

Python For The Digital Humanities

Manuel Huth
(<https://github.com/talant26>)

2026

Contents

1	Introduction	5
1.1	Why Python?	5
1.2	Helpful Tutorials and Websites	6
1.3	How to use this reader	7
2	Python Basics	9
2.1	Our environment	9
2.2	Comments	10
2.3	The print function	11
2.4	Arithmetic Operators	13
2.5	Variables	14
2.6	Basic data types	17
2.7	The input function	19
2.8	If statements and indentation	19
2.9	While Loops	23
2.10	Lists	26
2.11	For loops	30
2.12	Dictionaries	32
2.13	Functions	35
2.14	Interacting with files	39
2.15	String functions	41
2.16	Error handling	42
2.17	Importing modules	43
3	Useful modules for the Digital Humanities	45
3.1	RegEx	45
3.2	Interacting with CSV and JSON Files	45
3.3	Machine Learning	46
3.4	Working with Websites and XML-Files	46
3.5	Natural language processing (NLP)	46
3.6	Database management	46
3.7	Data analysis and visualization	47
3.8	Network Visualization	47
4	Outlook	49
4.1	Possible Next Steps	49
4.2	Goodbye	50

Chapter 1

Introduction

1.1 Why Python?

In recent years, the digital revolution has increasingly affected and transformed the humanities. There are now a variety of fields and software that are relevant to us and in which we can work. Maybe you are working with a database, maybe you are creating a digital edition or maybe you are working with artificial intelligence. **Whatever your field, Python can help.**



Figure 1.1: <https://clipground.com/images/wegweiser-clipart-1.jpg>
(CC BY 4.0 Deed)

Python...

- is one of the world's most popular programming languages.
- is a High-level programming language.
- is simple and efficient.
- has many, many modules for almost any purpose you can imagine.
- can do all kinds of time-consuming, repetitive work for you.

Overall Python is easy to learn, flexible and efficient.

1.2 Helpful Tutorials and Websites**Websites**

- <https://wiki.python.org/>
- <https://www.w3schools.com/python/>
- <https://www.geeksforgeeks.org/python-if-else/>
- <https://automatetheboringstuff.com/>

Youtube tutorials

- https://www.youtube.com/watch?v=_uQrJ0TkZ1c
- <https://www.youtube.com/watch?v=qwAFL1597eM>

Space for your personal notes

1.3 How to use this reader

I originally designed this course for the CSMBR STUDIOLO Winter School 2024. Its aim is to introduce humanities scholars with no prior knowledge to the basics of Python and demonstrate typical applications in the humanities, such as extracting and visualising information from texts. By the end of 2025, I had adapted the course materials to enable self-study.

This reader is split into two parts. The first part introduces the fundamentals of Python, while the second part provides a list of essential tools for working with texts in the field of digital humanities. It also provides references and links to further materials and websites for self-study. So you should have enough material to proceed directly with programming.

While the reader can be worked through independently, it is best used alongside the *BasicPython.ipynb* and *PracticalProject.ipynb* files. *BasicPython* contains sample scripts and exercises to accompany and supplement the reader. *PracticalProject* uses H. P. Lovecraft's novel *At the Mountains of Madness* as an example to demonstrate how to extract and visualise information from texts.

Copyright Notes

The text of the novel 'Mountains of Madness' comes from the Gutenberg Project (<https://www.gutenberg.org/ebooks/70652>). The text has been split into two parts 'Copyright.txt' and 'Mainpart.txt'. Please note the copyright!



Figure 1.2: <https://clipground.com/images/coder-clipart-1.jpg> (CC BY 4.0 Deed)

Chapter 2

Python Basics

2.1 Our environment

I recommend uploading the .ipynb files to Google Colab (<https://colab.research.google.com/>) and editing them there. It allows us to code directly in the browser and eliminates the need for any installations. You will need a Google account to do this. You will also need to upload the .txt files for *PracticalProject.ipynb* there.

Recommendations for working with Google Colab:

- Tutorial: https://www.tutorialspoint.com/google_colab/index.htm
- Change the editor so that line numbers are displayed. This way it is easier to find errors (Tools > Settings > Editor > show line numbers).
- Google Colab does not permanently store files. To keep your files, download them or save them to Google Drive https://www.tutorialspoint.com/google_colab/google_colab_saving_work.htm.
- If (unintended) errors occur when executing individual scripts, please click the *Run all* button, which can be found either at the top of the menu bar or in the *Runtime* > *Run all* tab. This will execute all scripts in the correct order. This helps to avoid errors that may occur if the session has been inactive for too long, for example, or if a script accesses variables that have not been set previously.
- If this does not work, you can try to restart a session (Runtime > Restart Session).

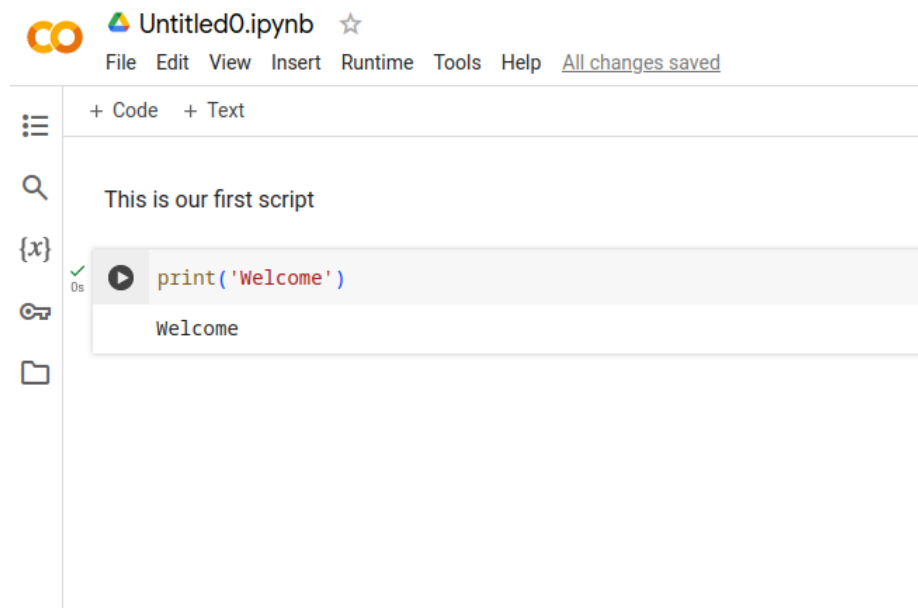


Figure 2.1: Google Colab

2.2 Comments

A *comment* is text that is ignored by the interpreter. It starts with a hashtag. In your editor, comments look like this:

```
1 # This is a comment
```

Why you should use comments

- Use comments to explain your code
- Use comments to structure your code
- Comments allow you to (temporarily) disable parts of your code. This is a great help when debugging (i.e. searching for errors).

Multiline comments

You can define multiple lines as a comment by enclosing them in three quotation marks (either single or double quotation marks).

```
1 '''This
2 is
3 a
4 multiline
5 comment'''
```

```
6
7  """This
8  is
9  a multiline
10 comment
11 as
12 well"""
```

Space for your personal notes

2.3 The print function

You can use the *print function* to display text. Just type the word "print" and then type the text you want to display in parentheses and quotation marks. You can use either single or double quotes.

```
1  # The print command using single quotation marks
2  print('Hello')
3
4  # The print function using double quotation marks
5  print("Hello")
```

If you want to display a number, you don't need quotation marks.

```
1  print(7)
```

This is because numbers are interpreted as a different data type, as we will see later (see 2.6).

Space for your personal notes

2.4 Arithmetic Operators

When it comes to coding, we need to do at least some mathematical operations. In this workshop, we will use only the four basic arithmetic operators you learned in elementary school:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /

For further arithmetic operators (like Modulus), see the section "Further References".

Arithmetic Operators and numbers

Using the print function we can perform all kinds of mathematical operations and display the results. Here are some examples:

```
1 print(7+2)
2 print(7-4+3-1+27)
3 print(2*2)
4 print(6/2)
```

Concatenating words or texts

We can use the "+" operator to concatenate words/texts (this can be an important tool if, for example, you want to automatically replace certain parts of a text):

```
1 print('Hi' + ' ' + 'there' + '.')
```

The output would be:

```
1 Hi there.
```

Further References

- https://www.w3schools.com/python/gloss_python_arithmetic_operators.asp
- <https://www.geeksforgeeks.org/python-arithmetic-operators/>

Space for your personal notes

2.5 Variables

Variables are like containers that can store different types of information (such as numbers or text) for you. In Python you declare variables by writing the name of the variable, an equal sign, and the value you want to assign to it. You can then access the variable in other functions (such as the print function):

```
1  # We assign the value 7 to the variable a.
2  # We do not need quotes, because 7 is a number
3  a = 7
4
5  # We assign the value 'Hi' to the variable b.
6  # We need quotes, because 'Hi' is a word / text
7  b = 'Hi'
8
9  print(a)
10 print(b)
```

The output would be:

```
1  7
2  Hi
```

You can name variables anything you like. You can even combine letters and numbers, but variables cannot begin with a number. It is also recommended to avoid special characters. So you could name a variable "RogueOne", "Rogue1", but not "1Rogue".

Changing Variables

You can change variables any time. Note how the interpreter reads line by line:

```
1  # We assign the value 7 to the variable a and display it.
2  a = 7
3  print(a)
```

```
4
5 # Now we assign another value to the variable a and display it.
6 a = 9
7 print(a)
```

The output would be:

```
1 7
2 9
```

We can define a variable relative to other variables. And unlike in mathematics, we can even define or change a variable relative to itself. In fact, this is an important operation, as we will see, when it comes to while loops (see the chapter 2.9):

```
1 # We assign the value 7 to the variable a and display it.
2 a = 7
3 print('Value of a:')
4 print(a)
5
6 # Now we define b in relation to a
7 # Then we display a and b
8 b = a
9 print('Values of a and b after equalization:')
10 print(a)
11 print(b)
12
13 # Now we change b in relation to itself and increment it by 2
14 b = b + 2
15 print('Values of a and b after changing b:')
16 print(a)
17 print(b)
18
```

The output would be:

```
1 Value of a:
2 7
3 Values of a and b after equalization:
4 7
5 7
6 Values of a and b after changing b:
7 7
8 9
```

As you can see, even after equalizing the variables, we can still change them independently!

Concatenating Variables

Like the words / texts we combined in the print function (see 2.4) you can also combine text variables:

```
1  # We assign the value 'Hi' to the variable a and 'you' to the variable b.
2  a = 'Hi'
3  b = 'you'
4
5  # Now we define the variable c as the combination of a, a white space and b
6  c = a + ' ' + b
7
8  # Now we display c
9  print(c)
```

The output would be:

```
1  Hi you
```

If you try to combine a variable, that contains a number with a variable that contains a word or text, you will get an error message. This is a common error. The reason is, that you are trying to combine different *data types*. The interpreter does not know whether to concatenate *strings* (=words or texts) or do an addition:

```
1  # We assign the value 'I am' to a, 9 to b and 'years old' to c
2  a = 'I am '
3  b = 9
4  c = ' years old'
5
6  # Now we define the variable c as the combination of a, b and c
7  print(a+b+c)
```

You will get the following error message:

```
1  -----
2
3  TypeError                                Traceback (most recent call last)
4
5  <ipython-input-19-c8b48e124d01> in <cell line: 7>()
6  5
7  6 # Now we define the variable c as the combination of a, b and c
8  ----> 7 print(a+b+c)
9
10 TypeError: can only concatenate str (not "int") to str
```

We will discuss *data types* in the next chapter (2.6)

Further References

- https://www.w3schools.com/python/python_variables.asp
- <https://www.geeksforgeeks.org/python-variables/>

Space for your personal notes

2.6 Basic data types

Data types are the kind of information that is stored inside a variable. Depending on the type you can do certain operations (for example do additions). In Python *data types* are automatically assigned when you declare a variable.

Data types used in this course

Type	Abbreviation	Description	Example
<i>string</i>	str	A sequence of characters like a word or text	'The Mandalorian'
<i>integer</i>	int	Whole numbers (positive or negative)	42
<i>floating point numbers</i>	float	decimal numbers	3.2
<i>boolean</i>	bool	True or False	True
<i>list</i>	list	A list of values, see chapter 2.10	[9 , 2 , 7]
<i>dictionary</i>	dict	A dictionary of key-value pairs, see chapter 2.12	{ "name": "Luke", "profession": "Jedi" }

Notes:

- For more datatypes see "Further References".
- Calculating with *floating point numbers* is imprecise with Python. This is due to rounding to certain digits. The math module is required to perform precise mathematical calculations.

How to read data types

With the *type function* we can read the data types of variables.

```
1  # We declare a variable a
2  a = 'hi'
3
4  # Now we use the type function to read the type
5  print(type(a))
```

The output would be:

```
1  <class 'str'>
```

Enforce data types

You can enforce a *data type* using the above mentioned abbreviations:

```
1  # We declare the variable a and read its type
2  a = 9
3  print(type(a))
4
5  # We enforce the datatype 'str' and read its type
6  a = str(a)
7  print(type(a))
```

The output would be:

```
1  <class 'int'>
2  <class 'str'>
```

Further References

- https://www.w3schools.com/python/python_datatypes.asp
- <https://www.geeksforgeeks.org/python-data-types/>

Space for your personal notes

2.7 The input function

The input function can be used to store user input inside a variable, making thus programs more flexible. The text in parentheses is displayed to the user.

```
1 age = input('How old are you?\n') #  
2 print(age)
```

The characters “\n” mark a line break. This allows the user to start writing on a new line.

Further Reference

<https://www.geeksforgeeks.org/python-input-function/>

Space for your personal notes

2.8 If statements and indentation

If statements

- If a condition is true, one or more commands are executed.
- **Indentations** (= one tab or 4 spaces at the beginning of a line) make it clear which code belongs to the *if statement* and which does not (see below for an example)

I recommend using the **tab key** instead of 4 spaces. This is faster and easier. It also minimizes the risk of miscounting spaces. You can undo an indentation by pressing **shift + tab**.

Comparison operators

Comparison operators are used to compare values:

<code>a == b</code>	a equals b
<code>a != b</code>	a does not equal b
<code>a < b</code>	a is smaller than b
<code>a <= b</code>	a is smaller than or equal to b
<code>a > b</code>	a is bigger than b
<code>a >= b</code>	a is bigger than or equal to b

Simple If statements

```
1 a = 4 # We define the variable 'a' that we want to compare
2
3 if a == 4: # We check if the variable 'a' is equal to 4
4
5     # The following indented text is only executed, if the condition is true.
6     print(a)
7
8 print('The program continues...') # This code is independent of the if statement,
9                                 # because there is no indentation.
```

Here the output would be:

```
1 4
2 The program continues...
```

Elif and else

```
1 a = 5 # we declare the variable 'a'
2 b = 5 # we declare the variable 'b'
3
4 if a < b: # We check if a is smaller than b
5
6     print(str(a) + ' is smaller than ' + str(b))
7
8 elif a == b: # We check if a is equal to b
9
10    print(str(a) + ' equals ' + str(b))
11
12 else: # If a is neither smaller than nor equal to b, the following code
13      # is executed:
```

```
14
15     print(str(a) + ' is bigger than ' + str(b))
```

Nested if statements

You can use if-statements within if-statements.

```
1  a = 5 # We declare the variable 'a'
2
3  if a < 5: # We check if a is smaller than 5
4
5      print(str(a) + ' is smaller than ' + str(b))
6
7  else: # If a is not smaller than 5, the following code is executed:
8
9      if a == 5: # We check if a equals 5
10
11         print('a equals 6')
12
13     elif a == 6: # We check if a equals 6
14
15         print('a equals 5')
16
17     else: # a must be bigger than 6
18
19         print('a is bigger than 6')
```

Further References

- https://www.w3schools.com/python/gloss_python_if_statement.asp
- <https://www.geeksforgeeks.org/python-if-else/>

Space for your personal notes

2.9 While Loops

While loops are similar to *if statements* (see chapter 2.8). But instead of asking if a condition is true, they ask how long the condition is true. That is, as long as (or: while) a condition is true, certain commands are executed.

```
1 i = 1 # We create a counter, that we want to increment each time we loop
2     # through the following function. When it reaches a certain value,
3     # we want the while loop to stop.
4
5 while i < 6: # We check if the counter is smaller than 6
6
7     print('The counter is ' + str(i) + '.') # display the counter
8     i = i + 1 # increase the counter by 1, then we will loop again
9     # through the function
```

Do not forget to increment the counter or you will create an infinite loop!

Just like in *if statements*, you can combine *while* with *else*:

```
1 i = 1 # our counter
2
3 while i < 6: # check if counter is smaller than 6
4
5     print('The counter is ' + str(i) + '.') # display the counter
6     i = i + 1 # increase the counter by 1
7
8 else:
9
10    print('The counter is no longer less than 6')
```

There are some commands that give us more control over the iterations:

continue	skips the current iteration
break	terminates the entire loop

Break

```
1 i = 1 # our counter
2
3 while i < 6: # check if counter is smaller than 6
4
5     if i == 3: # if the counter equals 3
6
7         break # the entire loop is immediately terminated
8
9     print('The counter is ' + str(i) + '.') # display the counter
10    i = i + 1 # increase the counter by 1
```

Here the output would be:

```
1 1
2 2
```

Continue

```
1 i = 1 # our counter
2
3 while i < 6: # check if counter is smaller than 6
4
5     if i == 3: # if the counter equals 3
6
7         i = i + 1 # increase the counter by 1
8         continue # the current iteration is skipped and the following code
9                 # is not executed
10
11     print('The counter is ' + str(i) + '.') # display the counter
12     i = i + 1 # increase the counter by 1
```

Here the output would be:

```
1 The counter is 1.
2 The counter is 2.
3 The counter is 4.
4 The counter is 5.
```

Note that we incremented the counter before the *continue* statement, otherwise we would have created an infinite loop.

[illegible]

- https://www.w3schools.com/python/python_while_loops.asp
- <https://www.geeksforgeeks.org/python-while-loop/>

2.10 Lists

"List" is a *data type* (like *string* / *integer* / *float* ...). A python list is a list of numbers or strings and it can be stored inside a variable. See the following code example for further explanations.

```

1  # Example for a simple list
2  list1 = [ "Wookie", "Ewok", "Java", "Rodian" ]
3
4  # Lists can have duplicates or be empty:
5  list2 = [1, 3, 5, 7, 9, 7, 7, 7, 7]
6  list3 = [ ]
7
8  # display the three lists
9  print(list1)
10 print(list2)
11 print(list3)
12
13 # with the len function you get the amount of elements inside a list
14 lengthList1 = len(list1)
15 print(lengthList1)
16
17 # There is a method for sorting lists. It will sort the list in an ascending
18 # order, either alphabetically (if the list contains strings) or numerically
19 # (if the list contains numbers).
20 list1.sort()
21 print(list1)

```

Lists are ordered and indexed. That means every entry of a list has an indexnumber, by which it can be accessed. The first entry has the index 0, the second one the index 1 and so on (In computer science we start by counting from 0 instead of 1). See the following code for examples and further explanations:

```

1  starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
2  # Indexnumber      0      1      2      3
3
4  print(starwarslist[0]) # the word "Wookie" will be displayed
5  print(starwarslist[1]) # the word "Ewok" will be displayed
6  print(starwarslist[2]) # the word "Java" will be displayed
7  print(starwarslist[3]) # the word "Rodian" will be displayed
8
9  # You can count backwards using negative indexes
10 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
11 # Indexnumber     -4      -3      -2      -1
12
13 print(starwarslist[-1]) # the word "Rodian" will be displayed
14 print(starwarslist[-2]) # the word "Java" will be displayed
15 print(starwarslist[-3]) # the word "Ewok" will be displayed
16 print(starwarslist[-4]) # the word "Wookie" will be displayed
17
18 # You can also access a range of indexes (i.e., a portion of a list).

```

```

19 # The first number (before the colon) indicates the starting point
20 # The second number (after the colon) indicates the end point.
21 # Note that the end point is not included. See the following examples:
22
23 # Entry one and two will be displayed
24 print(starwarslist[1:3]) # Output: [ "Ewok" , "Java" ]
25
26 # Entry zero and entry one will be displayed
27 print(starwarslist[:2]) # Output: [ "Wookie", "Ewok" ]
28
29 # The entire list starting with entry 1 will be displayed
30 print(starwarslist[1:]) # Output: ["Ewok", "Java", "Rodian" ]
31
32 # The entire list without the last two entries will be displayed
33 print(starwarslist[:-2]) # Output: [ "Wookie", "Ewok" ]

```

What we said about the indexes of lists, also applies to strings, which are basically nothing else than a list of characters. Therefore you can access the characters of a string the same way you would access the entry of a list.

```

1 examplestring = 'Mandalorian' # we declare a variable with the datatype string
2
3 print(examplestring[0])          # displays the first character of the
4                                 # variable 'examplestring'.
5                                 # The output therefore is: 'M'
6
7 print(examplestring[:-1])        # displays every character of the variable
8                                 # 'examplestring' except for the last
9                                 # character. The output therefore is:
10                                # 'Mandaloria'

```

You can check if an item is in a list with the *in* operator:

```

1 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
2
3 if "Wookie" in starwarslist:
4     print("'Wookie' is in the list.")

```

That also applies to strings. That means you can check if a string is part of another string with the *in* operator:

```

1 longString = 'Mos Eisley, you will never find a more wretched hive of scum...'
2
3 if 'scum' in longString:
4     print('Yes, the long string contains the word "scum".')

```

To check if an item is not in a list / a string is not in another string, type *not in* instead of *in*.

Adding / removing items

You can add items to a list with the *append method*. If you want to remove an item, you can either use the *pop method* (which uses the index number of the respective item) or the *remove method* (which uses the item itself).

```
1 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
2
3 # Add items with the append method.
4 starwarslist.append("Duros")
5
6 # now the list is [ "Wookie", "Ewok", "Java", "Rodian", "Duros" ]
7
8 # there are two methods to remove entries:
9 starwarslist.pop(0) # removes the first entry
10 starwarslist.remove("Duros") # removes the entry "Duros"
```

Copying lists

Note: Copying lists does the same way as with other variables. When you equate lists, they refer to the same location in the computer's memory. Any change you make to one list will also be made to the other. In fact, they are identical. If you want to copy lists so that you can edit them independently later, you need the *copy method*.

```
1 # Copying other variables:
2 a = 6
3 b = a
4 b = b + 1
5 print(a) # The output is 6
6 print(b) # The output is 7
7
8 # Wrong way of copying lists
9 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
10 newlist = starwarslist # equates the lists
11 newlist.remove("Ewok") # removes the item "Ewok"
12
13 print(starwarslist)
14 print(newlist)
15 # in both cases the output is:
16 # [ "Wookie", "Java", "Rodian" ]
17
18 # Right way of copying lists with the copy method
19 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
20 newlist = starwarslist.copy()
21 newlist.remove("Ewok") # removes the item "Ewok"
22
23 print(starwarslist) # output: [ "Wookie", "Ewok", "Java", "Rodian" ]
24 print(newlist)     # output: [ "Wookie", "Java", "Rodian" ]
```

[illegible]

- https://www.w3schools.com/python/python_lists.asp
- <https://www.geeksforgeeks.org/python-lists/>

2.11 For loops

With for loops you can iterate over sequences (e.g. lists or strings). That means you can access every element of a list or a string one after the other. See the following code examples for further explanation.

```
1  # We define a list
2  starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
3
4  # We iterate over each item and display it.
5  for element in starwarslist:
6      print(element)
7
8  # That means:
9  # We first look at the entry 'Wookie' and display it
10 # Then we look at the entry 'Ewok' and display it
11 # Then we do the same with 'Java' and 'Rodian'.
12 # In this example we used the variable "element".
13 # We could name the variable any way we like it
14 # (for example "item", "race")
15
16 # We can also loop through a string.
17 # Therefore we define the string "examplestring"
18 examplestring = "Mandalorian"
19
20 # Now we iterate over each item of the string and display it
21 for character in examplestring:
22     print(character)
```

You can use the same commands as in while loops to get more control over iterations:

continue	skips the current iteration
break	terminates the entire loop

```
1  # We define a list
2  starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
3
4  # We iterate over the entries of the list
5  for entry in starwarslist:
6
7      # We check, if the entry is equal to "Ewok"
8      if entry == "Ewok":
9
10         # We terminate the entire loop
11         break
12
13     print(entry)
14
15 # So the output will be [ "Wookie" ]
```

Space for your personal notes

[illegible]

Further References

- https://www.w3schools.com/python/python_for_loops.asp
- <https://www.geeksforgeeks.org/loops-in-python/>

2.12 Dictionaries

A dictionary is a bit like a list, but instead of index numbers you have keys. Like each index number in a list, each key in a dictionary has a value. This means that a dictionary consists of **key value pairs**. You can use dictionaries for example, to represent one entity in a database (e.g. a real person / thing):

Severin Göbel I. [\[Bearbeiten\]](#)

Profession	physician
Date of Birth	1530/06/25
Place of Birth	Königsberg
Date of Death	1612/01/05
Place of Death	Königsberg

Figure 2.2: An entity, as you would find it in a typical database. On the left side you can see the keys, on the right side the corresponding values.

Now let us see, how we can create a dictionary that corresponds to the entity 'Severin Göbel' from the image above.

```

1  # Dictionaries are enclosed in curly brackets. They are created according to
2  # the schema key : value. Note that we have to separate the key value pairs
3  # by commas
4
5  # We create the dictionary 'severinGoebelDict' for the person Severin Göbel
6  severinGoebelDict = {
7      'profession': 'physician',
8      'dateOfBirth': 1530,
9      'placeOfBirth': 'Königsberg',
10     'dateOfDeath': 1612,
11     'placeOfDeath': 'Königsberg'
12 }
13
14 # Now let us display the dictionary
15 print(severinGoebelDict)
16
17 # The output would be: {'profession': 'physician', 'dateOfBirth': 1530,
18 # 'placeOfBirth': 'Königsberg', 'dateOfDeath': 1612, 'placeOfDeath':
19 # 'Königsberg'}
```

You can access / add items by referring to the key (just like we used the index number of a list, when we wanted to know a certain value):

```

1  # We create the dictionary severinGoebelDict again
2  severinGoebelDict = {
3      'profession': 'physician',
4      'dateOfBirth': 1530,
```



```

5     'placeOfBirth': 'Königsberg',
6     'dateOfDeath': 1612,
7     'placeOfDeath': 'Königsberg'
8 }
9
10 # Now let us display his profession:
11 print(severinGoebelDict['profession'])
12
13 # Using the same principle we can add a key-value pair to the dictionary.
14 # Let us say we want to add an entry about his wife
15 severinGoebelDict['wife'] = 'Ursula'
16
17 # Now lets us display the dictionary again
18 print(severinGoebelDict)
19
20 # The ourput will be: {'profession': 'physician', 'dateOfBirth': 1530,
21 # 'placeOfBirth': 'Königsberg', 'dateOfDeath': 1612, 'placeOfDeath':
22 # 'Königsberg', 'wife': 'Ursula'}
```

You can remove a key-value pair by pointing to the key with the *pop method*.

```

1 # We create the dictionary severinGoebelDict again
2 severinGoebelDict = {
3     'profession': 'physician',
4     'dateOfBirth': 1530,
5     'placeOfBirth': 'Königsberg',
6     'dateOfDeath': 1612,
7     'placeOfDeath': 'Königsberg'
8 }
9
10 # Now let us remove the key value pair 'profession': 'physician',:
11 severinGoebelDict.pop('profession')
```

By pointing to the keys you can loop through a dictionary just like you can loop through a list:

```

1 # We create the dictionary severinGoebelDict again
2 severinGoebelDict = {
3     'profession': 'physician',
4     'dateOfBirth': 1530,
5     'placeOfBirth': 'Königsberg',
6     'dateOfDeath': 1612,
7     'placeOfDeath': 'Königsberg'
8 }
9
10 # Now let us loop through the dictionary and display the keys
11 print('These are the keys:')
12 for x in severinGoebelDict:
13     # we will display the key
14     print(x)
```

```
15
16 # Now let us loop through the dictionary and display the values
17 print('These are the values:')
18 for x in severinGoebelDict:
19     # we will display the values by using the keys
20     print(severinGoebelDict[x])
```

The output would be:

```
1  These are the keys:
2  profession
3  dateOfBirth
4  placeOfBirth
5  dateOfDeath
6  placeOfDeath
7  These are the values:
8  physician
9  1530
10 Königsberg
11 1612
12 Königsberg
```

Sometimes dictionaries are stored inside another dictionary or inside a list. The latter is the case, for example, when you want to display not one, but several entities of the same type within a database (for example, all persons). We will look at an example in the course. Otherwise, have a look at 'Further references'.

Space for your personal notes

Further References

- <https://www.geeksforgeeks.org/python-dictionary/>
- <https://www.geeksforgeeks.org/loops-in-python/>

2.13 Functions

A function is a block of code that is executed only when *called*, which means you can define one or more commands as a function that can be executed on demand. For this we use the keyword *def*:

```
1  # We define the function 'printfunction' that displays the word 'Hello'
2  def printfunction():
3      print('Hello')
4
5  # Now we call the function. That means we execute it.
6  # If we would not call the function, nothing would happen.
7  printfunction()
```

You should always give functions descriptive names that accurately describe their purpose (e.g., "listComparison" instead of "myfunction"). This makes the code easier to read and helps other people understand what the program is doing. I myself add a comment to all functions I write, explaining the purpose and operation of the function. That way, even a year later, I know why I wrote a function the way I did.

Arguments / parameters

When you call a function, you can pass *arguments* / *parameters* to the function. This means that each time you call a function, you can pass a different input to the function:

```
1  # We define the function 'addWhiteSpaces'. It adds a white space after
2  # each character of a word, that is passed to the function as an argument.
3  def addWhiteSpaces(word):
4
5      # We create a variable 'newWord' containing an empty string,
6      # where we will store our result
7      newWord = ''
8
9      # now we iterate over each character of the word:
10     for character in word:
11
12         # Now we add the character to the variable newWord and
13         # add a white space
14         newWord = newWord + character + ' '
15
16     # Now the loop is finished, but we have on white space too much at the end
```

```

17     # of 'newWord'. So we will display newWord without the last letter
18     print(newWord[:-1])
19
20     # Now we can call the functions and pass different words to it as arguments
21     addWhiteSpaces('Hello')
22     addWhiteSpaces('You')
23     addWhiteSpaces('Hi')

```

The output would be:

```

1  H e l l o
2  Y o u
3  H i

```

The number of arguments passed to a function must match the number of arguments defined in the function:

```

1  # We define a function 'combineWords'. It has two arguments: 'word1' and 'word2'
2  # The goal is to combine two words with a blank space between them.
3  def combineWords(word1, word2):
4      print(word1 + ' ' + word2)
5
6  # Now we call the function and pass the words 'Hello' and 'You' as arguments
7  combineWords('Hello', 'You')

```

Return values

Sometimes you just want a function to return a value so that you can move on. This can be the case, for example, if you want to generate an intermediate result to use in further code or in another function. This is done with the *return* keyword:

```

1  # We create the function "combineWords". It has two arguments: word1 and word2
2  def combineWords(word1, word2):
3      # it returns a string, that is the concatenation
4      # of word1, a white space and word2
5      return word1 + ' ' + word2
6
7  # We call the function combineWords and pass the words "Hello" and "you".
8  # The result is stored in the variable a
9  a = combineWords('Hello', 'you')
10 # We print the variable a
11 print(a)
12 # The output will be: Hello you

```

Local and global variables

A variable created outside a function is a *global variable*. A variable created inside a function is a *local variable*. *Global variables* can be accessed from anywhere, *local variables* can only be accessed within their respective functions.

This is why the following code will not work:

```
1  # we define the helloFunction. It will display 'Hello'
2  def helloFunction():
3      x = 'Hello' # We define x
4      print(x)    # We display x
5
6  helloFunction() # We call the function helloFunction
7  print(x) # We try to display x, but x is not a global variable.
8           # An error will occur
```

If you want to define a *global variable* inside a function, you can use the *global* keyword:

```
1  # we define the function globalVariableInsideAFunction. It will display 'Hi'
2  def globalVariableInsideAFunction():
3      global x # We define x as global
4      x = 'Hi' # We define x
5
6  # We call the function
7  globalVariableInsideAFunction()
8
9  # We display x
10 print(x)
```

Space for your personal notes

Further References

- https://www.w3schools.com/python/python_functions.asp
- https://www.w3schools.com/python/python_variables_global.asp
- <https://www.geeksforgeeks.org/python-functions/>
- <https://www.geeksforgeeks.org/global-local-variables-python/>

2.14 Interacting with files

Working with files can be very useful. I recommend using *with open*. It is similar to a function with a few arguments:

- First of all the filename in quotes (e.g. 'starwars.txt')
- Then we define the type of access we want to have. This is a single letter that tells the interpreter, what we want to do. This means we declare, if we want to read a file ('r'), write a file ('w') or append something to the end of the file ('a'). If it does not already exist, we need to add + to the letter (e.g. 'w+')
- Sometimes you need to specify the encoding (for example: encoding='utf-8'). This will be explained in more detail during the course.

Lets us look at an example. First of all let's open an already existing file for writing:

```
1 # We want writing access to the existing file 'starwars.txt'
2 # It is encoded with 'utf-8'
3 # We store the accessed file in the variable 'outputfile'
4 with open('starwars.txt', 'w', encoding='utf-8') as outputfile:
5     # Now we write 'A long time ago...' into the file
6     outputfile.write('A long time ago...')
```

Now we want to open an existing file and read its content. There are two ways to do this: the *read method* and the *readlines method*. With the read method we can store the entire content of the file in a single string variable, with the readlines method we can store all the lines of the file in a single list variable (a list, where each entry corresponds to a line of the file).

Now let us use the read method:

```
1 # We want reading access to the existing file 'starwars.txt'
2 # It is encoded with 'utf-8'
3 # We store the accessed file in the variable 'inputfile'
4 with open('starwars.txt', 'r', encoding='utf-8') as inputfile:
5     # Now we store the filecontent as a string in the variable content
6     content = inputfile.read()
7     print(content)
```

Space for your personal notes

Further References

- https://www.w3schools.com/python/python_file_handling.asp
- <https://www.geeksforgeeks.org/file-handling-python/>
- <https://automatetheboringstuff.com/2e/chapter9/>

2.15 String functions

Strings are similar to lists. To access certain parts of a string or to check if a string is inside another string, see chapter 2.10.

But there are also a number of methods specific to strings. Here are some important ones:

<code>endswith()</code>	Checks if a string ends with a specific value (e.g. a character).
<code>find()</code>	Finds a string inside another string and returns the indexnumber, where the string was found.
<code>isdigit()</code>	Checks if the characters of a string are digits.
<code>islower()</code>	Checks if the characters of a string are lowercase.
<code>isupper()</code>	Checks if the characters of a string are uppercase.
<code>rfind()</code>	see the <i>find method</i> , but <code>rfind</code> returns the indexnumber, where the string was found.
<code>split()</code>	Splits a string at a separator (like a comma or semicolon for example). The result is stored as a list.
<code>splitlines()</code>	Splits a string into lines. The result is stored as a list.
<code>startswith()</code>	Checks if a string starts with a specific value (e.g. a character).
<code>strip()</code>	Removes white spaces at the beginning and end of a string. Instead of white spaces other characters can be removed as well.

For further methods see Further References.

Further References

- https://www.w3schools.com/python/python_strings_methods.asp
- <https://automatetheboringstuff.com/2e/chapter6/>

Space for your personal notes

2.16 Error handling

Sometimes you do not want your entire program to quit, when a small error occurs. We can solve this problem with the statements *try* and *except*.

```
1  # We create a list with three words
2  wordlist = [ 'Maria', 'John', 'Joseph' ]
3
4  # We want to display the 5th character of each word, but the word 'John'
5  # has only 4 characters and would throw an error that would
6  # stop the entire program. So use 'try' and 'except'
7
8  # we iterate over the list
9  for word in wordlist:
10
11     try: # we check, if we can display the 5th character
12         print(word[4]) # remember: in computer science we start counting with 0
13
14     except: # if this is not possible, we want to display a message
15         print('This word has less than five characters.')
```

Space for your personal notes

Further References

- https://www.w3schools.com/python/python_try_except.asp
- <https://www.geeksforgeeks.org/python-exception-handling/>

2.17 Importing modules

A module is a set of ready-to-use functions that you can import and start using right away. This is done via the *import* statement.

Let us import the *time* module. With it we can make the program wait for a certain number of seconds.

```
1  # We import the time module
2  import time
3
4  # We make the program wait for 10 seconds
5  time.sleep(10)
6
7  # Now we display 'Ten seconds have passed.'
8  print('Ten seconds have passed.')
```

In general, most modules should already be installed in Google Colabs. If a module is not installed, you can install it with the following command, where MODULENAME is the name of the package you want to install:

```
!pip install MODULENAME
```

Space for your personal notes

Further References

- https://www.w3schools.com/python/python_modules.asp
- <https://www.geeksforgeeks.org/python-modules/>

Chapter 3

Useful modules for the Digital Humanities

There are many useful modules for almost every case you can imagine. The following list will show you some of the most useful ones for DH projects. Do not hesitate to try them out. After all, the goal of this workshop is to open up perspectives so that you can choose exactly what you need for your specific project.

3.1 RegEx

With **regular expressions** you can search for strings, that matter a specific pattern, like for example phonenumbers, dates, certain types of names... They can be very helpful and are essential, if you work with texts.

Tutorials:

- <https://automatetheboringstuff.com/2e/chapter7/>
- https://www.w3schools.com/python/python_regex.asp
- <https://www.geeksforgeeks.org/regular-expression-python-examples/>

3.2 Interacting with CSV and JSON Files

CSV and JSON files are common formats when it comes to store or exchange information. For each file type there is a certain module:

Tutorials:

- <https://automatetheboringstuff.com/2e/chapter16/>
- https://www.w3schools.com/python/python_json.asp
- <https://www.geeksforgeeks.org/python-json/>
- <https://www.geeksforgeeks.org/working-csv-files-python/>

3.3 Machine Learning

TensorFlow is a good tool for machine learning.

Websites and Tutorials:

- <https://www.geeksforgeeks.org/introduction-to-tensorflow/>
- <https://pypi.org/project/tensorflow/>
- <https://www.tensorflow.org/learn/>

3.4 Working with Websites and XML-Files

With the **request module** you can access websites, check if they are active and download their source code.

- **Short tutorial:** https://www.w3schools.com/python/module_requests.asp
- **Longer tutorial:** <https://www.geeksforgeeks.org/python-requests-tutorial/>

When you are working with HTML or XML files, **Beautiful Soup** is the module you want to have. It is very powerful and useful.

Tutorials and Documentation:

- <https://beautiful-soup-4.readthedocs.io/en/latest/>
- <https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/>

3.5 Natural language processing (NLP)

NLTK (Natural Language Toolkit): It is a tool for analyzing and working with language using computational linguistics. Its data is based on "50 corpora and lexical resources".

- **Website and introduction:** <https://www.nltk.org/>

CLTK (Classical Language Toolkit): It is the equivalent of the NLTK, but for the languages of pre-modern Eurasia.

- **Website and introduction:** <http://cltk.org/>

3.6 Database management

mysql.connector is a module that allows you to access MySQL databases.

- **Tutorial:** <https://realpython.com/python-mysql/>

Pywikibot is a module that allows you to access *MediaWiki databases* through a bot. It can be a handy tool, but the installation is tricky.

- **Website and Tutorial:** <https://www.mediawiki.org/wiki/Manual:Pywikibot>

3.7 Data analysis and visualization

Numpy is the module to use when it comes to *data analysis*. It is incredibly fast and offers various functions to perform all kinds of calculations.

- **Website:** <https://numpy.org/>
- **Tutorial:** <https://www.geeksforgeeks.org/python-numpy/>

Pandas is built on top of numpy, making the import and analyzation of data easier. You can combine it with **matplotlib** to visualize your data.

- **Data visualization tutorial:**
<https://www.geeksforgeeks.org/data-visualization-with-python/>
- **Matplotlib tutorial:**
<https://www.geeksforgeeks.org/matplotlib-tutorial/>
- **Pandas tutorial:**
<https://www.geeksforgeeks.org/pandas-tutorial/>

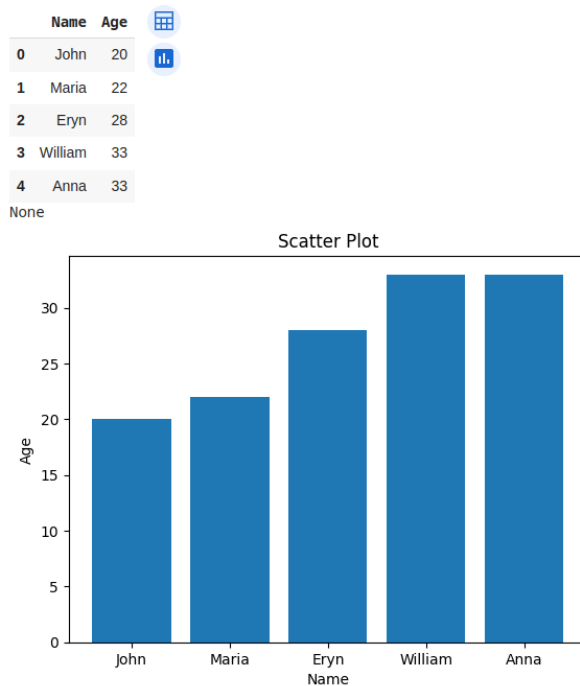


Figure 3.1: An automatically generated bar chart from a csv file (Matplotlib and pandas were used)

3.8 Network Visualization

Pathpy is a mighty and flexibel tool for network analysis and visualization.

- **Website and tutorial** <https://www.pathpy.net/>

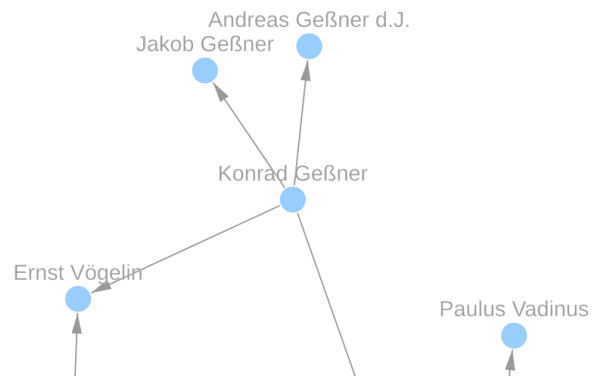


Figure 3.2: Visualization of a network section with Pathpy

Space for your personal notes

[illegible]

Chapter 4

Outlook

4.1 Possible Next Steps

Local installation

You may want to install python locally on your computer. There are many good editors you can use, among them *Visual Studio Code* and *PyCharm*.

Visual Studio Code allows you to work with almost any programming language. Pycharm is specifically designed for Python and has a free community version as well as a paid professional version. Both are great editors.

You can also use other editors. But for security reasons, make sure they work with *virtual environments* so that installing modules does not affect your system.

Advanced Studies

Chapter 3 introduced us to many new modules that do not require professional knowledge of Python. Do not hesitate to try them out.

However, this course did not cover some topics that are important if you want to work with Python at a professional level. These include the following not-so-simple topics:

- *Recursion*
- *Object Orientation*

If you want to get more than a basic understanding of Python, you should take a closer look at these topics. For this, I recommend the websites and tutorials mentioned in chapter 1.2.

I also recommend reading the following article on *clean code*:

<https://www.geeksforgeeks.org/best-practices-to-write-clean-python-code/>

4.2 Goodbye

No matter how deep you want to go with Python, hopefully this workshop has been fun, given you new insights, and shown you numerous perspectives on how to use Python. Most of all, it should have awakened your curiosity and made you want to start programming right away. I wish you a lot of fun!



Figure 4.1: <https://clipground.com/images/erfolg-clipart-4.jpg> (CC BY 4.0 Deed)

List of Figures

1.1	https://clipground.com/images/wegweiser-clipart-1.jpg (CC BY 4.0 Deed)	5
1.2	https://clipground.com/images/coder-clipart-1.jpg (CC BY 4.0 Deed)	7
2.1	Google Colab	10
2.2	An entity, as you would find it in a typical database. On the left side you can see the keys, on the right side the corresponding values.	32
3.1	An automatically generated bar chart from a csv file (Matplotlib and pandas were used)	47
3.2	Visualization of a network section with Pathpy	48
4.1	https://clipground.com/images/erfolg-clipart-4.jpg (CC BY 4.0 Deed)	50

Cheat Sheet

Data types used in this course

Type	Abbreviation	Description	Example
<i>string</i>	str	A sequence of characters like a word or text	'The Mandalorian'
<i>integer</i>	int	Whole numbers (positive or negative)	42
<i>floating point numbers</i>	float	decimal numbers	3.2
<i>boolean</i>	bool	True or False	True
<i>list</i>	list	A list of values	[9 , 2 , 7]
<i>dictionary</i>	dict	A dictionary of key-value pairs	{ "name": "Luke", "profession": "Jedi" }

print and input functions

```
1 print('Hello')
2
3 a = input('Please write something\n')
4 print('you wrote ' + a)
```

if statements

```
1 if a < 6:
2     print('a is smaller than 6')
3 elif a == 6:
4     print('a is equal to 6')
5 else:
6     print('a is bigger than 6')
```

while loops

```
1 i = 0
2 while i < 5:
3     print(i)
4     i = i + 1
```

lists

```
1 examplelist = [ 1, 2, 3 ]
2
3 for i in examplelist:
4     print(i)
```

for loops

```
1 starwarslist = [ "Wookie", "Ewok", "Java", "Rodian" ]
2
3 for entry in starwarslist:
4     print(entry)
```

functions

```
1 def myfunction(word):
2     returntext = 'your word is ' + word
3     return returntext
4
5 print(myfunction('Hello'))
```

dictionaries

```
1 exampleDict = {'name': 'Paul', 'age': 39}
2
3 for i in exampleDict:
4     print(exampleDict[i])
```

interacting with files

```
1 with open('hello.txt', 'r', encoding='utf-8') as inputfile:
2     text = inputfile.read()
3     print(text)
```

error handling

```
1  try:
2      print(a)
3  except:
4      print('Error')
```

Space for your personal notes

[illegible]