

IWGS – Informatische Werkzeuge in den Geistes- und Sozialwissenschaften – WS 2018/19

Michael Kohlhase
Informatik, FAU Erlangen-Nürnberg
FOR COURSE PURPOSES ONLY

October 17, 2019

Contents

1	Assignment 1 (First Steps with Python) – Given 26. 10. 2018, Due 9. 11. 2018	3
2	Assignment 2 (Control Flow in Python) – Given 09.11.2018, Due 16.11.2018	5
3	Assignment 3 (Data Structures) – Given 16.11.2018, Due 23.11.2018	7
4	Assignment 4 (Unicode and Number Systems) – Given 23.11.2018, Due 30.11.2018	9
5	Assignment 5 (Regular Expressions) – Given 01.12.2018, Due 07.12.2018	11
6	Assignment 6 (Regular Expressions 2 & HTML) – Given 08.12.2018, Due 14.12.2018	13
7	Assignment 7 (HTML 2 & Regular Expressions 3) – Given 15.12.2018, Due 21.12.2018	15
8	Assignment 8 (Project Design & Prototyping) – Given 22.12.2018, Due 11.01.2019	17
9	Assignment 9 (Trees in Python and XML) – Given 12.01.2019, Due 18.01.2019	18
10	Assignment 10 (Generating HTML with Python) – Given 18.01.2019, Due 25.01.2019	20

**11 Assignment 11 (Web-Applications with Bottle) – Given 26.01.2019, Due
01.02.2019**

21

1 Assignment 1 (First Steps with Python) – Given 26. 10. 2018, Due 9. 11. 2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 1.1 (Hello World)

Go to <http://pythonanywhere.com>, make an beginners account there and set the teacher's account to iwgsTeacher. 20pt

Write an extended "Hello World Program" in a file called `exthello.py`. The program should print information about you and your account. Specifically, the information should be:

```
        Hello World! I am <your name>.
    My account name is <account>, and my teacher is iwgsTeacher.
```

As you have given the teacher account access to our account, we can (and will, for grading) check your program on your account.

Submit only your account name to StudOn for this exercise.

Problem 1.2 (Variable Assignment and Output)

Write a program in `python` that calculates the total number of seconds in a leap year, stores the result in a variable and then displays that to the user. 20pt

Problem 1.3 (Variable Reuse)

Programming often has a efficiency as one of its goals. After all, why go through the trouble of telling a computer how to do something, if you could do it better and quicker yourself? 20pt

Write a program in `python` that prints the string “supercalifragilisticexpialidocious” five times, but *without* typing the word five times yourself.

Problem 1.4 (Human Readable Time)

In programming, it is often the case that your program collects a lot of data from various sources. It then becomes essential to present this data in a way that the user (usually a human!) can easily understand. For example, most humans don't know how long a longer timespan is if it is given only in seconds. 20pt

Write a program in `python` that first initialises a `variable seconds = 1234567`. Then, the program should calculate and print how long this timespan is in days, hours, minutes and seconds instead of just seconds.

Problem 1.5 (String Presentation)

Keeping with the importance of well-presented information: You can use certain special symbols in strings to give them a better formatting when they are ultimately printed. For 20pt

example, when you put “\n” into a string, instead of printing these symbols, the output switches to a *new line*.

Write a `python` program that prints your favourite haiku (a poem with five syllables on the first line, seven on the second and five on the third) on three lines, but using only *one print* statement.

P.S.: If you don't have a favourite haiku and can't think of one yourself, you can use this one:

My cow gives less milk,
now that it has been eaten,
by a fierce dragon.

*“If you are learning programming (or anything else) what you do when
you get "stuck" will be the sole determinant of your success.”*
Sarah Mount

2 Assignment 2 (Control Flow in Python) – Given 09.11.2018, Due 16.11.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 2.1 (User Input)

One of the most important things to learn about a programming language is how to get input from the user in front of the screen. In `python`, one way of doing this is the `input` instruction. 15pt

For example: if you write `answer = input("Do you like sharks?")`, this will print the message you gave (“Do you like sharks?”), wait for the user to submit a response and store it as a string in the variable `answer` when you run the program. You can then use it like any other value stored in a variable.

Write a simple program that prints a generic greeting message, then asks the user to input their name, stores the input in a variable and then finishes with a goodbye message that uses the name the user gave.

Problem 2.2 (Simple Branching)

The next important concept is `control flow`. A program that always does the same thing gets boring fast. We want to write programs that do different things under different circumstances. In `python`, one way to do this are `conditional statements`. 25pt

Write a `python` program that asks the user if they have a pet. If their answer was “yes”, the program should ask what kind of pet they have. Since sloths are the cutest animals (at least for this exercise), the program should print “awww!” if the user’s second answer was “sloth” and “cool!” if it was something else. If the user does not answer with “yes” the first time around, the program should quit with a goodbye message.

Problem 2.3 (Simple Looping)

Computers are very good at doing the same thing over and over again without complaining or messing up. Humans are not. In `python`, we can use a `loops` if we want something done multiple times. 25pt

Suppose your boss wants the string “Programming is cool!” printed exactly 1337 times (for some reason...). Typing up the string yourself takes about nine seconds each time, printing it in a loop takes no time.

To save time, write a `python` program that prints the sentence “Programming is cool!” 1337 times using a `loop`. Your program should also keep track of (store in a variable) how much time the loop saved the programmer in total (9 seconds per iteration of the loop). Print this value after the `loop` finishes.

Problem 2.4 (Temperature Conversion)

Write two `python` programs, named `celsius2fahrenheit` and `fahrenheit2celsius`, that given a 35pt

number as input from the user convert it to the respective other temperature scale and print the result.

The conversion formulas are as follows:

$$[^{\circ}C] = ([^{\circ}F] - 32) \cdot \frac{5}{9} \quad [^{\circ}F] = [^{\circ}C] \cdot \frac{9}{5} + 32$$

Remember that `input` will save the input as text, not as a number. You can convert a string to a number using the function `float`.

Example: `float("3.1415")` will evaluate to the *number* 3.1415. If the text given to `float` does not actually represent a number (e.g. `float("bad")`), python will throw an error.

Afterwards, please test your programs against another converter (easily found via your internet search engine of choice) to make sure that your functions produce the correct results.

3 Assignment 3 (Data Structures) – Given 16.11.2018, Due 23.11.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 3.1 (Basic Lists)

When working with lists, the first and the last elements of the list are often of special interest or significance. 20pt

1. Write a python function that, when given a list as a parameter, prints (on two separate lines, with some explanatory text) the first and last elements of the list.
2. Is it possible to do this without looping over the entire list to find the last element?
3. What happens when you give this function a list of only one element?
4. What happens when you give it the empty list?

Problem 3.2 (User Input II)

Often, when you are taking input from the user, it becomes important that the input is one of a certain set of “acceptable” answers. 30pt

Write a python program that asks the user for their favourite deadly sin. If the input it receives is not one of the acceptable answers (i.e. the strings "lust", "gluttony", "greed", "sloth", "wrath", "envy" and "pride"), it should keep asking again and again.

Hint: You can use a loop to achieve this. Using a “sin list” will also be helpful.

When the input is (finally) correct, it should print a message either complimenting or deriding the user on their pick (your choice!).

Problem 3.3 (Dictionaries)

In programming, it is important to gain familiarity with the most commonly used data structures. This exercise will make you more familiar with the dictionary data structure. 30pt

1. Write a python dictionary that associates names of famous peoples (i.e. strings as keys) with their year of birth (i.e. ints as values). The entries can be real or fictional people, as long as they have a clear year of birth.
2. Write a program that finds the oldest person (i.e. lowest year of birth) in that dictionary. (How) can you loop over all keys of a dictionary? Finally, your program should print in what year the oldest person in your dictionary was born (it does not have to say who that person is).

Problem 3.4 (Project 01)

During the remainder of the semester, you will be developing example applications in teams. Your task for this exercise is to form teams of three and think of an application that you would like to develop. Ideally, this would draw on or somehow be relevant to a background in the humanities. 20pt

The application should have to manage some sort of data (text, images, ...), and generate some sort of (simple) interface (web pages, graphical representations, ...).

Hand in

1. a unique name for your project,
2. your team composition, and
3. a short text (maybe two paragraphs) about what kind of application you would want to develop. This should include a description of what kind of user the application is for, how they will interact with it, what they can and cannot do with the program and what kind of data the program will have to manage.

Example: As our running example, the fictitious IWGS students Nadja, Lucas and Amira will be developing a order management program for the pizza delivery service "Planetary Pizza" (a deliberately not humanities-based topic, so that the example is neutral and different).

They write down that the user should be able to assemble and send a pizza order, including both pizzas from a menu (loaded from the pizzeria's database) and pizzas that they themselves choose the toppings for. During this process, the user should be able to see their selected pizzas so far and the individual prices, as well as a sum total.

4 Assignment 4 (Unicode and Number Systems) – Given 23.11.2018, Due 30.11.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 4.1 (Egyptian Hieroglyphs 1: Numerals)

Programming is a versatile discipline and applicable to a lot of very different fields, from space satellites to fast pizza delivery to Egyptian hieroglyphs. In the following exercises, you will take a closer look at the latter to familiarise yourselves with the Unicode encoding. 40pt

The Egyptian numeral system¹ is decimal, like our system, but is not position-based (similar to Roman numerals). Each hieroglyph has a certain Unicode encoding², i.e. a certain number that people have agreed upon to represent a certain hieroglyph.

The Egyptian number system is relatively simple (for numbers up to 1,000,000 or so). Learn about it. Then, write a `python` function `arabic2Egyptian` that takes a standard (positive) integer and returns a unicode string of a corresponding Egyptian number.



Note: The code here will be structurally similar to a previous exercise. Also recall that the Universal Character Set assigns every character a hexadecimal number n , e.g. 1F607 (smiling face with halo). If we want to use character n in a string in `python`, just use “\U0001F607” (i.e. n filled up with leading zeros to make it 8 hex digits).

Note: Note that we will *not* be awarding / deducting points on precise hieroglyph choice. As long as the hieroglyphs you chose roughly align with those presented in the number systems article, we will assume them correct. This goes for all exercises on this sheet.

Problem 4.2 (Egyptian Hieroglyphs 2: Text)

Suppose that word has gotten around that you know how to handle Unicode in `python` and one of your friends who is also an egyptology enthusiast wants your help. 40pt

The standard method of displaying Egyptian hieroglyphs (etched into stone or clay) can be slow in writing and just remembering longer messages can be hard to do³. A digital format would be so much simpler!

¹See, for example: https://en.wikipedia.org/wiki/Egyptian_numerals

²See [https://en.wikipedia.org/wiki/Egyptian_Hieroglyphs_\(Unicode_block\)](https://en.wikipedia.org/wiki/Egyptian_Hieroglyphs_(Unicode_block)) for details

³Compare “Ente, Auge, Zickzack” (ZDF, German): <https://www.youtube.com/watch?v=SbZXiDE6G04>

First, write a `python` dictionary that associates English or German words (keys) to fitting Unicode symbols (values). Your dictionary obviously does not need to translate *all* hieroglyphs, but should at least include five different ones.

Second, write a program that, using this dictionary, will ask the user again and again for input, looks up the value associated with that input in your dictionary and appends it to a string variable. When some special phrase to end the program is entered (e.g. `"exit"` or `"quit"`), the program should print the variable and exit.

This way, you can take a message that's easy to write on a Western keyboard and easily turn it into proper Egyptian hieroglyphs.

Problem 4.3 (Egyptian Hieroglyphs 3: Input Sanitising)

Whenever you ask a user for input that you want to use in a meaningful way later in your program, it is vital that you make sure the user has actually entered something sensible. Because often, they won't. 20pt

Concretely, if you look up a key in a dictionary that was never assigned a value, `python` will print an error message and your program will crash.

Amend your program from the previous exercise to check if the entered word is actually a key in the dictionary you are using. If it isn't, you can print an error message or simply ask again. Entering garbage should no longer crash your program.

* * *

Note: The following is a *bonus* exercise. You can hand in a solution for extra points, but correct solutions to the first three will already count as 100%.

Problem 4.4 (Basechange)

Colours are important for a plethora of things in software development and there are many ways of describing just which colour you are talking about. 40pt

Maybe the most common way to specify a colour is by giving a triple of numbers between 0 and 255, signifying the how strong the red, green and blue (*RGB*) components in the colour are. Often, these are given as values in base 16 (i.e. 00 to FF).

First, make sure that you understand how a hexadecimal number system works. Then, write a function that takes a string as an argument. This string will only have one (hexadecimal) character, either of the following:

```
["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"].
```

The function should return the *decimal* value of the input as a regular integer.

Then, using the function you just finished, write a program that takes strings of six hexadecimal characters (two for red, two for green and two for blue, in that order, e.g. 00FF88 or 326496) and prints their correct RGB components in decimal.

5 Assignment 5 (Regular Expressions) – Given 01.12.2018, Due 07.12.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 5.1 (Regular Expressions 1)

In this exercise we will explore regular expressions. Regular expressions allow us to find patterns in a given text and even modify the matched sections. In order to use regular expressions, you need to import the “re” package. This is done by typing “**import re**” at the top of your python file. 40pt

In the imperial unit system, weight is measured in pounds (lb). As Central Europeans are more used to expressing weight in kilograms (kg), we will use regular expressions to find occurrences of weight measurements in a text and convert it.

Consider the following text⁴:

Two-thirds of Americans report that their actual weight is more than their ideal weight, although for many, the difference between actual and ideal is only 10 pounds or less. But 30% of women and 18% of men say their current weight is more than 20 pounds more than their ideal weight. The average American today weighs 17 pounds above what he or she considers to be ideal, with women reporting a bigger difference between actual and ideal than men.

Use regular expressions to find all numbers in the text. Use the `re.findall()` function⁵, which returns a list of matches.

Take into consideration, that numbers can consist of more than one digit. Print the list of matches. Amend the program, such that it only matches occurrences of pound measurements, i.e. only numbers followed by the string “pounds”. The list for the above text should now be [“10 pounds”, “20 pounds”, “17 pounds”].

In regular expressions, you can group certain parts of the pattern by enclosing it in parentheses. This can be useful, if you want to further process the results of the matching.

Amend your program, such that `findall()` returns the following list: [“10”, “20”, “17”]. Note that these are still only the numbers followed by “pounds”, but the “pounds”-part is stripped away automatically.

Loop over your list of measurements. For each entry, convert the entry to kilograms using the following formula:

$$[kg] = [lb]/2.2046$$

⁴Source: <https://news.gallup.com/poll/102919/average-american-weighs-pounds-more-than-ideal.aspx>

⁵<https://docs.python.org/3/library/re.html#re.findall>

Print the conversion with some explanatory text, i.e. "10lb are 4.535970244035199kg".

Problem 5.2 (Regular Expressions 2)

In the real world, data processed by computers often comes from files read from the hard disk. Consider the following spreadsheet table: 60pt

	A	B	C
1	Dentist	11/29/2018	Example Str. 22
2	Exam	2/7/2019	Kollegienhaus
3	Hair cut	12/3/2018	Example Str. 25

It lists appointments line by line. Each line consists of the type of appointment, the date and the place. A common data format is the CSV file format. Most spreadsheets (like LibreOffice Calc or Microsoft Excel) support exporting to this format.

The resulting CSV file (also supplied for this exercise) looks like this:

```
Dentist;11/29/2018;Example Str. 22
Exam;7/2/2019;Kollegienhaus
Hair cut;12/3/2018;Example Str. 25
```

CSV is short for "Comma Separated Values". As the name implies it lists the entries, separated by commas (actually it's semicolons in this case).

The dates in this example are given in the American notation: Month/Day/Year. We will use regular expressions to convert it into German notation: Day.Month.Year, i.e. day before month and separated by dots instead of slashes.

Open the file using python's File I/O (input/output) functionality⁶. Read the whole file using the `readlines()` function, which returns a list of lines. Print this list.

Now loop over the list and perform the following for each entry: Use the string `split()` method⁷ to separate individual entries at the semicolons.

For example, splitting the entry "Dentist;11/29/2018;Example Str. 22" at the semicolons should give you the list ["Dentist", "11/29/2018", "Example Str. 22"].

The second value is the date we would like to convert. Use the `re.sub()` function⁸ to extract the day, month and year and reassemble them in the German notation. Afterward print some useful text for the appointment containing the converted date.

⁶If you need a refresher about file input/output, see: <https://www.pythonforbeginners.com/cheatsheet/python-file-handling>

⁷<https://docs.python.org/3/library/stdtypes.html#str.split>

⁸<https://docs.python.org/3/library/re.html#re.sub>

6 Assignment 6 (Regular Expressions 2 & HTML) – Given 08.12.2018, Due 14.12.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 6.1 (Regular Expressions 3)

One of the best uses of a computer’s enormous processing power is to have it filter quickly through large amounts of data that would otherwise take a human a long time to sift through. This is also often a task where regular expressions shine. 25pt

Along with this exercise, you will be supplied with a text file that contains the entire text of Lev Tolstoy’s “War and peace”⁹, slightly modified.¹⁰ This will serve as our “corpus data” for this exercise.

Somewhere in this text (more than 500 kilowords), you know that there are a few e-mail addresses and a few hexadecimal colour codes (in a format like the following: #10FFAA). Write a python program that reads the file and uses regular expressions to find these addresses and colour codes. Afterwards, display the results with some explanatory text.

Note: Simply searching for “#” or “@” will not help you here, because since the data is sadly a bit “degraded”, those characters are also interspersed a few hundred times at random intervals.

Problem 6.2 (HTML table)

In the lecture you saw the overview table for HTML below.

40pt

purpose	elements	purpose	elements
structure	html, head, body	metadata	title, link, meta
headings	h1, h2, . . . , h6	paragraphs	p, br
lists	ul, ol, dl, . . . , li	hyperlinks	a
multimedia	img, video, audio	tables	table, th, tr, td, . . .
styling	style, div, span	old style	b, u, tt, i, . . .
interaction	script	forms	form, input, button
Math	MathML (formulae)	interactive graphics	vector graphics (SVG) and canvas (2D bit-mapped)

Make a HTML file `html-table.html` that re-creates this table in HTML. Note that the table heading is boldface and all of the HTML element names in the right column are in typewriter font (but the commata, ellipses, and explanations are not.)

⁹As found on Project Gutenberg: <https://www.gutenberg.org> (currently not accessible from Germany due to copyright disputes)

¹⁰Found here: https://kwarc.info/teaching/IWGS/materials/war-and-peace_modified.txt

Hint: You do not have to re-create the lines in the table. If you want to have (some kind of) border around the table cells, just add `border="1"` to the `table` element (or the `tr` elements).

Problem 6.3 (A Simple HTML Page)

Have a look at <https://www.izdigital.fau.de/efi-digitale-souveraenitaet/>. This page has header and footer parts (in blue) and two columns of text in between. The left one has the main text of the page (the page payload) and the right one some information about other pages on the same web site. 35pt

Make a simple web page from the payload text and the page heading “EFI-Förderung für das Forschungsprojekt „Diskurse und Praktiken einer digitalen Souveränität“”.

1. Download the file <https://kwarc.info/teaching/IWGS/materials/efi.txt>, save it, and rename it to `efi.html`.
2. With the HTML tags we have introduced in the lecture mark up all structural parts: paragraphs, itemized lists, hyperlinks (Hint: you can obtain the link target by right-clicking on the hyperlink and selecting “Copy Link Address”. You only need to mark up five links total.)
3. Load your `.html` file into a browser of your choice (this acts as the HTML document viewer) and export the contents to PDF (call the file `efi.pdf`).
4. Use the HTML checker at https://validator.w3.org/#validate_by_upload to see what it thinks of your HTML. Correct your errors reported there (as much as reasonable). Briefly discuss what your experience has been with this tool.

Submit `efi.html`, `efi.pdf`, and your discussion from 4.

7 Assignment 7 (HTML 2 & Regular Expressions 3) – Given 15.12.2018, Due 21.12.2018

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 7.1 (Simple HTML Form)

For this exercise, you will construct a very simple HTML page with a basic form. Suppose 30pt you want to establish a basic pizza delivery service only for **FAU** staff and students. It is your task to make the first version of the website for the “front-end” (that is, the user-facing part of the application).

Create a `.html` file¹¹ with a title, a heading, a paragraph or so of descriptive text and a `<form>`-element that contains the following inputs:

- a text input field for people to enter their name,
- a dropdown menu with (at least three) FAU-related addresses,
- (at least three) radio buttons labeled with different pizza options (for the moment, we only allow one pizza to be ordered at a time).
- a form-submit button.

When the submit button is clicked by the user, they should be redirected to a second HTML page (hand this in, too, in a separate file), that tells the user their order has been received. Use the form `action` attribute to accomplish this. This second page does *not* need to use the data from the form.

Problem 7.2 (Simple CSS)

It is a well-known fact that nobody likes to buy from a pizza place that only uses plain 20pt HTML on their website. So now, we will improve upon the website from Problem 7.1.

Create an external style sheet (in a CSS-file called `styles.css`) to change the look of your website. You can load this style sheet by placing the following `head`-element into your website’s `html`-element:

```
<head>  
  <link rel="stylesheet" href="styles.css">  
</head>
```

You can make this style sheet as elaborate as you like. However, at least the following style changes should be implemented by your style sheet:

¹¹If you need a refresher: there is excellent documentation on how the basics work at https://www.w3schools.com/html/html_intro.asp and related pages.

- Center the heading.
- Give the `<body>` of your website a `background-color`.
- Set the `font-family` of all text to “Verdana”.
- Set the font size of your descriptive text to 14.

Problem 7.3 (Regex Parsing)

Suppose that you are now working on the `python` “back-end” (that is, the part of the software that is managing and manipulating the data) of your **FAU**-internal pizza delivery service from Problem 7.1. 50pt

Say you have a log file where in each line there is a percent-encoded¹² HTML POST request to your website. Each of them encodes the *name*, *address* and *pizza choice* of one order, like in the following examples:

```
POST name%3DTheo+McTestPerson%26address%3Dkollegienhaus%26pizza%3Dsalame
POST name%3DMax+Musterfrau%26address%3Dkollegienhaus%26pizza%3Dvegetaria
POST name%3DBea+Beispielname%26address%3Dmartensstrasse%26pizza%3Dsalame
...
```

Such a file is also being provided along with this exercise.¹³ Write a program that first reads that file and creates a list of `python` dictionaries (one for each order, with the keys “name”, “address” and “pizza”) out of the included data.¹⁴ Use regular expressions to find the corresponding values in the data.

The program should then do the following:

- Your program needs to compute (and print) what sorts of pizzas were ordered and how many of each are needed in total.
- Your program should also print all addresses that the delivery driver needs to go to.
- Lastly, your program should compute and display the total amount of money that you would expect to be paid for this delivery (you can assign an arbitrary price to each variety of pizza for this exercise).

¹²See: <https://en.wikipedia.org/wiki/Percent-encoding>

¹³Found here: <https://kwarc.info/teaching/IWGS/materials/console.log>

¹⁴You can read up on how to create and/or add key-value-pairs to dictionaries in a program here: https://www.w3schools.com/python/python_dictionaries.asp

8 Assignment 8 (Project Design & Prototyping) – Given 22.12.2018, Due 11.01.2019

Note: We realize that the workload of the last weeks was high, especially for those of you currently also attending GDI. Therefore, we would like to emphasize that this project is *purely optional*. If you feel like you still have trouble understanding some of the subjects of the lecture, we recommend that you don't participate in the project, and instead try to tighten your understanding of the topics of the last weeks.

Both Jonas and Philipp are more than willing to help you grasp the presented concepts. We strongly advise you to repeat the previous exercise sheets and we are happy to accept solutions to these assignments and give you feedback for them.

We know that programming can be hard and that learning a large amount of new concepts in rapid succession can feel overwhelming, but we are sure that if you go back to these exercise sheets now you will realize just how much you've learned so far. Please make use of this opportunity.

Again: This project is intended for those who can't get enough of programming. You will do just fine in the exam, if you did not complete the project, but instead have a solid understanding of the concepts presented in the lecture.

Problem 8.1 (Project 02)

Write a short summary (bullet points) of the core interactions the user is intended to perform on your project application. In which order are they expected to use the interface elements (checkboxes, buttons, dropdown menus, ...)? 40pt

How must these elements be wired internally? If the user interacts with a particular element, what is internally supposed to happen? What data elements will be shown to the user? Knowing these expected interactions, how is your data supposed to be organized in order to provide the wanted functionality? Which data structures (lists, dictionaries, lists of dictionaries, ...) do you expect to be using? Where does the data come from (files on the hard drive like CSV tables, from other websites like Wikipedia, ...)?

Problem 8.2 (Project 03 (Paper Prototype))

With the knowledge of your desired interaction scheme (from Problem 8.1), draw (on a sheet of paper) the basic layout of your application. An very simplistic example of what this could look like is shown on page 59 of the course notes. 20pt

Scan or photograph your design for the submission.

Problem 8.3 (Project 04 (HTML Prototype))

Create a first prototype of your design as a HTML/CSS website (whether or not your application was designed as a website, since this is only a *prototype*). 40pt

Try to position each element as planned in Problem 8.2. Since we are only working on the layout of the page in this exercise, your buttons etc. don't have to work yet.

The whole IWGS team wishes all of you happy holidays!

9 Assignment 9 (Trees in Python and XML) – Given 12.01.2019, Due 18.01.2019

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 9.1 (Trees in Python & Recursion)

During the lecture, you learned about the very important data structure of *trees*. In this exercise we will be taking a closer look at *binary* trees (trees where every non-empty node has exactly two children) of integers. 80pt

One way of implementing trees in Python is *nested dictionaries*. Every node in the tree is either the empty dictionary (`{}`, this is called a *leaf* of the tree) or a dictionary with the keys `"value"` (which for this exercise will be an integer), `"left"` and `"right"`. The latter two are both dictionaries that are again either empty or trees with a value and two children.

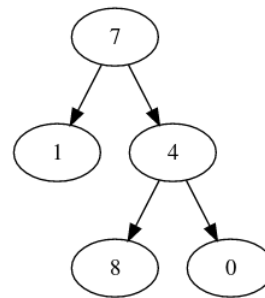
You can find an example tree constructed in this manner in the code snippet below and a visualization of the same tree below.

Example for a tree as nested dictionaries.

```
treeA = {"value":1, "left":{}, "right":{}}
treeB = {"value":8, "left":{}, "right":{}}
treeC = {"value":0, "left":{}, "right":{}}
```

```
treeD = {
    "value": 4,
    "left": treeB, "right": treeC
}
```

```
exampleTree = {
    "value": 7,
    "left": treeA, "right": treeD
}
```



A visual representation of the tree encoded as dictionaries on the left.

Write a Python function called `treeMinimum` that takes a (non-empty) tree as input (you can take `exampleTree` from above as a test case, but it needs to work for all trees constructed this way) and finds the *smallest* integer that any node in the tree carries. For example, for the tree above, your function should return 0.

Hint: To do this, you will have to understand and use *recursion*, another very important concept in informatics¹⁵ (read up on this first!). This means that, from within `treeMinimum`, you will need to call `treeMinimum` again on the children of the node you're currently inspecting. This will give you the smallest integers that are anywhere below the current node. The fact that you are

¹⁵For more on this, see: [https://en.wikipedia.org/wiki/Recursion_\(computer_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science)) or https://www.python-kurs.eu/rekursive_funktionen.php

solving the larger problem (minimum of the larger tree) by calling the same function on smaller problems (minimum of the left and right branches respectively) makes this *recursive*.

You might also find the `min`-function¹⁶ helpful.

Problem 9.2 (XML)

In this exercise, we will discuss the XML language family. Please answer the following 20pt questions (at most a few sentences each):

1. What is the difference between XML and HTML?
2. What roles do trees play for those two?
3. Name at least three uses of XML.

Give a short example of valid XML code that you have written yourself. Also give a small example of *incorrect* XML and explain why exactly your example is incorrect.

¹⁶This function finds the smallest element in a list. See: <https://docs.python.org/3/library/functions.html#min>

10 Assignment 10 (Generating HTML with Python) – Given 18.01.2019, Due 25.01.2019

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 10.1 (Generating HTML elements)

One of the biggest advantages in programming is *automation*, recognising structured tasks that come up a lot and replacing human effort with computation. In these exercises we will try and automate the “boring” parts of generating simple websites in HTML. 25pt

First, write two functions, `wrapH1` and `wrapP`, that take one argument and *return* (not to be confused with “print”!) a string. The return string should be an opening tag (<h1> and <p> tags respectively), followed by the argument to the function, and then the matching closing tag.

Problem 10.2 (Generating website skeleton)

Next, write a function `wrapQuickFacts` that takes 5 string arguments and returns a string describing a HTML table¹⁷ listing these arguments under the categories “Name”, “Job Title”, “Date of Birth”, “Email”, and “Website”. 25pt

Finally, write a `python` function `wrapSkeleton` that analogous to those in Problem 10.1, return the general structure of a basic HTML page¹⁸ as a string. The function should also take a string as an argument that is inserted between the opening and closing <body> tags in the returned string.

Problem 10.3 (Generating complete websites)

After we have solved the smaller problems, it is now time to combine the solutions into a (slightly) bigger program. 50pt

Using your results from Problem 10.1 and Problem 10.2, write a `python` function `generateWebsite` that, given a dictionary with appropriate data¹⁹ as input, generates (i.e. returns the HTML string that describes) the complete website including a heading, the table and a paragraph of flavour text and saves it into a `.html` file.

Generate one of these websites for all entires in `peopleList` using the functions you wrote.

¹⁷If you need a refresher on this, you can find this structure here: https://www.w3schools.com/html/html_tables.asp

¹⁸See also: https://www.w3schools.com/html/html_intro.asp

¹⁹You can find a file with example data here: <https://kwarc.info/teaching/IWGS/materials/people.py> You can either copy-paste these or have the file next to yours and use `import people` in your file to be able to use `people.peopleList`.

11 Assignment 11 (Web-Applications with Bottle) – Given 26.01.2019, Due 01.02.2019

Note: Exercises need to be handed in via StudOn at 23:59 on the day they are due or earlier. Please use only the exercise group of your tutor to hand in your work.

If any concepts here seem unfamiliar to you or you have no idea how to proceed, ask a fellow student, your tutor or consult the lecture materials at <https://kwarc.info/teaching/IWGS/>.

Problem 11.1 (Hello WebApp World)

In these exercises, we will take a closer look at web-applications, templating and HTML routing. Concretely, we will be using the Bottle framework²⁰, as demonstrated in the lecture. 10pt

Set up a web-application on your `pythonAnywhere.com-account`²¹ using the Bottle framework. `pythonAnywhere.com` has a quickstart option for bottle.

Set up the following routes (pairs of URLs and python functions that return strings):

- A client navigating to the root directory of your webapp ("/") should receive a standard "Hello World" message.
- A client navigating to `"/hello/<name>"` should find a greeting message personalised with the name given in the URL (`"/hello/Philipp"` greets Philipp, `"/hello/Jonas"` greets Jonas, ...).

Have at least one name (your choice) be treated differently than all others (for example: all names get a nice message by default, but the name "GrumpyCat" gets an annoyed message).

Problem 11.2 (Routing a HTML form)

In the following exercises, we want to build a small, but complete (!) web application where users can submit reviews for media (books, movies, ...) that get saved into a "database" and can be viewed later. A lot of these exercises will ask for HTML or python code that is similar to previous exercises. The challenge is to integrate the familiar code into the new context of web-applications and the bottle framework. 20pt

Add a `"/submit"` route to your web app that delivers a HTML form. The form should at least have `input` elements for a title (text), a synopsis (text) and a rating from 1 to 5 (number or radio buttons).

When the submit button (which also needs to be included in the form) is pressed, the form should redirect the user to the `"/submitted"` route (see Problem 11.3) via the `action` attribute. Make sure that the method used for this is a GET request (how can you specify this?).

Problem 11.3 (HTML GET Requests)

Now, add a route specifically for GET requests at `"/submitted"` (the target of your 40pt

²⁰See the documentation of bottle for reference: <https://bottlepy.org/docs/dev/tutorial.html>

²¹See this help page for instructions: <https://help.pythonanywhere.com/pages/WebAppBasics/>

submit-redirect from Problem 11.2). Since we're dealing with a GET request, the information submitted through the form will be encoded in the URL.

The corresponding function should read the title, synopsis and rating from the HTML request (see the bottle documentation or the lecture materials for examples) and append them to a file²² called `database.txt`²³.

You can append one line of text to the file per entry in the database, with the title, synopsis and rating separated by semicolons, for example.

Problem 11.4 (Displaying the database)

Finally, add a `"/database"` route to your web app that reads the aforementioned database file (`database.txt`) and displays its contents as a HTML page. This page should contain a heading and an unordered list (the `` element), in which each entry in the database (= line in the file) is one list item (`` element). 30pt

²²Even though the function must ultimately *return* a string from which a HTML page is constructed, it can write to a file before doing so as a side effect.

²³This file will appear next to your other files in your `pythonAnywhere` directory. It is enough to simply append to the file, `python` will create the file if it does not exist yet.