For revision practice

SCHOOL OF SCIENCE AND TECHNOLOGY, SINGAPORE

SECONDARY 3
END-OF-YEAR EXAMINATION

COMPUTING
Paper 2 7155/02
Practical (Lab-based)

9 October 2024 (Wednesday) **2 hours 30 minutes**

CANDIDATE NAME			
CLASS		INDEX NUMBER	

Additional Materials:
Electronic version of `AGE.ipynb` file
Electronic version of `coe_bid.txt` data file
Electronic version of `coe_id.txt` data file
Electronic version of `MASTERMIND.ipynb` file
Electronic version of `SALARY.xlsx` data file
Electronic version of `TEMP.ipynb` file
Insert Quick Reference Guide for Python

READ THESE INSTRUCTIONS FIRST
Approved calculators are allowed.

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Programs are to be written in Python.
Save your work using the file name given in the question as and when necessary.

The number of marks is given in brackets [ ] at the end of each question or part question.
The total number of marks for this paper is **70**.

For Examiner's Use		
Task 1	15	
Task 2	10	
Task 3	10	
Task 4	10	
Task 5	25	
Total		/70

Task 1

SSTea Inc, a bubble tea company, uses spreadsheet software to calculate their employees' salaries.

Open the file **`SALARY.xlsx`** and you will see the following data.

	A	B	C	D	E	F	G
1	**SSTea Inc Payslip for Sep**				Date and time		
2							
3	Full name (First and Last)		John Doe		Salary Reference Code		
4	Job rank		Junior		Part-time? (Yes/No)		No
5							
6	Total Hours Worked		201				
7	Normal hours (1x Salary Rate)						
8	Overtime hours (1.5x Salary Rate)						
9							
10	Salary for this month ($)						
11	Average earnings per week (rounded)						
12							
13	**Salary Rate (per hour)**						
14	**Job rank**	**Remarks**	**Salary rate ($)**				
15	Junior		8				
16	Senior		12				
17	Manager	Full-time only	20				

Save the file as:
`TASK1_<your class>_<index number>_<your name>.xlsx`
(e.g. *TASK1_S300_08_Jane Tan.xlsx*)

1 In cell **G1**, enter a function that displays the current **date and time**. **[1]**

2 In cell **G3**, enter a formula that uses (an) appropriate function(s) to create a **Salary Reference Code**. The reference code consists of the following information joined together with no extra characters in between:

- the employee's last name (the second word in **Full Name**)
- the 3-letter abbreviation of the month (as shown in cell **A1**)
- a random 1-digit number from 0 to 9

The **Salary Reference Code** must be calculated in a single formula. **[4]**

3 In cells **C7** and **C8**, enter a formula that uses (an) appropriate function(s) to calculate the number of **normal hours** and **overtime hours** the employee worked.

For a part-time employee, any hours after the 139th are considered overtime.

For a full-time employee, any hours after the 176th are considered overtime.

For example, if a part-time employee worked for 150 hours,139 hours are considered Normal Hours and 11 are considered Overtime Hours. **[4]**

4 In cell **C10**, enter a formula that uses (an) appropriate function(s) to calculate the **salary for this month**. The formula should look up the correct **salary rate** for the employee's **job rank** in the **Salary Rate (per hour)** table, then multiply the salary rate with the number of **normal hours** and **overtime hours** worked respectively.

The salary rate should be multiplied by 1.5 for the overtime hours.

The **salary for this month** must be calculated in a single formula. **[3]**

5 In cell **C11**, enter a formula that uses (an) appropriate function(s) to calculate the **average salary per week**, rounded to the nearest $0.10. You may assume that the month has 4 weeks. **[1]**

6 In cell **C6**, use a formatting tool to make the background red if the employee has at least 20 overtime hours. **[2]**

7 Save your work.

Task 2 begins on the next page.

Instruction to candidates:
Your program code and output for each of Tasks 2 to 5 should be saved in a single .ipynb file using JupyterLab. For example, your program code and output for Task 2 should be saved as:

`TASK2_<your name>_<centre number>_<index number>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in JupyterLab.

Task 2

Open the file **`TEMP.ipynb`**

Save the file as:
`TASK2_<your class>_<index number>_<your name>.ipynb`
(e.g. *TASK2_S300_08_Jane Tan.ipynb*)

The task is to edit program code that allows temperature readings for a week to be input and prints the average temperature out.

For each of the sub-tasks, add a comment using the hash symbol '#' at the beginning of your code to indicate the sub-task that the program code belongs to. For example:

```
In [1]: # Task 2.1
        Program code
        Output:
```

Task 2.1

Room temperatures in a biotechnology lab are highly controlled to ensure the optimum working conditions for the bacteria used in experiments. The following program accepts temperature readings for a week, from Day 1 to 7, and stores the temperatures in a list. It then prints out the average temperature.

```
templist = []

while len(templist) < 7:
    input_s = input("Enter temperature: ")
    temp = float(input_s)
    templist.append(temp)

average = sum(templist) / 7

print("Average temperature is", average)
```

Edit the program so that it:

(a) allows the user to enter any number of entries and terminates the loop only if an exclamation mark `'!'` is entered. **[3]**

(b) displays the number of the day with the highest and lowest temperatures. You may assume that the temperature for each day is different. The output must be displayed in the following format, where X, H, Y and L should be replaced with the appropriate values.:

```
Day X has the highest temperature at H degrees.
Day Y has the lowest temperature at L degrees.
```
[4]

(c) counts and prints out the number of days where the temperature was above the average temperature.

Suitable output messages must be used. **[3]**

Save your JupyterLab notebook for Task 2.

Task 3

Open the file **`AGE.ipynb`**

Save the file as:
`TASK3_<your class>_<index number>_<your name>.ipynb`
(e.g. *TASK3_S300_08_Jane Tan.ipynb*)

The task is to edit program code that calculates the age of a person.

The following program calculates a person's age after accepting today's date and the date of birth. For simplicity, we will assume that we are using a modified calendar where every month has exactly 30 days.

```
def validate_date(date):
    return True

def calculate_age(tdy, dob):
    d_tdy, m_tdy, y_tdy = int(tdy[0]), int(tdy[1]), int(tdy[2])
    d_dob, m_dob, y_dob = int(dob[0]), int(dob[1]), int(dob[2])

    d_age = d_tdy - d_dob
    m_age = m_tdy - m_dob
    y_age = y_tdy - y_dob

    return d_age, m_age, y_age

input_tdy = input("Enter today's date (DD/MM/YYYY): ")
input_dob = input("Enter date of birth (DD/MM/YYYY): ")

if validate_date(input_tdy) and validate_date(input_dob):
    tdy = input_tdy.split('/')
    dob = input_dob.split('/')
    d, m, y = calculate_age(tdy, dob)
    print("The age is {} years, {} months and {} days.".format(y,
m, d))
else:
    print("You have entered an invalid date. This program will be
terminated.")
```

For each of the sub-tasks, add a comment using the hash symbol '#' at the beginning of your code to indicate the sub-task that the program code belongs to. For example:

```
In [1]: # Task 3.1
        Program code
        Output:
```

Task 3.1

Edit the user-defined function `validate_date()` so that it returns `True` if all the four conditions below are met for the date to be valid. Otherwise, the function should return `False`.

A date is considered valid if:

- the number of characters in each entry is 10,
- the separator between day, month and year is a slash `'/'`,
- the day should be between 01 and 30 inclusive, and
- the month should be between 01 and 12 inclusive. **[5]**

Save your program.

Task 3.2

Copy and paste your program from sub-task 3.1.

Edit the user-defined function `calculate_age()` so that it corrects the issue of returning a negative value for the number of months and/or days for some inputs. **[5]**

Save your JupyterLab notebook for Task 3.

Task 4

Open the file **`MASTERMIND.ipynb`**

Save the file as:

`TASK4_<your class>_<index number>_<your name>.ipynb`
(e.g. *TASK4_S300_08_Jane Tan.ipynb*)

The task is to identify and correct errors in program code so that the program works according to the given rules.

The following program creates and plays a game of Mastermind with the user.

Mastermind is a two-player code breaking game. Sam wrote the program using the following rules.

- The computer will randomly generate a 4-digit code 'XXXX' where X is between 1 and 8, inclusive.
- Players will be given 10 tries to guess the code. Each guess is an input of 'XXXX'.
- For every guess:
  - if the player guesses correctly, the program will end with a congratulatory message and state the number of tries taken.
  - otherwise, the program will provide feedback: a 'R' is given for every correct digit that is in the correct position, and a 'W' is given for every correct digit in the wrong position. For example. Code = "1234" and Guess = "2764", the feedback will be 'RW'
- After 10 tries, the correct code will be revealed to the user.

There are several syntax errors and logic errors in the program.

```
import random

# Game variables
tries = 0
guess = ''
code = ''

# Generate the code
while i in range(4):
    temp = random.randint(1,8)
    code += temp

# Gameplay
while tries <= 10:
    tries += 1
    guess = input(f"Try #{tries}, enter code: ")
    feedback = ''
    # the following 2 lists as used to mark the positions
    # of guess & code that has been process for feedback
    guess_pos = (False,False,False,False)
    code_pos = (False,False,False,False)

    if guess == code:
        print(f"Congratulations! You broke the code on try #(tries)")

    # check for any correct digit in correct position
    for i in guess:
        if code[i] == guess[i]:
            feedback += "R"
            guess_pos[i] = True # mark this position as processed
            code_pos[i] = True  # mark this position as processed

    # check for any correct digit in wrong position
    for i in range(4):
        if not guess_pos:
            for x in range[4]:
                if not code_pos and code[x] == guess[i]:
                    feedback += "W"
                    code_pos[x] = True
    print(f"Feedback: {feedback}")

# Code reveal
if tries == 10:
    print(f"The code is {guess}. Better luck next time!")
```

Identify and correct the errors in the program so that it works correctly according to the rules above.

[10]

Save your JupyterLab notebook for Task 4.

Task 5

Open a new JupyterLab notebook and save it as:

`TASK5_<your class>_<index number>_<your name>.ipynb`
(e.g. *TASK5_S300_08_Jane Tan.ipynb*)

The task is to write a program that simulates the Certificate of Entitlement bidding exercise.

All vehicles in Singapore require a Certificate of Entitlement (COE). To register a vehicle, you must first place a bid for a COE in the corresponding vehicle category by stating a reserved price. A successful COE bid gives you the right to own a vehicle that can be used on the road for 10 years.

A simplified open bidding exercise is as follows:

- Bidders key in the reserved price that they are prepared to pay.
- The Current COE Price (CCP) is the reserved price of the highest unsuccessful bid plus $1. The number of successful bidders is limited by the COE quota.
- Bidders can adjust their reserved prices with an increment of at least $1 during the exercise.
- At the close of the exercise, bidders whose reserved prices are above or equal to the CCP will get a COE.

Example:

- COE quota for September exercise = 3.
- Number of bidders = 5, with reserved prices of $100, $88, $70, $70 and $41.
- At the end of the bidding exercise, the bid status of each of the 5 bids will be as follows:

Reserved price	**Bid Status**	**Remarks**
$100	Successful	Only the first 2 bids will be successful. The Current COE Price will be $71. The 3rd and 4th bids (both with reserved price of $70) are not accepted as then the number of successful bids would exceed the COE quota of 3. The remaining 1 unallocated COE quota, which is called unused quota, will be carried forward to the next COE bidding exercise in the following month.
$88	Successful	
$70	Unsuccessful	
$70	Unsuccessful	
$41	Unsuccessful	

You are tasked to write a program that simulates a bidding exercise. You will be given two data files: `coe_id.txt` which contains the bidders' identity codes, and `coe_bid.txt` which contains the bidders' reserved prices. The values in both files are entered in the same order. You are guaranteed that the number of entries in each file is the same.

For each of the sub-tasks, add a comment using the hash symbol '#' at the beginning of your code to indicate the sub-task that the program code belongs to. For example:

```
In [1]: # Task 5.1
        Program code
        Output:
```

All code should have appropriate comments and all identifiers should be appropriately named. **[4]**

Task 5.1

Write a function `load_data()` that has the parameters `id_file` and `bid_file` passed to it. The function must read the contents of both files, create and return a dictionary with the bidders' identity codes as keys and the corresponding reserved prices as values. **[4]**

Task 5.2

The bidders' identity codes are created by combining three parts "xyz" where

- ☐ x is the first character of the bidder's surname (or last name).
- ☐ y is the first character of the bidder's last given name (or first name).
  E.g. The first name "Matthew" will be "M", the given name of "Ah Beng" will be "B".
- ☐ z is a randomly generated 3-digit number.

Write a function `create_id()` that has the parameters `surname` and `given_name` passed to it. The function must generate an identity code based on the above requirements and return the created code. **[3]**

Task 5.3

Write a function `check_current_price()` that has the parameters `dict_data` and `current_quota` passed to it. The function must return the Current COE Price based on the `current_quota`. **[3]**

Task 5.4

Extend your program to create a simple text-based user interface for the bidding exercise. The user interface must have the following features:

- ☐ Set the COE quota
- ☐ Initialise a dictionary identifier `coe_data` using the `load_data()` function. Allow the user to input the name of the `id_file` and `bid_file`.
- ☐ Create a new identity code by asking the user for their surname and given name. The function `create_id()` must be used. Validate that the newly created identity code is unique. If validated, add the identity code to the `coe_data` dictionary with an appropriate default reserved price, which the user can update later if he or she wishes to do so. Otherwise, recreate the identity code using the same surname and given name.
- ☐ Update a bidder's reserved price. The program must validate that the new reserved price is at least $1 higher than the current reserved price.
- ☐ View the Current COE Price and the number of unused quotas based on the COE quota. You must use the function `check_current_price()`.
- ☐ End the current bidding exercise. **[8]**

Task 5.5

Electric cars are gaining popularity and their company's stocks are on the rise. Stock prices can gain or lose over the course of trading each day. You want to analyse the stocks from different companies over a consecutive number of days, to find the best performing stock for investment opportunities.

Given a list of projected changes, `pc = [d1, d2, d3, … , dn]`, where

- `n` is between 1 and 1825, inclusive, and
- `d` is between -100 and 100, inclusive, with positive values representing gains and negative values representing losses,

find the highest gains possible within any number of consecutive day(s).

Write a function `best_profit()` that has a parameter `pc` passed to it, where `pc` is a list with at least one integer. The function must return a single integer indicating the highest gain.

	Example 0	**Example 1**	**Example 2**
Input	[-2, -3, 4, -1, -2, 1, 5, -3]	[5]	[-10,-5]
Return value	7	5	0
Explanation	Starting from d_3 and ending at d_7 will yield the highest gain → 4 + (-1) + (-2) + 1 + 5 = 7		None of the days provide gain. No investment is required.

[3]

Save your JupyterLab notebook for Task 5.

--- END OF PAPER ---