

Instructions

This assignment will test your knowledge and skills in writing an application software for a given problem, understanding its business rules, tackling the challenge by implementing the problem-solving process, and coding a suitable solution using Python. As such, the assignment requires you to integrate and synthesise what you have learnt so far in this unit (content from week 1 to week 6), in order to design and create a proper working solution.

Background:

This programming assignment involves an escalation of the landing gear sensors problem discussed during the lectures. To guide you in this endeavour, an example case study - Planetary Exploration App- is provided to show you how would you approach the problem from the very beginning, the compulsory elements need to be included in your solution, and the final (and hopefully nice and useful) product.

This Assignment has three parts:

- **Part 1:** Use the problem-solving process to propose a solution to the challenge given.
- **Part 2:** Implement the solution you are proposing in Part 1 using Python via a shell-based menu.
- **Part 3:** Implement the solution you are proposing in Part 1 above using Python via an AI-based platform (such as ChatGPT)

The example case study *Planetary Exploration App* shows the integration of the three parts discussed above. **Please bear in mind that this is an example only.** Your solution does not need to be similar to this one. However, be sure that you include every element described at the end of each part.

Submission

- **This is a SOLO ASSIGNMENT.**
- Allocated marks are shown at the end of each item.
- This assignment requires you to complete Python code written in .py modules, in addition to supporting files.
- Create a folder called “uxxxxx_Assignment1” and drop all your files in there (problem-solving process answers, flowcharts, Word documents, and the like). **Python modules are compulsory. No modules, no marks.**
- Compress (zip) ALL your files and folders created above in one single file called xxxxxx_Assignment1.zip. Upload this file on Canvas by the due date using the drop box provided in the corresponding assignment.
- **There are 180 marks in total for this assignment.**

Your Challenge (Adapted from Floyd, Th. Digital Fundamentals, Pearson, 2015)

As part of an aircraft's functional monitoring system, a logic circuit is needed to display the status of the landing gears before landing. The system should illuminate a green LED if all three landing gears are fully extended when the 'gear down' switch is activated in preparation for landing (the pilots need an extra visual signal lasting 10 seconds as a warning about the activation of the switch). Conversely, a red LED should light up if any of the landing gears fail to extend properly (see Figure 1).

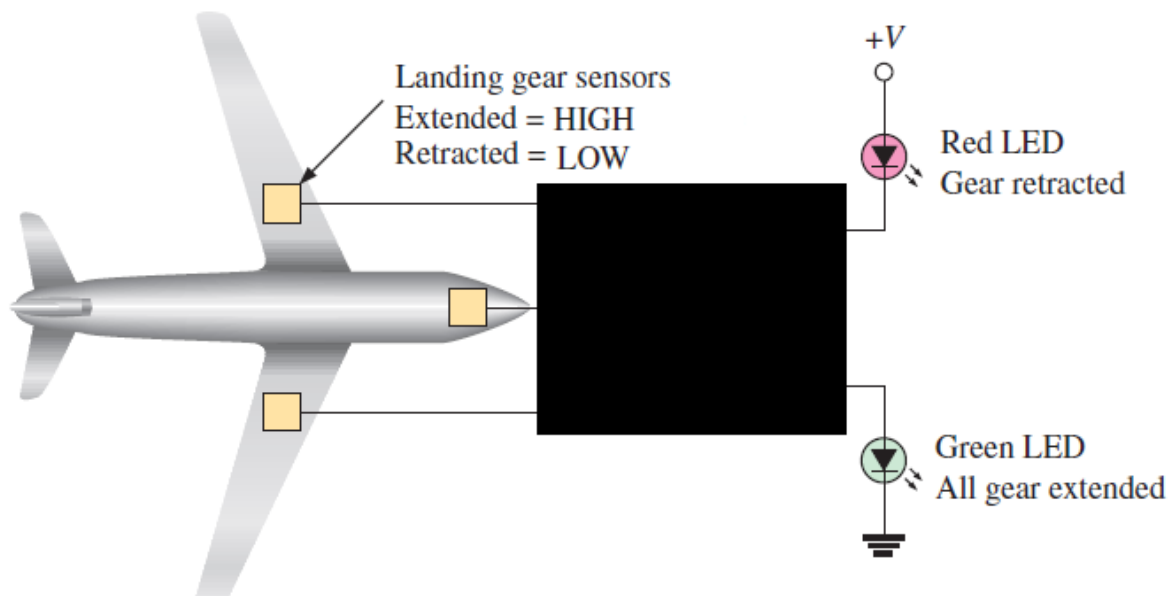


Figure 1: Airplane's Landing Gear Sensors.

In addition, a rotational sensor located in the directional gear below the cockpit, informs the pilots if the gear is not aligned with the aircraft's fuselage (set up as 0 degrees when properly aligned). If this is the case, the red led described above should also be activated (see Figure 2).

The circuit relies on sensors: a HIGH voltage indicates that a landing gear is extended, while a LOW voltage indicates it is retracted. All sensors are digital transducers. However, the directional sensor is an analogue transducer, that is, the signal it portrays happens to be continuous.

Aiming to preserve the integrity of the fuselage if the landing approach fails due to a gear system's malfunction, airplane's lift needs to be calculated and presented to the pilots in the cabin.



Figure 2: Airplane's Directional Gear in Wrong Position.
Credits: SamChui.com. "Batik Airbus A320 Lands with NLG Rotated 90 Degrees"

You are asked to design a software application in Python for the given challenge, using the **Solving Problem Process** explained during your lectures and tutorials. To better assist you in this endeavour, below you will find all required design specifications mapped by our aeronautical engineer (this is your marking rubric):

Part 1: [Total 70 Marks]

1. Briefly explain the problem to be solved. This step aims to make you clearly understand what the problem is about. Stick between one and two paragraphs. [5 Marks]
2. Describe the inputs and outputs. You may need to break the problem down in subsystems to make sure you cover them all. [5 Marks]
3. Develop all "hand-written" calculations. This stage is applicable whenever a calculation needs to be performed within the context of the problem. Make sure your calculations are legible. [10 Marks]

4. Create a list of all the tasks required to provide a suitable working solution. Once again, subsystems may need to be considered at this stage. The minimum tasks you must complete are as follows:
- (**Important:** You must create appropriate functions to get the tasks done)
- a. Design the logic of the circuit. This task involves three steps:
 - i. Write down the corresponding truth table. [5 Marks]
 - ii. Obtain the Boolean expression from the truth table above. [5 Marks]
 - iii. Depict the circuit based on the Boolean expression above. [5 Marks]
 - b. Create a log to store aircraft's landing conditions (its black box) that includes gears' positions, current angle of the directional gear, lift, and LED's status (red/green). To protect the integrity of the black box, all data must be stored using the binary system. (you should try using a dictionary here and convert all the incoming data to binary if it is not in that form already). [5 Marks]
 - c. Following step 4b above, as the directional sensor is analogue, convert the sensor's data into its binary equivalent, ranging from 0 to 9 (that is, 0 to 90 degrees) [5 Marks]
 - d. Calculate the airplane's lift (see appendix). Incorporate the value of the lift within the black box described in point 4b above. As the value must be binary, you may want to discretise it using suitable intervals. [5 Marks]
5. Following previous steps, produce your flowcharts. **Handwritten flowcharts will not be marked.** It is recommended the use of the [Draw.io](https://draw.io) app (or any similar app such as Microsoft Visio) [5 Marks]
6. Following number 5 above, create your pseudocodes. Although there isn't a general rule to create pseudo codes, follow the one shown in the case study as an example. [5 Marks]
7. Propose a set of tests to be used in the code you will create in Part 2. You need to rely in your handwritten calculations obtained in point 3 above. [10 Marks]

Part 2: [Total 70 Marks]

8. Starting from the pseudo codes you created in step 6 above, write the corresponding Python code. In addition to observe the [Python Style Guide](#) (or [simplified version](#)), you must consider the following:
- a. **Constants vs literals.** Using constants is important for the ease of maintenance. Not using constants will result in lower marks. For instance, in the example case study we consider constants for the mass multipliers for each celestial body and the upper limit for each type of astronaut. [5 Marks]
 - b. **Good names for your variables and constants.** Check style against the Python style guide. [5 Marks]

- c. You **must use functions for each task in your code**, for example a *printMenu* function to print a menu and *calculateAverageMass* to calculate the average available mass. [5 Marks]
 - d. Your code **must have a *main()* function** to appropriately direct the program. [5 Marks]
 - e. Program **code layout**. Separate blocks of code by a blank line. [5 Marks]
 - f. **Use of comments**. Comments are important for the maintenance of a program and should contain enough details but keep them concise. Do not comment every single line. [5 Marks]
9. In addition, the program **must work correctly**. That is:
- a. A text-based menu is displayed when first running your program. [5 Marks]
 - b. The menu works properly. [5 Marks]
 - c. Calculations are correctly performed. [5 Marks]
 - d. The output on the shell is properly formatted. [5 Marks]
10. Finally, and most importantly, the business rules are met.
- a. Your code solves the problem proposed (or at least contributes to its solution). [10 Marks]
 - b. The logic of the circuit does the intended job. [10 Marks]

Part 3: [Total 40 Marks]

11. Using any Artificial Intelligence-powered tool (such as ChatGPT or copilot), complete all the tasks given in Part 2. You must submit the Python code generated by your chosen AI (the .py module). [35 Marks]
12. In no more than three paragraphs, comment about the differences between the code developed by you and the one generated by AI. You may want to consider aspects such as speed in generating the code, complexity, and efficiency. [5 Marks]

Notes:

- Proper referencing is very important (including crediting your AI). **10 marks will be deducted if no references are found.**
- Once again, **you must include your Python modules** (.py files). Screenshots, code pasted in text editors (such as Word) will not be considered as valid pieces of code.

Example Case Study

Planetary Exploration App



Credits: NASA Solar System Exploration @solarsystem.nasa.gov

PART 1: Problem-solving process to create a simple Python program via the Shell.

I Background

Our astronauts are about to leave for a trip to outer space. They may be visiting the Moon, Mercury, Venus, Mars or one of the larger moons of Jupiter (IO, Europa, Ganymede or Callisto). Each astronaut has a mass allowance in kg for items that they can take along the trip, including tools and personal items. The amount they are allowed depends on their job description as follows:

Flight Crew are allowed a total of 100kg, and Mission Specialists are allowed a total of 150kg.

Each of the solar system bodies (celestial bodies) has less gravity than the earth. For example, if a mass of 100kg on earth weighs 100kg, on the Moon will weigh the equivalent of 16.6kg.

II Business Rules

a) The Problem at Hand

We need to calculate how much mass each astronaut has left over for personal items, the total available mass and the average available personal mass allowance across all six astronauts. We need to also calculate the weight of the average available personal mass allowance on the celestial body the astronauts are travelling to. We assume there are three crew astronauts and three mission specialists. **Be sure to clearly state your assumptions!!!**

b) Describing Inputs and Outputs

We need a list that contains the names of the celestial bodies that our astronauts may travel to and their Mass multipliers. This list must be stored using an appropriate data structure. Do not hard-code this table. **This list is part of our inputs.**

Celestial body	Mass Multiplier
Mercury	0.378
Venus	0.907
Moon	0.166
Mars	0.377
Io	0.1835
Europa	0.1335
Ganymede	0.1448
Callisto	0.1264

Also, the user needs to decide which celestial body the astronauts are travelling to, alongside their mass **(Second set of inputs)**.





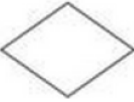


The code will display a list of celestial bodies that are possible destinations along with their mass multipliers. Our code needs to also display the weight allowances for the two different types of astronaut **(All of these are our outputs)**.

III Develop a “Hand” Calculation

We will create (pure creation, that it is!!!) a *flowchart* to start with.

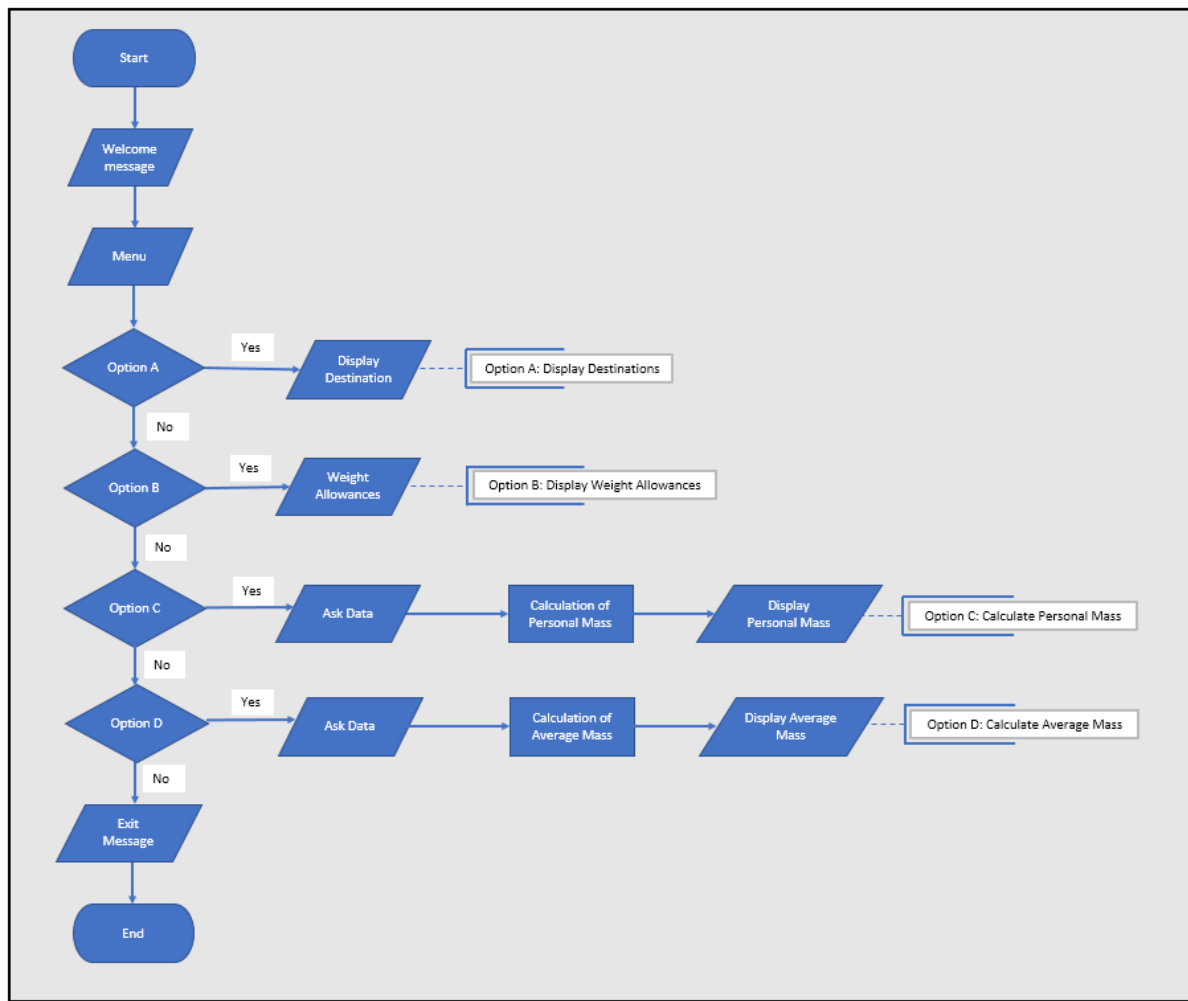
A flowchart consists of special symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur. The arrows connecting the symbols, called *flowlines*, show the progression in which the steps take place. Below is the ANSI standard for flowchart symbols (American National Standards Institute)¹:

¹ “An Introduction to Programming Using Python”, Schneider, D.I. Pearson, 2016

Symbol	Name	Meaning
	<i>Flowline</i>	Used to connect symbols and indicate the flow of logic.
	<i>Terminal</i>	Used to represent the beginning (Start) or the end (End) of a task.
	<i>Input/Output</i>	Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.
	<i>Processing</i>	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	<i>Decision</i>	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no."
	<i>Connector</i>	Used to join different flowlines.
	<i>Annotation</i>	Used to provide additional information about another flowchart symbol.

Once finishing with our flowchart, we'll develop a "*pseudocode*", which is an abbreviated version of actual computer code (hence, pseudocode). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudocode looks more like computer code than does a flowchart. Pseudocode allows the programmer to focus on the steps required to solve the problem rather than on how to use the computer language.

Ok then; let's start with the flowchart according to the problem at hand and input/output (I/O) description:



The above flowchart clearly shows the sequence of tasks that our program will be undertaking. We now translate this flowchart into a pseudocode:

```
Program: Calculate Astronauts' Mass Allowance
start
Print welcome message on the screen
Print (display) the menu on the screen
If [option A] then
    print destination on the screen.
Else [option B]
    print weight allowances.
Else [option C]
    Input destination
    input mass and personal items each crew member.
    Calculate personal allowable mass for each crew member as:
    Formula here [use functions for your calculations]
    Print personal mass for each crew member.
Else [option D]
    print weight allowances.
    Input destination
    input mass and personal items each crew member.
    Calculate average allowable mass for crew members as:
    Formula here [use functions for your calculations]
```

Print average mass for crew members.
Print exit message
end

PART 2: Implement the solution using Python via as shell-based menu.

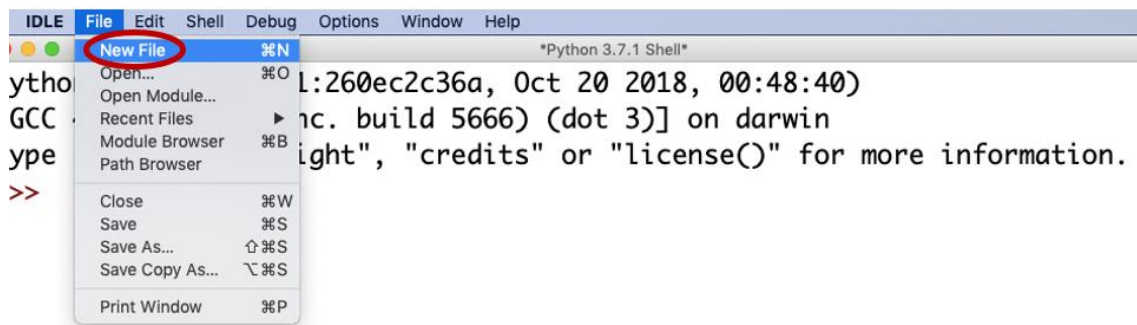
A running example will be shown during the lectures.

Hint: Developing a Command-Line menu.

```
>>>
RESTART: F:\Main Files\Lectures\Intro to Information Technology\Intro to Inform
ation Technology 2020-Sem 2\Python Assignment\Lecture Example\Stage1.py
Astronaut Mass Allowance Calculator
A: Display Program options
B: Display Destinations with Mass Multipliers
C: Display Weight allowances for astronauts
D: Calculate Personal Mass allowances
E: Calculate Average Available mass and weight
X: Exit
Enter A, B, C, D, E or X to proceed:
```

Step-by-Step Guide for Stage 1 (using the case study as an example)

1. Think about your strategy on how you will display the options for user to choose from according to your flowchart. How you will calculate the available personal item mass for each astronaut and how to calculate the average available mass and average available weight. Think about writing simple functions for the repetitive calculations.
2. Create a new Python file, you may call it **ProgAsgStage1**



3. In the Stage1 class (file ProgAsgStage1.py), you may put all code for the user interaction and the calculation into the main method. (You might still need to define global variables and constants outside the main method at the top of the editor window). You might like to use nested lists to hold the name and mass multiplier of the celestial bodies.
4. You will need to write a function to print a menu to the user. One possible example is:

```
def main():
    choice = printMenu()
    print(choice)

def printMenu():
    print("Astronaut Mass Allowance Calculator")
    print("A: Display Program options")
    print("B: Display Destinations with Mass Multipliers")
    print("C: Display Weight allowances for astronauts")
    print("D: Calculate Personal Mass allowances")
    print("E: Calculate Average Available mass and weight")
    print("X: Exit")

main()
```

5. Now add the code that implements your strategy to display the different destinations with their mass multipliers, **TEST**.
6. Add the code to display the weight allowances for the two different types of astronaut, **TEST**.
7. Add the code to calculate the personal mass allowances of each of the six astronauts along with the total available mass, **TEST**.
8. Add the code to calculate the average available mass allowance and weight on destination, **TEST**.
9. Finally, add print statements to print the output to the user.
10. Test your implementation with the test cases mentioned below (and additionally your own).

*For steps 7 & 8 work out your tests **before** you do any coding*

Examples of Test cases:

Test 1

Destination Selected	The Moon
Entered tool weights for crew	100, 100, 100
Entered tool weights for mission specialists	150, 150, 150

Available mass for astronauts:	0,0,0,0,0,0
Total Available Mass	0 kg
Average available mass:	0.0 kg
Average available weight on destination	0.0 kg

Test 2

Destination Selected	Mars
Entered tool weights for crew	90, 90, 90
Entered tool weights for mission specialists	140, 140, 140

Available mass for astronauts:	10,10,10,10,10,10
Total Available Mass	60 kg
Average available weight	10 kg
Average available weight on destination	3.77 kg

Test 3

Destination Selected	Venus
Entered tool weights for crew	99, 98, 101
Entered tool weights for mission specialists	150, 149, 151
Available mass for astronauts:	1,2,-1,0,1,-1
Total Available Mass	2 kg
Average available mass:	0.333333 kg
Average available weight on destination	0.302333 kg

Appendix: Calculating Lift (adapted from grc.nasa.gov)

We can calculate the airplane's lift using the following formula:

Where:
$$L = \frac{1}{2} \rho v^2 S_{\text{ref}} C_L$$

- **L** denotes lift force.
- **V** defines the velocity of aircraft expressed in m/s.
- **ρ** is air density, affected by altitude.
- **S_{ref}** is the reference area, or the wing area of an aircraft measured in square metres.
- **C_L** is the coefficient of lift, depending on the angle of attack and the type of aerofoil.

Assume you are flying an F-117A fully equipped. Hence, your aircraft weighs 52,500 pounds. You want to maintain the equilibrium in straight and level flight at an altitude of 30,000 feet, cruising at 400 knots to conserve fuel when a problem with the gears can be detected. The aircraft's wing area is 1,140 square feet. The coefficient of lift is related to the angle of attack (see Figure 3)

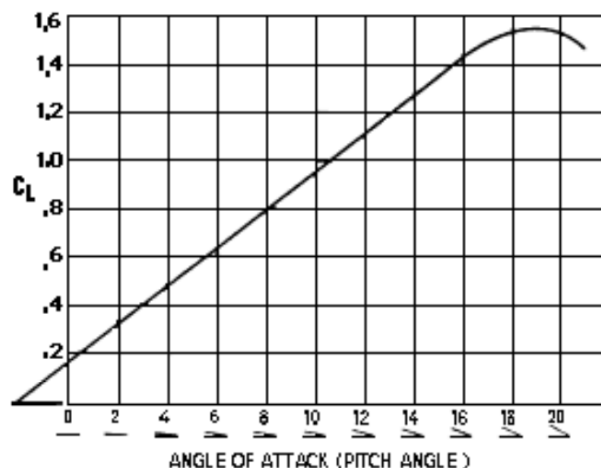


Figure 3: Courtesy of NASA.

As shown in the figure, angles of attack greater than 16 will stall the plane. Thus, you need to present to the pilots with the lift at 5 different pitch angles. Table 1 is also available for your reference.

I.C.A.O. Standard Atmosphere Table

Altitude (Feet)	Density (d)	Speed of Sound (Knots)
0	.002377	661.7
1,000	.002308	659.5
2,000	.002241	657.2
3,000	.002175	654.9
4,000	.002111	652.6
5,000	.002048	650.3
6,000	.001987	647.9
7,000	.001927	645.6
8,000	.001868	643.3
9,000	.001811	640.9
10,000	.001755	638.6
15,000	.001496	626.7
20,000	.001266	614.6
25,000	.001065	602.2
30,000	.000889	589.5
35,000	.000737	576.6
36,089*	.000706	573.8
40,000	.000585	573.8
45,000	.000460	573.8
50,000	.000362	573.8
55,000	.000285	573.8

* Geopotential of Tropopause

Table 1: Standard Atmosphere Table (I.C.A.O)