Table of Contents

1. Introduction

Welcome to the exercises book of the *assignment* **Coding**. Practically applying what you understand conceptually is probably the most important step in really acquiring the skills we aim to teach.

In the first chapter, the exercises will not (or very rarely) go beyond what is taught in the concepts book, but the further we get in the course, the more the exercises will be used to help you extend the knowledge of the specific language.

1.1 Feedback

As with the concepts, I would like you to report any issues you may find with the exercises. Additionally, if you're already well versed in one of the languages we teach, you may think of exercises that would be more applicable, or give another approach to the material. I'm keen to hear your suggestions!

Naturally, I'd like to hear any kind of feedback, so please don't hesitate to let me know your thoughts!

1.1.1 Time Keeping

For each exercise, there will be an indication of how much time we expect is needed to complete it. Don't spend more than twice that amount of time on any exercise, but rather make sure to discuss it and to have a look at the solutions later.

If you wouldn't mind noting the time each exercise takes you to complete (and reporting these back along with your own estimate of how well you knew the material before starting the course), that would help us better understand two things:

- how difficult an exercise is on average - we will be able to derive this from the mean time reported back.
- how well we explain the material in the concept book - the variation of times between students will provide data on that.

1.1.2 Journeyman's Piece

The chapters focused on real languages also feature some additional exercises which you are invited to complete should you be interested and have time left. They're called the "Journeyman's Piece" because they mark the end of your journey in a chapter. There's no need to complete these and there won't be any solutions for them, but I'm more than happy to discuss them during the sessions and to have a look at different ways of solving them.

1.2 Structure

The order of the exercises is the same as in the concept book, so it should be easy to find the right exercise to do at any given time! Whenever you're done with an exercise, it makes sense to go back to the concept book and to read the following section.

Of course, it's possible to first read through all the concepts and then do all the exercises, but that has two downsides to it:

- You'll already have additional tools at your disposal, so it's likely that you won't solve the exercise in the way intended. That isn't necessarily a problem, but it may mean that some skills aren't going to be as strong as we would like them to be.
- You may have to re-read large parts of the concept book because you'll have forgotten some parts you'll have read already. On the other hand, maybe that's an advantage, having re-read some parts is probably not wrong.

2. Formal Languages

2.1 Boolean Logic

2.1.1 Truth Tables & Operator Precedence (15')

Please draw truth tables for the following:

- $\neg A \vee \neg B \vee C$
- $\neg (A \wedge \neg (B \vee \neg C) \vee D)$

If you'd like, try to draw the circuits for one (or both) of the above!

2.1.2 Morgan's Law (15')

Now you know how you can transform expressions. Have a go at moving all the $\neg$ into the brackets (and removing the brackets no longer needed) in the following:

- $\neg (A \wedge \neg (B \vee \neg C) \vee D)$
- $\neg (\neg (\neg A \wedge B) \wedge \neg (C \vee \neg D))$
- $A \vee (C \wedge \neg (B \vee C))$

To be sure your solution is right, you can always draw truth tables for both the original and your (draft) solution to check! And if you think this is tedious, wait until you have to debug or reverse engineer some complicated code!

2.1.3 Simplifying Expressions (30’)

Please simplify the following expressions:

- $A \land (\neg A \land B)$
- $A \land (\neg A \lor B)$
- $(A \land \neg B) \lor (\neg A \land B)$
- $(A \lor \neg B) \land (\neg A \lor B)$
- $\neg (A \land \neg B) \lor A \land \neg B$

Does any of the expressions simplify to $\veebar$ (`XOR`)?

You should be done after a maximum of 20’.

2.2 Pseudocode

2.2.1 Control Structures

2.2.1.1 Conditions (15’)

Write pseudocode and draw the decision tree capturing the following logic:

“Depending on the type of customer, there are different animals we show them first: those coming in with children between the ages of five and ten are shown mice. Those with younger children will be shown bunnies initially. If the kids are older than ten, we choose depending on the kids’ gender: boys are shown kitten, while girls are introduced to rats. And adults coming in without kids: if they’re alone, we’ll show them puppies. If it’s a couple, fish are the way to go”.

2.2.1.2 Case Matching (15’)

Here’s another story - does it make more sense to use conditions or case matching? Or should you even mix and match?

“We have two kinds of fish - tropical and goldfish - and three types of rodents: mice, rats, and hamsters. The tropical fish get dry food whereas the goldfish are fed frozen food. The mice get nuts, the rats are additinally fed some meat. The hamsters get grain.”

Also draw a decision tree to show what you mean.

2.2.1.3 Loops (15’)

We are going to sort some mail. Please write pseudocode to handle the following:

- on a conveyer belt, we get bags full of mail: parcels, postcards and letters
- each bag has to be sorted item by item - parcels go left, postcards and letters have to be sent to the right

Also draw a control flow diagram to visualize what’s happening!

2.2.2 Data Structures

2.2.2.1 Constants and Variables (25’)

- To get some practice, use pen and paper (or some digital tool) to create a diagram of the code in the corresponding section of the concept book!
- Try to change the code such that the table has less rows (the code takes less steps to execute) and check your result by creating your own table to track variable values while the code executes! To make sure it still works, try assigning 4 to `input` (instead of 3) and see whether `result` is 24 at the end!

As a reminder, this is the original code:

Pseudocode

```
constant input = 3

variable result = 1
variable counter = 1

loop
  result = result * counter
  exit if counter == input
  counter = counter + 1
end loop
```

2.2.2.2 Arrays (15’)

Let’s create a little program to add up all the numbers in an array. Check your solution by tracking variable values along the control flow! Like before, just declare your input as a constant called `input`.

Before you start, though, let me introduce a (“shorthand”) syntax we will use: `counter++` means `counter = counter + 1`;

2.2.2.3 Maps (15’)

Please write a program that adds the number of corners of shape names given in an input array. Add a few more shapes to your map to make it more interesting.

2.2.3 Program Structures & Data Types

2.2.3.1 Methods (35’)

- Convert our code for adding up shape corners to a method.
  - Should it be a procedure or a function? Why?
  - How do you call the program?
- How do you combine this new method with the existing `calculateFactorial` to calculate the factorial of the sum of the corners of a triangle and a rectangle?
  - Remember, the method’s signature is `calculateFactorial(input: number) => number`.
- As a final step, let’s create a complete program which reads a user’s input and calculates the factorial of the number entered. Reuse the code you wrote before!

3. Regular Expressions

3.1 Matching

3.1.1 Literal matching (5')

Try searching for some characters and words. Can you think of things you might not be able to find this way? Take note of them, we'll be learning how to deal with these cases a bit later (if not, let me know)!

3.1.2 Alternatives (10')

Write a regular expression to look for either one of the following three words: `if`, `then`, `else`. Try searching in some code you've encountered in the Formal Languages [1] section. Do you notice anything? Are there any matches you would not have wanted?

3.1.3 Word Boundaries (10')

Re-write your expression from the previous exercise to only match entire words. If you're here after working through the entire concepts chapter, what's the shortest pattern you can use to do it?

3.1.4 Whitespace (10')

For once something to think about quickly: what case would not be covered by writing `/\sif\s/` instead of `/\bif\b/`?

3.1.5 Character Classes (10')

Write concise regular expressions to match the following exactly:

- nit, wit, kit, lit
- Aloof, aloof, AlooF, alooF

[1] `Formal%20Languages.md`

3.1.6 Quantifiers (10’)

How can you rewrite the quantifiers `?`, `+`, and `*` using the “between m and n” notation?

3.1.7 Wildcard (10’)

Write a regular expression that matches any amount (number of things), e.g. 10 cats, 5 mice, 743 monkeys.

3.1.8 Negation (10’)

Write regular expressions that:

- match anything that does not contain any digits
- match anything that does not contain lower case characters

3.1.9 Escaping Characters (10’)

Create regular expressions to match the following exact character sequences:

- `true / false`
- `price [EUR]`

3.1.10 Groups (15’)

Write regular expressions for the following:

- Match entire windows paths (e.g. “c:\users\”).
  - What possible cases do you need to cover? Write down a few test cases before you create your regular expression.
- Validate the typical format of a knock, knock joke [2]
- Replace named groups by positional ones and vice versa in your above expression. Make sure it still works!

3.1.11 Anchors and Multi-Line Matching (10’)

Write regular expressions that:

- Match any whitespace at the beginning or the end of a line
- Find lines that only consist of spaces

[2] https://en.wikipedia.org/wiki/Knock-knock_joke

3.1.12 Lookahead / Lookbehind (10')

Find a manual replacement for `\b`

3.1.13 Regex Delimiters (15')

Match entire UNIX paths (e.g. `/usr/bin/git` or `/`)

3.2 Replacing

3.2.1 Referencing Groups (15')

- Write a regular expression to reformat conditions (`if ... then ... else ... end if`) correctly - regardless of how they're formatted initially and independent of how many lines the "…" span. This is what it should look like:

```
if ... then
  ...
else
  ...
end if
```

- Given a list of web URLs, create a list of clickable links that always use HTTPS.

3.2.2 Case Conversion (10')

Write a regular expression that converts all condition keywords (that is, the "if", "then", "else", "end") to uppercase.

4. JavaScript

4.1 Tooling

4.1.1 Integrated Development Environment (IDE) (10')

Create a new file (e.g. `ide.js`), copy the following code over and run it.

JavaScript

```
console.log("Welcome to learning JavaScript!");
```

4.1.2 Browser (10')

Run the same code directly from your browser's console. In most browsers' console, you could just type your input expression without even having to put it within `console.log`. Try e.b. evaluating `navigator.platform` to see what OS you're running on according to your browser!

4.1.3 Debugging (40')

We haven't really started looking at the syntax yet, so feel free to skip this exercise and just debug some of the code we write later. If you're keen to get started, though, here's a snippet you can use to get into it:

JavaScript

```javascript
var i = 1;
var j;
while (i < 5) {
  j = 1;
  while (j < 4) {
    console.log(i * j++);
  }
  i++
}
```

Start by copying the code to a file, e.g. `debug.js`. Make sure the following are done:

- You are in VSCode
- You have code runner installed
- You have adapted your code runner configuration to suspend code execution immediately
- You have added a run configuration to attach to the debugging node application

Observe the variables `i` and `j` through the execution to see how they change.

> You can also debug directly in the browser! There's a nice little tutorial [a] I would recommend you go through after you've completed the *Basic Syntax* section.

[a] https://developers.google.com/web/tools/chrome-devtools/javascript

4.2 Syntax Mapping

4.2.1 Data Types (15’)

- Evaluate the types of the examples given to make sure the information in the concept book is correct!
- Have a look at the operators you can use on strings. Can you output all the types for the example in one call to `console.log`?

4.2.2 Data Structures

4.2.2.1 Constants and Variables (15’)

Let’s play around a bit (execute your code at each step):

- Declare both a constant and a variable each with types `string` and `number` assigned (four declarations in total).
- Try to assign a new value to all of these. What happens when you try to assign to the constants?
- Can you assign a string to a variable that was previously assigned a number?

4.2.2.2 Objects (20’)

Put down an object in JSON that describes yourself (e.g. age, family, pets, hobbies).

> Search the web to find out how to serialize [a] (convert from internal representation to a transportable format) dates in JSON.

4.2.2.3 Arrays & Maps (20’)

Create two arrays, one holding your favorite passwords (just kidding, use arbitrary ones, of course) and another one holding the URLs of sites these belong to. Because you’re clever, you won’t put them in the same order so that Eve doesn’t know which one to use (☺). Now, create a map that links the indexes of the URL array to the one holding the passwords.

[a] `https://en.wikipedia.org/wiki/Serialization`

4.2.3 Control Structures

4.2.3.1 Conditions (15')

Write the following down as a condition:

- output "fits" if all of the following are true
  - variables `x` and `y` are both smaller than 10
  - `x` multiplied by `y` is greater than 50
- output "uncomfortable" if `x` multiplied by `y` is smaller than 50
- output "out of bounds" if either `x` or `y` are greater than 10

Are we missing something? If so, what?

4.2.3.2 Case Matching (20')

You have a variable x, which has been assigned an arbitrary value. Based on that value, you need to output both value and type with some additional information based on type, e.g. "number: 5; my favorite number is 21".

4.2.3.3 Loops (20')

Let's implement a small sorting algorithm. We will get an array of numbers as input (`const input = [...]` for now) and at the end want that array to be sorted. While the algorithm is running, we want to output each intermediate step to the console (e.g. `Step 3: [1, 4, 9, 3, 26]`).

4.2.4 Program Structures

4.2.4.1 Methods (40')

Write a method that figures out whether a point is inside a cuboid (a box with its edges parallel to the x / y / z axis). The method should return a boolean and receives the following inputs:

- a cuboid defined by two corner points diagonally across from each other (an array containing two objects with x, y and z coordinates)
- a point in space (an object with x, y, and z coordinates)

The explicit syntax for returning an object is shown in the following example:

JavaScript

```
const person = (name) => {
  return { name: name };
}
```

Let's get you acquainted with some online resources that will make your life easier.

- Do some research on StackOverflow [1] to find out the shorthand syntax for
  - returning an object from a function
  - assigning a property value that has the same name as the property name (key)
- You don't even need an account to do this!

[1] https://www.stackoverflow.com

4.3 Language Features

4.3.1 Template Strings & Spread Operator (20’)

Go back to the JSON representation of yourself you created. Write code to transform it from JSON to a JS object and merge that with the following object (feel free to adjust your state of mind to match the reality…):

JSON

```
const meNow = {
  currentActivity: "learning JS",
  currentStateOfMind: "enlightened"
}
```

You don’t need to write much code - do a quick search if you haven’t yet seen how to deserialize JSON!

4.3.2 Variable as Property Key (30’)

Now we have the tools to write a method that gets a specific property from an object. Write a function that gets the following input:

- an object that conforms to the one describing you
- the name of a property

and returns

- the value of the property if it is defined
- a message saying it’s missing if it’s a property that exists in your object but the value is not set
- a message saying the property key is invalid if the property does not exist

4.4 Algorithms

4.4.1 Searching (20’)

If more than one element in an array matches what we’re looking for, `find` is not ideal. What other function can we use to get a new array with all the matching elements in it?

Write a method that returns an array with only even numbers from the input array retained.

This exercise requires a little bit of searching.

4.4.2 Sorting (15’)

Write two functions that sort arrays of numbers:

- `sortAlpha` sorts them alphabetically (12 before 2)
- `sortNumeric` sorts them numerically (2 before 12)

4.4.3 Mapping (20’)

We have an application which provides 3D coordinates as *triples* (arrays with three elements). Unfortunately, our application requires them to be in the form of an object with x, y, and z properties. Please write a function that transforms an array of triplets into an array of coordinate objects.

4.4.4 Aggregating (30’)

Now we have the tools to implement the following function:

JavaScript

```javascript
function objectify(names) {

}

console.log(objectify(["an", "apple", "must", "be", "red"])); // => { "an": 2, "apple":
    5, "must": 4, "be": 2, "red": 3 }
```

4.5 Journeyman's Piece (optional)

If you'd like to further improve your skills, here are two exercises you can do:

- Write three implementations calculating the factorial ($n!$, see Wikipedia [2] for more information) and submit them in file(s) with a `.js` extension.
  1. Use a manual loop
  2. Use `reduce`
  3. Use recursion [3] .
- Analyze the code below and write down answers to the following:
  - What would be good names for `whatAmI`, `foo`, and `bar`?
  - Explain line by line what the code does - you may have to do some research.

JavaScript

```javascript
const whatAmI = (foo, bar) => {
  if (!bar) { return foo; }

  const result = JSON.parse(JSON.stringify(foo));
  Object.keys(bar).forEach((key) => {
    const value = bar[key];
    if (value === null) {
      result[key] && delete result[key];
    } else if (value !== undefined) {
      if (Array.isArray(value)) {
        throw "Array elements not supported!";
      } else if (typeof value === "object" && foo[key]) {
        result[key] = whatAmI(foo[key], value);
      } else {
        result[key] = value;
      }
    }
  });
  return result;
}
```

[2] https://en.wikipedia.org/wiki/Factorial
[3] https://en.wikipedia.org/wiki/Recursion

5. Shell Scripts

5.1 Syntax

5.1.1 Data Structures: Constants and Variables (20')

Write a small script which asks the user what they would like to talk about and responds to their input in a sensible way.

5.1.2 Control Structures

5.1.2.1 Conditions (15')

- Why did we not use `x < 5`, but rather wrote `x -lt 5`? Have a look at the *man pages* of the `[` program (type `man [` in your console) and find out whether `x < 5` would work as well. Would it? If so, why? If not, why not?

5.1.2.2 Loops and Case Matching (30')

- The "for"-loop in the concept book outputs a very specific set of numbers, did you recognize the series? They're called Fibonacci numbers [1] . Write a shell script that outputs the first `count` such numbers.
- Create a little quiz program that checks if your users know the names of regular shapes with a given amount of corners (e.g triangles, rectangles,...)
  - hold the mapping between number of corners and shape names in a map
  - allow the "admin" to provide an array configuring the order of questions.
  - loop through that array and ask the user about the shape with the number of corners at the current position
  - count how many answers the user gets right and inform them about how many were right / wrong at the end
- How can you simulate the map lookup if you're in an old version of bash? Re-write your program to use your suggested answer.
- What does the following do? And why (you may need to do a bit of research to figure out the "why" part)?

Bash

```bash
for i in *
do
  echo $i
done
```

[1] https://en.wikipedia.org/wiki/Fibonacci_number

5.1.3 Program Structures: Methods (20’)

Look at the below program code (the same as you saw in the concept book) and answer the following questions:

- How would a call to the `binary` method look in order to calculate all powers of 2 up to 1024?
- What does `[ -z $base ]` test for?
- Why are we writing `[ $base = "-h" ] || [ $base = "--help" ]` instead of `[ $base = "-h" || $base = "--help" ]`?

Bash

```bash
power_usage() {
  echo "Usage: power BASE EXPONENT"
}

power() {
  base=$1 # first argument passed
  exponent=$2 # second argument passed

  if [ $base = "-h" ] || [ $base = "--help" ]; then power_usage && exit 1; fi
  if [ -z $base     ] || [ -z $exponent     ]; then power_usage && exit 1; fi

  echo $(($base**$exponent))
  return 0
}

binary_usage() {
  echo "Usage: binary LIMIT"
}

binary() {
  if [ -z $1 ];                             then binary_usage && exit 1; fi
  if [ $1 = "-h" ] || [ $1 = "--help" ]; then binary_usage && exit 1; fi

  limit=$1
  i=1
  while [ $i -le $limit ]; do
    power 2 $((i++)) # call "power" with arguments 2 and "i" (then increase i)
  done

  return 0
}
```

5.2 Language Features

5.2.1 Builtin Parameters (15')

- What is the difference between `"$@"` and `$@`?
- Create a small program that shows the difference between `"$*"`, `$*`, `"$@"`, and `$@`.

5.2.2 Expansions

5.2.2.1 Parameters (20')

Implement a method which attempts to read user input and times out if none is provided within 5 seconds. In any case, it prints the output "Continuing with ABC", where ABC is either the input read (if any) or a default value (if timed out).

5.2.2.2 Output (20')

Call the method implemented above and extract the original user input (or default) from its output.

5.2.2.3 Curly Braces (30')

1. Create the following directory structure using one call to `mkdir` and one call to `rm`:

```
+- Appendix A
   +- Page 1
   +- Page 2
   +- Page 3
   +- Page 4
+- Appendix B
   +- Page 1
   +- Page 2
   +- Page 3
+- Appendix C
   +- Page 1
   +- Page 2
   +- Page 3
+- Appendix D
   +- Page 1
   +- Page 2
   +- Page 3
   +- Page 4
```

2. Create a file called "docker-compose.yml" with the below content in every directory with an even page number

```yaml
version: '3.7'
services:
  hello_world:
    image: alpine
    command: [/bin/echo, 'Hello world']
```

5.2.3 Commands: Built-In & Chaining (30')

Create a script which loops over all direct subdirectories (of the current directory), and if there is a file named “docker-compose.yml” runs the command `docker-compose up` (this will fail if you don’t have docker installed - never mind).

6. Unix Tools

6.1 Information Gathering: find (20')

- Locate all files on your computer that have been modified during the past hour
- Run the same command again, but don't show any error output
- Find all log files in your home directory

6.2 Content

6.2.1 cat (10')

Write the names of all files / directories located directly in your home directory and the root directory to the console.

6.2.2 tail (15')

Run the search for files modified in the past 60 minutes in the background and redirect standard output into a file. Use `tail` to output the results as they are being written to the file.

6.2.3 column (10')

Output a CSV (comma separated values) file in the console as a nice table.

6.2.4 wc (10')

What's the difference between the following two commands?

- `wc -l <(ls)`
- `ls | wc -l`

6.3 Text Manipulation

6.3.1 sort (10')

Write the unique names (no duplicates) of all files / directories located directly in your home directory and the root directory to the console.

6.3.2 grep (15')

You want to see some errors when running `find`, but ignore "Permission denied" and "Operation not permitted", how can you do that?

6.3.3 cut (15')

- Show each user's id and username (from `/etc/passwd`)
- Use `cut` to show only the last 4 characters in each line of a file.

6.3.4 awk (20')

- Write a script to kill all you user's processes running bash
- Use `awk` to show each user's id and username - in this order - (from `/etc/passwd`)
- Parse a list of e-mail addresses and create an html document with hyperlinks out of them

6.3.5 sed (15')

There's one more edit command in `sed` we haven't mentioned: `p` prints a given line. How do you print all lines of `/usr/share/dict/words` which start with "a" and end with "z"?

6.4 Miscellaneous

6.4.1 seq (10')

- Write an expression that prints every multiple of three smaller 100 that is not a multiple of seven.
- If you have used a loop above, try again without one. If you haven't, use one now!

6.4.2 curl (10')

Create an expression that will output (exactly and only) one line with the following format: `$ip:$port -> $status`, where `$ip` is the remote IP (v4, not v6), `$port` is the remote port and `$status` is the response code (HTTP) for a given URL.

6.4.3 Journeyman's Piece (optional)

Will write a little program that keeps track of our IT inventory - we want to be able to

- show our inventory
- add an item to the inventory
- remove items from the inventory

Here are the detailed requirements:

- generate the file if it doesn't exist yet
- each item can have the following properties
  - ID (mandatory)
  - date added (mandatory)
  - name (mandatory)
  - type (mandatory)
  - notes
  - date removed
- adding an item
  - reads item properties "name", "type", and "notes" from the command line and
  - generates "ID" and "date added"
  - adds the new item to the inventory
- removing an item by ID
  - sets the date it was removed
  - keeps it from being shown in the inventory listing
- the inventory listing shows the following columns
  - ID
  - type
  - name
  - date added
  - notes
  - date removed (if you implement the functionality to show removed items)

> I suggest saving the inventory in a file. You may simply use a line number as ID

7. Python

7.1 Syntax Mapping

7.1.1 Data Structures

7.1.1.1 Arrays -> Lists (15')

Implement a function `multiplyAt` which takes two arrays and an index as input and returns the product of the values at the given index.

Here's one test case

- `multiplyAt([2, 8, 3], [7, 1, 13], 2)` → `39`

What special cases do you have to deal with? What test cases would you use to check if they work?

> You don't yet have to deal with special cases in your code since we haven't looked at conditions yet. . .

7.1.1.2 Maps -> Dictionaries (15')

Implement a function `resolve` which for a given map checks if there's an existing entry, if so returns it and if not returns a fallback value.

The following are example inputs / outputs:

- `resolve({ "dna": "desoxyribonucleid acid" }, "dna", "nothing found")` → `"desoxyribonucleid acid"`
- `resolve({ "dna": "desoxyribonucleid acid" }, "dns", "nothing found")` → `"nothing found"`

7.1.2 Control Structures

7.1.2.1 Conditions (20')

1. Improve your implementation of `multiplyAt` to deal with the special cases you identified above.

   - If you don't have anything sensible to return, you can return the special value `None`.
2. Write an explicit version of your implementation of `resolve`, not using the `get` builtin method. Do it both with
   - a regular condition
   - a conditional statement

7.1.2.2 Loops (25')

1. Implement a function `sumproduct` which takes two arrays and returns the sum of the multiplied values at the same indexes.
   - Test case: `sumproduct([2, 8, 3], [7, 1, 13])` → `61`
   - What special cases do you have to deal with? Add test cases for these.
2. Implemente a function `feed` which asks the user if they're still hungry. If so, feed them one of the items according to their taste.
   - Test case: `feed({ "greens": [ "broccoli", "beans" ], "carbs": [ "potatoes" ] })`
     1. Are you hungry?
     2. Yes
     3. Would you like greens or carbs?
     4. greens
     5. Have some broccoli!
     6. Are you still hungry?
     7. Yes
     8. Would you like greens or carbs?
     9. proteins
     10. Sorry, we don't serve proteins. Would you like greens or carbs?
     11. greens
     12. Have some beans!
     13. Are you still hungry?
     14. No
     15. See you next time!

> You may want to randomize which item you offer for a given type of food. If so, you can use the following snippet:

Python

```python
from random import randrange
maxNumber = 5
rnd = randrange(maxNumber) # a pseudo-random number from 0 to 4
```

7.2 Languages Features

7.2.1 Modules & Packages (10')

Create a module `misc` containing all the functions you have implemented so far.

Only include the last version of each function (so if two exercises asked you to implement a function with the same name, use the most recent one).

7.2.2 Sets (15’)

Write a function `gcpf` which finds the largest prime factor all numbers given as input have in common.

- Test case: `gcpf([21, 56, 77])` → `7`

You can use the following implementation of a prime number decomposition (or create your own ;-)):

Python

```python
def pnd(number):
  if -1 <= number <= 1:
    return number

  factors = []

  while number % 2 == 0:
    number /= 2
    factors.append(2)

  current = 3
  while abs(number) >= current:
    if number % current == 0:
      number /= current
      factors.append(current)
    else:
      current += 2

  return factors
```

7.2.3 Strings (25’)

Make your `resolve` function interactive and allow for typos. If there is no exact match, check if there is a key where only the first or last character differs and if so, ask the user if they meant that.

Test Case: `resolve({ "dna": "desoxyribonucleid acid" })`

1. What are you looking for?
2. dns
3. Did you mean dna?
4. yes
5. desoxyribonucleid acid
6. anything else I can help you with?
7. karma
8. Sorry, I don’t know anything about that!
9. Anything else I can help you with?
10. No
11. Ok, goodbye!

7.2.4 Unpacking (15’)

1. Rename your function `gcpf` to e.g. `fixedGcpf` and create a new function `gcpf`, which accepts any number of arguments. Pass these to the renamed `fixedGcpf`.
   - Test case: `gcpf(21, 56, 35)` $\rightarrow$ same behavior as before.
2. Rename your function `feed` and create new function `feed`, which accepts any amount of food types. Pass these to the renamed method.
   - Test case: `feed(greens=[ "broccoli", "beans" ], carbs=[ "potatoes" ])` $\rightarrow$ same behavior as before.

7.2.5 Classes (30’)

Create a class `LogEntry`, which contains the following data:

- level (ERROR, WARNING, INFO)
- message

It should be able to

- print a log line from its state. If the instance is initialized with an optional file name parameter, write to the given file instead of printing to the console.
- set its state by parsing a log line

The log format will simply be `$level: $message`, so the following test case should work:

Python
```python
entry = LogEntry("ERROR: file foo.txt not found")
print(entry) # => ERROR: file foo.txt not found
```

7.2.6 Docstrings (10’)

Add documentation to your `LogEntry` class.

7.2.7 Exceptions (20')

Add exeception handling to your `LogEntry`'s parsing of a log line. What could go wrong when parsing? Are there cases in which we can do something sensible?

7.2.8 Operator Overloading (20')

Overload the operator < (`__lt__`) in your class `LogEntry` to allow sorting by log level. Sort order should be `ERROR` > `WARNING` > `INFO`.

7.3 Algorithms

7.3.1 Filtering (15')

Create a function `startsWithA`, which removes words which don't start with an "a".

Test case: `startsWithA(["apple", "pear", "grape"])` → `["apple"]`

7.3.2 Mapping (30')

Implement a function `pnds` to transform a list of numbers into a list of tuples containing the prime number decomposition of these numbers.

Test case: `pnds([15, 99, 100])` → `[(3, 5), (3, 11), (2, 5)]`

7.4 Built-In Libraries

7.4.1 functools (25')

Implement a function `favorites` asking a user to prioritize a list of items according to their taste. Print a summary to make sure you got it right.

- Test case: `favorites(["tea", "coffee", "water", "soda"])`
  1. Please prioritize the following: 1) tea, 2) coffee, 3) water, 4) soda
  2. 2, 3, 1, 4
  3. You like coffee best and soda least!

> - To convert a string representing a whole number to an integer, you can use `int(...)`

7.4.2 datetime (15')

Add time information to your `LogEntry` class. Lines should now look like this example:

```
2020-12-04 21:45:11.947 - ERROR: could not find file a.txt
```

7.4.3 json (15')

Extend your `LogEntry` to add information about the user and an origin ID:

```
2020-12-04 21:45:11.947 - { "level": "ERROR", ↩
"message": "could not find file a.txt", "user": "dooleyt", ↩
"id": "asdflk238dfg347" }
```

7.4.4 re (10')

Change your log line to match `$datetime: $json` instead of `$datetime - $json`. Use a regular expression to parse a log line.

7.4.5 urllib.request (30')

Call the echo service at https://postman-echo.com/headers and print the response as follows: `$httpResponseCode - $headers`. As an example, the response could be `200 - Content-Type: text/plain, Content-Length: 50`.

7.5 Journeyman's Piece (optional)

Sometimes there is no centralized logging facility, forcing us to work with different log files, each of which can contain data in different formats.

Your task is to write a program which will read two log files and write a consolidated version in a standardized format to an output file. Specifically, you will read two files, one with the format we ended up with above and one which could contain the following example:

```
error at 7:45:03 on Tue, 26/5/2020 - https://www.myserver.com not available ↩
- tracking id is asdflk238dfg347 - user is doej.
```

The log entries should

- be in a standardized format
- contain all the information available
- be sorted according to timestamp

To test your program, generate random log data in the two formats (feel free to do this any way you'd like, but make sure you have at least 25 log entries in each input file).

8. C

8.1 Syntax Mapping

8.1.1 Methods (15’)

There’s not much to do yet that would be sensible, but in order to compile and run your first C program, create a method `add` that adds two numbers and returns the result. Because we don’t know how to print output yet, I’ll give you a test harness:

C

```c
#include <stdio.h>

// your method here

int main() {
  printf("%i\n", add(1, 5)); // => 6
  printf("%i\n", add(-2, 4)); // => 2
}
```

8.1.2 Input and Output: Printing (20’)

Create methods for multiplying (`mul(int a, int b)`) and dividing (`div(int a, int b)`) numbers. Print your results.

C

```c
#include <stdio.h>

// your methods here

int main() {
  // mul(4, -3) => prints -12
  // div(28, 6) => prints 4.66666666667
}
```

You may have to do some research about why your program prints `4.00000000000` instead of `4.66666666667` ;-)

8.1.3 Arrays (40’)

Now that we know about arrays, we can access arguments passed to us on the command line.

What we will do is the following: we will write a program that accepts **exactly** four arguments on the command line (if there are more or less, feel free to just crash, we don't yet know how to do checks...). These arguments will be integer values. We will use the four values to do the following three things:

- create an integer array on the heap (using `malloc`) which has space for three integers. These three integers will be the sums of the pairs of input values (an example will follow). Don't forget to use `free` when you don't need the memory any longer!
- create a local integer array (on the stack) which has space for two integers. These integers will be the sums of the pair of values in the array on the heap.
- print the sum of the two values in the local array in the following form: `And here it is: $n`, where `$n` is the sum calculated.

As an example, we call our program (say, `prog`) on the command line: `./prog 1 8 5 7`.

- In our memory on the heap, we want the values `9` (1 + 8), `13` (8 + 5), and `12` (5 + 7).
- In our local array, we want the values `22` (9 + 13) and `25` (13 + 12)
- Finally, we want to print `And here it is: 47` (22 + 25)

You will need a method to convert strings to numbers in order to add them. Use `atoi` in `<stdlib.h>` to do so.

8.1.4 Conditions & Loops (35')

And now we're ready to write a little program that makes a bit of sense!

Our program is passed an ASCII string as command line argument. We will do a very simple encoding / decoding of it (based on a Caesar Cipher): every character in the input string gets shifted up by a given input number (between 0 and 127), "wrapping" numbers outside of the ASCII range around (subtracting 127). Our program should be able to encode and decode such strings.

As an example, we would like to get the following:

- `./prog encode "Hello, World!" 7` => `Olssv'^vysk(`
- `./prog decode "Olssv3'Dvysk(" 7` => `Hello, World!`

For simplicity, we're not going to allow double quotes in the input.

> You may wish to compare two strings. >ou can do so using the function `strcmp` in the library `<string.h>`. The function takes two strings (`char*`) as input and returns `0` if they are equal.

8.2 Journeyman's Piece {#c-journeymans-piece} (optional)

Remember our journeyman's piece from the Bash chapter? Let's create something very similar, but without persistence to a file. Instead, we're going to use a two-dimensional array to store the information we require, and we're going to keep the program running, waiting for user input.

You can use the below method `currentDate` to get the current timestamp:

C

```c
#include <time.h>
#include <stdio.h>
char* currentDate(char* s) {
  time_t t = time(NULL);
  struct tm *tm = localtime(&t);
  strftime(s, 25, "%c", tm);
  return s;
}

int main() {
  char s[25];
  currentDate(s);
  printf("%s\n", s);
}
```

Here is the basic functionality required:

- show our inventory
- add an item to the inventory
- remove items from the inventory

Here are the detailed (simplified) requirements:

- each item can have the following properties
  - ID (mandatory)
  - date added (mandatory)
  - name (mandatory)
  - notes - up to 100 characters
  - date removed
- adding an item
  - reads item properties "name" and "notes" from the command line and
  - generates "date added"
  - adds the new item to the inventory
- removing an item by ID
  - sets the date it was removed
  - keeps it from being shown in the inventory listing
- the inventory listing shows the following columns
  - ID
  - name
  - date added
  - notes
  - date removed (if you implement functionality to show removed items)

9. Assembly

9.1 Play and Understand (90’)

We’re not going to make you write much assembly code since it’s very unlikely you’ll ever really have to write larger amounts of it. On the other hand, it does help a lot to play around with the code and figure out what works and what doesn’t.

So, here’s the deal: I suggest you get the code from the repository (ask me to add your user to get access!) and try to see what you can change without breaking it and how it affects the program’s behavior. The more you play with it, the better you will understand!

9.2 Journeyman’s Piece {#as-journeymans-piece} (optional)

- Write a program which calculates the factorial up to a command line argument given.
  - As an example, if you compile to an executable file named `prog`, we will call `./prog 5` to have `Here it is: 120` (followed by a newline) printed to the console.
- If you’re really keen, you can try to write the program both iteratively and recursively ;-)

10. Java

10.1 Syntax Mapping

10.1.1 Data Structures

10.1.1.1 Constants and Variables (15’)

Let’s play around a bit (execute your code at each step):

- Create variables of types `int` and `char` and assign a value to each of them.
- Increase both your `int` and `char` typed variables by 5 and print them

10.1.1.2 Arrays (30’)

- Create two arrays, the first one (`arrayA`) containing `int`s and the second one (`arrayB`) containing Strings. Print the item in `arrayB` which is at the index represented by the third item in `arrayA`. In other words, if the first array was `[4, 1, 8, ...]`, print the element at index 8 from the second array.
- Use `arrayB` to initialize an `ArrayList`. Now remove the first item in this list which has the same value as the one you printed above and print the resulting list. So if your second array wass `["Foo", "Bar", "Bar", "Foo",` and you previously found the value `"Bar"`, print an array starting with `["Foo", "Bar", "Foo", ...]`.
  - You may want to use the method `Arrays.asList(...)` to convert your array.

10.1.1.3 Maps (30’)

Create a map `shapeNames` containing the inverse mapping for our shapes. As an example, we want `3` to resolve to `"Triangle"`. To ensure consistency, use the entries in our existing `shapeCorners` to create your new map. Here’s some code to get you started:

Java

```java
// your code here

System.out.println(shapeNames); // will print in "arbitrary" order
// => {3=Triangle, 4=Rectangle, 5=Pentagon}
```

10.1.2 Program Structures

10.1.2.1 Interfaces, Classes & Inheritance (75')

We will start creating a tiny program, which we're going to extend in the following exercises. Let's get started:

- Create an executable Java class (one with a `main` method)
- Add an interface `Shape` with the following methods:
  - `calculateArea`
    - Triangle: $\frac{a*b*\sin gamma}{2}$
    - Parallelogram: $base*height$
  - `calculateCircumference`
    - Triangle: $a+b+\sqrt{a^2+b^2-2*a*b*\cos gamma}$
    - Parallelogram: $2*(base+(\frac{height}{sin(angle)})$
- Add classes `Triangle` and `Parallelogram` implementing `Shape`
  - Add constructors to create the shapes from the relevant properties
  - Implement the methods inherited from `Shape`
  - Add `toString` to print a sensible representation of the shapes
- In your main method, create one shape each and print the shape's string representation along with it's area and circumference

> You may find necessary methods in Math [a] .

10.1.2.2 Packages & Imports (15')

We should structure our program a bit. Move your app into package `com.javaproject` and the shape classes into a package `com.javaproject.shapes`. Import the shape classes from `App`.

[a] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html

10.1.3 Control Structures

10.1.3.1 Conditions (30')

Now we can do something useful: when the program starts, ask the user for the shape to print. Depending on their input, create a triangle or parallelogram. Create new constructors for triangle and parallelogram which ask the user for the necessary information to create the shape. In the end, print circumference and area.

10.1.3.2 Case Matching (30')

We might want to add further shapes to our program. To prepare for this, convert your condition to use case matching.

10.1.3.3 Loops (30')

Your users probably don't want to start the program multiple times if they need to calculate areas / circumferences for multiple shapes. Ask them if they want to calculate anything further and only exit the program if they say "no".

10.2 Journeyman's Piece (optional)

For once, we'll do something a bit more fun: we will create a little game of *hangman*. Here are the requirements:

- We need the ability to provide a list of all possible words. It would be nice if this could be configured, but you can also hard-code it.
- The user will be given a short introductory text
- A word will be selected randomly from the wordlist (have a look at `java.util.Random`
- The user will be presented with a representation showing how many characters the word has
- The program will prompt the user to enter a single character
- If the character is part of the word, the representation will be updated to show the position(s) of the character
  - Make sure you ignore the case of the character the user enters
- If the character is not part of the word, the ascii-art representation of a hanging person should be updated (e.g. arm added,...)
  - The game ends after 6 wrong guesses (head, body, arm 1, arm 2, leg 1, leg 2).

I'm sure you'll find lots of different implementations of this game if you look for them. I would suggest you attempt this by yourself first even if you decide to look for them later ;-)

11. VBA

11.1 Syntax Mapping

11.1.1 Program Structures (90')

Create a procedure `Main` which can be called from Excel. Have `Main` call the following three procedures:

1. a procedure `Start`, which simply writes "starting" to the debug window
2. a procedure `WriteToExcel`, which takes two mandatory parameters `forename` and `surname` and an optional parameter `hobby`. Write your forename and surname into the first two cells of the top row of your excel sheet. Define your procedure such that you don't need conditions to decide whether or not to print the optional hobby to the third cell in the row (assuming the cell is initially empty).
3. a procedure `Terminate` which show a dialog with a message passed as parameter.

Execute the `Main` procedure by pressing "Play" (▶) to test it.

11.1.2 Data Types (30')

Go back to the previous exercise and add typing to your methods!

11.1.3 Objects (45')

Have a look at the documentation and find out how you can paste only the formulas or only the values you've copied. Apply that in the code.

11.1.4 Input and Output (45')

Copy an entire row from worksheet "Sheet1" and paste the formulas into "Sheet2". Then copy the row you just pasted and paste it as values in the same place.

Why might we be doing this?

11.1.5 Control Structures (90’)

Loop through all worksheets in your workbook and disable calculation (`.EnableCalculation = False`) if the value in Cell “A1” is empty.

> - To get all worksheets, use `ActiveWorkbook.Worksheets`.
> - The type of the worksheets is `Worksheet`.

Write a procedure `printNumbers` which takes two parameters: a maximum number and an optional parameter specifying “even” or “odd”. Put all numbers from one to the given maximum into your worksheet. If “even” or “odd” are specified, only add those numbers.

11.2 Journeyman’s Piece (optional)

When you have Excel sheets with many formulas, calculations become very CPU intensive. In many cases, lots of rows contain the exact same formulas for each cell. There is one main way to optimize work with such sheets, and we’re going to get you started on it!

You get a workbook which already contains some worksheets:

- Cockpit: starting point
- Data: here’s the data we actually want to have, which would typically contain rows with the same formula over and over again
- Parameters: inputs for calculations
- Raw: raw data, e.g. from an input or selected from a database
- Enrich: additional information to add to the raw data
- Formulas: the formulas which would normally be found in “Data”

There's also some VBA code already available (also listed below). Your task is to complete it in order to get one row of data in "Data" for each row in "Raw", with the formulas in "Formula" applied to calculate it. There should not be any formulas in "Data", only values.

> The data that's already in the sheet "Data" is the solution you should get when running your macro. Make sure to have your macro delete it first!

VBA

```vba
Option Explicit

Public Sub Enrich()
  ' save calculation mode
  Dim Calc As XlCalculate: Calc = Application.Calculation
  Application.Calculation = xlManual
  ' disable updating information shown
  Application.ScreenUpdating = False

  ' TODO: disable calculation in all sheets except "Formulas"
  ' NOTE: you can compare worksheet's `Index` to see if they're the same

  ' TODO: here's your main code getting the data into "Data"

  ' TODO: re-enable calculation of each worksheet
  ' TODO: set calculation of the workbook back to what it was originally
  ' TODO: re-enable information updating

  ' TODO: Make sure the user knows we're done
End Sub

Private Function FindLastRow(ws As Worksheet)
  FindLastRow = FindLastCell(ws, xlByRows).Row
End Function

Private Function FindLastColumn(ws As Worksheet)
  FindLastColumn = FindLastCell(ws, xlByColumns).Column
End Function

Private Function FindLastCell(ws As Worksheet, searchOrder As Long) As Range
  Set FindLastCell = ws.Cells.Find( _
    What:="*", _
    After:=ws.Range("A1"), _
    Lookat:=xlPart, _
    LookIn:=xlFormulas, _
    searchOrder:=searchOrder, _
    SearchDirection:=xlPrevious, _
    MatchCase:=False _
  )
End Function
```