish)

Basic Math

- Addition

```
print(1 + 1)

> 2
```

- Multiplication

```
print(2 * 2)

> 4
```

- Division

```
print(10 / 3)

> 3.3333333333333335
```

- Integer Division

```
print(10 // 3)

> 3
```

- Power

```
print(2 ** 3)

> 8
```

Python Data Types I

- Boolean

```
x = True
y = False
print(x)
print(y)

> True
> False
```

Python Data Types II

- String

```
x = "banana"
print(x)
y = 'banana'
print(y)
print(x[0])
print(x[0:3]) # Slicing
print(x[0:1])
print(x[2:]) # Not giving the beginning/end of a slice goes until the end of the string
print(x[-2]) # Negative indices count from the end
z = "one"
print(z + " " + y) # Strings can be concatenated
print(f"{z} yellow {y}") # The f-string can also be used for concatenation. You need to write an 'f'
                                                    before the quotation marks

> banana
> banana
> b
> ban
> b
> nana
> n
> one banana
> one yellow banana
```

- You can imaging the indices in Python being between characters for the purpose of slicing: ${}_0b_1a_2n_3a_4n_5a_6$

Python Data Types III

- List

```
empty_list = []
print(empty_list)
x = [1, 2, 3, 4]
y = ["banana", "apple", "coconut"]
print(x)
print(y)
print(y[0])
print(y[1:3])
empty_list.append("mango") # Items can be added to list via append method
print(empty_list)
print(y + empty_list) # Lists can be concatenated
y[0] = "orange" # List items can be changed
print(y)

> []
> [1, 2, 3, 4]
> ['banana', 'apple', 'coconut']
> banana
> ['apple', 'coconut']
> ['mango']
> ['banana', 'apple', 'coconut', 'mango']
> ['orange', 'apple', 'coconut']
```

Python Data Types IV

- Dictionary (like *HashMap* in Java)

```
empty_dict = {}
print(empty_dict)
d = {"banana": "yellow", "apple": "red", "coconut": "brown"}
print(d)
print(d["coconut"])
d["cherry"] = "red"
print(d)
d["apple"] = "green"
print(d)
d2 = {"banana": ["yellow", "brown"], "apple": ["red", "green"]}
print(d2["banana"])
d3 = {"banana": {"color": "yellow", "sweet": True}, "coconut": {"color": "brown", "sweet": False}}
print(d3["coconut"]["sweet"])

> {}
> {'banana': 'yellow', 'apple': 'red', 'coconut': 'brown'}
> brown
> {'banana': 'yellow', 'apple': 'red', 'coconut': 'brown', 'cherry': 'red'}
> {'banana': 'yellow', 'apple': 'green', 'coconut': 'brown', 'cherry': 'red'}
> ['yellow', 'brown']
> False
```

If Statements I

```
x = ["banana", "apple", "coconut"]
if x[0][-1] == "a":
    print(f"The final letter of '{x[0]}' is 'a'")
else:
    print(f"The final letter of '{x[0]}' is not 'a'")

> The final letter of 'banana' is 'a'
```

```
x = ["banana", "apple", "coconut"]
if x[2][-1] == "a":
    print(f"The final letter of '{x[2]}' is 'a'")
elif x[2][0] == "c":
    print(f"The first letter of '{x[2]}' is 'c'")
else:
    print(f"The final letter of '{x[2]}' is not 'a' and the first letter is not 'c'")

> The first letter of 'coconut' is 'c'
```

- Python does not use curly brackets {} to group if statements and loops, but indentations
- Conventionally, one indentation should be a single tab or four spaces (spaces are preferred)
- The condition of the if statement is terminated via a colon :

If Statements II

```
x = ["banana", "apple", "coconut"]
if x[1][-1] == "a":
    print(f"The final letter of '{x[1]}' is 'a'")
elif x[1][0] == "c":
    print(f"The first letter of '{x[1]}' is 'c'")
else:
    print(f"The final letter of '{x[1]}' is not 'a' and the first letter is not 'c'")

> The final letter of 'apple' is not 'a' and the first letter is not 'c'
```

- The evaluation of a condition is a Boolean value

```
print(x[0][-1] == "a")
print(x[0][0] == "a")

> True
> False
```

If Statements III

- The `in` operator checks if the value exists in a string or list

```
print("banana" in x)
print("b" in "banana")
print("y" in "banana")

> True
> True
> False
```

Python Loops I

- For Loop

```
for i in [1,2,3,4]:
    print(i)
for i in range(1,5):
    print(i)
# enumerate() creates a generator to iterate over list items plus an index
for i, fruit in enumerate(["banana", "apple", "coconut"]):
    print(i, fruit)
# zip() creates a generator to iterate over two lists in parallel
for item1, item2 in zip(["banana", "apple", "coconut"], ["yellow", "red", "brown"]):
    print(item1, item2)

> 1
> 2
> 3
> 4

> 1
> 2
> 3
> 4

> 0 banana
> 1 apple
> 2 coconut
```

Python Loops II

```
> banana yellow
> apple red
> coconut brown
```

Python Loops III

- While Loop

```
i = 1
while i <= 4:
    print(i)
    i+=1
> 1
> 2
> 3
> 4
```

- You can create infinite while loops by adding a statement that is always true
- The while loop ends as soon as the condition is not true anymore

```
condition = True
i = 1
while condition:
    print(i)
    i = i + 1
    if i == 5:
        condition = False

> 1
> 2
> 3
> 4
```

Functions

```
def split_words(text):
    # The split() method operates on strings and outputs a list with items coming from the operation when the
                                                    string is split at the separator given to the
                                                    function
    return text.split(" ")
print(split_words("I want to split this text into a list containing its words"))
def split_lines(text):
    return text.split("\n")
# Three quotation marks around strings allows for multi-line strings
print(split_lines("""This text contains multiple lines.
I want to split it into a list containing one line per item."""))
def count_word_length(words):
    for word in words:
        print(len(word))
count_word_length(split_words("Count the word length of this text"))

> ['I', 'want', 'to', 'split', 'this', 'text', 'into', 'a', 'list', 'containing', 'its', 'words']
> ['This text contains multiple lines.', 'I want to split it into a list containing one line per item.']

> 5
> 3
> 4
> 6
> 2
> 4
> 4
```

- Input and output of functions are not typed in Python, you need to keep track of the type of a variable
- If a function does not have a return value, it does not need to be declared as *void*

Reading User Input

- You can wait for user input and write the input into a variable
- When the user hits “Enter”, Python stops waiting for input and reads in the string so far

userinput.py

```
1  user_input = input()
2
3  print(f"User input: {user_input}")
```

```
$ python userinput.py
Janis Pagel<Return>
User input: Janis Pagel
```

Reading and Writing Files I

data.txt

```
1  William Shakespeare
2  As You Like It
3
4  ACT 1
5  Scene 1
6
7  Enter Orlando and Adam.
8  ORLANDO
9  As I remember, Adam, it was upon this
10 fashion bequeathed me by will but poor a thousand
11 crowns, and, as thou sayst, charged my brother on
```

- `with ... as ...:` opens the file in a separated environment, so you don't need to take care of closing the file
- `file_object.read()` returns the file content as a string

```
with open("data.txt", "r") as file_object:
      file_read = file_object.read()
print(file_read)
print(file_read.split("\n"))

> William Shakespeare
> As You Like It
>
> ACT 1
> Scene 1
>
> Enter Orlando and Adam.
> ORLANDO
> As I remember, Adam, it was upon this
> fashion bequeathed me by will but poor a
                        thousand
> crowns, and, as thou sayst, charged my
                        brother on

> ['William Shakespeare', 'As You Like It',
                        '', 'ACT 1', 'Scene 1'
                        , '', 'Enter Orlando
                        and Adam.', 'ORLANDO',
                         'As I remember, Adam,
                         it was upon this', '
                        fashion bequeathed me
                        by will but poor a
                        thousand', 'crowns,
                        and, as thou sayst,
                        charged my brother on']
```

Reading and Writing Files II

data.txt

```
1  William Shakespeare
2  As You Like It
3
4  ACT 1
5  Scene 1
6
7  Enter Orlando and Adam.
8  ORLANDO
9  As I remember, Adam, it was upon this
10 fashion bequeathed me by will but poor a thousand
11 crowns, and, as thou sayst, charged my brother on
```

- `readlines()` directly splits the file content by newline and returns a list, but preserves the newlines

```
with open("data.txt", "r") as file_object:
      file_read = file_object.readlines()
print(file_read)

> ['William Shakespeare\n', 'As You Like It\
                        n', '\n', 'ACT 1\n', '
                        Scene 1\n', '\n', '
                        Enter Orlando and Adam
                        .\n', 'ORLANDO\n', 'As
                         I remember, Adam, it
                        was upon this\n', '
                        fashion bequeathed me
                        by will but poor a
                        thousand\n', 'crowns,
                        and, as thou sayst,
                        charged my brother on'
                        ]
```

Reading and Writing Files III

data.txt

```
William Shakespeare
As You Like It

ACT 1
Scene 1

Enter Orlando and Adam.
ORLANDO
As I remember, Adam, it was upon this
fashion bequeathed me by will but poor a thousand
crowns, and, as thou sayst, charged my brother on
```

- `open(..., "w")` writes to a file, creates it if it doesn't exists yet and overwrites it if it does exist (without asking for confirmation!!!)
- `"sep".join()` takes a list as argument and returns a string with `"sep"` as the separator

```
with open("data.txt", "r") as file_object:
      file_read = file_object.read()
file_split = file_read.split("\n")
file_sorted = sorted(file_split)
with open("sorted.txt", "w") as file_object:
      file_object.write("\n".join(
                              file_sorted))
```

sorted.txt

```

ACT 1
As I remember, Adam, it was upon this
As You Like It
Enter Orlando and Adam.
ORLANDO
Scene 1
William Shakespeare
crowns, and, as thou sayst, charged my brother on
fashion bequeathed me by will but poor a thousand
```

04

EXERCISE 2

Exercise 2

- Exercise 2 can be found on `https://github.com/IDH-Cologne-Deep-Learning-2024/Exercise-2`
- Deadline: October 24, 2024, 08:00:00 CEST

UNIVERSITY
OF COLOGNE

Janis Pagel
Institut für Digital Humanities

eMail janis.pagel@uni-koeln.de
Website https://janispagel.de
Phone +49 221 470 5749

References

Chacon, Scott and Ben Straub (2014). *Pro Git*. 2nd ed. Apress. ISBN: 978-1484200773. URL: https://git-scm.com/book/en/v2.