Foundation Assessment 2

Section 1 - Theory Questions

1.1 SDLC stands for Software Development Lifecycle.

1.2 When you try to divide by zero, a ZeroDivisionError is raised.

1.3 To move code from the local repository to the remote repository, use the command ‘git push’

1.4 In a database, NULL represents a value that doesn’t exist or is missing.

1.5 Two responsibilities of a Scrum Master are ensuring that the scrum framework is being followed, and prioritising tasks to be completed from the product backlog

1.6 One method of debugging is by using brute-force - this is where the developer manually searches through their code for the bugs, often used as a last resort when time is short and/or no other method is working.

Another debugging method is cause-elimination - it’s where symptoms of the code failure are noted, and the developer hypothesises causes of the bug. This method is used because it leads to some of the shortest debugging times (when the developer understands the software well).

1.7

```
def can_pay(price, cash_given):
    if cash_given >= price:
        return True
    else:
        return False
```

Case: inputting a string instead of an integer value for one or both of the can_pay() arguments. Doing so raises a TypeError because the ‘>=’ arithmetic operator is incompatible with string data types. To handle this, you could either re-write the code to handle this:

```
def can_pay(price, cash_given):
    if price != int(price) or cash_given != int(cash_given):
        price = int(price)
        int(cash_given)
    if cash_given >= price:
        return True
    elif cash_given < price:
        return False
```

```
if can_pay('4',5) == True:
    print("YES")
elif can_pay('4',5) == False:
    print("NO")
```

Or you could define a TypeError message using a Try Except block:

```
def can_pay(price, cash_given):
    try:
        if cash_given >= price:
            return True
        elif cash_given < price:
            return False
    except TypeError:
        print("One of your inputs is a string, not an
integer!")
    except Exception:
        print("Something went wrong”)
```

```
if can_pay('4',5) == True:
    print("YES")
elif can_pay('4',5) == False:
    print(“NO”)
```

1.8 A Git branch is a version of the repository that diverges from the main working project. Here are some examples of branch commands: git branch <branchname> (creates branch), git checkout <branchname> (switches to another branch). The main benefit of using branches is creating a separate space to try out new ideas without risking the main codebase. The main disadvantages stems from having an extreme number of branches; it leads to increased maintenance, storage requirements, and complexity. You have to submit a pull request to add your code to a remote branch (it acts as a review stage before adding your code to the main codebase).

1.9 Design a restaurant ordering system. You do not need to write code, but describe a high-level approach: [10]

a. Draw a list of key requirements
- clearly displaying menu options in a dictionary format (key-value pairs for dishes and prices)
- price transparency - no hidden costs worked into final bill (no mandatory tip or tax)
- have the user pick which option they want using ‘input’
- all code should be in try-except blocks to catch errors

- there should be options to add drinks, sides, and condiments to the order

b. What are your main considerations and problems?
- limiting chance for the user to input an answer that isn't a menu option (on this same note, a TypeError)
- maybe having a 'what's your budget' input so users can keep adding to their order - use a while loop to check if meal price is < budget
- will users find this system easy to use?
- could users place an order without having to have an account?

c. What components or tools would you potentially use?
- my ordering system connects to an SQL database to store unique usernames, passwords, and delivery addresses.
- SQL is also the place that the system pulls menu options from once the user picks a restaurant
- I'd use HTML and CSS to create and style the website
- the app will be created using Objective-C
- The python Requests library will be used to pull correct pricing from receptive restaurants

SQL CHALLENGE

Question 1:

```sql
USE foundation_assessment_ii;

CREATE TABLE movie_info (
Movie_Id INTEGER PRIMARY KEY,
Movie_Name VARCHAR(50),
Movie_Length DATETIME,
Age_Rating VARCHAR(50));

CREATE TABLE screens (
Screen_Id INTEGER PRIMARY KEY,
Four_K BOOLEAN);

CREATE TABLE showings (
Showing_Id INTEGER PRIMARY KEY,
Movie_Id INTEGER NOT NULL,
Screen_Id INTEGER NOT NULL,
Start_Time DATETIME,
Available_Seats INTEGER,
CONSTRAINT fk_movie_id
FOREIGN KEY (Movie_Id)
REFERENCES movie_info (Movie_Id)
);

ALTER TABLE showings
ADD CONSTRAINT fk_screen_id
FOREIGN KEY (Screen_Id)
REFERENCES screens (Screen_ID);
```

Question 3:

```sql
USE foundation_assessment_ii;

SELECT Movie_Id, COUNT(Movie_Id) AS MOST_FREQUENT
FROM showings
GROUP BY Movie_Id
ORDER BY COUNT(Movie_Id) DESC;

SELECT Movie_Name
FROM movie_info
WHERE Movie_Id = 3;
```