Section 1: Theory Questions [31 marks]

1.1 What does SDLC stand for? 1 mark

SDLC stands for Software Development LifeCycle.

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

When you divide a number by 0 in python, a ZeroDivisionError will be thrown.

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

To move code from the local repository to the remote repository, the git push command is used.

1.4 What does NULL represent in a database? 1 mark

In a Database (DB) a NULL value represents an absence of a value. So, in a field where there is no value entered, a NULL value is essentially used as a placeholder for fields where there is no data entered. It does ***not*** represent zero. NULL values are not permitted in columns specified as NOT NULL, where a value must be entered in order to insert the data into the DB.

1.5 Name 2 responsibilities of the Scrum Master. 2 marks

The Scrum Master supports the workings and progress of the scrum (and sprints) and those working in it. There are many responsibilities that a Scrum Master has.

One responsibility of a Scrum Master is to organise the meetings for the Scrum. This includes things such as Daily Stand Ups, where each member talks about what they did the day before, what they will do that day, and if they have any challenges that they need the Scrum Master to help tackle. Other examples of this are the Sprint Review and Sprint Retrospective meetings that are held after the sprints have been finished. These meetings allow the Scrum team to reflect on the performance of the sprint, what could be improved for the future, and any key takeaways.

A second responsibility of the Scrum Master is to help the Scrum work more efficiently. This can be done through helping Scrum members minimise the amount of meetings they attend during the sprint, by acting as an intermediary between the Scrum and other members in the organisation, for example. The Scrum Master also helps to remove challenges by collaborating with the Product Owner and stakeholders, to ensure that the project has a realistic deadline and milestones for the Scrum team.

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

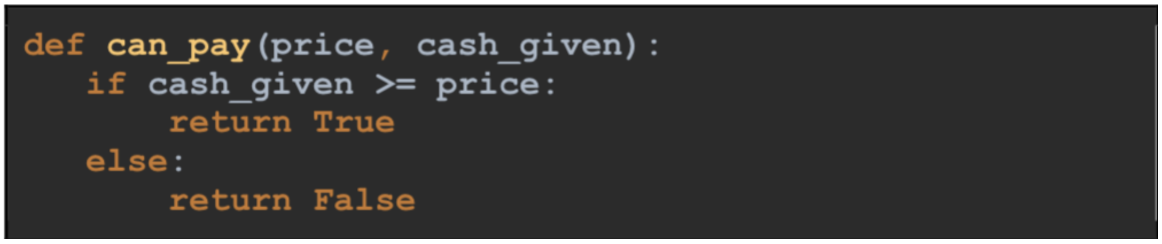
Debugging is used to solve / prevent errors in code. We can usually classify any errors into two categories : syntaxical and logical errors.

One method of debugging is to use the print() statement to display the issue in the console. You would usually use this type of debugging for shorter pieces of code that do not have high numbers of different variables. This is because for print() as a debugging method, you have to decide manually where to insert the print() statement in your code to try to understand which line(s) the bugs occur in, or which variable. So, using this for a long piece of code with multiple values would take quite a bit of time. For short, simple codes, however, using a print statement is a very effective way to quickly identify where the bug is for you to correct it.

Another method of debugging is to use the breakpoint() and pdb module in Python. This enables you to go through your code very thoroughly (one line of code at a time) to identify where there is an error. You can isolate the bug by importing the pdb (python debugger module) and using the breakpoint() function to create breaks (sometimes called pauses) in blocks of code that you are concerned about, or think have a bug. This type of debugging is particularly useful when you have an extremely long program that is hundreds of lines of code long and takes multiple variables, with a complex logic. Breakpoint() is more applicable than using print() to debug for these types of programmes and code because it allows you to check multiple variables faster, calling the blocks of code to query automatically (and therefore faster) than if you had manually used the print() statement at intervals throughout your program to console display up until which point in your code the program worked. It essentially helps you to isolate the bug with less steps than the print() debugging method, which requires you to manually test the different steps in your code.

Debugging enables you to identify and resolve a bug in your code. To help handle bugs in code, you can use Error handling and exceptions (such as raising Assertion Errors, Validation Errors and using try & except). These are particularly relevant for logical errors. We would use Assertion Errors to check the sanity of an input, and try and except to gracefully inform the user if part of the code cannot run and instruct them on how to resolve it (such as changing an object data type, or informing the user if a database could not be connected to, for example).

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. 5 marks

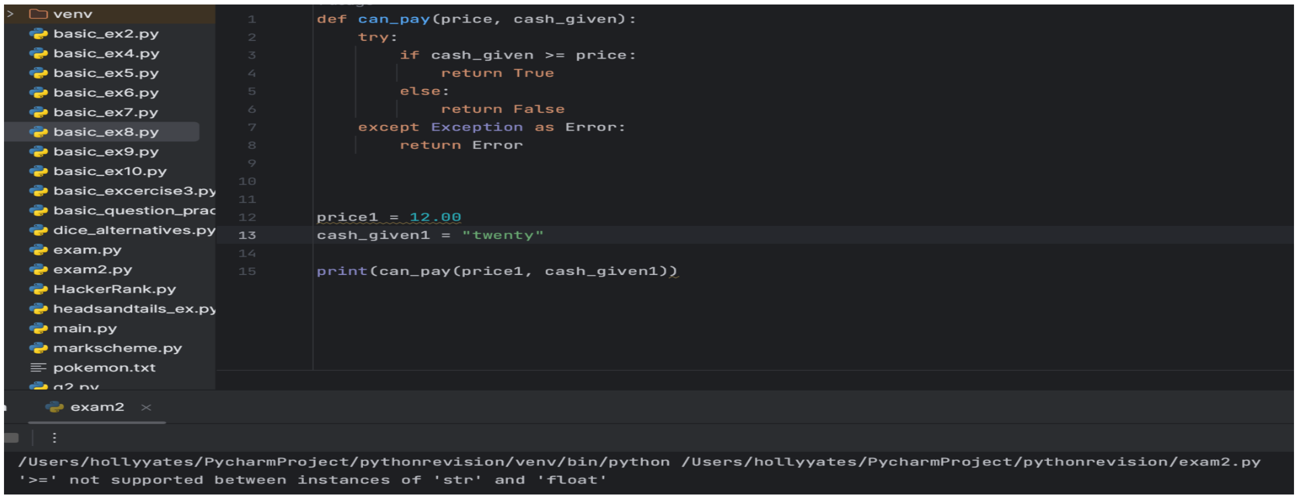


```python
def can_pay(price, cash_given):
    if cash_given >= price:
        return True
    else:
        return False
```

A case where this function would throw an error, would be if one of the inputted variables were not an int or float, say for example it was a string. This would be because the comparison operator, more than or equal to (">="), cannot compare an integer or float with a string. One method of exception handling to solve this could be to use a try and except with a TypeError. I would put the "try:" before the main body of code in the function, and then the "except" after it. Within the "except" I would specify a "TypeError:" exception as this is raised for unsupported/incorrect object types. This would, therefore, handle the incompatible data types for this error case, because it would be raised if one of the variables entered into the function were a string. In the TypeError exception, I would also return a message that would explain why the function couldn't run and instruct the necessary steps to resolve this. Please see this below:

```python
def can_pay(price, cash_given):
   try:
       if cash_given >= price:
           return True
       else:
           return False
   except TypeError:
       return ("Invalid data type for comparison, please convert any strings to
integers or floats.")
```

It is worth noting that the except TypeError above would only catch the error spoken about in this case (if a variable were a string). If I were unsure of which type of error may be thrown, I would use a general Exception rather than a specific exception like TypeError. In order to get this Exception to explain the issue, I would use 'except Exception as Error': and then indented below 'return(Error). This would mean that if an error were thrown, the Exception would return the error in the console. Please see this (and the console message it would generate) here:



```python
def can_pay(price, cash_given):
    try:
        if cash_given >= price:
            return True
        else:
            return False
    except Exception as Error:
        return Error


price1 = 12.00
cash_given1 = "twenty"

print(can_pay(price1, cash_given1))
```

```
/Users/hollyyates/PycharmProject/pythonrevision/venv/bin/python /Users/hollyyates/PycharmProject/pythonrevision/exam2.py
'>=' not supported between instances of 'str' and 'float'
```

<u>1.8 What is git branching? Explain how it is used in Git. 6 marks</u>

Git branching is a form of version control that enables users to edit, develop or change a file of a repository/ project in isolation, whilst simultaneously maintaining an original 'untouched' version. It is essentially creating copies of a file in new branches to be worked on without changing the integrity of the original file in the base branch.

Once a new branch has been created, the user or other users can alter and make changes to the project in this new branch, whilst maintaining the in-tact, unchanged original version of the project in the base branch (this is also referred to as the "main" or "master" branch). Once a user has finished their changes in the new branch, a git pull request can be used to inform other users / collaborators on the project that the code is ready for them to review or check. At this point, further changes can be committed to the branch if need be. Once finalised, the new branch is then merged with the original branch and the changes between the two branches are resolved (updated).

There can be multiple branches created in a repository, and these can be pulled to a local server, which enables multiple users in multiple different locations to work on the branch on their own device. By using a Git push, the changes made to the branch in the local repository are then committed back to the remote repository (branch) for everyone to see. So, another way branching is used in Git, is to enable people all over the world to work on a project together, and peer review each other's work remotely.

Branching is also used in git to track changes and updates made to the code, as each commit is recorded, enabling users to follow the developments or debugging work in the branch. Therefore, branching on git is also used to enable users to demonstrate their workflow to their collaborators to explain the changes they have made to their code before merging back with the main branch.

Branching can also facilitate teamwork, as an entire program written in the main branch can be split into new branches of smaller segments of the program to have different people working on different parts of the code at the same time.

How we use git to create and use branching:
Whilst a new branch can be created directory from Github, a new branch is created in the terminal using git with the following syntax:

- git branch -c <new_branch_name_goes_here>

To switch to the new branch in git from the terminal:

- git checkout <new_branch_name_goes_here>

To add the changes made in the new branch::

- git add .

To commit the changes just added to the new branch - and add an explanation message to the commit:

- git commit -m <"explanationl_message_goes_here">

To push the committed the branch, and the new changes from the local to the remote repository on github:

- git push origin <branch_name>

From this point onwards, the new branch and its changes will be compared and pulled in Github to then go onto merge them to the main branch when desired.

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

a. **Draw a list of key requirements**
b. **What are your main considerations and problems?**
c. **What components or tools would you potentially use?**

Key requirements:

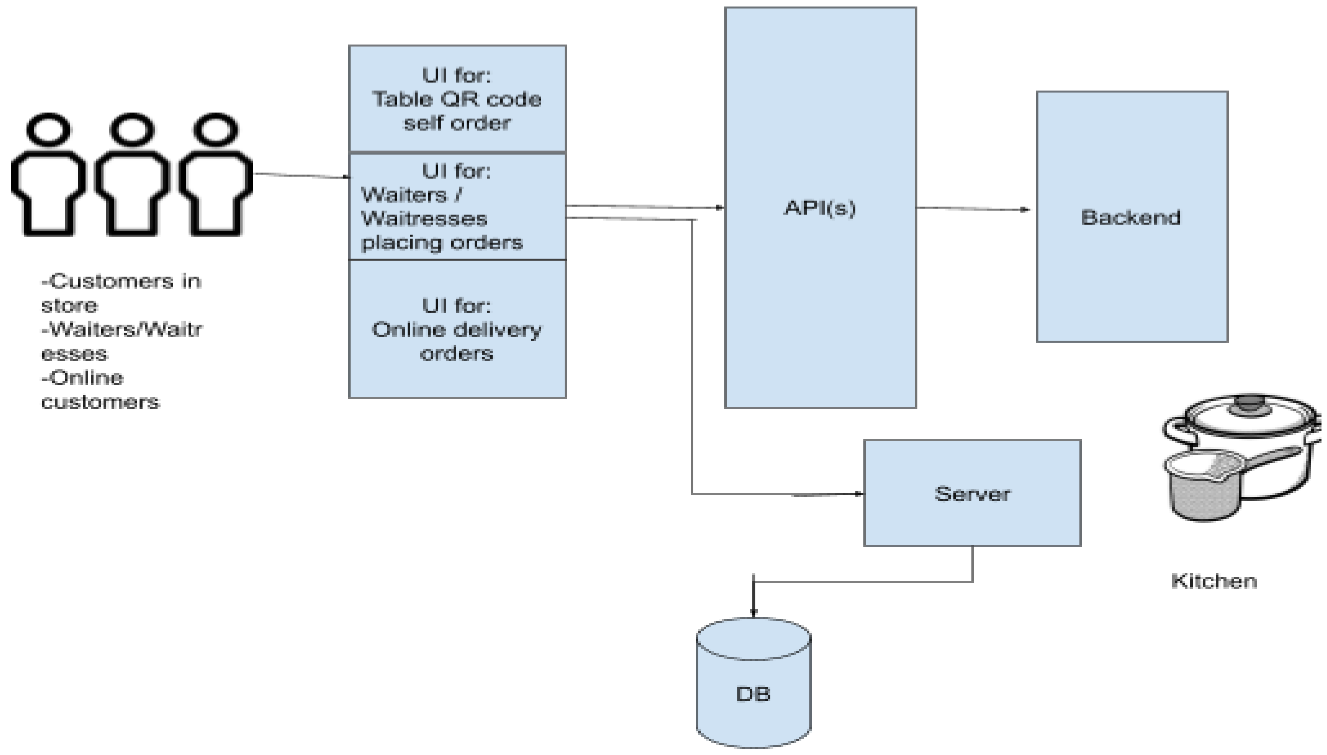
- A system where orders in a restaurant can be placed.
- Orders can be placed by wait staff at the tables.
- Customers can scan a QR code at the table and place their own orders to the system.
- Ordering system must show all restaurant menu options and prices.
- Orders are immediately sent to the kitchen upon being placed.
- Able to make comments and notes (such as allergies) when placing orders.
- System linking to the restaurant's website for delivery orders to be placed.
- Online payment Methods for online delivery orders.
- Interaction with other services /sites (such as justeat, deliveroo etc).
- Orders placed can be linked to the table that made the order to serve the order when ready.
- Online orders placed can be linked to customers / addresses of the order.
- Verify online accounts of customers to place orders.

- Match delivery drivers that arrive at the restaurant to orders placed.

Considerations and problems:
- System traffic - at peak times how many orders can be placed at once before the system lags or crashes.
- Internet connectivity of the system - if there is a connection failure, will orders recently placed be dropped, will the kitchen still get their orders or will re-ordering be necessary.
- Would there be a delay or any issue if it was a large order - say a table of 12 people ordering three courses at once.
- Resource availability / changing menu - can the system be immediately updated if the restaurant kitchen runs out of an ingredient, to prevent orders being placed for dishes that are no longer available.
- Able to make comments and special requests with order for allergens.
- For multiple staff in one restaurant, can all wait staff have access to orders taken by other waitstaff - in case a table of customers asks a different server for the bill etc.
- Age verification - if someone tries to order alcohol from the QR code, their age would need to be verified as 18+ before the order would be placed.

Components/ tools potentially needed:
- User Interface for wait staff to place an order.
- User Interface for in-store customers to place orders at table via QR code.
- User Interface for online customers placing delivery orders.
- Database to store menu, prices, offers, seasonal dishes, ingredients, allergens etc.
- Database Management System.
- API(s) to connect User Interfaces to database & backend.
- Python/Javascript for backend logic of system.
- Server.
- Online payment service (such as paypal etc) / website builder with in-built transactional processing capacity such as Shopify, for example.
- CRM to store customer data/information.
- Authentication software to check online customer's log-ins and accounts match.
- Potentially use analysis software like a Customer Data Platform to analyse order system data (such as which dishes are ordered the most, when are the most orders placed, what is the average order cost, etc). This would be useful for the restaurant, but would not change the ordering system.

Basic Design of System:



-Customers in store
-Waiters/Waitresses
-Online customers
UI for: Table QR code self order
UI for: Waiters / Waitresses placing orders
UI for: Online delivery orders
API(s)
Backend
Server
DB
Kitchen