Assessment Task 7

Café Au Latte

LAPUZ Frinze

8/13/18

Computer Science ATAR Year 12

Features Required & How it is to be implemented: (Initial Planning)

- Add, remove, edit, existing employees, enter other additional payroll related data.

Employee Table is to be created for easy, efficient and reliable data storage and processing. This is to be done using Structured Query Language (SQL)

- Retrieve Data -> *SELECT*
- Edit Data -> *UPDATE*
- Remove Data -> *DELETE*

Although the use of export and import is also available with SQL if it is ever needed in the future. Along with that SQL is compatible with most software dealing with databases and replaces the requirement of constructing algorithm in different language with loops and decision structure [LOOP + IF = SELECT + WHERE] possibly decreasing time to produce solutions.

SQL also makes filtering and searches easier.

Eg. Menu – show all managers in alphabetical order of their first name

SQL:

*SELECT * Employee WHERE Role = 'Manager' ORDER BY Fname ASC*

Employee Table Fields & Examples:

- EmployeeNo -> PRIMARY KEY
  - 2447532
- First Name / Fname
  - Frinze
- Last name / Lname
  - Lapuz
- Role
  - Manager
- Sup_An
  - 0.04
- Helt_Ins
  - 15

- Allow secure means of "clocking-in" and "clocking out". Default Values maybe offered to speed data entry.

As my experience working in McDonalds, a smart clock is used to "clock-in" and "clock-out" which can be done by the use of just inputting Employee-Number(7-digit pin).This is deemed as 'secure enough' as it is easy to implement and other employees will not be able to 'clock-in' and 'clock-out' for other employees since a 7-digit pin is long enough that will most of the time difficult to remember in the perspective of other employees, and this 7-digit pin is not shared.

If necessary a pin and password combination could be used for extra security, but in this project only PIN will be used for 'clocking-in' and 'clocking-out'.

For default values that can speed up data entry, sometimes a person will receive the same shift (length, days) for every week from the previous weeks. A **Schedule Table** could be used that will store the schedule of an employee for a week with and also the starting day of the week (if implementing storing all schedule that ever existed from the beginning).

Although in this project, schedule table will only store the schedule for the previous week.

Schedule Table Fields & Examples:

- ScheduleID -> PRIMARY KEY
  - 1
- EmployeeID/7-digitPin (FK)
  - 2447532
- Monday Hours
  - 5
- Tuesday Hours
  - 2

... From Monday to Sunday Hours

For Example:

	Role	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Harry Grapes	Barista	9	9	9	0	9	6	10

Harry could be asked, *"Is the hours to be inputted EXACTLY similar to last week's?"* if he did the same shifts as last week. Which speeds up the data entry.

In short: the default value set is the value previously used for a specific day

- Calculate Wages of the previous Week

There are two functions to be created to achieve specific result. Here are the name and parameter/data required for it to run.

1. ***GrossPay(Int/String Role/RoleID, Int Array Weekhours)***
   a. Int/String Role/RoleID
      i. Different Role have different pay rate
   b. Int Array Weekhours
      i. Contains the hours of each day for calculation
   c. Return Gross Pay
      i. Floating Point Calculation

Function GrossPay will calculate for the raw wage(no tax or deductions)

2. ***TotalPay(Double GrossPay, String Role, Double Sup_ann, Double Helt_Ins)***
    a. Double GrossPay
        i. Takes the raw wage
    b. String Role
        i. For the task, it is assumed that managers and barrista have different TaxRate.
        ii. Role is used to determine taxrate
    c. Double Sup_ann
        i. Superannuation deductions to be calculated
    d. Double Helt_Ins
        i. Health Insurance deductions to be calculated
    e. Return Total_Pay
        i. Floating Point Calculation

Function **TotalPay** will calculate the income which takes into consideration taxation and deductions.

Since both managers and baristas differ in terms of their tax brackets and income. A table of Roles and Wages could be Created. In this project, it is **assumed** that there is a tax bracket specified for the salary.

Roles Fields & Examples:

- RoleID -> PRIMARY KEY
    - 0
- RoleName
    - Barrista
- TaxRate
    - 0.3
- PayRate
    - $23.00

Changing the original **Employee Table**, **Role attribute to RoleID**. Hence everything related to Roles in the previous sections could be changed to ***Int RoleID***. This adds flexibility as the salary and tax bracket could be adjusted for the role.

Employee Table Fields & Examples: (NEW)

- 7-digit pin / PIN -> PRIMARY KEY
    - 2447532
- First Name / Fname
    - Frinze
- Last name / Lname
    - Lapuz

- RoleID (FK)
  - 1
- Sup_An
  - 0.04
- Helt_Ins
  - 15

- Data should be stored in and retrieved from external text files

This is where databases could be used in various formats such as .xml, .accdb, .db, .sqlite, csv, which could be considered as a text file since every database is divided into two parts, data and field formats which could be opened in most text editors.

During Development:

- **Paying Employees**

This feature development went straightforward and is the same as to how it was planned.

Paying employees required data given.

- ID (foreign key for other tables)
- RoleID/Role
  - Pay_rate
  - Tax_rate
- Superannuation Percentage
- Health Insurance
- Hours
  - Normal Hours
  - Overtime Hours

To pay employee, calculations such as their gross, superannuation deduction, health insurance deduction, tax, and net pay.

The formula for the following required calculations are:

- $Gross = Gross + Pay\_rate * Hours * (Increase\ Rate\ If\ Overtime)$ FOR WEEKDAYS
- $Gross = Gross + (Pay\_rate + Additions) * Hours * (Increase\ Rate\ if\ Overtime)$ FOR WEEKENDS

This is looped for 7 times. Weekday length

This is expressed here:

```
def GrossPay(role,weekhours,weekhours_overtime): #CALCULATES GROSS PAY
   total_norm_hour = 0
   total_ot_hour = 0
   gross = 0
   pay_rate = (23.0,30.0)[role==1] #role: 2 - Barista, 1 - Manager
   pay_day = 0

   if len(weekhours) != 7:
       return "Number of Weekday is 7 -> 7 Input is required"

   for index in range(0,7): # count = 0 to 6
       norm_hour = weekhours[index]
       ot_hour = weekhours_overtime[index]

       total_norm_hour += norm_hour
       total_ot_hour += ot_hour


       if index<5: #WEEKDAYS
           if ot_hour>3:
               ot_hour = (ot_hour-3)   #pay = (pay_rate * hours) * % increase  Hence: pay = (hours *
%increase) * pay_rate
               pay_day +=  pay_rate * ot_hour * 1.45                #calculation of hours increase
relative to normal hours
               ot_hour = 3

           pay_day = pay_rate * (norm_hour + ot_hour*1.25)

       else:       #WEEKENDS
           pay_day = (pay_rate+(3,4)[index==6])*(ot_hour*1.5+norm_hour)  #index = 5 or 6; $3 for SAT, $ $4
for SUN
       #print(str(index) + ": " + str(pay_day))
       gross = gross + pay_day



   return gross,total_norm_hour,total_ot_hour #MULTIPLE RETURN VALUE
```

- $Superannuation\ Deduction\ =\ Gross\ *\ Superannuation\_Percentage$
- $Health\ Insurance\ Deduction\ =\ Health\ Insurance$
- $Wage\ W/\ Ded. =\ Gross\ -\ Superannuation\_Deduction\ -\ Health\ Insurance\ Deduction$
- $Tax\ =\ Wage\ w/\ Ded\ *\ Tax\_Rate$
- $Net\ Pay\ =\ Wage\ w/\ Ded\ -\ Tax$

```
def Calculate_Tax(GrossPay, tax_rate, Sup_an, Helt_Ins): #CALCULATES DEDUCTIONS AND TAX
   Sup_an_val = GrossPay *(Sup_an / 100.0) #CALCULATES VALUE OF SUPERANNUATION DEDUCTION
   Wage_w_deduction = GrossPay - Sup_an_val - Helt_Ins #TAKES THE WAGE WITH DEDUCTION
   Tax = tax_rate*(Wage_w_deduction) #TAXATION CALCULATION

   Net_Pay = Wage_w_deduction - Tax #NET PAY

   return Sup_an_val,Helt_Ins,Tax,Net_Pay #RETURNS MULTIPLE VALUES
```

- **Clocking-on / Clocking-off**

This portion of the programming took some time to develop particular about the **change in feature**. At first, I developed a system where employee has to input hours, yet there's a change in feature to be implemented as entering in time in and time out.

To access this feature, an authentication has to go through with the use of inputting the Employee ID. This is how most company use smart clocks.

- **Clock-In**

Clock-in is simple, as it asks for the Employee to input the time in **with error checking limiting only integers** when it asked for time in the format of (HH:MM). This string

format is converted into a valid DB date format to preserve which day it is. **(Year – Month – Day Hour:Minute:Seconds).**

At the moment, year is set to 2018, month to January because January 1,2018 is Monday, January 2,2018 is Tuesday… Then this is saved into Schedule Table as timesheets. **(This is where the initial planning of storing hours is discarded Schedule Table is Renewed)**

- **Clock-Out**

Clock-out is much **more complex** as it asks for date like Clock-in, but the big difference is that Clock-out also tries to find which is the equivalent Clock-in. Let say Harry clock in at Monday 7pm, and ends at Tuesday 3am, but later on he starts work at Tuesday 7pm.

The Clock-out functions determines whether he started at Monday or Tuesday. This is done my figuring out **the negative operation upon time subtraction**. If no clock-in is found, then the clock-out is not saved.

Otherwise, the number of normal and overtime hours are calculated using boundary of 6:30am – 3:30 pm time subtraction and is save into **Calculation Table.**

The need of **Calculation Table** arose since there is an import with no timesheets, but had hours, and also to preserve the need for processing the number of hours again.

- **Time Subtraction**

Time Subtraction is handled by a datetime library that utilizes string format of time. This must be maintained as the library is flexible, yet hard to call upon with the different formats of time. Essentially it converts everything to seconds and subtracts the second and converts it to its original format using division and modulo.

```
def days_hours_minutes_second(delta): #CONVERTS CHANGE IN TIME INTO ARRAY
    days = delta.days
    hours = delta.seconds // 3600
    minutes = (delta.seconds // 60) % 60
    seconds = delta.seconds % 60
    return 2018,1,days,hours,minutes,seconds,2, 317, 0
```

Tables/Relations Used

Employee Table

- EmployeeID(PK)
  - Integer
  - Eg. 2447532
- FName
  - String
  - Eg. Frinze
- LName
  - String
  - Eg. Lapuz
- RoleID(FK)
  - Integer
  - Eg. 1
- Sup_An
  - Float
  - Eg. 8
- Helt_Ins

- Float
- Eg. 25

Stores all employee Related Data

Role Table

- RoleID(PK)
  - Integer
  - Eg. 1
- RoleName
  - String
  - Eg. Manager
- TaxRate
  - Float
  - Eg. 40
- PayRate
  - Float
  - Eg. 30

Stores all role related data

Schedule Table

- ScheduleID(PK)
  - Integer
  - Eg. 1
- EmployeeID(FK)
  - Integer
  - Eg. 2447532
- MonDateIn
  - String
  - Eg. 01-Jan-18 1:30:01 PM
- TueDateIn
  - String
  - Eg. 02-Jan-18 1:30:01 PM

... SunDateIn

- MonDateOut
  - String
  - 01-Jan-18 6:30:01 PM

...SunDateOut

Stores Timesheets of Employee

Calculation Table

- CalculationID
  - Integer
  - Eg. 1
- EmployeeID(FK)
  - Integer
  - Eg. 2447532
- MonH
  - Float
  - Eg. 2
- TueH
  - Float
  - Eg.2

... SunH

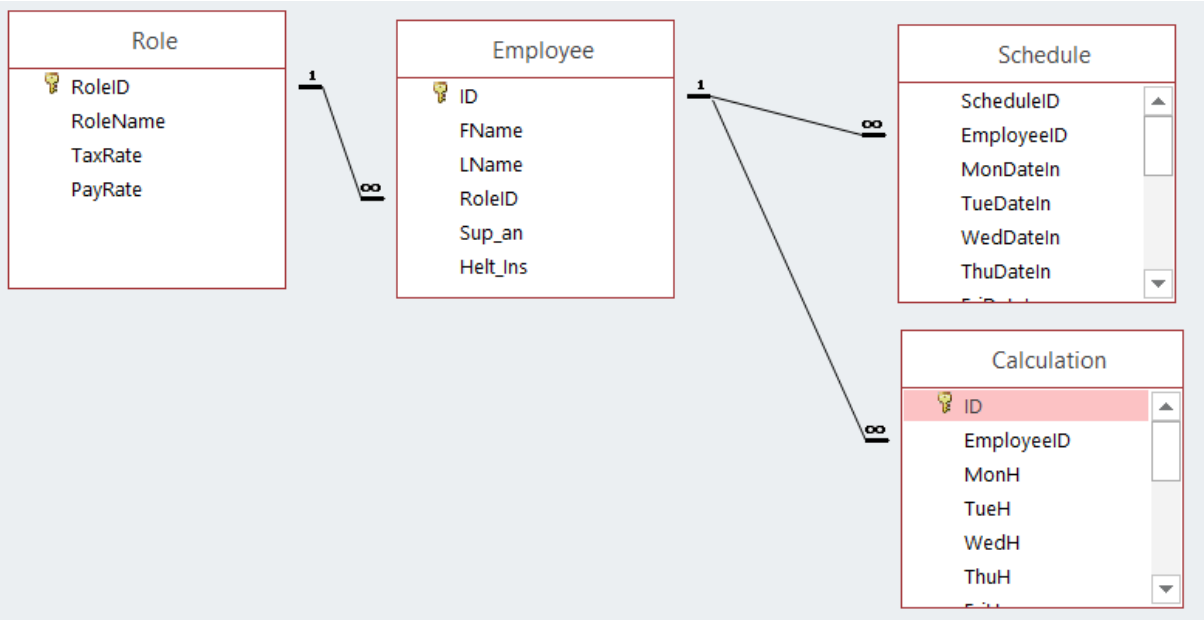
- MonOH
  - Float
  - Eg. 3

...SunOH

Stores Normal Hours (H) and Overtime Hours (OH) of Employee

Calculation Table looks redundant, but it saves processing time for the project especially when dealing with huge calculations at once, and it helps storing import data without timesheets. It is saved in separate from Schedule Table for simplicity in terms of index calculation.

How the relationship looks like:



Pseudocode Display:

Special Display Related Module:

```
MODULE header(title): #PRINTS A NICE HEADER

    PRINT()
    PRINT()
    PRINT("----------------------------  " + title + "  ----------------------------")
```

```
MODULE select(choices_arr, Message): #LIMITS THE INPUT FROM CHOICES
    # ONE OF THE MOST USED FUNCTION
    # ACCEPTS A ARRAY FULL OF CHOICES
    # DISPLAYS MESSAGE (LIKE A PRINT STATEMENT AT THE BEGINNING)
    # RETURNS THE INDEX OF THE SELECTED CHOICE

    choices_arr_len <- len(choices_arr)#

    print(Message)
    FOR i <- 0 TO choices_arr_len: #PRINTS THE CHOICES
        PRINT(str(i+1) + ". " + choices_arr[i]) #listing of the choices

    WHILE True:
        PRINT()
        choice_inputS <- INPUT("Choice: ")


        IF str_to_int_verify(choice_inputS): #INPUT SHOULD BE AN INTEGER
            choice_input <- int(choice_inputS)
            IF choice_input>=1 AND choice_input <= choices_arr_len: #test if the choice input is within the
choices
                BREAK #breaks out of the loop
            ELSE:
                PRINT("Error: input should be in the choices")
                PRINT("You have inputted: "+ choice_inputS)
        ELSE:
            PRINT("Error: input should be an integer")
            PRINT("You have inputted: " + choice_inputS)

    RETURN (choice_input-1) #RETURNS INDEX FROM CHOICE
```

Main Menu Module FROM Internal_Processing

```
MODULE display_menu(): #DISPLAY MAIN MENI
    header("MAIN MENU")
    choices <- {}
    choices[0]<-"Add/Edit/View Employee Details"
    choices[1]<-"Employee Section"
    choices[2] <-"Exit"
    input_choice <- select(choices,"Select From the functionalities below:")

    IF input_choice = 0: #FUNCTIONS CORRESPONDING ON CHOICES
        IF (authent_manager()):
            display_submenu_AE_EmpDet()
        else:
            PRINT("Invalid Authentication")
            display_menu()

    ELSE IF input_choice = 1:
        returns<-authent_employee()
        Employee_ID <- returns[1]
        IF (returns[0]):
            wage_calculations.display(Employee_ID)
        else:
            PRINT("Invalid Authentication")
            display_menu()
    ELSE IF input_choice = 2:
        Exit()
```

Submenu ONLY FOR MANAGERS

```
MODULE display_submenu_AE_EmpDet(): #DISPLAY SUBMENU
        header("SUB-MENU ADD/EDIT/VIEW EMPLOYEES")
        choices <- {}

        choices[0] <- "Display ALL Employees & Details"
        choices[1] <- "Display ALL Managers & Details"
        choices[2] <- "Display ALL Barista & Details"
        choices[3] <- "Add New Employee"
        choices[4] <- "Edit Employee"
        choices[5] <- "Delete Employee"
        choices[6] <- "Import from CSV file"
        choices[7] <- "Export Data to a CSV file 'payroll.csv' "
        choices[8] <- "Back to previous menu"
        choices[9] <- "Exit"

        input_choice <- select(choices,"Select from the functionalities below:") #FUNCTIONS CORRESPONDIDNG
TO CHOICES
        IF input_choice =  0 :
            DisplayEmployee(1)
        ELSE IF input_choice =  1:
            DisplayEmployee_byRole(1)
        ELSE IF input_choice =  2:
            DisplayEmployee_byRole(2)
        ELSE IF input_choice =  3:
            Ins_Employee()
        ELSE IF input_choice =  4:
            DisplayEmployee(0)
            Edi_Employee()
        ELSE IF input_choice =  5:
            DisplayEmployee(0)
            Del_Employee()
        ELSE IF input_choice =  6:
            import_export.Imp_data_from_csv()
        ELSE IF input_choice =  7:
            import_export.export_payroll()
        ELSE IF input_choice =  8:
            display_menu()
        ELSE IF input_choice =  9:
            Exit()
```

Submenu FOR ALL EMPLOYEES

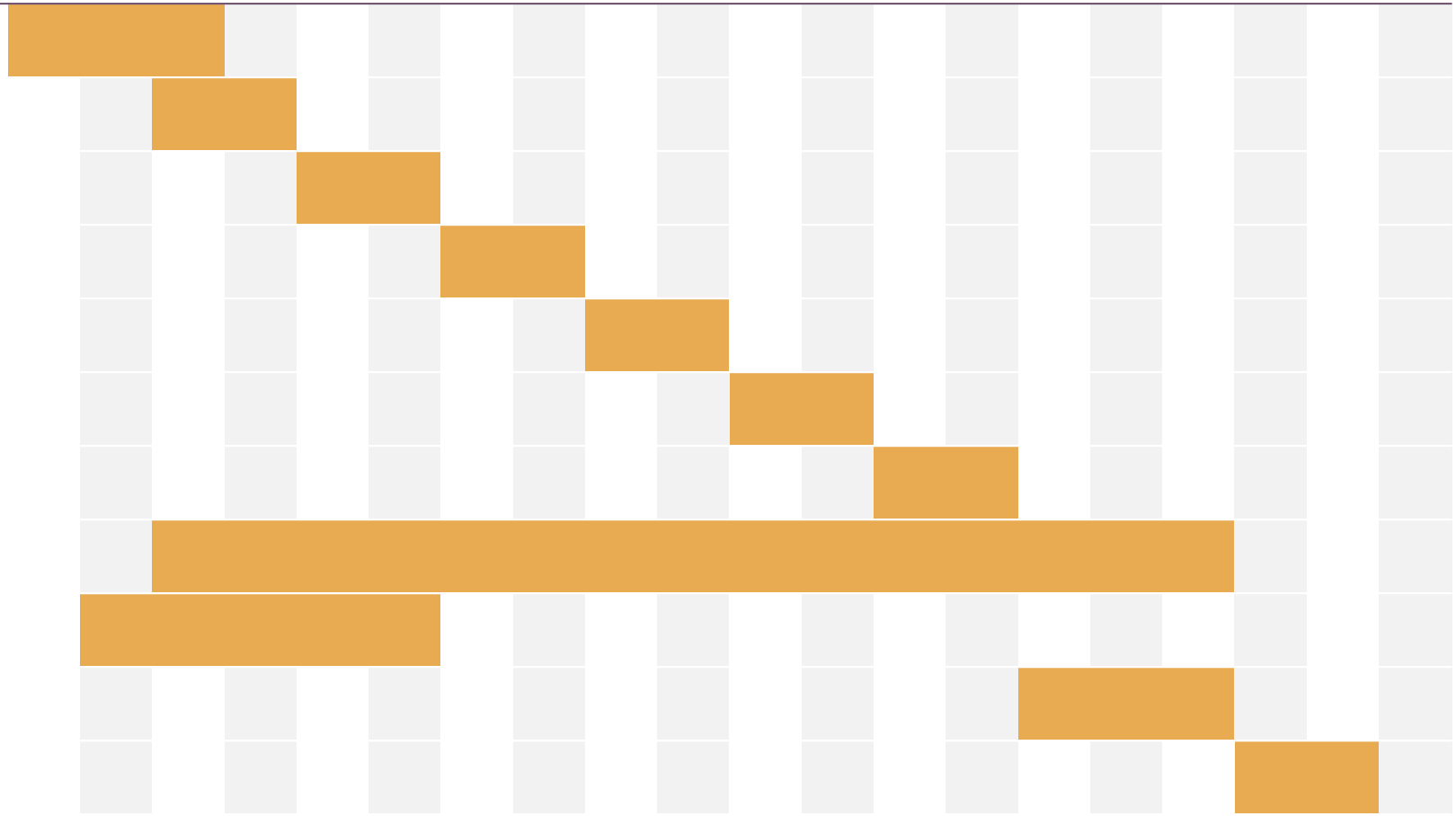
```
def display_submenu_EMPLOYEE_CLOCK(Employee_ID):
    choices <- {}
    choices[0] <- "Clock-In"
    choices[1] <- "Clock-Out"
    choices[2] <- "Calculate your wage"
    choices[3] <- "Go back to previous menu"

    Internal_Processing.header("CLOCK-IN | CLOCK-OUT | CALCULATE WAGE")
    choice <- Internal_Processing.select(choices,"Please select from the functionalities below:")

    IF choice = 0:
        Internal_Processing.header("CLOCK-IN")
        clock_in(Employee_ID)
    ELSE IF choice = 1:
        Internal_Processing.header("CLOCK-OUT")
        clock_out(Employee_ID)
    ELSE IF choice = 2:
        Internal_Processing.header("CALCULATE YOUR WAGE")
        Calculate_Wage_One_Person(Employee_ID)
    ELSE IF choice = 3:
        Internal_Processing.display_menu()
```

Café Au Lait Procedural Plan

ACTIVITY	START	DURATION
Pre-Planning	1	3
Payroll Calculator Function	3	2
DB_Connect (Initial)	5	2
DB_Connect (INSERT,DELETE,UPDATE)	7	2
Menu System	9	2
Clock-in/Clock-out Feature	11	2
Tax Calculations	13	2
Internal Documentation	3	15
External Documentation	2	5
Detailing/Finishing	15	3
Reflection	18	2



Tasks:

- Pre-planning
    - Includes orientating a general idea for all the tasks to be able to determine the order at which tasks are to be done, and how long
    - Allotted: 3 hours
- External Documentation
    - Starts midway in pre-planning as to list and document possible methods and solutions for each individual tasks available
- Payroll Calculator Function
    - One of the main function of the program
    - Calculates raw wage
- DB_Connect Function (Initial)
    - Preparation of database construction and possible fields
    - Testing of Connection with source code running and database
- DB_Connect Function (Insert,Delete,Update)
    - Creating a functionality for updating, inserting, and deleting employees in the system
    - Possibly also inserting schedule/default shifts of employees
- Menu System
    - Making Main Interface (Command Line Interface [CLI])
    - Includes sub-main menu
- Clock-in/Clock-out Feature
    - PIN authentication
    - Schedule Table update
- Tax Calculation
    - Put on long after Payroll Calculator Function due to need of research before starting and clarifications
- Internal Documentation
    - Starts as soon as building source code is started (Payroll Calculator Function)
- Detailing/Finishing
    - Finishing of the project including simply testing and correcting minor bugs or problems
    - Adding more comments on how the program is done
- Reflection
    - Evaluating the finished project
    - Looking for possible room for improvements if project will ever be continued

Test Data with Justifications (FLtestdata.csv):

GNAME	SNAME	ROLE	TRATE	SUPER	HLTH	MON	TUE	WED	THU	FRI	SAT	SUN	Justification
Frinze	Lapuz	Manager	40	8	25	9	0	0	0	0	0	0	Working – all normal hours in weekday manager(boundary)
Danie	Roach	Barista	30	8	25	9	0	0	0	0	0	0	Working – all normal hours barista (boundary)
Ibrahim	Something	Manager	40	8	25	5	0	0	0	0	0	0	Working – all normal hours manager
Yooo	ABC	Barista	30	8	25	5	0	0	0	0	0	0	Working – all normal hours barista
Carlin	Lapuz	Barista	30	8	25	12	0	0	0	0	0	0	Working – overtime(2nd stage) + normal hours - barista
Aldwin	Lapuz	Barista	30	8	25	17	0	0	0	0	0	0	Working – overtime + normal hours - barista
Shernel	Lapuz	Barista	30	8	25	24	0	0	0	0	0	0	Error – invalid working hours
Anand	Karna	Administrator	40	8	25	0	0	0	0	0	9	0	Error – invalid role
Dhruv	Jobanputra	Barista	40	8	25	0	0	0	0	0	9	0	Working – weekend all normal hours (boundary)
Chaitany	Goyal	Manager	40	8	25	0	0	0	0	0	15	0	Working – Weekend normal + overtime hours - manager
Rey	Lapuz	Manager	40	8	25	9	15	3	10	11	6	6	Working – Complete timesheet filled with all the weekdays and weekend

Reflection

- Calendar System instead of a Weekend System
    - For time subtraction, the program uses a hidden date system in the format %Y %M %D - %H:%M:%S (%H – Hour, %M – Minute, %S – Second)
    - %Y %M %D is not shown for the user because at the moment the program has no capability to input DATE (%Y -Year, %M – Month, %D -Day)
    - Hence, the hidden date system of Assigning the Year = 2018, Month = 1 or January
        - Day = 1 for Monday
        - Day = 2 for Tuesday …

Why does it matter?

This creates a bug for when the clock-in is in Sunday, yet clock-out is in Monday. Wherein:

$$T_{Clock-out} - T_{Clock-in} = Negative\ Day\ /\ Carryover\ on\ Hours$$

Yet this bug will happen if calendar system will be NOT used instead of a weekend system, and can be something to be improved upon.

- Overnight Working Feature (Today pm To the Next day am)
    - At the moment, the program can and will detect if the clock-out is the day before or the day
    - It works by using time subtraction

```
delta_tuple = time_subtraction(string_to_time_array(today_in),time_clock_out)
if delta_tuple[3]<0 or delta_tuple[2]!=0: # negative hour -> OVERNIGHT
    if today_in == "None" or yesterday_in == "None":
        print("No Clockout Available for that day ")
        print("Cancelling Clockout")
        print("Going back to menu ")
        display(EmployeeID)
    else:
        delta_tuple = time_subtraction(string_to_time_array(yesterday_in), time_clock_out)
        IsOvernight = True
```

When a negative day or hour occurs, it is considered as an overnight. This feature also detects whether an available clock-out time is available.

E.g.

Clock-In: MON 7:30pm

Clock-Out: TUE 6:30 am

Clock-In: Tue 7:30pm

For Tuesday (Evaluating Same Day)

$$T_{Clock-out} - T_{Clock-in} = Negative\ Hours$$

For Tuesday Clock Out – Monday Clock-in

$$T_{Clock-out} - T_{Clock-in} = Positive\ hours$$

Another code, ensures that the time difference shouldn't be more than a day so that it refuses to take values days before.

Considering, Overnight shifts also opens a new complexity. The range of values it could take. Take these scenarios for simplicity:

- M for morning (before 6:30am)
- A for afternoon (from 6:30am – 3:30pm [normal hours])
- N for night (after 3:30pm)

Originally in the project, it is assumed that all employees can only clock-in after A.

Range of scenarios allowed:

- AA (start afternoon end at afternoon [all normal hours])
- AN (start afternoon end at night [normal hours + overtime])
- NN (start night end at night [all overtime hours])

If we consider that overnight and mornings is allowed

Other Range of scenarios that will be allowed:

- MM (start morning end at morning [all overtime])
- MA (start morning end at afternoon[normal hours + overtime])
- MN (start morning end at night [normal hours + overtime]))

For Overnight:

- AM (start afternoon end at morning the next day [normal hours + overtime])
- NM (start night end at morning the next day[all overtime])
- NA (start at night end at afternoon the next day [overtime + normal])

Each scenario has a particular time subtraction to differentiate between normal and overtime hours. Hence the development time will take longer, yet will be more flexible in the future as the feature will be available if the Café eventually plans to start overtime shifts. E.g. McCafé

Why does it matter?

The program that is working on has two methods in storing the values of timesheets and hours. One for Schedule Table (time sheets), and another for Calculation Table (normal hours + overtime hours recording).

Since development time is short, there are only 3 scenarios that are implemented which are AA, AN, NN. Therefore, the others scenarios save timesheets in Schedule Table when clocking-out, yet does not calculate the number of hours to be saved in Calculation Table.

Note: Calculation Table is only updated for when the scenarios are AA, AN, NN

- Importing Data

Importing Data is not flawless, the program relies on the correctness of equivalent headers in the program and the csv in which it imports from. This means that the set of data should have equivalence on what the program requires.

This also means that interchangeable header names could result in unwanted imports and could not be evaluated by the program which means it relies on the user end.

E.g. Program asks for Equivalent of First Name, but the user inputs “SNAME”, but the program could not reject the import since it is in the same data type and functions, only values are different.

The use of the program asking for the equivalent headers, makes it flexible when the data to be imported is not in the order of interest as the program will adjust the order depending on the headers.

- All the other features

All the other features are working and are great. Error checking is done all throughout the program to limit runtime error and responding with appropriate error message.

The overly use function of selection is great since it limits the amount of keypress and logic a user need to enter data. Yet selection is not always possible E.g. Entering Name. This is where most of the error checking lines are allocated. Since a freestyle input requires functionalities such as alphanumeric or even alphabetic restriction on characters allowed.