Instructions: use the online assignment administration platform to indicate which problems you have completed *and are able to present.* Create one short plain text file (`.txt` or `.md`) **per problem** in which you discuss your solution. Name these files `N.{txt, md}`, where `N` is the problem number in this list. Upload your solutions to our Moodle course.

1. Create a file that is 100 years old by learning about the `touch` command. Can you also create a file that says it is from the year the University of Graz has been founded (1585)?
2. Create three empty files named `file1.txt`, `file2.txt`, `file3.txt` in your current directory. Then, list all files in the directory sorted by their creation time (oldest first, newest last).
3. Use `curl` or `wget` to download the text file that is hosted at https://gist.githubusercontent.com/MattIPv4/045239bc27b16b2bcf7a3a9a4648c08a/raw/2411e31293a35f3e565f61e7490a806d4720ea7e/bee%2520movie%2520script, which contains the script of the *Bee Movie*. Use `grep` to find all lines starting with the word *Bee*. How many such lines are there?
4. Complete the free demo part of https://vim-adventures.com/. What is in the chest in the maze?
5. Complete the tutorial shipped with every installed version of vim (command: `vimtutor`) or neovim (command: `nvim`, followed by `:Tutor`). *Alternatively:* Learn how to navigate, edit, search, replace using any CLI / terminal editor of your choice.
6. The file `webserver.log` contains the following content:

```
203.0.113.24 - - [25/Mar/2024:10:15:32 +0000] "GET /index.html HTTP/1.1" 200 2326
198.51.100.10 - - [25/Mar/2024:10:15:33 +0000] "GET /images/logo.png HTTP/1.1" 200 5432
203.0.113.24 - - [25/Mar/2024:10:15:34 +0000] "GET /css/style.css HTTP/1.1" 200 1054
192.0.2.42 - - [25/Mar/2024:10:15:35 +0000] "POST /login HTTP/1.1" 302 0
203.0.113.24 - - [25/Mar/2024:10:15:36 +0000] "GET /about.html HTTP/1.1" 200 3571
192.0.2.42 - - [25/Mar/2024:10:15:37 +0000] "GET /dashboard HTTP/1.1" 200 5123
198.51.100.10 - - [25/Mar/2024:10:15:38 +0000] "GET /products/item1234 HTTP/1.1" 404 1762
203.0.113.24 - - [25/Mar/2024:10:15:39 +0000] "POST /contact HTTP/1.1" 200 982
```

   Use the programs `cut`, `uniq`, `wc` to determine the number of different visitor IP addresses in this logfile. If you are satisfied, test your solution against the real sample logfile from here: https://raw.githubusercontent.com/elastic/examples/refs/heads/master/Common%20Data%20Formats/apache_logs/apache_logs.
7. At https://www.gutenberg.org/cache/epub/84/pg84.txt you can find a copy of the book *Frankenstein; Or, The Modern Prometheus.*
   - How many words and how many individual (string) characters are in the book?
   - Print all lines containing the word *Frankenstein* (including a context of 1 line before and after).
8. There is a compressed `.tar` archive located at

   https://imsc.uni-graz.at/hackl/teaching/cherry_tree.tar.gz
   - Use `curl` or `wget` to download the archive.
   - Learn about the `tar` command, figure out how to unpack the downloaded archive.
   - Use `find` to find all files with a `.txt` file ending. How many are there?
   - How many files contain the word `cherry`? How many of these "cherry-files" have a file name ending with `.txt`?
9. Write a simple bash script `greeting.sh` that asks the user for their name and then greets them personally, e.g., `Hello, Benjamin!`. Use `read` to get input from the user, and make sure your script is interpreted by `bash`. What do you need to do to try out your script?
10. The code snippet `:(){ :|:& };:` defines a bash function and calls it; the code is, however, written in a rather obfuscated way. What does it do? (Try running it, if you are brave.)

11. Learn about the terminal multiplexer `tmux` by following the short tutorial at https://hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/.
12. Read the first chapter[1] of the book Software Engineering at Google and prepare to discuss the content in the exercise session.
13. Learn about the command `ssh-keygen` and create a new SSH key pair using the option `-t ed25519` (what does it do?). Add the public key (usually located at `~/.ssh/id_ed25519.pub`) to your GitHub account (or any other Git hosting service of your choice if you don't want to use GitHub).
14. Read Chapter 16 (https://abseil.io/resources/swe-book/html/ch16.html) on Version Control of *Software Engineering at Google* and prepare to discuss the content in the exercise session.
15. Complete the chapters *Ramping Up*, *Moving Work Around*, *A Mixed Bag* from the interactive browser tutorial on branching with Git at https://learngitbranching.js.org/, take notes with your solutions.
16. Use `git log` to determine how often the file at `sympy/abc.py` in the repository that can be cloned from https://github.com/sympy/sympy has been modified.
17. Learn about *aliases* in Git (https://git-scm.com/book/ms/v2/Git-Basics-Git-Aliases) and setup the alias `lg` to run the command

    ```
    log --graph
        --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
    blue)[%an]%Creset'
        --abbrev-commit
        --date=relative
    ```

18. Create a new repository (`git init`) and commit a file `fruit.txt` containing a list of some of your favorite fruits (one fruit per line). Add some more fruits in a second commit. Explain what happens in each step to your repository when you run the following commands in order:
    - `git revert HEAD`
    - `git reset --hard HEAD~1`
    - `git restore --source=HEAD~1 fruit.txt`
    - `git stash`
    - `git stash pop`
19. The Miller–Rabin (Pseudo)primality test contained in the `openssl` library can be found in their repository at https://github.com/openssl/openssl, in the file `crypto/bn/bn_prime.c`. Use `git blame` to find all line numbers of the "oldest" lines of code in this file. Who has authored them?
20. Create a new repository (or use one of the ones from the other exercises) with at least 3 different commits. Let us call their hashes C_1, C_2, and C_3. Keep an eye on the contents of the `.git/refs` directory in your repository while you carry out the following tasks:
    - Create a new branch `feature` starting at commit C_1, then apply the changes introduced in commit C_3 on top of it using `git cherry-pick`.
    - Tag commit C_3 with the tag name `v1.0`.
    - Create a new commit on top of `main`, then reset `main` to point to commit C_2 using a *hard reset*.

    What are the contents of the files in `.git/refs` after each step? And what about the `.git/HEAD` file?

[1] https://abseil.io/resources/swe-book/html/ch01.html

21. Lean about `git bisect` (https://git-scm.com/docs/git-bisect), then complete the assignment available in the repository at https://imsc.uni-graz.at/git/behackl/swdev-git-bisect.

22. *Interactive Rebase.* Clone the demo repository from

    https://imsc.uni-graz.at/git/behackl/swdev-git-interactive-rebase.git

    Unfortunately, the history in this repository is a (linear, but still) mess! Rebase all commits starting at `a5b4556` and ranging to the tip of `main` interactively to address the following issues:
    - In `ae8efe0`, a `.env` file containing an important secret has been added by mistake. Remove the file!
    - The changes happening in commits `a5b4556`, `ae8efe0`, `950f370` should be consolidated into one commit (with a more verbose commit message).
    - Commit `2a41313` should be split into two separate commits: one for the `multiplication` function, one for the `division` function. Write verbose commit messages.

    (Hint: if fixing everything at once is too intimidating or confusing, go through multiple rounds of interactive rebasing!)

23. Learn about *pre-commit hooks* in Git repositories (https://git-scm.com/book/ms/v2/Customizing-Git-Git-Hooks). Create a new Python project (`uv init`) in a Git repository and add `ruff` as a development dependency, then setup a pre-commit hook running linting and formatting (`ruff check`, `ruff format`). Test your hooks by trying to commit a simple Python file with bad formatting.

24. Type hints: make yourself familiar with the basic syntax introduced by PEP 484, then add missing type hints to the file included in the practice repository at

    https://imsc.uni-graz.at/git/behackl/python-typehint-playground

    Make sure `mypy --disallow-untyped-defs string_operations.py` does not raise any errors or warnings.

25. Read Chapter 21 (https://abseil.io/resources/swe-book/html/ch21.html) on Dependency Management of *Software Engineering at Google* and prepare to discuss the content in the exercise session.

26. Use the Python library `fastapi` (https://fastapi.tiangolo.com/) to write your own simple web API with the following endpoints:
    (a) • `/hello` should return a response with a `message` attribute containing the string `"Hello World!"`.
        • `/hello/<name>` should do the same, but the returned message should be `"Hello <name>!"` (where `<name>` can be any string).
    (b) • `/convert-temperature?value=<number>&from=<unit>&to=<unit>` where `<number>` is a floating point number and `<unit>` is one of `C`, `F`, or `K` should return an object with a `converted` attribute that corresponds to the converted temperature as a floating point number as well as a message attribute that describes the conversion. For example, querying `/convert-temperature?value=100&from=C&to=F` should return

```
{
  "converted": 212.0,
  "message": "100.0°C --> 212.0°F"
}
```

    Make sure that your API replies with a response with status code `400` if the input parameters are invalid (e.g., unknown units).

(c) The `fastapi` library uses *Swagger* to automatically generate documentation for your API. Where can you find it? What do you need to do to have your API support more complex data types like a `Lecturer` object with `name: str`, `id: int`, and `courses: list[str]` attributes?

27. Work through the *Getting started* guide of the `polars` library (https://docs.pola.rs/user-guide/getting-started/), then load the `iris` dataset included in the `seaborn` library (`seaborn.load_dataset("iris")` returns a `pandas.DataFrame`, which you can convert to a `polars.DataFrame`). Print the statistical summary of the data set (`describe`), and find the indices of the flowers with the smallest / largest petal length.

28. *Path of Exile* is an online action role-playing game developed by the New Zealand-based developer Grinding Gear Games. It is particularly well known for its highly complex skill tree system (https://poeplanner.com/). The complete skill tree for the latest version of the game (v3.27 just released on 31. October 2025!) is available as `data.json` in the repository at

    https://github.com/grindinggear/skilltree-export.

    (a) Use the `load` function of the `json` module to load the data as a dictionary. Data on individual nodes is stored in a dictionary under the `nodes` key. Construct a `polars.DataFrame` whose rows correspond to the individual node data dictionaries. How many nodes are there in total?
    (b) Find all nodes whose `name` is the longest or the shortest, respectively. Which are the nodes that award the most distinguished `stats` (with respect to the length of the list under the `stats` key)?

29. Use the GeoSphere Austria API to determine which day has been the hottest and which has been the coldest at 12:00 in the past 3 years? Use the measurements from the weather station at the university (ID: 30).

30. Learn about the Python library `beautifulsoup4` (https://beautiful-soup-4.readthedocs.io/en/latest/) and write a short Python script that fetches the *Top Events* page from Graz Tourism (https://www.graztourismus.at/en/events/event-calendar/top-events) and stores it in a `BeautifulSoup` object.
    (a) Extract the names of all events by finding all `h3` elements with the `img-teaser__title` CSS class.
    (b) If you additionally know that the `div`-containers holding the content of the individual event cards have the CSS class `img-teaser__body`, can you also extract the date(s) for each event?

31. Read Chapter 10 (https://abseil.io/resources/swe-book/html/ch10.html) on Documentation of *Software Engineering at Google* and prepare to discuss the content in the exercise session.

32. Turn the code at https://imsc.uni-graz.at/git/behackl/python-typehint-playground into a proper package with Sphinx-generated documentation.
    (a) Add docstrings to all functions (containing at least a one-line summary and documentation for the parameters), then use the `autodoc` extension for Sphinx to automatically generate HTML documentation for this module.
    (b) Pick a theme of your choice from https://sphinx-themes.org/ and use it to style your documentation. Additionally, learn about admonition blocks and include at least two different ones in your documentation.
    (c) References to individual members of a package (modules, classes, functions) can be created using so-called *roles* in Sphinx, written as (for example) ``:func:`some_function` ``. (Note: these are backticks, not single quotes.) Read about roles in the Sphinx documentation and add some to your documentation.

33. Create a simple Python package with `sphinx`-generated documentation. Setup Sphinx such that it can also process Markdown files (MyST: https://myst-parser.readthedocs.io/). Configure Sphinx to correctly parse NumPy or Google style formatted docstrings (https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html).

34. Use the CPU and memory profilers discussed in the lecture (`snakeviz`, `line_profiler`, `memray`) to analyze the sorting algorithms hosted at https://missing.csail.mit.edu/static/files/sorts.py. Which algorithm "wins" in terms of run time, which in terms of memory consumption?

35. There are various ways to accelerate Python code. Two such mechanisms are implemented in form of the Python library `numba` (https://numba.pydata.org) and the C extension framework `Cython` (https://cython.org). Read up on one of the two and work through the respective introductory tutorial (https://cython.readthedocs.io/en/stable/src/tutorial/cython_tutorial.html for `Cython`, https://numba.readthedocs.io/en/stable/user/5minguide.html for `numba`).

36. (a) Read parts of Chapter 11 (https://abseil.io/resources/swe-book/html/ch11.html, in particular the sections *Why Do We Write Tests?*, *Designing a Test Suite*, *The Limits of Automated Testing*) on Testing, ...
    (b) ... as well as the *Preventing Brittle Tests* section, the *Conclusion* and *TL;DRs* of Chapter 12 (https://abseil.io/resources/swe-book/html/ch12.html) of *Software Engineering at Google*

    and prepare to discuss the content in the exercise session.

37. Add unit tests (using `pytest`) to bring the test coverage (determined via `pytest-cov`) for our demo project in https://imsc.uni-graz.at/git/behackl/python-typehint-playground to 100%.

38. Learn about *parameterized tests* in `pytest`. Write a parameterized test for the function

```python
def is_sum_even(x: int, y: int) -> bool:
    return (x + y) % 2 == 0
```

    that checks that the sum of two even numbers and the sum of two odd numbers is even, and that the sum of an even and an odd number is not even.

39. The Python core library `logging` offers rich features for (properly – i.e., not by `printing`) emitting log messages to users running your code.

    (a) *Setup:* Create a new (minimal) Python package and paste the code snippet

```python
import logging

logging.basicConfig()
logger = logging.getLogger("navi")
logger.setLevel(logging.INFO)

logger.debug("Wake-up call ...")
logger.info("Hey! Listen!")
```

    into your `__init__.py` file. Run the code and consult the `logging` documentation to find out what the code snippet does and why the output looks the way it does.

    (b) *Testing:* Implement a function that, when called, uses the `logger` to emit a warning. Learn about the `pytest` fixture `caplog` and use it to write a test checking that your warning really has been emitted.

40. Create a GitHub repository containing a (minimal) Python package, then add a linting pipeline running `ruff format` and `ruff check`. Make sure to pass the correct options such that the

pipeline fails when "bad" code is pushed to your repository. Push at least one *bad* commit to test your pipeline.

41. Create a GitHub repository containing a (minimal) Python package with some functions that have docstrings and setup rendering your documentation with Sphinx (you are free to use the repository from one of the previous exercises). Setup a pipeline that runs the documentation build and stores the output as a *build artifact* on every commit pushed to the `main` branch. *Optional, if you are interested: deploy the rendered documentation from the artifact to GitHub Pages.*

42. PostgreSQL is a production-grade database system whose Docker image (`postgres`) is hosted at https://hub.docker.com/_/postgres.
    (a) Create a simple `compose.yml` file with only one service which runs the `postgres` image; the instructions in the readme from DockerHub are useful. **Do not, however,** put the username / password for the database in your `compose.yml`: instead, use environment variables defined in a `.env` file.
    (b) Start (`docker compose up`) the container and spawn a shell (`docker compose exec -it servicename psql --user postgresuser`). Run the following commands to create and populate a table:

    ```
    CREATE TABLE fruits (name TEXT, color TEXT, origin TEXT);
    INSERT INTO fruits (name, color, origin)
    VALUES
      ('Apple', 'Red', 'Italy'),
      ('Banana', 'Yellow', 'Ecuador'),
      ('Kiwi', 'Green', 'New Zealand'),
      ('Strawberry', 'Red', 'Spain'),
      ('Grape', 'Purple', 'France'),
      ('Lemon', 'Yellow', 'Italy');
    ```

    You can verify that the data is stored by running `SELECT * FROM fruits;` in the database shell. Stop and remove (`docker compose down`) your container, then bring it back up. Are the fruits still there?
    (c) Again, stop and remove your container, then modify `compose.yml` and specify the directory `/var/lib/postgresql/data` inside the container to be a named volume. Repeat the steps from (b) – did anything change? What is the output of `docker volume ls`?

43. Read this two-part blog post on *How to do Code Reviews Like a Human* by Michael Lynch: https://mtlynch.io/human-code-reviews-1/, https://mtlynch.io/human-code-reviews-2/. Prepare to discuss the content in class; do you agree with his suggestions?

44. Head to https://github.com/KFU-DATB31UB-25W/animal-fun-facts, read `CONTRIBUTING.md`, create a new issue or comment on an existing one – then fork the repository and submit a pull request.

45. Find the repository for any open source project of your choice and prepare a short *case study* on their development habits (cf. Slides 14 + 15 in the slide set of Chapter 7).