Assessment 2

1. Python theory questions

1. What is a program?

A program is basically an instruction for the computer which tells it what to do, how, where and when to do it. It's an instruction which is written for the computer, interpreted, read and executed by the computer. It can be an operating system or an application. And by a computer we mean any device with a programmable computer, not necessarily just personal computers like PCs or laptops. Servers and washing machines are also programmable and execute programs – one can program the washing machine to perform the laundry according to the type of clothing input – one chooses from a preprogrammed, closed set of options, can set the temperature and type of fabric, and run the program so that the washing machine executes it (does the laundry according to instructions). This can even be done remotely with nowadays laundry machines which come with their own mobile applications.

What is important here, the program has to be written in a language which the computer (machine) can "understand", read and execute the command/instruction. Computer science employs a range of programming languages for different purposes and machines, not every machine reads every language. Depends on the attributes of the machine (type and size of memory and processing power – mobile phones used to have a separate type of OS, washing machines also have different OS from PCs and laptops) and the OS installed on the machine (a Windows application will not run on a Macintosh OS computer, despite its intended for computers, but due to incompatible OS it won't be read and executed).

2. What is a process?

A process is an instance of a program that runs (is being executed) on the computer using its computing memory. In Windows if you open the Task Manager, an application that lists all the currently running programs, it lists them as processes and if you want to end one of them, it asks if you want to "kill the process". The processes exist in one of 5 stages of a process lifecycle: start, ready, running, waiting, and terminated; which are leveraged to allow better CPU allocation.

3. What is Cache?

In Polish we call it "handy memory" or "at hand memory" – it's one of the types of computing memory on the computer – in order to enable faster processing of data, so faster work of the computer, it has a kind of a handynotepad for the most recent and most commonly accessed data – if it is frequently accessed, it makes work faster if we do not need to go all the way to where the data is stored and open it up each time to access it, but use the handy shortcut of this data we have in cache. Cache is a temporary memory – it's not for long time storage, but for frequent access and short time storage. As far as I know, the word "cache" in English actually pretty much carries the meaning of what this memory's core function is, PL name is more descriptive. Might be worth to mention that cache is not only 1 per computer – some people might be inclined to think that 1 computer has 1 cache, namely the RAM memory (rapid access memory, if I remember correctly) – however, cache is also employed by the computer's subsystems, browsers have their temporary files, which is their cache of operating, rapid access memory, and due to their voraciousness, graphics cards also have their own cache memory, so as to unburden their demands on the computer's memory and computing resources.

4. What is Thread and Multithreading?

This is complicated and the best description I've encountered was in CodeCademy, so I'll just quote it: “A thread represents the actual sequence of processor instructions that are actively being executed. Each process contains at least one thread and can contain many such structures that all share resources among each other to allow for faster communication and context switching between them. This all allows them to be “lighter” and require fewer system resources. With the hardware advancement of multithreading, individual cores can also execute multiple threads at once, further improving system utilization and responsiveness by more efficiently splitting up tasks. Threads behave differently depending on the environment they were created in. Kernel threads are constructed through system calls to the kernel while user threads are constructed using local function calls. User threads, therefore, allow for more fine-grained control by developers that can be more efficient than their kernel counterparts. However, these user threads have to be mapped to their kernel counterparts in order to be actually executed.”

In my words attempt: 1 process has at least 1 thread, but usually more. 1 process can be for example a Word.exe running. It can concurrently run several threads: one for tracking the cursor & clicks, one for keyboard input, one for auto saving etc. A thread is an end-to-end task, doing 1 particular thing, while a process is a complex instance of a program that requires multiple threads to execute. Execution of multiple threads concurrently is multithreading.

5. What is GIL in Python and how does it work?

GIL is a mutex/process lock which disallows multithread execution – it only allows 1 thread to be executed at any given time, enforces single thread execution. It works by forcing single thread execution on Python programs, even multi-thread ones, because Python has a variable reference counter which is used for memory management – when the counter of total reference number of an object reaches zero, the memory assigned to it is freed up. With multi-thread execution enable there's a risk of this feature of Python (reference counter) getting messed up (counting the same variable several times in threads running concurrently) – it caused leaked memory which was never released (never reached zero to be released, because the counter was upped by the multi-threading) and Python crashes.

6. What is Concurrency and Parallelism and what are the differences?

Concurrency is when we are running several tasks/processes/threads on one and the same CPU – pretty much just like with humans with 1 single brain, 1 computer with 1 single CPU can only process 1 single task at any given time – the fact that we can speak of multitasking in both cases only accounts for the ability to promptly switch between the tasks. I can listen to music while typing this text, but what in fact happens is, I concentrate on typing and when I listen to lyrics, I briefly switch my attention to the song text. Computers are fast enough to allow for seamless switching, so that multithreading is performed (switched from task to task) so fast, that it seems like a computer is doing several things at the same time, when in fact it's performing one task after another in overlapping time periods, just that performing a task takes a split second, and switching also takes a split second, making it seamless. Parallelism is when in the same time the computer truly performs more than 1 task (at the exact same time) – but that can't be done using the same CPU. With more than 1 CPU (i.e. a multi-core processor) a thread/task/job/process can be delegated to the other CPU or other CPUs and performed (run) at the exact same time as another task on CPU 1. Quote: “Differences between concurrency and parallelism: - Concurrency is when multiple tasks

can run in overlapping periods. It's an illusion of multiple tasks running in parallel because of a very fast switching by the CPU. Two tasks can't run at the same time in a singlecore CPU. Parallelism is when tasks actually run in parallel in multiple CPUs. - Concurrency is about managing multiple instruction sequences at the same time, while parallelism is running multiple instruction sequences at the same time. - In Python, concurrency is achieved by using threading, while parallelism is achieved by using multitasking. - Concurrency needs only one CPU Core, while parallelism needs more than one. - Concurrency is about interruptions, and parallelism is about isolation."

7. What do these stand for in programming: DRY, KISS, BDUF

DRY stands for "don't repeat yourself" and its key idea is non-duplication – it underlines the importance of each piece of knowledge/command having only 1 representation, in 1 place. If 1 thing is defined in more than 1 place, when we want to make changes to it, it makes life harder cause we must then make sure to correctly coordinate all of the occurrences of this definition to ensure it's not suddenly in conflict with the newly implemented rule/update. "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." - the Pragmatic Programmer.

KISS stands for "Keep It Simple, Stupid", and it's a neat design idea stemming from US Navy in the 60s (according to wiki). The idea is to keep all and any things as simple as possible – the simpler, the better. Avoid unnecessary additions, waterworks, fireworks, decoration – this is clutter. It's another iteration of the idea that "if you can't explain something simply, you don't understand it well" in my opinion. Simplification is the utmost sophistication, even ancient philosopher said that things are perfect not when not more things can be added, but no more things can be detracted from our "thing". It's easy to over-complicate and add unnecessary things, but harder to prioritise, realise what is really necessary and needed, and how to write it succinctly, shortly, precisely, in a clear way. I don't really like the modern iterations of this principle which try to cover the last S standing for "stupid" with something else – like "Keep It Super Simple" for graphics design – I personally feel it is really useful to stigmatise over-complication this little bit, as it makes life harder for everyone and is just unwise, this word, "stupid", fits well. I prefer the original name of this principle, minimalism is good and clear – also good for the person who will read my code, who is not me, and does not know my code – it's good to write with other people & readability on mind, and it is really unwise to write code in a convoluted, cryptic way that only I will understand. That's counterproductive.

BDUF "Big Design Up Front" – an idea saying that the full design of the program should be readied before work on the implementation begins. That sounds very waterfall alike. What if the client changes their mind? Market changes and we need a new option/functionality added? I think agile approach is better for programming. From what I heard from friends it's a rarity that a client would not have any changes to the project on the go, or that they would clearly define their needs at day 1. This BDUF seems like a good idea, ideally, and that's how I try to do things – i.e. plan a trip or something – but when some changes comes around to be implemented or just befalls us, we must adapt. Maybe then BDUF again with that change and start implementing again. A bit of agile approach, adapt, and back to waterfall BDUF. Depends on the situation and the project I guess, if it is possible to make the A-Z design and then implement it – sure, that's better, faster, cheaper, less problematic. Maybe in reality not many things can be done so – my manager friend said waterfall is mostly used in things like construction, we know what we want to build, and there are not as many unknowns, changes (a completely new type of windows every 2 years or so,

or client suddenly switching their mind from concrete to wood), so it's more possible to plan everything A-Z in steps and then implement it in steps accordingly. From what I've heard from friends in IT and also from my Scrum courses, in IT, agile seems to be a better fit than waterfall approach, but of course both approaches have pluses and minuses.

8. What is Garbage collector? How does it work?

Simply put garbage collection is when the previously allocated memory which is no longer necessary (in use, referenced) is removed (wiped, freed up, stops being occupied by the program which first allocated some object to that memory). Python has an in-built garbage collection in form of this reference counter (see the above answer #9 about GIL for more details), but also had a referencable garbage collector which can be enabled or disabled, set debugging options and the collection frequency for garbage that the reference counter cannot free up.

9. What are 'deadlock' and 'livelock' in a relational database?

Two different types of a standstill / pat type situation where the database queries conflict with one another and are indefinitely stuck, will not be able to finish their work because of codependencies prohibiting them from it.

Deadlock: "In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt." My own words attempt: it's a pat type situation where two (or more) operations collide with each other – they have locked (reserved for use – access, read & write) some resources (i.e. some table or db access) but cannot complete their action because some other resource necessary to complete their action is locked (by another operation, which also cannot be completed due to some resources it requires being locked) – and because they are cross locked, they will keep waiting forever, unless this standstill is detected and one of the operations aborted, to let one of them finish, and then perform the aborted one (or fix the issue which caused the deadlock and then perform the aborted operation). "Necessary Conditions for Deadlock: To successfully characterize a scenario as deadlock, the following four conditions must hold simultaneously: - Mutual Exclusion: At least one resource needs to be held by a process in a non-sharable mode. Any other process requesting that resource needs to wait. - Hold and Wait: A process must hold one resource and requests additional resources that are currently held by other processes. - No Preemption: A resource can't be forcefully released from a process. A process can only release a resource voluntarily once it deems to release. - Circular Wait: A set of a process {p0, p1, p2,.., pn} exists in a manner that p0 is waiting for a resource held by p1, pn-1 waiting for a resource held by p0."

Livelock: "In the case of a livelock, the states of the processes involved in a live lock scenario constantly change. On the other hand, the processes still depend on each other and can never finish their tasks. […] In a deadlock, processes involved in a deadlock are stuck indefinitely and do not make any state change. However, in a live lock scenario, processes block each other and wait indefinitely but they change their resource state continuously. The notable point is that the resource state change has no effect and does not help the processes make any progress in their task." My own words attempt: A live lock is when the processes collide with each other in the way that they keep accessing and changing the same resources and due to codependency (and possibly also the constant overwriting) they can never complete their work. "A real-

world example of livelock occurs when two people make a telephone call to each other and both find the line is busy.”

10. What is Flask and what can we use it for?

Flask is a Python module (or: library) which facilitates development of websites and web applications. It is written in Python and made for Python. It s a web framework, meaning it contains a lot of useful code (modules) for creating a web layer for the Python application, so a website / interface or a web app or an API. Flask with adds already coded functionalities which would facilitate connection management and contains web templates for the visual website interface (in Jinja engine).

2. Discuss the difference between Python 2 and Python 3

The only difference I personally knew was that switch case was added in Python 3 (in Python it s “match case”) because previously I was searching for this function in Python (knew it from Java) and the article describing its used said it was implemented in Python 3, so prior to that it seemingly was not featured. Everything else is knowledge I ve Googled: 1. syntax: Python3 - print (“hello”) versus Python2 - print “hello” 2. integer division result: Python3 – float versus Python2 – integer 3. Unicode text: Python3 – text in Unicode per default versus Python2 – strings needed to be marked with “u” to be considered as Unicode. 4. iteration: Python3 – uses range() versus Python2 – uses xrange() 5. exceptions: Python3 – enclosed in parenthesis versus Python2 – in notations 6. legacy: can port Python2 in Python3, but can t port Python3 in Python2 7. libraries: possibly more of them in Python3, new ones created mainly for Python3, libraries created for Python2 not compatible with Pyton3 8. basically the answer on the internet is that Python3 is easier to understand, has more libraries and features, is widely used, and most importantly, is still being developed while Python2 was discontinued 2 years ago. Since my answer for the Theory Questions assignment I also learned that they wanted to deprecate maps too but in the end didn't because of the backlash from the community telling the devs it would be a bad idea and they still need it.

3. Write a function that can define whether a word is a Palindrome or not (a word, phrase, or sequence that reads the same backwards as forwards, e.g. madam).

Kindly please see the attached .py file named "assessment2_answer3.py" with the Python code including the solution.

4. Write tests for the newly created Palindrome function. Provide a brief explanation for your test case options.

Kindly please see the attached .py file named "assessment2_answer4.py" with the Python code including the solution.

5. Agile methodology, Scrum: name at least 3 types of meetings that are exercised by Agile teams and describe the objective of each meeting.

I actually passed a PSM1 sometime ago, I'll try keeping brief and answer tot he point :) One type is the spring planning meeting, where the backlog items are being review, discussed (I learned a thing from Nneka - the one about "grading" the backlog items to assess their difficulty + complexity blend, so to assess how much time it would need in order to complete - that's a neat idea!) and chosen by the developers team (according to what I was taught on Scrum.org classes, the term "developer team" already includes everyone on board, however, the actual to-do items so backlog items for the sprint are actually chosen by only the developers, as they are the only ones who work on them, so it's a bit like "their choice, their responsibility") for the

upcoming sprint. The duration of each sprint is determined at the beginning of a new project, so at the sprint planning they will know how long a sprint is and how much they are taking on upon them, how many tasks they plan to complete. The idea of each sprint is to deliver completed backlog items, so the plan is to choose enough but not too much also. The chosen items will be the sprint backlog (not the general backlog with all items for the whole project. Also, the difference is, sprint backlog is developers' choice and responsibility, while the actual general backlog is the product owner's // product manager's thing to manage, it's a different thing with a different purpose). Another type of a meeting is the daily meeting - it's brief and it's mandatory. All accounts I've heard of it is that it is a very useful practice to stay atop the current state of things, exchange info, indicate obstacles and ask for help if needed. 3 questions are asked, I don't remember exactly, but their sense is to determine 1 - what WAS done, so what each person did actually do yesterday, 2 - what is planned to be done today, and 3 - if there are any obstacles, and this can really be anything at all that affects the work environment and individual productivity. The third example is the retro meeting, it is where the team (developers, scrum master, product owner, everyone involved in the project unit) discusses the just completed sprint + customer's feedback. This meeting comes after the sprint results (if I remember correctly that was called a sprint review) were presented to the customer, discussed (what is good, what is bad, what ideas customer/devs have, what should be progressed from now on) so they already know the customer's POV as well. The team discusses what went well during the sprint, what went so-so or wrong, and this is the right place to make any changes to the way team works during the sprint, during the daily meetings such changes should not be discussed (no time, not the place and time for that - daily scrum should take like 15mins). I hope my answer is sufficient :) I spoke from memory, the exam for this was like 2 years ago :)

6. Exception handling in Python, explain what each of the following blocks means in the program flow: Try, except, else, finally

TRY, followed by some command that does something, means "try performing this command and see if it goes on without a flaw". If the command that follows "try" succeeds, the result is positive, as expected, and the result is provided. It can however go wrong, for example asked for a number a user can input the word "two" instead of number "2" - if our command is a function working on numbers, it normally requires integers and won't work if provided words like "two" or "three" (unless there is also a logic translating word written numbers to arithmetical, numerical numbers somewhere in our code, but that's besides the point of this answer & example).

EXCEPT means that we are trying to imagine what can go wrong - how our command might end up producing a wrong result. In the above example, when we ask for user input, a lot of things can go wrong with user input, but, staying with our example - if they provide values "two" and "three" when asked for 2 numbers for a numbers calculating formula, then we can anticipate that a person might give an input of a wrong type, and already equip our code with a way to cope with that - ValueError, so "if the value input is string not integer// not the value we expected/require" then ValueError will be produced. Why this is handy: normally if there is an error, the function would throw the error in the console and shut down, stop working. End user would have received some message they might not be able to read (with comprehension). The point of catching errors is so that 1 - our program continues to work despite the error, and 2 - we can customise the error message to send text tot he user which would let them know what was wrong and how to avoid it in the future.

ELSE - after this word there should be a piece of code we want to execute if there are no exceptions, so if our code in "try" executes properly. So if the code in try executes with success here we can add something additional, an additional function or comment or anything.

FINALLY - is a piece of codde that will execute ALWAYS, no matter the result of all the above. If there are exceptions, it will execute, if there are no exeptions, it will also execute. Could be something like "Goodbye" or anything we want to always do in our function.

7. How can we connect a Python program (process) with a database? Explain how it works and how do we fetch / insert data into DB tables from a python program.

Short answer is - using connectors, usually. There are modules (libraries) which greatly facilitate the process (and make it more secure). We can use either mysql.connector library, or sqlite3. In order to use these modules, we need to import them into our project code:

```
# with mysql.connector
import mysql.connector
OR
# with sqlite3
import sqlite3
```

Then we can connect to the database (create a connection to the database - name it and state to what we are connecting, using what method (so in our case, a module/library):

```
# with mysql.connector
db = mysql.connector.connect("my_database.db")
OR
# with sqlite3
db = sqlite3.connect("my_database.db")
```

Then we create the cursor, which is kind of like our operative on the database - shows us where "the connection" is, executes commands / queries we want to do on the database we've accessed. Same code for mysql.connector and sqlite3:

```
cursor = db.cursor()
```

At this point we can formulate and execute queries and commands, so perform the tasks for which doing we've connected to the database - again, same code on both connectors:

```
cursor.execute("SELECT * FROM table")
```

The queried data (if we query, not execute commands) is fetched for us using fetchone / fetchmany / fetchall method to get data from last executed statement. Here for example we get all the data rows - data is stored as 1 object, here called "data":

```
data = cursor.fetchall()
```

We can insert one row by using execute method

```
cursor.execute("INSERT INTO table (name, city) VALUES ('Rob', 'Dublin')")
```

Or we can insert more rows by using executemany. Additionally we can set our SQL code to be variable, which will be called for all rows we want to insert, and we can set values to be inserted as list:

```
sql = "INSERT INTO table (name, city) VALUES (%s, %s)"
val = [
    ('Peter', 'London')
    ('Ben', 'Paris')
]
cursor.executemany(sql, val)
```

At the end we need to commit our changes using commit method but this time on our connection to database

```
db.commit()
```

Then we can close our connection - very important!

```
db.close()
```

8. Given two SQL tables below: authors and books. ● The authors dataset has 1M+ rows ● The books dataset also has 1M+ rows Create an SQL query that shows the TOP 3 authors who sold the most books in total!

This works, although I'm not convinced this is the best method (over 2 millions of rows to be ordered, right?) but for now this is the best (working!) answer I can come up with:

```
USE BookStore;

SELECT DISTINCT author_name, a.book_name, b.book_name, sold_copies
FROM Authors AS a
LEFT JOIN Books AS b
ON b.book_name = a.book_name
ORDER by b.sold_copies DESC LIMIT 3
```

(this SQL script is also attached as a separate file - assesment2_answer8.sql - but this script was short enough to also be placed here without wrecking the readbility)

9. TWO NUMBER SUM: ● Write a function that takes in a non-empty array of distinct integers and an integer representing a target sum. If any two numbers in the input array sum up to the target sum, the function should return them in an array, in any order. If no to numbers sum up to the target sum, the function should return an empty array. ● Note that the target sum has to be obtained by summing two different integers in the array. You cannot add a single integer to itself in order to obtain the target sum. ● You can assume that there will be at most one pair of numbers summing up to the target sum. Sample Input: numbers = [3, 5, -4 ,8, 11, 1, -1, 6] target_sum = 10 Sample Output: [-1, 11] the numbers can be in any order, it does not matter.

Kindly please see the attached .py file with the Python code including 2 solutions.