



**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]**

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

Software Development Lifecycle

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

ZeroDivisionError

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

git push

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

NULL represents a missing or unknown value that has not been assigned or determined.

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

1. Coaching the team on Scrum principles and practices

2. Making sure that Scrum events, such as Sprint Planning, are positive and productive and that the team achieves desired outcomes.

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

1. Assertion: can be used in testing such as unittest to ensure that the code is behaving as expected.

2. Exception handling: can be used in try-except block to handle errors that occur during the execution of the code.

| **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** |
| --- | --- | --- |

If either ‘price’ or ‘cash given’ is not numeric (e.g. a string) , then a TypeError will be thrown when operator ‘>=’ is used.

We could add a try-except block to catch the error and handle it appropriately.

def can\_pay(price, cash\_given):

try:

if cash\_given >= price:

return True

else:

return False

except TypeError:

print("Error: one of the arguments is not a number")

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

In Git, branching is the process of creating a separate line of development for a project. It allows developers to work on different features or versions of a project simultaneously without affecting the main codebase. When a branch is created, it is essentially a copy of the main branch (usually called "master" or "main") that can be modified independently.

Branching is an essential feature of Git that allows for efficient collaboration, experimentation, and organization of code. Developers can create and switch between branches to work on different features or versions of a project, merge changes from one branch to another, and collaborate on a single codebase without interfering with each other's work.

Git provides several commands for branching, including ‘git branch’, ‘git checkout’, and ‘git merge’. ‘git branch’ is used to create a new branch, ‘git checkout’ is used to switch between branches, and ‘git merge’ is used to merge changes from one branch to another.

By using branching, developers can work on different features or versions of a project in parallel, allowing for faster development and more efficient collaboration. Branching also makes it easier to manage code and track changes over time, as each branch represents a distinct stage of development.

| **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:

1. Displaying menu items with with prices and descriptions
2. Ability to select items and add special requests or modifications to menu items
3. Ability to cancel order before payment
4. Integration with payment processing system
5. Inventory management system to track the availability of ingredients or dishes

Main considerations and problems:

1. Friendly user interface
2. Ensuring the security of payment and other private information; compliance of GDPR
3. Scalability to handle high volumes of orders during peak times

Components and tools:

1. Front-end web development tools such as HTML, CSS and JavaScript
2. Database management tool such as MySQL
3. Using Python for building framework at back-end

**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 is\_isogram(input):

if not input.isalpha():

return False

else:

count\_dic = {}

for letter in input:

if letter not in count\_dic.keys():

count\_dic[letter] = 1

else:

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** |
| --- | --- |

import unittest

from foundation\_assessment2 import is\_isogram

class MyTestCase(unittest.TestCase):

def test\_not\_word(self):

predicted = False

result = is\_isogram(input = "jiw\*\*")

self.assertEqual(predicted, result)

def test\_valid\_isogram(self):

predicted = True

result = is\_isogram(input = "isogram")

self.assertEqual(predicted, result)

def test\_not\_isogram(self):

predicted = False

result = is\_isogram(input = "racecar")

self.assertEqual(predicted, result)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**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 calc\_minimum\_classes(student):

if student < 0 or isinstance(student, int) == False:

raise ValueError("Do not accept negative numbers or decimals")

try:

if student <= 60:

class\_count = 2

print(f"Proposed Allocation: {class\_count} classes")

class\_dic = {}

class\_dic['Class 1'] = round(student/2)

class\_dic['Class 2'] = student - class\_dic['Class 1']

print(class\_dic)

else:

if student % 30 == 0:

class\_count = student // 30

else:

class\_count = student // 30 + 1

print(f"Proposed Allocation: {class\_count} classes")

class\_dic = {}

for i in range(class\_count):

if i < class\_count - 1:

class\_dic[f"Class {i+1}"] = round(student / class\_count)

else:

class\_dic[f"Class {i+1}"] = student - round(student / class\_count) \* i

print(class\_dic)

except TypeError:

print("Number of student invalid")

**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 | | **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. |
| --- |

CREATE TABLE movie\_info (

Movie\_ID INT PRIMARY KEY,

Movie\_Name VARCHAR(50),

Movie\_Length TIME,

Age\_Rating VARCHAR(20)

);

CREATE TABLE screens (

Screen\_ID INT PRIMARY KEY,

Four\_K BIT

);

CREATE TABLE showings (

Showing\_ID INT PRIMARY KEY,

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)

);

| **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 m.Movie\_Name, sh.start\_time FROM showings sh INNER JOIN movie\_info m ON sh.Movie\_ID = m.Movie\_ID

WHERE sh.start\_time > '12:00:00' AND sh.available\_seats >= 1

ORDER BY sh.start\_time;

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

SELECT c.movie\_name FROM (

SELECT COUNT(sh.movie\_id) AS count\_time, m.movie\_name FROM showings sh INNER JOIN movie\_info m

ON sh.Movie\_ID = m.Movie\_ID GROUP BY sh.movie\_id ORDER BY count\_time DESC) AS c

LIMIT 1;