CODING? ISN'T THAT FOR ENGINEERS?

Once organizations reach a certain size they can no longer store all their data in Excel sheets with names such as "customer_main_file_final(3).xlsx" and are forced to move to a database— in our experience, this usually happens much too late. This matters because once data is in a database it is usually beyond the reach of your average office drone, and to get things done in a modern organization you have to be able to retrieve data, analyze it, present it, and argue your case. To do this you must be able to communicate with databases and applications. Computers run on code, and if you yourself can learn how to read and write code you can harness the immense capacity of computers to do boring and repetitive tasks for you. We designed this course to grant you this power and to also give you an unfair advantage over your future competition in the workplace. We have therefore based the curriculum on our real-life managerial, analytical, and engineering experience.

This experience tells us that the first and most important part before we jump into modern machine learning is to give you a foundation in coding. This foundation consists of learning how to:

- Read and Write Python code
- Understanding the basics of tabular databases and SQL
- Breaking complex projects down into smaller modules
- Being able to test and troubleshoot these modules
- Having the modules interact with one another to perform more complex tasks
- Understand and explain your code in a code review

In this assignment, we would like you to construct a rule-based system and at the same time get used to writing and tinkering with Python and SQL code. If you are new to coding you will have to spend some time before it begins to feel natural, and later fun. It is easy to feel overwhelmed in the beginning, so starting early and attending the help sessions is strongly encouraged, even if you have some prior coding experience.

We have designed the assignment with the goal in mind that it should be modular, which means it consists of a few functions that can be independently run and independently tested, and you should test them. There is an old saying that goes *"You can save hours of testing through days of coding"*, remember this and the assignment will be much easier.

Formal requirements

- Please work in teams of two. If you can't find a partner, contact the course director for help with matchmaking.
- The assignment will be evaluated in three (3) parts:
  - The JSON file will be tested against the correct answers
  - Your code will be inspected by us for readability and efficiency
  - Your participation and exhibited understanding of your own code during the code review

What does success look like?

Since we want to put you in as realistic a situation as possible we will not impose any unrealistic restrictions. But we will introduce realistic restrictions and evaluations.
What this means is:

(1) You do not have to strictly follow the order or naming of the steps in the assignment, but since we will be reviewing your solution together in English it would be wise to not stray too far from the proposed order.
(2) **Both members of the group must understand the code they have submitted and be able to explain and justify their solution in the code review.** If we have cause to suspect that you have submitted code you do not understand we reserve the right to lower your score or invalidate the assignment in its entirety.
(3) You may use AI tools such as ChatGPT or CoPilot to help you <u>learn</u> Python or SQL, but be mindful of (2).
(4) To facilitate (2) and achieve a full score on the assignment adhering to the coding style guide is highly recommended. Time we have to spend on deciphering your code is time we'd rather spend on helping you learn.
(5) You will be working in pairs of two, cooperation outside of this group is not allowed.

The skills you acquire through this assignment will help you in all future assignments we will do together. We therefore humbly suggest that you prioritize this and start working on it as soon as you have the coding environment up and running.

Avoid losing points by not making these silly mistakes

Connecting to the database is a costly operation. Don't open more connections than necessary; this will bog down the database and make your code run slower. As a rule of thumb – if you need to download the same table more than once, you can definitely improve your code.

Grading will be based on the quality of your code and how well it solves the problem. Your code is expected to be well-structured and readable. Use meaningful names for variables and functions and add comments to make your code easier to follow. See appendix C for the coding style guide we use in this course.

While this assignment would take a long time to do by hand, for a computer this is actually quite easy. If your program takes longer than five minutes to run you probably have some major inefficiency in your code.

Stockholm Bank of Economics

If you owe the bank a small loan it feels like having a millstone around your neck. But if you owe the bank a large loan that's like standing on a mountaintop where you can see a whole world of possibilities - Refaat El-Sayed

In this assignment, we will recreate part of the mortgage application process for Swedish banks. Although this assignment has been slightly simplified and altered to fit with the course's intended learning outcomes, it is very close to real-life applications built by regulated financial institutions. As this is an introductory assignment meant to introduce you to coding and working with large amounts of data, we have done our best to remove ambiguities that would be present in a real-life data science project. **In reality, and in later assignments, how you deal with ambiguous requirements and data will be one of the most important parts deciding the quality of your work.**

Regulations issued by the Swedish Financial Supervisory Authority (SFSA) require Swedish banks to ensure that any consumer mortgage provided for the purchase of a house or an apartment will be financially feasible for the customer to pay. **The bank must therefore ensure that customers have sufficient income to support the loan's interest rate, amortization, and also ensure that they can meet the daily needs of themselves and their dependents.** As mortgages and loans are not on fixed interest rates they also have to ensure that the customer would be able to pay if interest rates were to rise by several percentage points.

The decisions of machine learning models are not always easy to interpret and their inner workings can often feel like a black-box. They are also highly dependent on the data they are trained on. Even if we have access to historical default rates, this does not guarantee the ML-model will be able to accurately predict the effects of a major financial crisis. As the SFSA also wants to be able to review whether the Stockholm Bank of Economics adheres to its regulations and wants transparent and explainable rules for any individual decision this task is more suited for a classical rule-based system.

The bank has recently been flooded with loan applications from customers[1], which are helpfully provided to you in a MySql Database. Your end goal is to return the maximum amount of money the bank should be willing to lend to each one of the customers who have requested a loan. We also want to create a system that can integrate with the rest of the bank's infrastructure, therefore we will be writing our decisions into a JSON file that will be readable by other applications within the organization. Your hand-in will consist of both the code and your decisions as two separate files (see Step 9).

[1] *All data used in this assignment has been generated and any resemblance to real people is purely coincidental.*

Step 0 – Connecting to the database

The appendix contains some code for connecting and downloading data from MySQL into Python. Before we begin, see if you can get the connection up and running. Some of the steps below can be solved in either Python or straight SQL. But a good place to start is to make sure you can download the tables into Pandas. We can then go back and modify the code later if need be.

Step 1 – Amortization rate:

Swedish financial regulations only mandate a yearly amortization of mortgages if certain conditions are true.

> *Households taking new mortgages with a loan-to-value ratio over 70 percent must amortise two percent of their loan every year, while households with loan-to-value ratios between 50 and 70 percent must amortize one percent. Households that have mortgages that are more than 4.5 times larger than their total income before tax need to amortise an additional one percent a year.*
>
> \- The Swedish FSA (Finansinspektionen)

The actual regulation is written by lawyers, not coders. This leaves an ambiguity which we clarify below.

- If the loan-to-value ratio > 50% amortize 1 percent yearly.
- If the loan-to-value ratio > 70% amortize 1 **additional** percent yearly.
- If the loan > 4.5 times the gross annual income, amortize 1 additional percent yearly.

Create a function amortization_rate that accepts the following arguments:
property_valuation, requested_loan_size, gross_yearly_income

```
def amortization_rate(property_valuation, requested_loan, gross_yearly_income):
    #Your code here
    return
```

Where property_valuation is the market value of the real estate in question (usually the selling price).

The function should return a number (float) indicating the amortization rate as a decimal number, either {0, 0.01, 0.02, or 0.03}.

Step 2 – Children

A child costs 3700 SEK per month to support. To offset this cost, the government provides a monthly child support to families. The size of this support depends on how many children the family has. The 2024 years numbers can be found in the table below.

No. Of Children	Monthly Child Support
1	1250
2	2650
3	4480
4	6740
5	9240
6	11740

For each additional child, an extra 1250 is given in support.

Write a function that returns the total **yearly** net cost to the family when called with an integer representing the number of children in the family.

Step 3 – Calculate Taxes

Income tax calculation in Sweden involves two taxes: municipality tax (kommunalskatt) and state income tax (statlig inkomstskatt). The rates for municipality tax vary slightly across different municipalities while state income tax is an additional 20 percent tax on annual income earned above 598 500 SEK.

Sweden has some 290 municipalities with varying tax rates, which can be found in the TaxRate table in MySQL. As we only have gross income stored on the customer, we must calculate net income ourselves. This can be done either in SQL or in Python.

STEP 4 – COST OF SERVICING EXISTING LOANS

Customers may have existing loans, such as car loans, credit cards, or similar. The costs of servicing these existing loans must be factored in but can be simplified as the loan amount multiplied by the interest rate. Customers may have an arbitrary number of loans, and each loan may have different amounts and interest rates.

These loans are on non-fixed interest rates. To ensure the customer can pay for their existing loans even if the base rate of interest rises you should **increase the rate of interest on each loan by 3% units** from what is in the database. However, due to consumer protection laws the maximum rate of interest for these existing loans can go **no higher than 20%.**

You can solve this step either in Python with a function or already in your SQL query.

STEP 5 – CALCULATING DISPOSABLE INCOME

To be able to give the requested loan to the customer, the customer must be able to pay all their already existing recurring costs, loans, and taxes, as well as any additional payments stemming from a new loan, such as interest and amortization. Therefore, the credit model needs to consider costs relating to daily living, housing costs, and children. Any money remaining after all existing and additional costs is disposable income.

- The cost of daily living (clothes, food, necessities, etc) for an adult is **10 000** per month.
- Housing costs depend on the form of housing; if the customer lives in an apartment the monthly cost is **4200 per month** and **4700 for a house.**
- The bank's policy is to use a 7**.0% yearly** interest rate when calculating the monthly interest cost for the new loan

You don't need to consider that the loan may decrease over time due to amortization; just calculate based on the initial situation.

Write a function that, when given a customer, calculates the remaining yearly disposable income rounded to the nearest integer.

Step 6 – So, how big can the mortgage be?

What is the largest mortgage that we are legally allowed to offer to a customer?

Answering this question is more challenging than it may seem, given that the loan's size affects the disposable income, which in turn affects the loan's size. This makes solving it using classical calculus a non-trivial task. Instead of solving it algebraically, we will take an iterative approach.

In this step, we will develop a new function that takes the same customer input as the disposable income function. This function will use a loop to iterate over possible loan values. It should call the disposable income function from the previous step to see if the loan can be granted.

Some things to keep in mind when trying to find the maximum loan:

- Customers aren't allowed to have a negative disposable income after the loan has been granted.
- The maximum granted loan can't exceed 85% of the property value by law.
- The bank cannot grant negative loans either, obviously.
- If the customer has a negative disposable income even without a loan, the maximum loan the bank can offer should be 0.

A margin of error of +/- 1000 SEK is sufficient, considering that mortgages are typically large. Round any answer to the nearest integer.

This is the first part of the assignment which might run too slow. If you've written a really inefficient solution it might take days or millennia to complete, and that won't do. If you encounter such a situation, try to look at a single customer in isolation. Are we doing redundant guessing? If you had to do this guessing game yourself, how would you speed it up?

Step 7 – Storing the output

To create a rule-based system that can integrate with the bank's infrastructure, we need to write the data back so as to be readable by other programs within the bank.

```
import json

kitchen_items =[ # A list containing multiple dictionaries
  {"item_id":"101", "item_type": "plate"},
  {"item_id": "102", "item_type": "fork"}
]

with open('12345.json', 'w') as file: # Use this to "dump" the list as a json file
  json.dump(kitchen_items, file, indent= 2)
```

To do this, we will now generate a JSON file containing a list of dictionaries, where each dictionary has the keys listed below. Each key after the application_id corresponds to one step in the assignment.

- application_id # This is the ID of the application as seen in the applications table
- answer_amortization # This is the number (float) from our amortization function, e.g., 0.02
- answer_total_child_cost # The integer containing the output from step 2
- answer_taxes # A float from step 3 containing the total taxes paid by the customer
- answer_existing_loans_cost # An integer of the total loan cost for this customer
- answer_disposable_income # Integer with customers' disposable income at the original requested loan
- answer_max_loan # Total amount the bank would be willing to lend to this customer

Step 8 – Submitting the work

We will review both your Python code and JSON output. As you work in teams of two, add a comment in the beginning of your python code with your enrollment number, and your partner's enrollment number.

```
import json

# Student Enrollment Number: 12345
# Partner Enrollment Number: 54321

# Assignment 1 - 7313
# Etc etc etc
```

(1) Save your code file as .py or .ipynb and name the file: assignment1_<enrollment-number>.py, e.g. assignment1_25123.py
(2) Name your JSON file accordingly: assignment1_<your-registration.number>.json, e.g. assignment1_25124.json

BOTH TEAM MEMBERS HAVE TO UPLOAD BOTH FILES ON THEIR CANVAS.

Appendix A: Connecting your Python script to the database

Installing drivers for MySql

Anaconda comes preinstalled with many packages allowing you to handle relational databases in general. As they come in many different flavors, individual drivers for each database need to be installed separately. In this course, we will be using MySql.

The recommended drivers for PyMySql can be installed either using conda or pip depending on preference.

conda install pymysql

or

pip install pymysql

Connecting to MySql from Python

The generic relational database driver in Anaconda is called SqlAlchemy, which will do all the heavy lifting for us. The following code sets up a connection to MySql, runs a SQL query, and downloads the dataset into a data frame. Copy paste and run the code below to ensure you can connect to the database.

```
from sqlalchemy import create_engine, text
import pandas as pd

host = "mysql-1.cda.hhs.se"
username = "7313"
password = "data"
schema = "MortgageApplications"

# format replaces {} in the string with the respective inputs to the function
connection_string = "mysql+pymysql://{}:{}@{}/{}".format(username, password, host, schema)
# this creates a connection from the string we just created
connection = create_engine(connection_string)
# TODO HERE YOU SHOULD PUT YOUR OWN SQL QUERY
query = """

SELECT * FROM Customer

"""

# use the connection to run SQL query and store it into a dataframe
df = pd.read_sql_query(con=connection.connect(), sql=text(query))
# OPTIONALLY, you can convert the dataframe to a list of dictionaries
list_of_dictionaries = df.to_dict(orient='records')
print (list_of_dictionaries)
```

Appendix B: Testing Your Code

If you're encountering problems during your coding a good idea is to first look at customers and functions in isolation. Break down the problem into smaller parts and try to isolate where the error occurs. Once you've done this, there are multiple ways to find out why your code is behaving weirdly. We'll show you some below.

Please note that none of this is required learning in the curriculum and that **the actual code you use to debug is not something we expect or want to see in your handed-in assignment,** but as you're doing the assignment this can help you unblock yourself.

Print statements:
The absolute simplest way is just to ask the program to show what it's doing by using **print**. This can really slow down the run speed of the program so we highly recommend you only use them for debugging.

We've written a short script to find how quickly a cinema will fill up but it's not really doing what we expect.

Try running the code below.

Warning: You will need to interrupt the script manually.

Select the terminal panel and use CTRL + C on PC and Mac to stop the code from running forever.

```python
remaining_chairs = 1600
people_entering_cinema = 1
cycle_count = 0

# This loop will run forever
while remaining_chairs != 0:

    remaining_chairs -= people_entering_cinema # decrease remaining chair count
    people_entering_cinema += people_entering_cinema # double amount of people entering
    cycle_count += 1
```

Now let's try to figure out why the code runs forever. Try running this code instead.

```
remaining_chairs = 1600
people_entering_cinema = 1
cycle_count = 0

# This loop will run forever
while remaining_chairs != 0:

    remaining_chairs -= people_entering_cinema # decrease remaining chair
count
    people_entering_cinema += people_entering_cinema # double amount of people
entering
    cycle_count += 1
    print("Remaining chairs: " + str(remaining_chairs))
    print("People entering cinema: " + str(people_entering_cinema))
```

Again, you'll need to interrupt the script with CTRL + C. The terminal window should now be full of very long numbers. It might immediately be obvious to you what the bug is, but let's go a little deeper.

Break:

If a function, for loop, or while loop reaches a **break** statement, it will immediately stop what it's doing and move to the next step in the script.

```
remaining_chairs = 1600
people_entering_cinema = 1
cycle_count = 0

# This loop will break after one iteration
while remaining_chairs != 0:

    remaining_chairs -= people_entering_cinema # decrease remaining chair
count
    people_entering_cinema += people_entering_cinema # double amount of people
entering
    cycle_count += 1
    break

print("Remaining chairs: " + str(remaining_chairs))
print("People entering cinema: " + str(people_entering_cinema))
print("Number of cycles: " + str(cycle_count))
```

Your terminal should show something like this.

```
>>> print("Remaining chairs: " + str(remaining_chairs))
Remaining chairs: 1599
>>> print("People entering cinema: " + str(people_entering_cinema))
People entering cinema: 2
>>> print("Number of cycles: " + str(cycle_count))
Number of cycles: 1
```

This would probably not allow us to find the bug in the while statement. So, let’s let it run a little bit longer and see what happens.

```
# This loop will break after twenty iterations
while remaining_chairs != 0:

    remaining_chairs -= people_entering_cinema # decrease remaining chair
count
    people_entering_cinema += people_entering_cinema # double amount of people
entering
    cycle_count += 1

    if cycle_count == 20: # Stop the while loop after 20 iterations
        break

print("Remaining chairs: " + str(remaining_chairs))
print("People entering cinema: " + str(people_entering_cinema))
print("Number of cycles: " + str(cycle_count))
```

```
>>> print("Remaining chairs: " + str(remaining_chairs))
Remaining chairs: -1046975
>>> print("People entering cinema: " + str(people_entering_cinema))
People entering cinema: 1048576
>>> print("Number of cycles: " + str(cycle_count))
Number of cycles: 20
```

So, we’re getting negative remaining chair amounts. That should not be possible. Upon closer inspection the issue seems to be that since the remaining chairs count never hits exactly 0, the while loop continues indefinitely. Let’s fix it.

```
remaining_chairs = 1600
people_entering_cinema = 1
cycle_count = 0

# This while loop will complete with a fixed condition
while remaining_chairs > 0:
    remaining_chairs -= people_entering_cinema # decrease remaining chair
count
    people_entering_cinema += people_entering_cinema # double amount of people
entering
    cycle_count += 1

print("Remaining chairs: " + str(remaining_chairs))
print("People entering cinema next iteration: " + str(people_entering_cinema))
print("Number of cycles: " + str(cycle_count))
```

Alright, there’s our answer: The cinema will fill up during the 11th iteration and out of the 1024 people trying to enter 447 will have to be turned away during that final iteration.

Assert:
Errors aren't always a bad thing. Sometimes we want the program to cease operation and tell us that something happened. For this you can use **assert.**

```
assert 1 + 1 == 2
assert 1 + 1 == 0
```

If you run this code the terminal should give you this.

```
>>> assert 1 + 1 == 2
>>> assert 1 + 1 == 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

When assert encounters a logically false statement (in this case that 1 + 1 is 0) it will throw an error and stop the script. This might look trivial but it can be used to ensure that functions only return certain values or types of values.

Let's say we for some reason think that all squared integers are also even numbers but aren't certain of this. Perhaps it's good to test our assumption.

```
#Function to square an integer
def square_integer(integer):
    return integer**2

#Some integers we want to test
integer_test_cases = [2,4,6,8,3]

#Iterate over test cases
for integer in integer_test_cases:
    #Let's ensure the squared integer is even
    assert square_integer(integer) % 2 == 0 # "% 2" finds the remainder when
dividing by 2
```

This will break once it reaches 3, since 3 squared is 9, which is obviously not an even number.

Try & except blocks:
Just halting the script doesn’t always help us. Maybe we want some more information or make a note that something isn’t working as it should but let the script complete. By using **try** and **except** blocks we can “catch” errors and have the program finish running. Let’s use this to really disprove our earlier hypothesis.

```python
#Function to square an integer
def square_integer(integer):
    return integer**2

#Some integers we want to test
integer_test_cases = [*range(21)] #All numbers from 0 up to 20 in a list
even_when_squared = []
not_even_when_squared = []

#Iterate over test cases
for integer in integer_test_cases:
    #Let's ensure the squared integer is even
    try:
        assert square_integer(integer) % 2 == 0
        even_when_squared.append(integer)
    except AssertionError: # Assert throws an AssertionError, we are catching
it here
        print("The integer " + str(integer) + " is not even when squared.")
        not_even_when_squared.append(integer)


print("These integers are even when squared: " + str(even_when_squared))
print("These integers are not even when squared: " +
str(not_even_when_squared))
```

As you can see from the terminal output the script keeps running even though it encounters false statements, but as soon as the **assert** error is triggered it instead jumps to the **except** block and saves the offending integer in a separate list.

This was a super short introduction to testing and debugging but there’s much more to learn. Lots of senior developers consider writing proper tests the most important part of coding rather than the code itself. If you are doing a quantitative thesis with lots of complicated mathematical or statistical analysis, learning proper testing procedures will help you keep your sanity as the project becomes more complex. If you’re interested in learning more, you can have a look at the actual documentation: https://docs.python.org/3/library/unittest.html

Appendix C: Coding Style

An organization will usually have a specific way of writing code to more easily allow different developers and analysts to understand each others' code. We have based these requirements on PEP-8 which is typically the style guide recommended in Python.

Put all import statements at the top of your Python file; that way, you can easily see what dependencies on other modules your program has.

```python
from sqlalchemy import create_engine
import pandas as pd
import json

# Like this
```

```python
import random
def some_func():
    pass
example_variable = 4

import json
# Not like this
```

Name variables with lowercase letters and words separated by underscore (snake_case): e.g. loan_to_value_ratio

```python
# Name variables like this
example_numeric_variable = 10
maximum_loan_to_value_ratio = 0.7
favorite_municipality ='Grönköping'
```

```python
# Not Like this
A = 15560
Favorite_Municipality = 'Nykvarn'
testvariablefordisposableincomeiterationfour = 0
ThIsIsReAlLYaNnOyInGtOrEaD = True
```

If you define a constant variable (that is a variable that doesn't change value during the program) you should write it in upper case with underscores, e.g. MONTHLY_APARTMENT_COST. This makes it obvious to someone reading that this is unchangeable.

```python
# Name constant variables like this
NUMBER_OF_HOURS_IN_A_DAY = 24
FAVORITE_COURSE_AT_SSE = '7313'
```

Avoid magic numbers in your code. Magic numbers are unique values that have no explanation and the reader is left wondering.

```
## Avoid magic numbers

# Weekly budget calculation
WEEKLY_FIXED_OFFICE_COSTS = 1300 + 6700 + 7400  # Where are these figures
coming from? // Your boss
```

```
## Do this instead

# Constant values for budget calculation from steps 4 and 5
INSURANCE_WEEKLY_COST = 1300
OFFICE_WEEKLY_RENT = 6700
AWS_WEEKLY_FIXED_COST = 7400

# Weekly budget calculation
WEEKLY_FIXED_OFFICE_COSTS = INSURANCE_WEEKLY_COST + OFFICE_WEEKLY_RENT +
AWS_WEEKLY_FIXED_COST
```

Name functions and variables something descriptive.

```
# This is easy to understand
def calculate_max_loan():
    pass
def find_smallest_divisor():
    pass

# This is not
def cml():
    pass
def hello():
    pass
def test23():
    pass
```

If a variable is used in a function it should be passed as an argument into the function.

```python
x = 3
y = 2
def multiply(x):
    # y is declared outside the function
    # y is not constant and may be changed or deleted by other functions,
    # making this code unreliable
    return x * y

# Instead make sure all variables needed are sent in to the function
x = 3
y = 2
def multiply(x,y):
    return x * y
```

One exception to the above rule:

```python
# This is fine as the WEEKLY_FIXED_OFFICE_COSTS should never change
def calculate_weekly_office_expenses(days_in_office):
        COST_PER_DAY = 55
        return ((days_in_office * COST_PER_DAY) + WEEKLY_FIXED_OFFICE_COSTS)
```

These are some of the most classic mistakes we have seen but it is impossible to list everything that would make your code difficult to read ahead of time, use common sense and discuss within your group, or better yet, ask us during the help sessions.