CP1401/CP5639 2021 SP1 Assignment 1

Task:

For this assessment, you are to plan and then code 3 separate console-based programs in Python 3. This assignment is designed to help you build skills using:

1. Pizza Pay Calculator: **Input, Processing and Output**
2. Tennis Results: **Decision structures**
3. Sleep Debt Calculator: **Repetition structures**

Do not define any of your own functions or use any code constructs that have not been taught in this subject. 100% of what you need to know to complete this assignment successfully is taught in the lectures and practicals.

Each program should be written in a separate Python file with the prescribed file name.
Each file should follow the structure provided below by example. That is, each file should start with a module docstring comment at the very top containing your own details and your pseudocode, then your solution code should follow. Replace the parts in <brackets>, which are there to show you where to put your details and work.

Example for program 1:

```
"""
CP1401 2021-1 Assignment 1
Program 1 - Pizza Pay Calculator
Student Name: <your name>
Date started: <date>

Pseudocode:

<pseudocode here>
"""

print("Warm Pizza Pay Calculator")
<code here>
```

Coding Requirements and Suggestions:

- We suggest you work incrementally on these tasks: focus on completing small parts rather than trying to get everything done at once.
- Sample output from the programs is provided with each program description.
  Ensure that your programs match these, including spacing, spelling, etc.
  Think of this as helpful guidance as well as training you to pay attention to detail. The sample output is intended to show a full range of situations so you know how the programs should work. It should be obvious what parts of the samples are user input.
- You do **not** need to handle incorrect types in user input. E.g., if the user is asked for the number of minutes or tennis games won and enters "none" instead of an integer, your program should just crash. That's fine.
- Make use of named constants as appropriate, e.g., for things that would otherwise be "magic numbers", like pay rates or thresholds.
- You are expected to include appropriate comments in each of your programs (not just the module docstring). Use # block comments on their own line for things that might reasonably need a comment. Do not include unnecessary or many comments as these are just "noise" and make your program harder to read.
- Check the rubric below carefully to understand how you will be assessed. There should be no surprises here – this is about following the best-practices we have taught in class.

Program 1 – Pizza Pay Calculator:

Learning outcome focus: Input, Processing, Output
File name: a1_1_pizza_pay_calculator.py

Warm Pizza is a new company revolutionising the pizza home delivery experience by paying its drivers by the trip and minute.
(The faster drivers go, the more trips they can do, but the fewer minutes they get paid for. Interesting...)
This program is a simple calculator for pizza delivery drivers to calculate their pay for a shift based on how many trips they do and how long they drive for.
Warm Pizza pays $1.45 per trip and $0.95 per minute.

Note: there is no looping or error-checking in this program.
The sample output below shows the currency values displayed with two decimal places. This can be achieved using string formatting, like:

```
print(f"Money be like ${value:.2f}")
```

Sample Output from 2 different runs:

```
Warm Pizza Pay Calculator
  Number of trips: 17
Number of minutes: 2
For 17 trips, your pay is: $24.65
For 2 minutes, your pay is: $1.90
Your total pay is $26.55


Warm Pizza Pay Calculator
  Number of trips: 7
Number of minutes: 93
For 7 trips, your pay is: $10.15
For 93 minutes, your pay is: $88.35
Your total pay is $98.50
```

Program 2 – Tennis Results:

Learning outcome focus: Decision Structures
File name: a1_2_tennis.py

This program helps determine the results of junior tennis matches.
Players play for a fixed time, not to a goal score, then one player enters the scores.
The system then determines their match result.
The player is asked for two scores - the number of games they won, then the number of games their opponent won.
Based on their score, they either win, lose or draw.
If a match has at least 8 total games, then they are congratulated for playing a fast match.

With (next to) your pseudocode for this question, include a brief justification/explanation of which decision pattern(s) you chose to use and why.

Note: there is no looping or error-checking in this program.

Sample Output from 3 different runs:

```
Welcome Player 1. How was your match?
    Your score: 3
Opponent score: 2
You won! :)

Welcome Player 1. How was your match?
    Your score: 4
Opponent score: 72
You lost :( Keep trying.
Congratulations on playing a fast match!

Welcome Player 1. How was your match?
    Your score: 4
Opponent score: 4
It's a draw.
Congratulations on playing a fast match!
```

Program 3 - Sleep Debt Calculator:

Learning outcome focus: Repetition Structures
File name: a1_sleep_debt.py

A “sleep debt” represents the difference between a person’s desirable amount of sleep and how long they actually sleep for. Write a program that prompts the user to enter how many hours they slept each day over a work-week period of 5 days, then informs them of their sleep debt status.
Using 8 hours per day as the desirable amount of sleep, determine their sleep debt by calculating the total actual hours of sleep and subtracting that from the total desirable hours of sleep.
Display results messages as demonstrated below.
Note that in this program, you must use an error-checking loop to ensure that the user's inputs are within the range 0-24 inclusive.
As with the other programs, you should think about using constants to make it easy (in one place) to change the program, such as calculating sleep debt for a period of 7 days instead of 5.

With (next to) your pseudocode for this question, include a brief justification/explanation of which repetition pattern(s) you chose to use and why.

Sample Output from 2 different runs:

```
Sleep Debt Calculator
Night 1 hours sleep: 7.5
Night 2 hours sleep: -3
Invalid number of hours.
Night 2 hours sleep: 25
Invalid number of hours.
Night 2 hours sleep: 0
Night 3 hours sleep: 8.75
Night 4 hours sleep: 6
Night 5 hours sleep: 7

Recommended total sleep is: 40
Your total hours of sleep : 29.25
Your sleep debt over this time is: 10.75
```

```
Sleep Debt Calculator
Night 1 hours sleep: 8
Night 2 hours sleep: 8
Night 3 hours sleep: 8
Night 4 hours sleep: 8
Night 5 hours sleep: 8

Recommended total sleep is: 40
Your total hours of sleep : 40.0
You are getting enough sleep. Keep it up!
```

Submission:

Submit 3 separate Python files, correctly named as in the instructions.
DO NOT ZIP/COMPRESS YOUR FILES.
Upload your 3 separate .py files on LearnJCU (under Assessments as instructed).
Submit your assignment by the date and time specified on LearnJCU. Submissions received after this date will incur late penalties as described in the subject outline.

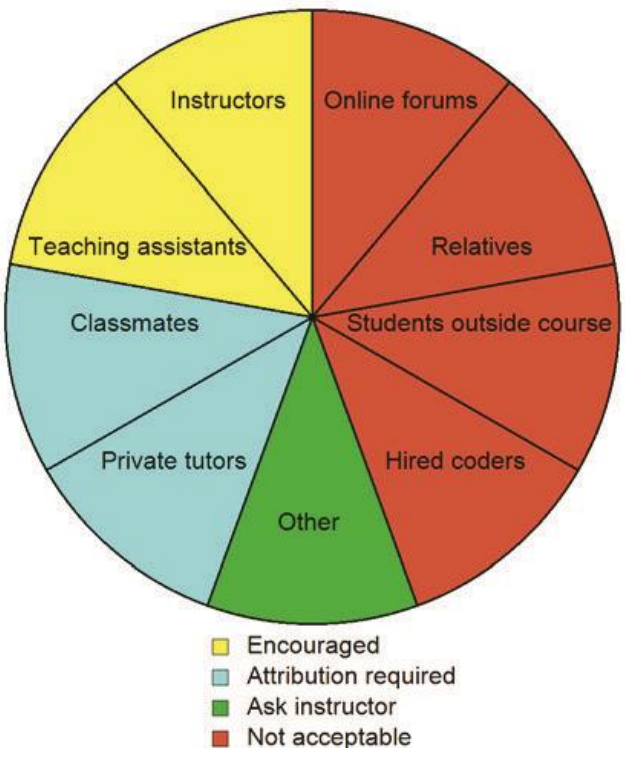
Integrity:

The work you submit for this assignment must be your own. Submissions that are detected to be too similar to that of another student or other work (e.g., code found online) will be dealt with according to the College procedures for handling plagiarism and may result in serious penalties.

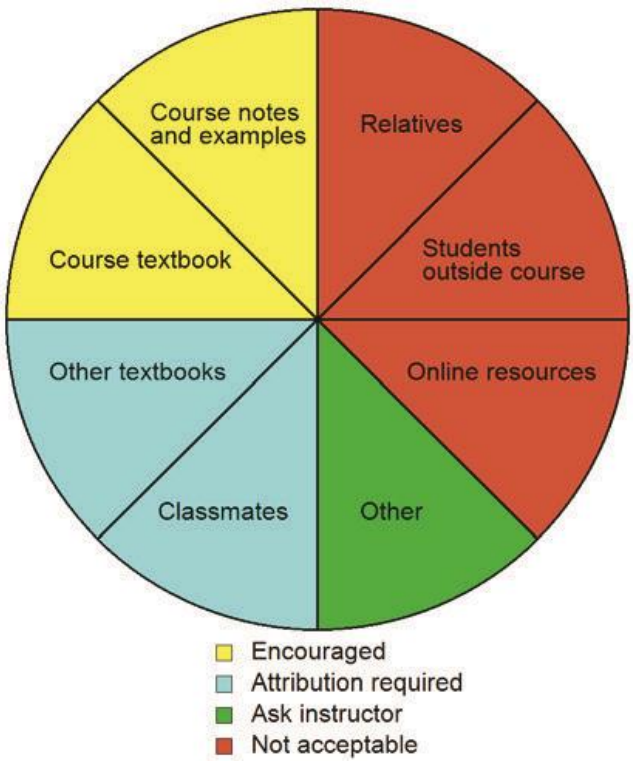
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. You may discuss the assignment with other students and get general assistance from your peers, but you may not do any part of anyone else's work for them and you may not get anyone else to do any part of your work. **Note that this means you should never give a copy of your work to anyone or accept a copy of anyone else's work, including looking at another student's work or having a classmate look at your work.**
If you require assistance with the assignment, please ask **general** questions in #cp1401 in Slack, or get **specific** assistance with your own work by talking with your lecturer or tutor.

The subject materials (lectures, practicals, textbook and other guides provided in the subject) contain all of the information you need for this particular assignment. You should not use online resources (e.g., Google, Stack Overflow, etc.) to find resources or assistance as this would limit your learning and would mean that you would not achieve the goals of the assignment - mastering fundamental programming concepts and skills.

Assistance: Who can you get help from?
Use this diagram to determine from whom you may seek help with your programs.



Resources: Where can you get code from?
Use this diagram to determine where you may find code to use in your programs.



Marking Scheme:

Ensure that you follow the processes and guidelines taught in class in order to produce high quality work. Do not just focus on getting your code working. This assessment rubric provides you with the characteristics of exemplary to very limited work in relation to task criteria.
This rubric will be applied as an average for the 3 programs.

Criteria	Exemplary (9, 10)	Good (7, 8)	Satisfactory (5, 6)	Limited (2, 3, 4)	Very Limited (0)
Algorithm 15%	Clear, well-formatted, consistent and accurate pseudocode that completely and correctly solves the problem.	Exhibits aspects of exemplary (left) and satisfactory (right)	Some but not many problems with algorithm (e.g. incomplete solution, inconsistent use of terms, inaccurate formatting).	Exhibits aspects of satisfactory (left) and very limited (right)	Many problems or algorithm not done.
Correctness 20%	Program works correctly for all functionality required.		Program mostly works correctly for most functionality, but there is/are some required aspects missing or that have problems.		Program works incorrectly for all functionality required.
Similarity to sample output 15%	All outputs match sample output perfectly, or only one minor difference, e.g. wording, spacing.		Multiple differences (e.g. typos, spacing, formatting) in program output compared to sample output.		No reasonable attempt made to match sample output. Very many differences.
Identifier naming 15%	All variable and constant names are appropriate, meaningful and consistent.		Multiple variable or constant names are not appropriate, meaningful or consistent.		Many variable or constant names are not appropriate, meaningful or consistent.
Use of code constructs 20%	Appropriate and efficient code use, including good logical choices for calculations, selections and loops.		Mostly appropriate code use but with definite problems, e.g. unnecessary code, poor choice of selections or loops.		Many significant problems with code use.
Formatting 5%	All formatting meets PEP8 standard, including indentation, horizontal spacing and consistent vertical line spacing. PyCharm shows no formatting warnings.		Multiple problems with formatting reduce readability of code. PyCharm shows formatting warnings.		Readability is poor due to formatting problems. PyCharm shows many formatting warnings.
Commenting 10%	Helpful block/inline comments and top docstring contains all program details, no 'noise' comments.		Comments contain some noise (too many/unhelpful comments) or some missing program details in top docstring or some inappropriate or missing block/inline comments.		Commenting is very poor either through having too many comments (noise) or too few comments.