1. Assignment 1 Reflections and Lessons

a1_sleep_tracker.py

Feedback 1:

Variable and constant name are be simple, meaningful and short.

Reflection:

My variable names below are too long. They should be short, meaningful and simple.

```
    DESIRABLE_HOURS_OF_SLEEP_PER_DAY
    recommended_total_sleep _hours
    total_hours _of_sleep
```

Modified name:

```
    DESIRABLE_SLEEP_PER_DAY
    recommended_total _sleep
    total_sleep
```

Lessons:

The code readability drops when it has long variable names.
It is easy to have typos with long variable names, which causes some bugs.

Feedback 2:

The message is display when total sleep > 40

Reflection:

My condition structure is difficult to be understand.

```
    sleep_debt = recommended_total_sleep_hours - total_hours_of_sleep
    if sleep_debt > 0:
        print(f"Your sleep debt over this time is: {sleep_debt:.2f}")
    else:
        print("You are getting enough sleep. Keep it up!")
```

Modified structure:

```
    If total_sleep < recommended_total _sleep:
        sleep_debt = recommended_total _sleep - total_sleep
        print(f"Your sleep debt over this time is: {sleep_debt:.2f}")
    else:
        print("You are getting enough sleep. Keep it up!")
```

Lessons:

The condition in if-else should be clear and simple.

a1_1_pay_calculator

Feedback:

Magic number with fix value should declare as constant.

Refection:

I should declare a contant variable for 0.05

Modified code:

```
BASE_PAY = 45
GROWTH_RATE = 0.05
print("Experience Counts Pay Calculator")
number_of_hours_worked = int(input("Number of hours worked: "))
experience_level = int(input("      Experience level: "))
hourly_pay = BASE_PAY * (1 + GROWTH_RATE * experience_level)
total_pay = hourly_pay * number_of_hours_worked
print(f"Based on your experience level ({experience_level}):")
print(f"Your hourly pay rate is ${hourly_pay:.2f}")
print(f"Your total pay is ${total_pay:.2f}")
```

Lessons:

Using constant varialbe can help make the code more readable and maintainable.
The constant varialbe name is more meaningful and helpful than a number.
It is easier and quicker to modify contant variable value only at one place than to modify number at multiple places.

a1_2_space_cadet.py

Feedback:

Nested if-else statement

```
if total_score <50
   display "You failed. Please try again next year."
else if plactical_score >= exam_score
    display "You will become a field agent."
else
    display "You will become a desk officer."
```

Repeated print statement

```
elif practical_score >= exam_score:
    print("You will become a field agent.")
else:
        print("You will become a desk officer.")
```

Reflection:

Use nested if-else statement is more clear and more readable.
The two print statement is repeated. The code need to be dry.
Modified code:

```
if total_score < 50:
    print("You failed. Please try again next year.")
else
    If  practical_score >= exam_score:
        result = "field agent"
    else:
        result = "desk officer"
print(f"You will become a desk {result}.")
```

Lessons:

- Sometimes nested if-else statement is clear and readable.
- Use DRY rule to review the code after finishing it. Make the code is DRY when there is repeated code.
-

Rubric Feedback

Algorithm:

Feedback: good, 16 out of 20
Reflection:
- The pseudocode Algorithm is a great method to help me plan my code. Give me a clear picture on how to organize my code before starting coding.

Lessons:
- The code might be messy without writting pseudocode Algorithm.
- It is actually time saving to write pseudocode Algorithm before starting coding.

Correctness:

Feedback: good, 16 out of 20
Reflection:
- The assignment documentation is very clear. It helps to understand the requirement clearly.

Lessons:
- I need to double check if my code match the requirement.

Similarity to sample output:

Feedback: Exemplary, 9 out of 10
Reflection:

- The example output help me understand the logic.

Lesson:

- I need to test my code more carefully to meet the output requirements.

Identifier naming:

Feedback: Good, 10.5 out of 15
Reflection:

- My variable name is too long.
- I didn’t use constant variable when a constant variable is needed.

Lessons:

- I need to make variable name short, simple, and meaningful.
- Long variable name is not only readable but also easier to generate typo bug.
- Constant variable is more readable and meaningful than a number.
- With constant variable, it is much easier and quicker to change contant variable value then to change a number value at multiple places.

Use of code constructs:

Feedback: Satisfactory, 12 out of 20
Reflection:

- I do need to improve the code constructure.

Lessons:

- Sometimes, a nested if-else structure is clearer.
- I need to compare different structure and choose the best one according to
    - Readability
    - Simple
    - maintenance

Commenting:

Feedback: Satisfactory, 5 out of 10
Reflection:

I didn’t do good job on commenting.

Lessons:

I need add comments in my code. Sometimes, I thought the code is clear enough from my side. However, the code is not clear when other people read it. Therefore, comments is needed to improve code readability.

Formatting:

Feedback: Exemplary, 4.5 out of 5

Reflection:

The lecture and example code help me learn on what is a good coding format.

Lessons:

Good formatting improve the code readability.

2. Work Entries:

Date	Hours	Work on
02/01/2023	2	prac 8
03/01/2023	1	prac 8
05/01/2023	0.5	prac 8
06/01/2023	4	prac 8
07/01/2023	3	prac 8
08/01/2023	4	prac 8
10/01/2023	1	prac 9
11/01/2023	1	prac 9
12/01/2023	2	prac 9
13/01/2023	3	prac 9
14/01/2023	4	prac 9
15/01/2023	5	prac 9
19/01/2023	1	prac 10
20/01/2023	1	prac 10
21/01/2023	2	prac 10
22/01/2023	3	prac 10
23/01/2023	4	prac 10
24/01/2023	5	prac 10
25/01/2023	1	Assignment 2
26/01/2023	2	Assignment 2
27/01/2023	3	Assignment 2
28/01/2023	4	Assignment 2
29/01/2023	5	Assignment 2
30/01/2023	5	Assignment 2
31/01/2023	6	Assignment 2

3. Summary:

Pseudocode

Pseudocode is very helpful. I can use pseudocode to develop algorithms and solutions without concern about code syntax. I can focus on the algorithms and solutions.
After finishing the pseudocode, it is easy to transfer it into code. I have already built the structure and logic on pseudocode. What I need to do is just change it with the correct syntax.

Variable name

Give a variable a meaningful name, which improves code readability.
Avoid using long variable names. Choose a short name if a short meaningful name is available. A long name may produce many typo bugs.

Constant Variable

Using constant varialbe can help make the code more readable and maintainable. The constant varialbe name is more meaningful and helpful than a number. It is easier and quicker to modify contant variable value only at one place than to modify number at multiple places.

Don't Repeat Yourself (DRY)

It is difficult for me to understand the importance of DRY. I learn a little bit on DRY when I review the code. I used DRY as guidance to modify my code. Comparing the first version, the second version (DRY version) is cleaner and well-organized. The second version has less redundancy. In my own word, the DRY code is more elegant. The more I learn and practice coding, the deeper I can learn about DRY.

Decision structure

I have been struggled to choose the best if-else decision structure. The instructor taught me several if-else patterns, which are really helpful.

1. if, no else
   a. Use this if you want to do something when the condition is true, but do nothing when it's false.
2. if, else
   a. Use this if you want to do something when the condition is true, and something different when it's false.
3. if, elif, else

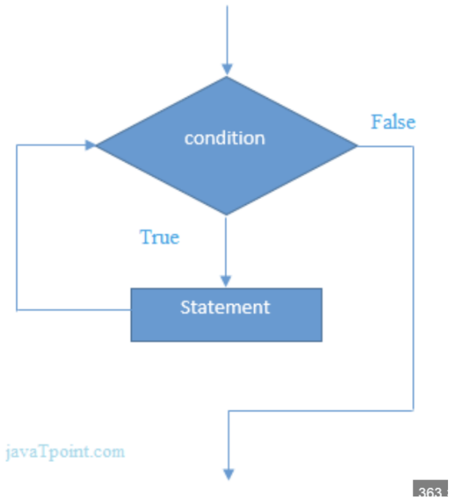
    a. Use this pattern to handle all scenarios.
4. if, elif, no else
    a. use this when plan to handle multiple possible cases, but there will be some cases that don't need to be handled.
    b. Pattern 3 handles all cases, while pattern 4 handles part of cases.

5. if, if, if
    a. Use this when you want multiple outcomes.
    b. Different from the patterns above, all if-cases are not mutually exclusive. One condition/case being true does not affect the other conditions/cases.

Repetition structures

The repetition structure performs the same kind of task multiple times. It makes code shorter and more readable.

- ❖ While loop
    - ➢ Indefinite iteration – repeat an unknown number of times
    - ➢ Used to continue doing something while the condition is True and until the condition is False
    - ➢ while loop is controlled by a condition
        - ■ For a while loop to stop executing, something must happen inside the loop to make the condition False.
        - ■ Otherwise it would loop forever.
    - ➢ While loop flowchart



    - ➢ Standard while loop pattern

```
<Initialize condition>
while <condition based on something from above>
    <body of the loop - do the thing you want to repeat>
    <Update condition>
<do next thing now that the loop is finished when the
condition was false)>
```

- ➢ Menus pattern

```
display menu
get choice
while choice != <quit option>
    if choice == <first option>
        <do first task>
    else if choice == <second option>
        <do second task>
    ...
    else if choice == <n-th option>
        <do n-th task>
    else
        display invalid input error message
    display menu
    get choice
<do final thing, if needed>
```

- ➢ Error checking pattern

```
<get input>
while <input is invalid>
    display error message
    <get input again>
do next thing now when the input is valid
```

- ❖ For loop
  - ➢ Definite iteration – the number of times is known
  - ➢ used to do something with each item in a sequence.
  - ➢ For loop pattern

```
For item in sequence
    Do something with the item
```

  - ➢ range(start, end, step)
    - ■ The range function represents a sequence of integers
      - ● Start: the start of the range. The default value is 0 if not provided
      - ● End: the end of the range, but not inclusive. Required.
      - ● Step: the step of the range. Assumed to be 1 if not provided.

Comments

- ❖ Comments help other people to understand your code easily.
- ❖ Comments add clarity in situations where it is not clear what is going on in the code
- ❖ comments should be used to explain complex pieces of code
- ❖ Too many comments are noise. It interrupts the flow when reading the code.
- ❖ Don't write bad/unnecessary comments.
- ❖ Comments help you understand your own code.

Functions

- The most important aspect of function design is the Single Responsibility Principle (SRP), which means that functions should "do one thing".
- Function makes it easier to reuse code.
- Function is the code that performs the same task in multiple places.
- Functions allow us to break larger programs into smaller, more manageable pieces
- A function is a named group of statements within a program that performs a specific task
- Function structure

```
                    def function_name(parameters):
                            statement statement
```

- Argument
    - An argument is a piece of data that is sent into a function
- Parameter
    - Parameter is what we call the variable inside the function that is assigned the value of an argument when the function is called
- Arguments get passed to function parameters
- Function name
    - A function's name should say what it will do
    - Functions that return Booleans (True or False) are usually used as conditions, and should usually be named like is_*, such as is_large
- Benefits to using functions
    - Reuse code
    - Code is more readable
    - Easier testing and debugging
    - Good for teamwork and large project
- Functions should be testable

Program Structure

```
"""module-level docstring"""
import statements
CONSTANTS
def main():
  statements
def do_step_1() statements
def do_step_2() statements
main()
```

Collections:

- A list is an object that contains multiple data items
    - Each item in a list is called an element
    - Lists can hold items of different (any) types
- Lists, tuples and strings are ordered sequences
- Python sequences include lists, tuples and strings
    - lists are mutable - the elements can be modified
    - tuples and strings are immutable, the elements can not be modified.
        - to change a tuple or string you must create a new one
- iterate over a list using a for loop

```
subjects = ["CP1401", "CP1404", "CP2406"]
for subject in subjects:
   print(subject)
```

- Name sequence with plural name
- Index
    - Index of the first element in the list is 0, second element is 1, and nth element is n-1
    - Negative indexes identify positions relative to the end of the list
        - -1 identifies the last element,
        - -2 identifies the second-last element
    - An IndexError exception is raised if an invalid index is used
        - IndexError: list index out of range
- functions with collections
    - Len, min, max and sum
    - del statement: removes an element from a specific index in a list.
        - del scores[1]
- Methods
    - append(item): used to add item to the end of the existing list
    - sort(): used to sort the elements of the list in ascending order
    - reverse(): reverses the order of the elements in the list
- In operator
    - Use "in" to determine if an item is in a list
- Tuples
    - Tuples are like lists, but immutable
    - Once created, cannot be changed
- Strings
    - Strings are sequences too
    - Much of what works with lists also works with strings (but not modifying)
    - Each character in a string has an index
    - String methods
        - Lower, upper, title

        - endswith(substring), startswith(substring): check if the string ends or starts with substring
        - find(substring): searches for substring within the string
    - + is the "concatenation" operator
        - String + String = String
- Slicing
    - Use slicing to access slices of a sequence (list, tuple, string)
    - A slice is a span of items taken from a sequence • Known as a substring for string slices
    - Slicing format: sequence[start:end]
        - Start is included.
        - End is not included.

Files

- Operating steps
    - 1. Openthefile
    - 2. Processthefile(read or write)
    - 3. Closethefile
- file_object = open(filename, mode)
    - The open function creates a file object and associates it with a file named filename
    - Mode
        - 'r' = read (the default)
        - 'w' = write
        - 'a' = append (write to the end)
- Always close open files
    - file.close()
- Example

```
in_file = open("letter.txt")
for line in in_file:
    print(line.strip())
in_file.close()
```

Summary

- I have learned so many wonderful skills over this semester. The instrutor Miss Cynthia Chan is a great teacher. The documentation is very helpful.
- I still need a lot of practice to help to fully master all those skills.