



**FOUNDATION EXAM - 2 HOURS**

|  |  |
| --- | --- |
| **SECTION** | **MARK** |
| **Theory Questions** | 31 |
| **Concept Questions** | 19 |
| **Challenges: students should choose two out of the three to answer** | |
| **Python Challenge** | 25 |
| **SQL Challenge** | 25 |
| **Javascript Challenge** | 25 |
| 1. **TOTAL** | **100** |

**Important notes:**

* Any code files written **must be submitted via a Pull Request to your marker**.
* You can submit theory questions through an edited version of this document on Slack, or on the Pull Request by adding python comments into a new file, or using a text or markdown file.
* You are allowed to submit everything on Slack if it is close to the deadline, as long as you work on getting a pull request up soon after.
* It is an open book exam.
* You are allowed to use PyCharm, MySQL Workbench, and/or CodePen for this assessment.
* If you are using CodePen, please add your CodePen URL in a text or markdown file.
* You only need to complete **two coding challenges**. Please choose two coding challenges of the three provided (Python, SQL and Javascript) to answer.

**Section 1: Theory Questions [31 marks]**

|  |  |
| --- | --- |
| **1.1 What does SDLC stand for?** | **1 mark** |

Software Development Lifecycle (SDLC)

|  |  |
| --- | --- |
| **1.2 What exception is thrown when you divide a number by 0?** | **1 mark** |

ZeroDivisionError Exception

|  |  |
| --- | --- |
| **1.3 What is the git command that moves code from the local repository**  **to the remote repository?** | **1 mark** |

Use Git Push to move code from the local repository to the remote repository.

|  |  |
| --- | --- |
| **1.4 What does NULL represent in a database?** | **1 mark** |

NULL represents a field with no value.

|  |  |
| --- | --- |
| **1.5 Name 2 responsibilities of the Scrum Master** | **2 marks** |

Facilitating the Scrum process – they are responsible for running the Sprint planning sessions to allow members of the development team to effectively share what they have been working on, what they will work on, and any impediments that they are facing.

Addressing any impediments raised by the development team during the daily Sprint meetings and clearing these obstacles where possible.

|  |  |
| --- | --- |
| **1.6 Name 2 debugging methods, and when you would use them.** | **4 marks** |

You can debug using the pdb module in Python. This allows you to run programs in debugging mode. It is a simplistic command-line debugger and is therefore useful for smaller programs.

Another debugging method are GUI-based debuggers, such as pycharm, which can be used when using an Integrated Development Environment (IDE) like Python. Pycharm allows you to run a program in debugging mode. Because pycharm includes several tools which makes debugging faster and easier, it is especially useful for debugging larger, more complicated programs, and when using tools such as Flask.

|  |  |  |
| --- | --- | --- |
| **1.7 Looking at the following code, describe a case where this function**  **would throw an error when called.** Describe this case and talk about  what exception handling you’ll need.   |  | | --- | | **def can\_pay(price, cash\_given):**  **if cash\_given >= price:**  **return True**  **else:**  **return False** | | **5 marks** |

Comparison operators, such as >=, will only work if the variables ‘price’ and ‘cash\_given’ are either both integers or both strings. If the function was to be called using “can\_pay(13, ‘12’)” a TypeError would be thrown.

We can deal with this using exception handling. By adding a try-catch block, if the parameters are inputted in the incorrect format we can correct it prevent this from crashing the program.

def can\_pay(price, cash\_given):  
 try:  
 if cash\_given >= price:  
 return True  
 else:  
 return False  
 except TypeError:  
 if int(cash\_given) >= int(price):  
 return True  
 else:  
 return False

In the above amended program, the function would first try comparing the variable cash\_given and price. If the input values are in the incorrect format and a TypeError thrown, the except block would then compare the integer types of the variables. In this way, the exception handling means the variables can be inputted in either string or integer format and the function would still give a True or False output.

This try-catch block would not cover cases where the input was e.g., ‘ten’, therefore additional exception handling would be needed to deal with these cases.

|  |  |
| --- | --- |
| **1.8 What is git branching?** Explain how it is used in Git. | **6 marks** |

A branch in Git is a separate version of the main repository. By creating a new branch, you can make changes to the code without effecting the main, or ‘production’, branch. It also means that if, for example, testing highlights issues with the code, you can revert back to a previous version.

Diagram

Description automatically generated The main branch is highlighted in green and new divergent branches are shown in blue and orange.

Git branching allows us to, for example, fix a bug or add a new feature, in separate diverging branches without disrupting the code in the main branch. Once these changes in the diverging branches have been thoroughly tested, they can then be merged with the main branch. New branches can also be created from existing branches, making it easier for large development teams to implement changes without breaking the program.

By using git branches we can keep track of all the changes and ensure nothing is lost before a branch is removed. This is because a branch cannot be deleted if any pull requests exist on the branch or the branch has not been fully merged. In these instances, an error will be raised until all open pull requests have been resolved and the branch can be fully merged.

|  |  |
| --- | --- |
| **1.9 Design a restaurant ordering system.**  You do not need to write code, but describe a high-level approach:   * 1. Draw a list of key requirements   2. What are your main considerations and problems?   3. What components or tools would you potentially use? | **10 marks** |

Key requirements:

* We would need a welcome screen and a display to list all the items sold within our restaurant.
* We need a shopping basket so that items the customer selects can be stored somewhere.
* We would need to set up a database that effectively interacts with our program. This database should include a list of items sold and their identifiers, stock levels of each item to confirm whether a customer can order that item, and a log of placed orders so the kitchen can keep track of which items they need to prepare. A customer table would also be necessary for delivering the order and updating the customer, but also for analytics and marketing.
* We need to include payment options and the option to include loyalty cards, discounts, and promotional offers.
* Upon successful completion of an order a receipt should be printed, detailing the order, customer details and an order ID. This information should be stored in and / or update our database.

Main considerations and problems:

* One key consideration is the welcome display and how the available items will be displayed. We need to ensure that these are shown clearly and can easily be interacted with. This means customers should be able to select and modify based on e.g., dietary requirements and allergens.
* One problem is ensuring that items which are not in stock are not displayed. Also, if multiple customers are ordering at the same time, ensuring that items are removed as soon as stock has run out.
* Another issue is ensuring items are displayed clearly enough that things like allergen information is clearly accessible.
* Another consideration is the maintainability and functionality of the program. We need to ensure that changes to the item list can be easily made to take into consideration, for example, seasonal produce such as berries in summer, and special menus such as Christmas items, that do not feature permanently on the menu.
* Another consideration is storing information securely in the database. This is especially important when handling customer information, so that things like personal emails, home addresses etc., are not widely accessible.

Components and tools:

* SQL can be used to store and manage our database and the information we need for our restaurant ordering system.
* An appropriate cloud system, such as Microscoft Azure, could be used to store database information securely.
* A private GitHub repository would allow changes to the program to be made more easily, especially if multiple people are managing the system or someone leaves the project.
* Python can be used as the IDE, to help build the code and then run it once complete. Debuggers like pycharm can then be used to help debug the program.
* JIRA can be used to help facilitate an agile software development methodology.
* JavaScript, HTML, and CSS can be used to develop the GUI.

**Section 2: Concept Questions [19 marks]**

|  |  |
| --- | --- |
| **2.1 Write a function that takes in an input and checks to see if it’s an**  **isogram. The function should return True or False.**    An isogram is a word where no letter is repeated.  Examples include:   * "isogram" * "uncopyrightable" * “ambidextrously” | **7 marks** |

def check\_if\_isogram(word):  
 letters = [letter.lower() for letter in word]  
 for i in letters:  
 if letters.count(i) > 1:  
 return False  
 return True

|  |  |
| --- | --- |
| **2.2 Make a new test file and write comprehensive unit tests for the**  **function you wrote in 2.1**  For each test case add a comment stating why you chose that case. | **12 marks** |

from unittest import TestCase  
from main import check\_if\_isogram  
  
class TestifIsogram(TestCase):  
 def test\_true\_isogram(self):  
 result = check\_if\_isogram(word='isogram')  
 self.assertTrue(result)  
  
 def test\_false\_isogram(self):  
 result = check\_if\_isogram(word='lolly')  
 self.assertFalse(result)  
  
 def test\_edge\_case(self):  
 result = check\_if\_isogram(word='helLo')  
 self.assertFalse(result)

**Section 3: Python Challenge [25 marks]**

You are tasked with calculating the minimum classes we need to have so we know how many people to employ. Write a function which when given a number of students, calculates and prints out a string for your proposed number of classes, and a dictionary showing the allocation.

***Key Constraints:***

* The maximum size of a class is 30
* There needs to be a minimum of 2 classes
* The distribution of each class should be as even as possible.
* We want to hire as little people as possible - so where possible focus on bigger classes, and less of them!

***Inputs/Outputs***:

* If 31 was the input, the output would be:

|  |
| --- |
| Proposed Allocation: 2 classes  {'Class 1': 16, 'Class 2': 15} |

* If 59 was the input, the output would be:

|  |
| --- |
| Proposed Allocation: 2 classes  {'Class 1': 30, 'Class 2': 29} |

* If 87 was the input, the output would be:

|  |
| --- |
| Proposed Allocation: 3 classes  {'Class 1': 29, 'Class 2': 29, 'Class 3': 29} |

max\_class\_size = 30  
  
def allocate\_students\_to\_class(number\_of\_students):  
 class\_dict = {}  
  
 # if the number\_of\_students is less than max\_class\_size allocate students evenly between two classes  
 if number\_of\_students <= max\_class\_size:  
 halved\_students = number\_of\_students // 2  
 class\_dict['Class 1'] = halved\_students + (number\_of\_students % 2)  
 class\_dict['Class 2'] = halved\_students  
  
 # if the number\_of\_students is greater than max\_class\_size determine the minimum number of classes needed  
 else:  
 classes\_needed = (number\_of\_students + max\_class\_size - 1) // max\_class\_size  
 students\_per\_class = number\_of\_students // classes\_needed  
 remaining\_students = number\_of\_students % classes\_needed  
  
 # allocate the students evenly between the number of classes  
 for i in range(1, classes\_needed + 1):  
 class\_dict[f'Class {i}'] = students\_per\_class  
  
 for i in range(1, remaining\_students + 1):  
 class\_dict[f'Class {i}'] += 1  
  
 print(f'Proposed Allocation: {len(class\_dict)} classes')  
 print(class\_dict)

**Section 4: SQL Challenge [25 marks]**

In this section you will be fleshing out a database and performing queries.

**Starter Code:**

CREATE DATABASE foundation\_exam;

USE foundation\_exam;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4.1 Write (and execute) syntax to create the following tables:**  Example data is included to help you choose suitable data types  **A] *movie\_info*** *Table*     |  |  |  |  | | --- | --- | --- | --- | | Movie\_ID | Movie\_Name | Movie\_Length | Age\_Rating | | 1 | The Movie | 1:35:00 | 12A |   **B] *screens*** *Table*     |  |  | | --- | --- | | Screen\_ID | Four\_K | | 1 | False |   **C] *showings*** *Table*     |  |  |  |  |  | | --- | --- | --- | --- | --- | | Showing\_ID | Movie\_ID | Screen\_ID | Start\_Time | Available\_Seats | | 1 | 1 | 1 | 12:00:00 | 23 | | **10 marks** |

**CREATE DATABASE foundation\_exam;**

**USE foundation\_exam;**

**CREATE TABLE movie\_info (**

**Movie\_ID INTEGER PRIMARY KEY,**

**Movie\_Name VARCHAR(100),**

**Movie\_Length VARCHAR(100),**

**Age\_Rating VARCHAR(50)**

**);**

**CREATE TABLE screens (**

**Screen\_ID INTEGER PRIMARY KEY,**

**Four\_K VARCHAR(50)**

**);**

**CREATE TABLE showings (**

**Showing\_ID INTEGER PRIMARY KEY,**

**Movie\_ID INTEGER,**

**Screen\_ID INTEGER,**

**Start\_Time TIME,**

**Available\_Seats INTEGER,**

**FOREIGN KEY (Movie\_ID) REFERENCES movie\_info(Movie\_ID),**

**FOREIGN KEY (Screen\_ID) REFERENCES screens(Screen\_ID)**

**);**

|  |
| --- |
| **Populate the database!**  Use the file*foundation\_exam.sql* to fill your tables with the needed data.  You may need to change the names of the tables in the SQL file if yours don’t align. |

|  |  |
| --- | --- |
| **4.2 Write a query to return the name and time of all movies that play after**  **12:00 given there is at least 1 available seat. Display the results in time**  **order.** | **6 marks** |

**SELECT mov.Movie\_Name,sho.Start\_Time**

**FROM movie\_info AS mov**

**INNER JOIN showings AS sho ON mov.Movie\_ID = sho.Movie\_ID**

**WHERE**

**sho.Start\_Time BETWEEN '12:00:00' AND '23:59:59'**

**AND sho.Available\_Seats > 1**

**ORDER BY sho.Start\_Time ASC;**

|  |  |
| --- | --- |
| **4.3 Return the name of the movie with the most showings.** | **9 marks** |

**SELECT mov.Movie\_Name**

**FROM movie\_info AS mov**

**INNER JOIN showings AS sho ON mov.Movie\_ID = sho.Movie\_ID**

**GROUP BY sho.Movie\_ID**

**ORDER BY COUNT(sho.Movie\_ID) DESC**

**LIMIT 1;**

**Section 5: JavaScript Challenge [25 marks]**

Create a simple To-Do List web application using HTML, CSS, and JavaScript. The application should have the following features:

1. A text input field for entering tasks.
2. A "Add Task" button to add tasks to the list.
3. A list to display added tasks.
4. Each task should have a checkbox to mark it as complete.
5. A "Delete" button to remove completed tasks.
6. Style the application with CSS to make it visually appealing
   1. Make all the items centered on the page
   2. Display the list below the text input

Provide a clear and organized code for this web application.