UNIVERSITÄT
ZU KÖLN

SPRACHVERARBEITUNG: ÜBUNG

SoSe 2025

Janis Pagel

Institut für Digital Humanities

2025-04-08

01

COURSE LANGUAGE

Language of the course

- Spoken course language is German
- All slides, exercise sheets and material in English
- If you have questions or comments, feel always free to ask in English

02

ABOUT ME

Janis Pagel

Figure: Janis Pagel

- Bachelor of Arts in Linguistics and German Studies (Ruhr University Bochum)
- Master of Science in Computational Linguistics (University of Stuttgart)
- PhD in Computational Linguistics (University of Stuttgart)
- Currently Postdoc at the Institut für Digital Humanities (IDH) in Cologne
- Sprechstunde / Office hours: Appointment via email or via https://calendly.com/janispagel/sprechstunde
- janis.pagel@uni-koeln.de
- https://janispagel.de

Janis Pagel

- (Character) references in literary texts (coreference, bridging)
- Applying NLP methods (rule-based, machine learning, neural networks, large language models) to Computational Literary Studies tasks ...
  - ...and making the results interpretable/usable for literary scholarship
- Practises of good annotation and annotation as a means to learn about texts/tasks
- Multi-word expressions, compounding and compositionality in diachronic settings

03

THIS CLASS

Basismodul “Grundlagen der Computerlinguistik”

- Winter term
  - Seminar: Computerlinguistische Grundlagen (Jürgen Hermes)
    - Annotation, linguistic foundations

Basismodul “Grundlagen der Computerlinguistik”

- Winter term
  - Seminar: Computerlinguistische Grundlagen (Jürgen Hermes)
    - Annotation, linguistic foundations
- Summer term
  - Vorlesung: Sprachverarbeitung (Janis Pagel)
  - Übung: Sprachverarbeitung (Janis Pagel)
- Modulprüfung / Module examination
  - Klausur / Written Exam: Thursday, July 17, 10:00–11:30

Aufbaumodul “Anwendungen der Computerlinguistik”

- Winter term
  - Übung: Deep Learning (Judith Nester)
  - Hauptseminar: Experimentelles Arbeiten in der Sprachverarbeitung (Nils Reiter)
- Modulprüfung / Module examination
  - Hausarbeit mit computerlinguistischem Experiment / Term paper with computational linguistics experiment

Studienleistung

- In total 8 exercise sheets
- For 5 out of 8 exercises, the solution must be uploaded on Ilias for the Studienleistung
- Deadline: Every Monday before the next Übung session at 23:59:59
- Solution for exercise will be uploaded after deadline

Learning Goals

After this class (and the module), you will practically and conceptually

- be able to handle corpora
- have an overview over multiple machine learning algorithms
- be able to train your own models, know how to interpret and evaluate them
- know about pros and cons of various representations of language
- have an insight into corpus statistics

Weekly Flow

- Time slots
  - Tuesdays, 16:00–17:30: Exercise
  - Thursdays, 10:00–11:30: Lecture
- Course information
  - General information will be found here: https://lehre.idh.uni-koeln.de/lehrveranstaltungen/sommersemester-2025/sprachverarbeitung/
    - Slides will also be uploaded there
  - Literature will be accessible on Ilias (if not publicly available)

04

PYTHON CRASH COURSE

Motivation

- Not a full-fledged Python course
- Enough knowledge to be able to use Python in the *Übung* and solve the exercises
- Python is used in a variety of research and industry-relevant fields, e.g. data science, machine learning, deep learning, web development, etc.
- Assuming existing knowledge of Java
- If you do not know Java or any other programming language, it is still possible to attend the Übung, but it will be much harder

First Python Script

```
print("Hello world!")

~> 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)
- Some goals of Python: being nice to read, writing concise code
  - For that, see also https://peps.python.org/pep-0008/

Python Data Types I

- String
  - Representation of chains of characters

```
x = "banana" # Strings are declared with double quotes
print(x)
~> banana

y = 'banana' # Single quotes are equivalent to double quotes
print(y)
~> banana

print(x[0]) # Single characters can be accessed by their index
~> b

print(x[0:3]) # Slicing allows to retrieve characters from a range of indices: BEGIN:END
~> ban

print(x[0:1]) # Slicing from index 0 to 1 is the same as requesting the character at index 0 (see comment below)
~> b

print(x[2:]) # Not giving the beginning/end of a slice goes until the end of the string
~> nana

print(x[-2]) # Negative indices count from the end of a string
~> n
```

Python Data Types II

```
z = "one"
print(z + " " + y) # Strings can be concatenated with the "+" sign
~> one 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
  - Is a collection of items

```
empty_list = [] # Lists are written in square brackets. Leaving them empty creates an empty list
print(empty_list)
~> []

x = [1, 2, 3, 4] # Items of lists can for example be integers...
print(x)
~> [1, 2, 3, 4]

y = ["banana", "apple", "coconut"] ... or strings (among others)
print(y)
~> ['banana', 'apple', 'coconut']

print(y[0]) # The items of lists can be accessed by indices
~> banana
print(y[1:3]) # Slicing can also be used on lists
~> ['apple', 'coconut']

empty_list.append("mango") # Items can be added to lists via the "append" method
print(empty_list)
~> ['mango']

print(y + empty_list) # Lists can be concatenated
~> ['banana', 'apple', 'coconut', 'mango']
```

Python Data Types IV

```
y[0] = "orange" # List items can be changed at their index
print(y)
~> ['orange', 'apple', 'coconut']

print("orange" in y) # The "in" keyword checks if an item exists in a list
~> True
```

Python Data Types V

- Dictionary
  - Creates a mapping between a key and a value

```
empty_dict = {} # Dictionaries are written between curely brackets. Leaving it empty creates an empty dictionary
print(empty_dict)
~> {}

d = {"banana": "yellow", "apple": "red", "coconut": "brown"} # "banana" is a key and "yellow" the value for this key
print(d)
~> {'banana': 'yellow', 'apple': 'red', 'coconut': 'brown'}
print(d["coconut"]) # The value of a key can be accessed by writing the key inside squared brackets behind the dictionary
~> brown

d["cherry"] = "red" # New key-value pairs can be added if the key does not exist yet in the dictionary
print(d)
~> {'banana': 'yellow', 'apple': 'red', 'coconut': 'brown', 'cherry': 'red'}

d["apple"] = "green" # Values of existing keys can be overwritten
print(d)
~> {'banana': 'yellow', 'apple': 'green', 'coconut': 'brown', 'cherry': 'red'}

d2 = {"banana": ["yellow", "brown"], "apple": ["red", "green"]} # The values of dictionaries can be complex, like lists
print(d2["banana"])
~> ['yellow', 'brown']

d3 = {"banana": {"color": "yellow", "sweet": True}, "coconut": {"color": "brown", "sweet": False}} # The values can also
                                                                be dictionaries themselves
```

Python Data Types VI

```
print(d3["coconut"]["sweet"])
~> False

print("coconut" in d3) # The "in" keyword checks if a key exists in a dictionary
~> True
```

If Statements I

```
x = ["banana", "apple", "coconut"]
if x[0][-1] == "a": # A double equal sign checks for equality of two values
    print(f"The final letter of '{x[0]}' is 'a'") # The f-string can contain variables in curly brackets, the evaluation of
                                                  the variable is included in the final string
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)

If Statements II

- The condition of the if statement is terminated via a colon :

```
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'
```

Python Loops I

- For Loop

```
for i in [1,2,3,4]: # for-loops iterate over the items of a list ...
    print(i)
~> 1
~> 2
~> 3
~> 4

# ... or the keys of a dictionary
for i in {"banana": "yellow", "apple": "red"}:
    print(i)

~> banana
~> apple

# The range method creates a generator of integers from a start- to an end-value
for i in range(1,5): # "5" is not included in the generated integers
    print(i)
~> 1
~> 2
~> 3
~> 4

# enumerate() creates a generator to iterate over an automatically created index and the list items
for i, fruit in enumerate(["banana", "apple", "coconut"]):
    print(i, fruit)
```

Python Loops II

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

# 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)
~> banana yellow
~> apple red
~> coconut brown
```

Python Loops III

- While Loop

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

Functions I

```
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"))
~> ['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."""))
~> ['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)) # The len() function counts the number of characters in a string
count_word_length(split_words("Count the word length of this text")) # Functions can be arguments of other functions. The
                                                                    outer function takes the output of the inner function as input
~> 5
~> 3
~> 4
~> 6
~> 2
~> 4
~> 4
```

Functions II

- 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*

05

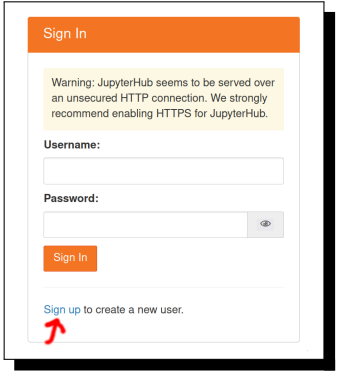
JUPYTER

Jupyter Notebooks

- Browser-based development tool for (mostly) Python code
- Code can be written interactively
- You can use a Jupyter instance hosted at the institute to write your code for the exercises, but do not have to
- The hosted Jupyter instance already has all required packages installed, so no setup needed from your side

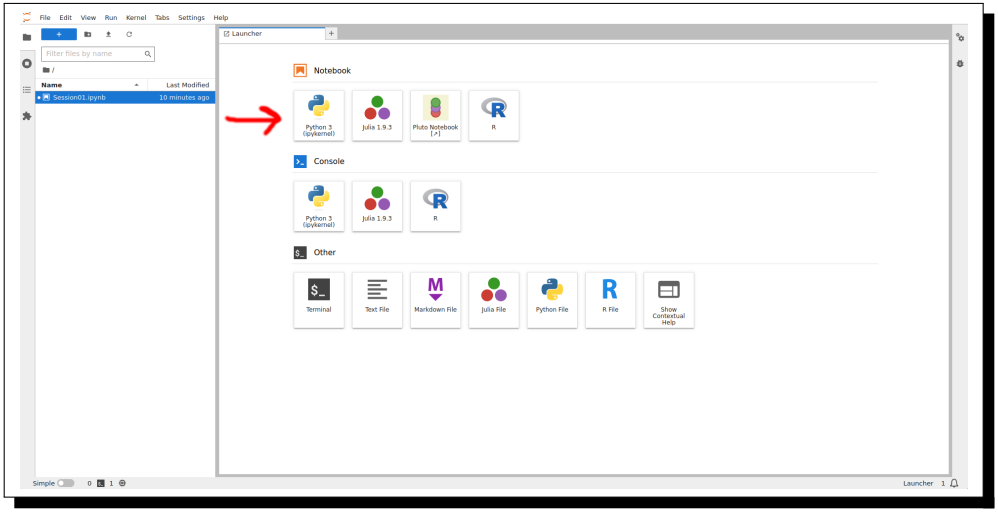
Jupyter Login

- http://compute.spinfo.uni-koeln.de/
  - This is only reachable from the University of Cologne Network
  - https://rrzk.uni-koeln.de/internetzugang-web/netzzugang/vpn for VPN access

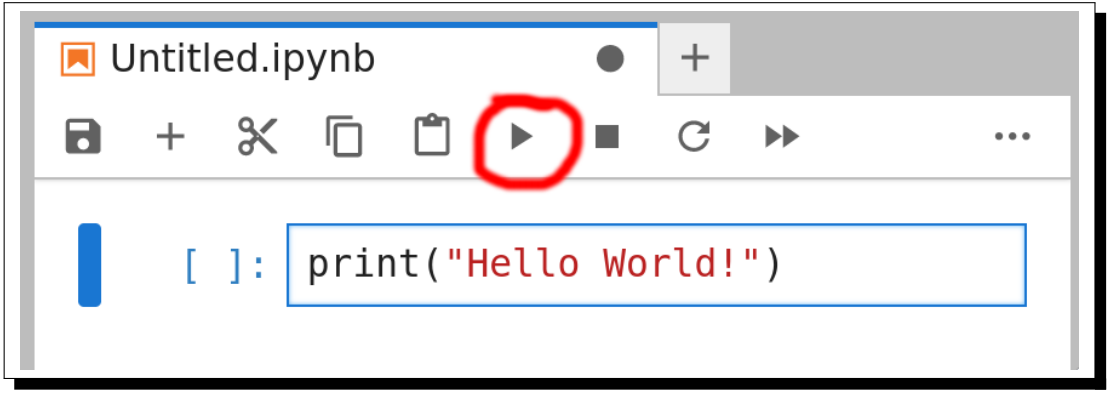


- Sign up with a new username and password
- Go back to “Sign In” and sign in with your chosen username and password

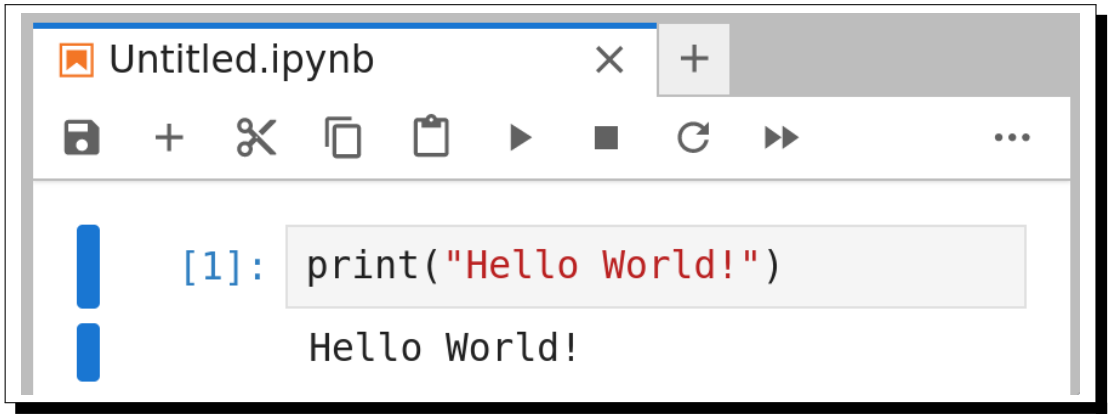
Start Notebook



Run Code



Result of Running Code Cell



06

OUTLOOK ON NEXT WEEK

Outlook

- Classes and Methods?
- Reading and writing files
- Useful libraries
  - pandas (Data Engineering)
  - seaborn (Plotting)

07

FURTHER MATERIAL ON PYTHON

Python Resources

- Install Python: https://www.python.org/downloads/
- Popular Python Development Environments
  - IDLE (built into Python): https://docs.python.org/3/library/idle.html, https://www.python-lernen.de/python-idle.htm
  - PyCharm: https://www.jetbrains.com/pycharm
  - spyder: https://www.spyder-ide.org/
  - VSCode: https://code.visualstudio.com/
- Python Tutorials
  - https://automatetheboringstuff.com/
  - https://docs.python.org/3/tutorial/index.html
  - https://python.land/python-tutorial


UNIVERSITY
OF COLOGNE
Janis Pagel
Institut für Digital Humanities
eMail janis.pagel@uni-koeln.de
Homepage https://janispagel.de
Phone +49 221 470 5749
Foto: Gregor Hübl