Fall 2024 – Programming Assignment 2

Functions

This assignment is an Individual Effort assignment.

The purpose of this assignment is to gain experience using functions and selection statements, Loops as well as effectively combining them with basic calculations.

Learning Objectives:

- Using selection statements (**if/elif/else**)
- Using loop (**While/for**)
- Defining functions
- Using function calls
- Using function return statements
- Combining with basic calculations
- Combining loops and lists with selection statements (`if, elif, else`)

This assignment helps us become familiar with function definitions, function calls, and selection statements and loops. It allows us to write code that can execute different statements based on the current values observed during a particular run of the program. We will use this knowledge to write programs that perform calculations and selectively report on various properties of the calculated values.

Restrictions

Allowed Things:

- **Any arithmetic/comparison/boolean operators:** +, -, *, /, //, %, and, or, >=, <, etc.
- **Data types: int, float, str, bool**
- **Casting functions: int(), float(), str(), bool()**
- **The range(), len() functions**
- **The .append(), .extend(), .remove(), .index() methods on lists**
- **The +, +=, * operators on lists (concatenation, repetition)**
- **The in operator on lists (membership test)**
- **del statements**
- **Building a list by comprehension**

Disallowed Things:

- You are **not allowed to import** anything.
- You are **not allowed to use the function input() and print().** For this assignment you will be defining functions and handling input/output differently.

I. Part1

The following are the 5 functions (within 3 tasks) you must implement for PA2:

Task 1

```
def gift_recommender(preference, budget):
```

Description: For this function, your task is to recommend a gift based on user's preference and budget, the parameters of the function. The function should return a string recommending a gift based on the following conditions:

preference	budget	gift (return string)
'electronics'	Equal or under $100	"gaming earbuds"
'electronics'	Above $100 but under or equal $200	"headphone"
'electronics'	Above $200	"smart watch"
'clothing'	Equal or under $25	"tops"
'clothing'	Equal or under $50 but above $25	"jackets"
'clothing'	Equal or under $100 but above $50	"shoes"
'clothing'	Above $100	"suits"
'jewelry'	Under $500	"ring"
'jewelry'	Equal or above $500 but under $1000	"necklace"
'jewelry'	Equal or above $1000	"bracelet"
not listed above	any price	"decorative gifts!"

Parameters: `preference` (`str`, indicating user's preference or gift category)

`budget` (`float`, indicating user's budget)

Assumptions: `budget` is non-negative (>= 0)

Return value: A `str`, indicating the recommended gift.

Examples:

```
gift_recommender('jewelry', 100.0)       → "ring"
gift_recommender('clothing', 66.0)       → "shoes"
gift_recommender('electronics', 202.0)   → "smart watch"
gift_recommender('clothing', 31.5)       → "jackets"
gift_recommender('jewelry', 1500.0)      → "bracelets"
```

Task 2

To complete this task, you need to write two functions to calculate student's final grade: `calculate_student_score` and `calculate_letter_grade`, where the output of the first function will be used as the parameter for the second function.
Below are the required function signatures:

```
def calculate_student_score(PA, mid_term_exam, final_exam):
```

Description: For this function, your task is to calculate a student's weighted score based on their Programming Assignment (PA), mid-term, and final exam scores. If a student has a score of zero in the final exam, they will receive a weighted score of zero, regardless of their other scores. If the mid-term exam is not taken (score is zero), the score of the final exam will be used as a replacement.

Additionally, please take into account the following weightage for each score:

- PA score carries a weight of 40%.
- Mid-term exam score carries a weight of 30%.
- Final exam score carries a weight of 30%.

Parameters: `PA` (`float`, denoting student's PA score)

`mid_term_exam` (`float`, denoting student's mid-term exam score)

`final_exam` (`float`, denoting student's final exam score)

Assumptions: scores are non-negative (>= 0) with a maximum value of 100.

Return value: A `float`, indicating the students' weighted score.

Examples:

```
calculate_student_score(100.0, 90.0, 95.0)      → 95.5
calculate_student_score(50.0, 70.0, 0.0)        → 0.0
calculate_student_score(80.0, 0.0, 50.0)        → 62.0
```

```
def calculate_letter_grade(score):
```

Description: For this function, your task is to assess a student's score and determine their final letter grade based on the following criteria:

Scores between (90 – 100] ➔ "A"
Scores between (80 – 90] ➔ "B"
Scores between (70 – 80] ➔ "C"
Scores between [60 – 70] ➔ "D"
Scores below 60 ➔ "F"

Parameters: `score` (`float`, denoting student's score)

Assumptions: `score` is non-negative (>= 0) with a maximum value of 100.

Return value: A `str`, indicating the students' final grade.

Examples:

```
calculate_letter_grade(95.5)      → "A"
calculate_letter_grade(0.0)       → "F"
calculate_letter_grade(62.0)      → "D"
```

Task 3

To complete this task, there is a need to write two functions where the first function needs to be called (`is_discount_applicable`) within the body of the second function (`book_price`). Below are the function signatures:

```
def is_discount_applicable(age, is_military, major, gpa):
```

Description: For this function, your task is to check if a customer is eligible to receive a discount based on their age, military status, major, and GPA, according to the following criteria:

- Persons in military services: are eligible.
- Seniors (60 and above): are eligible.
- Students in the 'CSE' major with a GPA of at least 3.7 are also eligible.

Parameters: `age` (`int`, indicating customer's age)

`is_military` (`bool`, indicating the military status of customer)

`major` (`str`, indicating customer's major)

`gpa` (`float`, indicating customer's gpa)

Assumptions: `gpa` is non-negative (>= 0) with maximum value of 4.

Return value: A `bool`, indicating whether the customer is eligible for a discount or not.

Examples:

```
is_discount_applicable(16, False, "CSE", 3.8)  →  True
is_discount_applicable(37, True, "", 0.0)      →  True
is_discount_applicable(24, False, "ISt", 4.0)  →  False
is_discount_applicable(71, False, "", 0.0)     →  True
```

```
def book_price(age, is_military, major, gpa, book_category):
```

Description: For this function, your task is to call '`is_discount_applicable`' function (defined earlier) and calculate the book price that a customer needs to pay. The bookstore has the following pricing structure for different book categories:

- "science" and "fiction": $30
- "novel" and "horror": $20
- "mystery": $10
- "comic": $15
- Not among the categories listed above: $0

The bookstore offers discounts based on the user's eligibility for the following book categories:

- 20% discount for "comic" and "fiction" categories of books.
- 40% discount for "novel" books.
- No discount for other book categories.

Using the above information, calculate the book price for the customer after discounts have been applied.

Parameters: `age` (`int`, indicating customer's age)

`is_military` (`bool`, indicating the military status of customer)

`major` (`str`, indicating customer's major)

`gpa` (`float`, indicating customer's gpa)

`book_category` (`str`, indicating the category of book)

Assumptions: `gpa` is non-negative (>= 0) with maximum value of 4.

`book_category` is all lowercase

Return value: A `float`, representing the book price for the customer.

Examples:

```
book_price(16, False, "CSE", 3.8, "fiction")  →  24.0
book_price(37, True, "", 0.0, "mystery")      →  10.0
book_price(24, False, "IST", 4.0, "comic")    →  15.0
```

Part2

The following are the 5 functions (within 2 tasks) you must implement:

Task 1: Manage a single parking lane

```
def empty_or_full (parking_lane, capacity):
```

Description: This function determines whether there is room in the parking lane and whether it is empty.

Parameters: `parking_lane` (list of strings (`list[str]`) indicating the currently parked license plates)

`capacity` (`an int`, indicating the maximum number of cars that fit into the parking lane)

Assumptions: `len(parking_lane) <= capacity,` and `capacity` is positive (>= 1)

Return value: A `str`, as follows:

- If `parking_lane` has no elements, return string `"empty"`
- If the number of elements in `parking_lane` equals `capacity`, return string `"full"`
- If the number of items in `parking_lane` is neither `0` nor `capacity` , return `"neither"`

Examples:

```
empty_or_full(['RTY-5655', 'FF 22', 'LKJ-7250'],  3)  →  "full"
empty_or_full(['RTY-5655', 'FF 22', 'LKJ-7250'],  10) →  "neither"
empty_or_full([], 1)  →  "empty"
```

```
def park_cars (parking_lane, capacity, cars_to_park):
```

Description: This function places more cars from `cars_to_park` into `parking_lane`, without exceeding `capacity`.

Parameters: `parking_lane` (list of strings (`list[str]`) indicating the currently parked license plates)

`capacity` (`an int`, indicating the maximum number of cars that fit into the parking lane)

`cars_to_park` (list of strings (`list[str]`) indicating the cars to add to `parking_lane`)

Assumptions: `len(parking_lane) <= capacity,` and `capacity` is positive (>= 1)

Return value: A `list`, representing the `parking_lane` after receiving updates from `cars_to_park` up to capacity. The cars in the returned list should preserve their original ordering from parking_lane followed by cars_to_park.

Examples:

```
park_cars(['RTY-5655'], 2, ['FF 22', 'LKJ-7250']) → ['RTY-5655', 'FF 22']
park_cars(['RTY-5655'], 1, ['FF 22', 'LKJ-7250']) → ['RTY-5655']
park_cars(['RTY-5655'], 2, []) → ['RTY-5655']
```

```
def retrieve_cars (parking_lane, cars_to_retrieve):
```

Description: This function removes from `parking_lane` any cars that are in the list `cars_to_retrieve`.

Parameters: `parking_lane` (list of strings (`list[str]`) indicating the currently parked license plates)

`cars_to_retrieve` (list of strings (`list[str]`) indicating the cars that need to be removed from `parking_lane`)

Assumptions: `parking_lane` does not contain duplicate strings (no equal strings at different locations).

Return value: A `list`, representing the `parking_lane` list after removing cars from `cars_to_retrieve`. The cars in the returned list should preserve their original ordering.

Examples:

```
retrieve_cars(['FF 22', 'LKJ-7250'], ['RTY-5655']) → ['FF 22', 'LKJ-7250']
retrieve_cars(['RTY-5655'], ['FF 22', 'LKJ-7250']) → ['RTY-5655']
retrieve_cars(['RTY-5655'], []) → ['RTY-5655']
retrieve_cars(['RTY-5655'], ['RTY-5655']) → []
```

```
def check_cars (parking_lane, cars_to_check):
```

Description: This function verifies whether all the cars in `cars_to_check` are in `parking_lane`.

Parameters: `parking_lane` (list of strings (`list[str]`) indicating the currently parked license plates)

`cars_to_check` (list of strings (`list[str]`) indicating the cars to check)

Assumptions: No assumptions.

Return value: A `bool`. This function returns `True` if all of the cars in `cars_to_check` are in `parking_lane`, and it returns `False` otherwise.

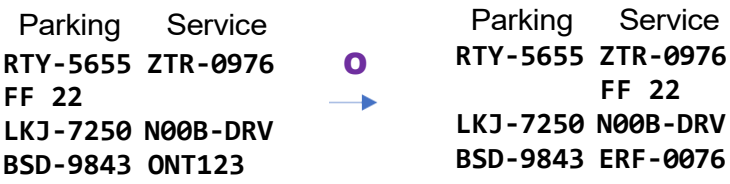
Examples:

```
check_cars(['RTY-5655'], ['FF 22', 'LKJ-7250']) → False
check_cars(['FF 22', 'LKJ-7250'], ['RTY-5655']) → False
check_cars(['FF 22', 'LKJ-7250'], ['FF 22']) → True
check_cars(['RTY-5655'], []) → True
```

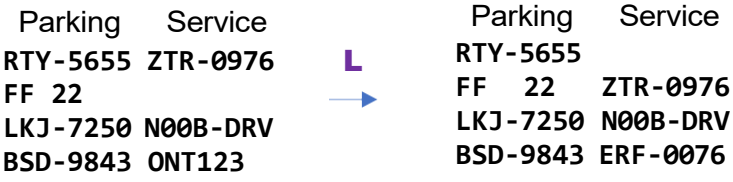
Task 2: Manage two full lanes with the help with an empty spot, the *bubble*.

The possible moves of a bubble are illustrated below.

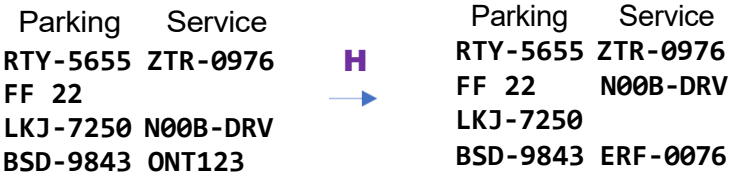
Code O ("shift bubble to the **O**ther lane"):



Code L ("shift bubble to the next **L**ower index):



Code H ("shift bubble to the next **H**igher index):




```
def swap_to_front (parking_lane, service_lane, car):
```

Description: This function returns a list of move codes for the bubble to swap places with other cars so that eventually the specified car shifts to the front of its lane.
Parameters: `parking_lane` (list of strings of the license plates or empty slot in the parking lane)

`service_lane` (list of strings of the license plates or empty slot in the service lane)

`car` (str representing the license plate of the car that needs to be brought to the front)

Assumptions: `car` is an element in one of the `parking_lane` or `service_lane` lists

`parking_lane` and `service_lane` have equal lengths

`parking_lane` and `service_lane` together do not contain any duplicate strings

`parking_lane` and `service_lane` together contain the empty string `''` in exactly one item.

Return value: A `list[str]` representing the codes of bubble moves that bring `car` to occupy the slot at index 0 in the lane list that contains `car`.

Submit your colab file under mini_Project2

- Do not submit screen shots, only submit your assignment code (.ipynb) file.
- Don't forget to name your file: Student Name and ID
- Don't forget to comment your code!
- No hard coding!

Grading Rubric:	
Well-Documented:	5
Code works correctly:	10 (each function)
TOTAL:	100