# A1 Assessment Description

**Directions:**

For this assessment you will code three small programming exercises in Python (one exercise per Python file). You will be focusing on implementation using selections, repetition, and error-handling. These exercises are similar to those you will have seen in your pracs. Each problem should be coded using main only. No additional functions are necessary.

For example:  
def main():

<your code goes here>

Ensure your code follows the coding standards listed on p3.

**NOTE:** You **may not** consult with anyone other than your lecturer or prac tutor regarding this assignment. Activities such as showing your code to someone else to debug or asking how to solve the exercises are considered plagiarism. The purpose of this assessment is to show what **YOU** have learned so far.

**A screenshot of a cell phone

Description generated with very high confidence**

**Submission:**

You are to submit three Python 3 code files (\*.py) which have been zipped into one file. Name this file LastnameFirstname.zip (e.g. if your name were Darth Vader, the filename would be VaderDarth.zip)

Each file within the zipfile is to be named: prog1.py, prog2.py and prog3.py.

This file is to be submitted to the dropbox for your campus, or enrolment mode, on LearnJCU.

# Exercise 1: How much is your sleep debt? (prog1.py)

A “sleep debt” represents the difference between a person’s desirable and actual amount of sleep. Write a program that prompts the user to enter how many hours they slept each day over a period of 5 days.

Using 8 hours per day as the desirable amount of sleep, determine their sleep debt by calculating the total number of hours of sleep they got over the five-day period and subtracting that from the total hours of sleep they should have got.

If the user has a sleep debt display a message saying they need more sleep. If the user does not have a sleep debt, display a message expressing your jealousy.

**Sample output: (user inputs in red)**

**This program will calculate your sleep-debt over 5 days.**

**Please enter Day 1 sleep:**

**6**

**Please enter Day 2 sleep:**

**7**

**Please enter Day 3 sleep:**

**8**

**Please enter Day 4 sleep:**

**7**

**Please enter Day 5 sleep:**

**6**

**Your total hours of sleep were:**

**34**

**Your sleep debt over this time is:**

**6 hours**

**You need more sleep!**

# Exercise 2: Restaurant Selector (prog2.py)

You have a group of friends coming to visit from out of town and you want to take them out to eat at a local restaurant. You’re not sure if anyone has any dietary restrictions, but your restaurant choices are as follows:

* Joes’ Gourmet Burgers – Vegetarian: No, Vegan: No, Gluten-Free: No
* Main Street Pizza – Vegetarian: Yes, Vegan: No, Gluten-Free: Yes
* Corner Cafe – Vegetarian: Yes, Vegan: Yes, Gluten-Free: Yes
* Luigi’s Fine Italian Restaurant – Vegetarian: Yes, Vegan: No, Gluten-Free: No
* The Chef’s Kitchen – Vegetarian: Yes, Vegan: Yes, Gluten-Free: Yes

**Sample output: (user inputs in red)**

**Is anyone in your party a vegetarian (Yes/No)**

Yes

**Is anyone in your party a vegan (Yes/No)**  
No

**Is anyone in your party gluten-free (Yes/No)**

Yes

**Here are your restaurant choices:**

**Main Street Pizza**

**Corner Café**

**The Chef’s Kitchen**

# Exercise 3: Salary Calculator (prog3.py)

A salary calculator will let a user see how much they could earn over several years given some basic information. The calculator assumes that the user will receive a 2% raise in salary at the beginning of each following year.

Write a Python program that will ask the user for the user’s name, the beginning salary, and how many years the person will be working. The program should check for the following values:

* Name must be at least 1 character (i.e. not blank)
* Beginning salary must be at least 10,000
* Years worked must be at least 1, maximum of 10

The program should output on each line, the year and the salary. The final line should list the total salary over the lifetime of the job.

**Sample output: (user inputs in red)**

**Welcome to the Salary Calculator**

**Please enter your name:**

**Di**

**Please enter your beginning salary:**

**500**

**Sorry, your beginning salary must be at least 10000**

**Please enter your beginning salary:**

**20000**

**Please enter the years worked:**

**5**

**Thank you.**

**Your salary will be**

**Year 1: 20000.00**

**Year 2: 20400.00**

**Year 3: 20808.00**

**Year 4: 21224.16**

**Year 5: 21648.64**

## General Principles:

In this assignment, you will be focusing on implementation using selections, repetition, and data validation.

Use the techniques and patterns that you have learned and seen demonstrated in class.

* Code must be indented properly
* Variables should be named according to the standards discussed in lectures 1 and 2
* Each of the programming exercises requires you to obtain input from the user. Ensure that you use an error handling loop to check for valid values (i.e. within an expected range).
* If you use coding syntax that differs from that taught in class, you must explain (via a comment in the code) why you are using this technique
* Good coding standards:
  + Variables:
    - Should be named appropriately using either camelCase or under\_scores
    - Should not be a single character such as “x” or “y” unless they are used in a loop as an index.
    - Names should be meaningful, and describe their use and/or purpose
  + Selections:
    - Should use the appropriate pattern, i.e. no blank condition states (if/else with a blank else)

**ASSESSMENT 1 RUBRIC**

(This rubric will be used with each of the three programming exercises.)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Exemplary (9, 10)** | **Good (7, 8)** | **Satisfactory (5, 6)** | **Limited (2, 3, 4)** | **Very Limited (0, 1)** |
| **Program Execution**  **Correctness** *Worth double* | Program works correctly for all functionality required. | Exhibits aspects of exemplary (left) and satisfactory (right) | Program mostly works correctly for most functionality, but there is/are some required aspects missing or that have problems. | Exhibits aspects of satisfactory (left) and very limited (right) | Program works incorrectly for all functionality required. |
| **Similarity to sample output (including all formatting)** | All outputs match sample output, or only minor differences, 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. |
| **Quality of Code Identifier naming** | All variable names are appropriate, meaningful and consistent. | Multiple variable or names are not appropriate, meaningful or consistent. | Many variable names are not appropriate, meaningful or consistent. |
| **Use of code patterns** | Appropriate and efficient code use, including good logical choices for 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** | All formatting is appropriate, including correct indentation, horizontal spacing and consistent vertical line spacing.. | Multiple problems with formatting reduces readability of code. | Readability is poor due to formatting problems. |
| **Error checking** | Invalid inputs are handled well using control logic as instructed, for all user inputs. |  | Invalid inputs are mostly handled correctly as instructed, but there is/are some problem(s), |  | Error checking is not done or is very poorly attempted. |