



**FOUNDATION ASSESSMENT II - 2 HOURS**

|  |  |
| --- | --- |
| **SECTION** | **MARK** |
| **Theory Questions** | 31 |
| **Concept Questions** | 19 |
| **Python Challenge** | 25 |
| **SQL 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 a closed book exam.
* You are allowed to use PyCharm and MySQL Workbench for this assessment.

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

|  |  |
| --- | --- |
| * 1. **What does SDLC stand for?**   Software Development Life Cycle | **1 mark** |

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

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

|  |  |
| --- | --- |
| * 1. **What does NULL represent in a database?**   Null represents a non-existing value (missing or unknown). Not the same as empty string or 0. | **1 mark** |

|  |  |
| --- | --- |
| * 1. **Name 2 responsibilities of the Scrum Master** * Ensures that everyone in the team is focussed on the key deliverables for the sprint and that everything is on track * Acts as a middleman between the development team and the product manager and stakeholders, ensuring that both sides are informed about important decisions and updates and have most recent information | **2 marks** |

|  |  |
| --- | --- |
| * 1. **Name 2 debugging methods, and when you would use them.**  1. Using the debugging tools (included in most IDEs). This is a more reactive approach and is particularly helpful when we are not sure where the bug might be. It allows us to step through the program one line at a time and monitor variables (their value, type etc) at each stage, which is especially useful when we are working with while and for loops, if statements etc., where there is a lot of data manipulation, change of data types and checks for conditions. 2. Using unit testing, which is more a preventative measure, that can be implemented at the early stages of development. Because of this it helps to identify potential bugs early on, which in turn helps stopping issues from becoming more complex and difficult to find. Since it tests individual parts of code in isolation it is very useful when debugging complex systems where it can be difficult to find the source of a bug. | **4 marks** |

|  |  |  |  |
| --- | --- | --- | --- |
| **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** | |  |   The program will throw an error (TypeError) if the two arguments provided for the function are of different data types (e.g., one is string and another number, one is number and another a list etc). To handle this error, we would need to wrap the code in a try and except block, printing a statement that provides clear information about the issue and how the user can avoid it that would replace an inbuild error information provided by Python.  *def* can\_pay(*price*, *cash\_given*):  try:  if cash\_given >= price:  return True  else:  return False  except:  print("Something went wrong. Please make sure that price and cash\_given are numbers.”)  It is important to note, however, that handling this error does not prevent users from providing input that is invalid for the purpose of this function, since comparisons between the same data types are allowed and do not throw an error. This is to say, that the user can provide two words, lists etc., and the function will still work (will not throw an error) and return True or False, so additional checks/custom exceptions would be needed to ensure the user is providing numbers, but this seems to be outside the scope of this question. | **5 marks** |

|  |  |
| --- | --- |
| **1.8 What is git branching?** Explain how it is used in Git.  Git branching is a feature of git that allows diverging from main (or any other) “line” of development and continuing developing in a parallel branch while the main code remains unchanged. When creating a branch, we create a pointer to a particular commit in another branch, which makes it very lightweight comparing to creating a copy of the code. We can also merge the code back to another branch at any point and whenever we are ready, assuming there are no conflicts with the code in the branch we are merging into.  This has many useful applications:   * Facilitates collaboration, where many people can work on various parts of the code at the same time. * Allows for parallel development of various features and fixes without affecting live code. * Keeps the development and production code separate thus protecting the live code/application from not working as expected any vulnerabilities and bugs before the code is fully tested. * Separate branches can be created for features and then each sprint to keep better track and visibility of changes and development progress. * Such granularity can help with locating and fixing bugs quicker. | **6 marks** |

|  |  |
| --- | --- |
| **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?   Note: By *ordering system* here I understand a system used within a restaurant to handle orders rather than an online system for take-aways.   1. **Key requirements:**  * The system has to be able to keep track of items ordered per each table, their quantity and price. * Keep track of menu items availability. * It needs to allow for updates and modifications to accommodate special requirements of the client (allergies, preferences etc.). * The system needs to be fast and reliable to support very busy restaurant environment. * Should integrate with the system used by the kitchen staff. * Should integrate with payment system (POS) that allows to split bills and add tips. * Should be able to track which waiter served which table to monitor performance and tips.  1. **Key considerations:**  * It needs to be easy to use, supporting quick selection of food/drinks to add – categorisation/sizes etc., ideally items represented with images. * Reliability of service and tolerance for downtime - online or local application solution (desktop or mobile/webapp)? Mixture of the two? * How best to integrate with other systems already used by the restaurant, e.g., payment system? * How to track availability of menu items real-time to avoid customers having to change their orders? * Type of device/devices used on – touch screen? Normal computer? mobiles? * Security of customer details and payment? Any other industry standards? * Does the system need to support take aways with deliveries as well, in which case customer address needs to be stored as well? * Depending on existing systems and solutions preferred by the restaurant, decisions need to be made on how to handle waiter logging/tracking to prevent lengthy login/logouts – mobile devices? System logging using fobs/fingerprints or other solutions?  1. **What components/tools use:**  * HTML/CSS/Bootstrap/JavaScript or React or Angular for frontend. * Python/Django/Flask for backend to process the data and link frontend with the database. * Database to store the information on menu items, customers, waiters, orders etc. * APIs for integration with the POS and other systems used by the restaurants.   The above is just a general draft and is not exhaustive. A simplified representation of how the system would work is represented below. | **10 marks** |

**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”   import collections  """Write a function that takes in an input and checks to see if it’s an isogram. The function should return True or False."""  *def* check\_if\_isogram(*word*):  # find out counts for all letters in the word  word\_counts = collections.Counter(word)  # if the max count is greater than 1, then it's not an isogram  if max(word\_counts.values()) > 1:  return False  return True  print(check\_if\_isogram("isogram"))  print(check\_if\_isogram("uncopyrightable"))  print(check\_if\_isogram("ambidextrously"))  print(check\_if\_isogram("where")) | **7 marks** |

|  |  |
| --- | --- |
| **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.  import unittest  from foundation\_assessment2 import check\_if\_isogram  *class* TestCheckIfIsogram(*unittest*.*TestCase*):  *def* test\_input(*self*):  word = "isogram"  self.assertEqual(check\_if\_isogram(word), True)  *def* test\_input2(*self*):  word = "uncopyrightable"  self.assertEqual(check\_if\_isogram(word), True)    *def* test\_input3(*self*):  word = "ambidextrously"  self.assertEqual(check\_if\_isogram(word), True)    *def* test\_input4(*self*):  word = "where"  self.assertEqual(check\_if\_isogram(word), False)  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() | **12 marks** |

**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} |

*def* get\_classes(*students\_no*):

# create a dictionary to store the allocation

allocation = {}

# base case when we have fewer than 30 students

if students\_no <= 30:

no\_classes = 2

allocation = {"Class 1": students\_no - students\_no // 2, "Class 2": students\_no // 2}

# if students number is higher, divide by 30 usign floor division to check how many classes we need and then find the remainder of students that are left over

else:

no\_classes = students\_no // 30

if students\_no % 30 > 0:

no\_classes += 1

students\_per\_class = students\_no // no\_classes

remainder = students\_no % no\_classes

# loop through the number of classes and allocate students to each

for i in range(no\_classes):

# if the remainder is greater than 0, add 1 to the number of students in each class

if remainder > 0:

allocation[*f*"Class "] = students\_per\_class+1

# else:

print(*f*"Proposed Allocation: {no\_classes} classess \n{allocation}")

get\_classes(15)

get\_classes(31)

get\_classes(59)

get\_classes(87)

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

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

**Starter Code:**

CREATE DATABASE foundation\_assessment\_ii;

USE foundation\_assessment\_ii;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |   CREATE DATABASE foundation\_assessment\_ii;  USE foundation\_assessment\_ii;  CREATE TABLE movie\_info(  Movie\_ID INT PRIMARY KEY,  Movie\_Name VARCHAR(50),  Movie\_Length TIME,  Age\_Rating VARCHAR(5));  CREATE TABLE screens(  Screen\_ID INT PRIMARY KEY,  Four\_K BOOLEAN);  CREATE TABLE showings(  Showing\_ID INT,  Movie\_ID INT,  Screen\_ID INT,  Start\_Time TIME,  Available\_Seats INT,  FOREIGN KEY (Movie\_ID) REFERENCES movie\_info(Movie\_ID),  FOREIGN KEY (Screen\_ID) REFERENCES screens(Screen\_ID)); | **10 marks** |

|  |
| --- |
| **Populate the database!**  Use the file*foundation\_assessment\_ii.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.  Done, had to change capitalisation, sample data below |

|  |  |
| --- | --- |
| **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.**  SELECT mov.Movie\_Name, sh.Start\_Time  FROM movie\_info mov  JOIN showings sh  ON mov.movie\_id = sh.movie\_id  WHERE sh.Start\_Time > '12:00:00' AND sh.Available\_Seats >= 1  ORDER BY sh.Start\_Time; | **6 marks** |

|  |  |
| --- | --- |
| **4.3 Return the name of the movie with the most showings.**  SELECT mov.Movie\_Name, count(\*)  FROM movie\_info mov  JOIN showings sh  ON mov.movie\_id = sh.movie\_id  GROUP BY mov.Movie\_Name  ORDER BY count(\*) DESC  LIMIT 1 | **9 marks** |