

## Comments on COMPS258 Assignment 1 (Autumn 2021 Term)

Authored by Dr. Andrew Kwok Fai Lui, Hong Kong Metropolitan University  
Copyright © Andrew Kwok-Fai Lui 2021

*All rights reserved. No part of this note may be reproduced, transmitted, or stored in a retrieval system, in any form or by any means, without permission in writing by the author.*

Dear students,

Hope you are safe, healthy, and well!

The submission rate is very high this year. We are happy to know that most of you have started studying the course, and submission of the assignment is an indication. We know the submission of Assignment 1 (and 2) is important to passing the course. Well done! You have passed the first hurdle.

The performance of Assignment 1 looks good so far. This assignment covers the easy part of the course and therefore we expect a higher mean. However, the other assignments are more challenging and they are not easy to get good marks.

The purpose of the assignments is to keep you on track with your study in this course, in addition to the need for assessment. If you get a good score, then keep going. If not, please try again.

In this course you will solve unseen problems with programming. You will design programs that thoughtfully put the basic components together. The basic components covered in Assignment 1 include statement execution, operators, selection structures and repetition structures.

Programming requires practices. You will learn more skills by writing more programs. So your performance will only improve. Please note that you cannot learn programming just by reading and thinking alone. You must write them and write a lot of them. I hope by Assignment 4 you will have written at least 30 Python programs of various sizes.

If you have not studied well enough, so that you find it difficult to work on the programs in Assignment 1, then I would strongly encourage you to study and/or revise the Study Guide and the textbook. I have also written a number of tutorial notes, which are useful for specific types of programs. Please download these notes from the COMPS258 page in the OLE. You may find the Project Perform and the Programming Library helpful if you are not confident in writing programs. Please also take time to study this discussion paper and hopefully you can learn something from it.

Keep the hard work going. 😊

### Administrative Issues

#### Plagiarism

Please read the copy "Academic Writing: Acknowledging your Sources" that has been sent to you with the course material. It is a matter of having good values of honesty and personal integrity. My colleagues at the School of Science and Technology have developed a research system for detecting plagiarism. We will do a trial run on your submission to test the system. 😊

#### Late Submissions

Late submission without prior approval may result in zero marks. We are very glad that the absolute majority of students submit it on time. We need to manage the assessment tool well and we should approve extension only to those with a genuine reason (health problems, travel abroad, etc).

If you wish to apply for extension, you should do it before the cut-off date (or the extended cut-off date).

We are under pressure from the senior management to manage the cut-off date and submission well. So we may not be able to grant an extension without an adequate reason. Please understand our difficulty.

## Academic Issues

I do not intend to provide a full set of solutions. I would rather focus on a few important concepts covered in Assignment 1 here.

There is no 'model solutions' for programs. In fact, programming is a creative process that the solutions produced by two students should be different at least in some parts. I don't want to fool you by telling you that this or that is a model solution. Rather we should look at the common features shared by the number of acceptable solutions.

Although there may be many acceptable solutions, some solution may be better than others. The better one may excel in the performance, the style of programming, the flexibility to changes, and so on. These factors play an important role in the assessment as time goes on. My suggested solution may not be the best one. I have seen some students could produce better solutions than I would. I am glad about it. 😊

### Basic Input and Output

#### Question 1(a) to 1(c)

These questions are warm up questions.

Python provides the function `print()` to print a text message to the screen (console). In combination with string message formatting (covered in later units), a lot of useful text messages can be printed for users.

For q1(a), the simplest way is to draw the MU by dot matrix and then use the string format method of three consecutive double quotation marks (for defining the start and the end). The following shows an example.

```
print("""
*      * *      *
**      ** *      *
* *      * * *      *
* *      * * *      *
* * * * * * *      *
*  *      * *      *
*      * * *      *
*      *  * * * *
""")
```

Of course, alternatively, you can use a sequence of print statements.

```
print("*      * *      *")
print("**      ** *      *")
print("* *      * * *      *")
print("* *      * * *      *")
print("* * * * * * *      *")
print("*  *      * *      *")
print("*      * * *      *")
print("*      *  * * * *")
```

A common problem is the use of the tab escape sequence `\t`. Note that the tab is handled differently on different operating systems, and it may mean 4 spaces on one computer but 8 spaces on another computer. So using `\t` in drawing may have your figures distorted on different computers. Be careful with it.

For q1(b), the `print()` function allows the concatenation of multiple text messages (strings) in a function call.

```
name = input('Enter your name: ')
print('Happy birthday to me. Happy birthday to me. Happy birthday to', name,
end='. ')
print('Happy birthday to me!')
```

The `input()` function call will stop (i.e. blocked) and wait for the user to enter something on the keyboard and press the Enter key. After that, the program will continue execution. The something entered is stored in the variable `name` (through the assignment operator on line 1). The variable `name` is one of the text messages in the first `print()` statement. The `end` parameter specifies that the ending string is a full-stop with a space (instead of the default newline).

For 1(c), both input and output are involved. A sequence of alternative input and output is often called interaction.

The input function returns the input as a string. If the input is to be handled as a number, then a function call to `int()` or `float()` is used, depending on what type of number you wish to process.

```
print('Rate a Hotel System')
hotel = input('Enter hotel name: ')
score = input('Enter score (0-10) for the hotel: ')
score = float(score)
print('Thank you for given', format(score, '.1f'), 'to', hotel)
```

In this case, the program allows fractional values as input and so the function `float()` is used for the conversion. Finally, the rounding to one decimal place is done by the `format` function with the conversion character `'.1f'` used.

Python usually allows a number of different ways to do the same thing. A convenient alternative (not covered in study unit) is to use the *formatted string literals*. This is a Python feature that allows the embedding of variables and formatting guideline inside a string literal. The string literal should start with the letter `f`. These are also called f-strings.

```
print(f'Thank you for given {score:.1f} to {hotel}')
```

The `score` and `hotel` are variables, and they can be embedded in place with a pair of curly braces. Additional formatting requirement (such as decimal places) can be added after a colon.

## Operators and Calculation

### Question 1(d) – (e)

An important point of these questions is the required programs follow the input – processing – output pattern. This can be called data processing pattern.

- At the input stage, the end-user will enter some data for processing (i.e. say the number of fish cakes).
- At the processing stage, the data is used for calculation and processing (i.e. calculate the amount to pay).

- At the output stage, the result is formatted appropriately in the output for the end-user.

Variables are the programming construct so that data can be used inside a program and can be involved in calculation, input, and output.

Python is a programming language that does not require variables to be "declared" before use. Some programming languages like Java and C requires declaration of all the variables needed for a block of code first. One role of variable declaration is to define the type of variables and such variables can only hold data of that type. This is known as strong typing. However, in Python declaration is not needed and the type of variables is implicit.

The programmers are responsible for knowing the type of data in variables, regardless of the programming languages. Python just gives programmers more flexibility to store any type of data to a variable, but requires the programmers to keep track of the type being used in variables.

Solutions for Q1d to Q1e are given below.

Q1d

```
mangos = input('Enter the number of super mangos: ')
amount = int(mangos) * 10.5
print('Amount to pay is $', format(amount, '.2f'), sep='')
```

Variables are important items in a program that carries data from one part of the program to another part. Pay attention to the variables in the above program and see how the data is transferred from the input to the calculation and then to the output. The programmer must be aware that amount is used as an integer.

Q1e

```
print('Made-to-measure seed block')
length = input('Enter the length(m): ')
width = input('Enter the width(m): ')
height = input('Enter the height(m): ')
volume = float(length) * float(width) * float (height)
print('The price is $', format(volume * 550000, '.2f'), sep='')
```

The programmer must be aware that length, width and height are string first at the input statement, and then becomes an floating point value after the float() conversion. So the program itself does not make the type of data explicit. The programmer must keep track of the types being used in the variables.

### ***Conditional Calculation with Selection Structure***

#### ***Question 1(f) – (h)***

An important point of these questions is the required programs uses the selection structure to do calculation on conditions.

Q1f is based on a selection structure that implements these differential payment conditions:

- If number of mangos is 9 or less, \$10.50 each (as in Q1d)
- If the number of mangos is 10 or more, 1 mango is free (meaning \$0)

```
mangos = input('Enter the number of super mangos: ')
mangos = int(mangos)
if mangos < 10:
    amount = mangos * 10.5
else: # >= 10 mangos
```

```

    amount = (mangos - 1) * 10.5
amount = mangos * 10.5
print('Amount to pay is $', format(amount, '.2f'), sep='')

```

The two cases are found in the if part and the else part of the selection structure.

The following program is also acceptable. The logic is focussed on getting the correct number of mangos for calculation.

```

mangos = input('Enter the number of super mangos: ')
mangos = int(mangos)
if mangos >= 10:
    mangos -= 1 # free mango
amount = mangos * 10.5
print('Amount to pay is $', format(amount, '.2f'), sep='')

```

Q1g can be implemented with at least three if-else structures.

- One checks the validity of the input (number is negative)
- One checks the number of chicken pieces for calculation of amount
- One checks if the purchase amount is over \$200 or more for additional discount.

Be mindful if the position of the if structures in the program.

```

print('Metro Chicken')
numberstr = input('Enter the number of chicken pieces: ')
number = int(numberstr)
if number > 0:
    if number <= 6:
        amount = number * 15
    else:
        amount = 6 * 15 + (number - 6) * 12
    if amount >= 200:
        amount = amount * 0.9
    print('Amount to pay is $', format(amount, '.2f'), sep='')
else:
    print('Sorry the input is not an integer')

```

Q1(h) has more rules, and the 9 different pricings are summarised in a table. The 9 pricings comes from 2 sets of 3 options. In addition there is error checking needed. In summary, we need the following in the program.

- A (nested) if structure for checking correct input
- A top-level if-elif-else structure for checking the first set of 3 options of type.
- A second-level if-elif-else structures for checking the second set of 3 options of number of colors.

```

print('MU Name Cards')
type = input('Enter the required paper type (A, B, C): ')
color = int(input('Enter number of colours : '))
lot = int(input('Enter number of lots of 500 : '))

correct = False
if type == 'A' or type == 'B' or type == 'C':

```

```

if color == 1 or color == 2 or color == 4:
    if lot > 0:
        correct = True

if correct:
    if type == 'A':
        if color == 1: rate = 140
        elif color == 2: rate = 180
        else: rate = 250
    elif type == 'B':
        if color == 1: rate = 100
        elif color == 2: rate = 120
        else: rate = 160
    else:
        if color == 1: rate = 80
        elif color == 2: rate = 90
        else: rate = 120
    amount = rate * lot
    print('The amount to pay is $', format(amount, 'd'), sep='')
else:
    print('Error in the input')

```

An alternative is given below.

```

if correct:
    if type == 'A':
        if color == 1: rate = 140
        if color == 2: rate = 180
        if color == 4: rate = 250
    if type == 'B':
        if color == 1: rate = 100
        if color == 2: rate = 120
        if color == 4: rate = 160
    if type == 'C':
        if color == 1: rate = 80
        if color == 2: rate = 90
        if color == 4: rate = 120
    amount = rate * lot
    print('The amount to pay is $', format(amount, 'd'), sep='')
else:
    print('Error in the input')

```

There are two features that this alternative is not as good as the first program.

- The execution speed is slower
- The intrinsic relation between the if conditions is not clear. For example, the selection for fabric A B and C is supposed to be coming from the same selection logic. In the second program, they are three different if structures instead of one if-elif structure.

Please refer to Appendix for more discussion about the differences between if, if-else, and if-elif-else structures.

## Loops

### Question 2(a) – (c)

Question 2(a) ask you to write a single loop. The solution uses a for loop. For loops are suitable for counter-controlled loops, and the start, the end, and the step size is known

Q2a

```
for x in range(0, 501, 2):
    print(x, end=' ')
```

Question 2(b) also ask you to write a single loop. A while loop is used because the number of iterations is not known. The stopping condition is given instead (which is when the savings reaches the target amount).

Q2b

```
target = input('Enter target ($): ')
target = float(target)
savings = 10000
N = 0
while savings < target:
    N += 1
    savings = savings * 1.15
    print('After ', N, ' months saving is $', savings, sep='')
print('Reached $', target, ' after ', N, ' months', sep='')
```

Q2c has some calculation needed based on the math module.

```
import math

print(format('Degrees', '>10s'), end='')
print(format('Radians', '>10s'), end='')
print(format('Sin', '>10s'), end='')
print(format('Cos', '>10s'))
for angle in range(0, 91, 5):
    radian = math.radians(angle)
    sin = math.sin(radian)
    cos = math.cos(radian)
    print(format(angle, '10.4f'), end='')
    print(format(radian, '10.4f'), end='')
    print(format(sin, '10.4f'), end='')
    print(format(cos, '10.4f'))
```

## Pattern Printing

### Question 2(d)-(e)

The data validation issue is the most talked about one on this question. Note that data validation here means that the value entered is logically correct. Logically correct means it makes sense. In this context, the size of the square shape would only make sense if it were positive. So the program should check for it.

You need not check for non-integer input.

Q2d

```
#program to print a square on screen
size = int(input('Enter the size of the pattern: '))

if ((size < 1) or (size > 20)) :
    print ('The entered size should fall between 1 to 20')
else :
    for row in range(size) :
        for col in range(size) :
            if (col == 0 or col == size - 1) :
                print ('$ ', end = '')
            else :
                print ('+', end = '')
        print()
```

Q2e

```
#program to print a square on screen
size = int(input('Enter the size of the pattern: '))

if ((size < 1) or (size > 20)) :
    print ('The entered size should fall between 1 to 20')
else :
    for row in range(size) :
        for col in range(size) :
            if (col == 0 or col == size - 1) :
                print ('$ ', end = '')
            else :
                print (col % 10, end = '')
        print()
```

### Combined Loops and Logic

#### Question 2(f)

This question requires you to integrate the skills in handling logic and loops in one program.

A better way to approach this question is to consider and list out the tasks needed for each loop.

- Read in a daily IAPI
- Check the validity of input, and print error and go back if invalid input
- Sum the total
- Check if the score is the maximum or minimum score so far
- Check if it is very unhealthy
- Check if this IAPI is higher than the last day.

The variables `maxAPI` and `minAPI` play the key role in tracking the minimum and maximum API. As the case of all variables, they must be initialised. There are two ways to initialize variables used for capturing the maximum and minimum.

- Before the loop



- At the time of the first loop with the condition based on `day`. Consider why this is so.

Q2f

```

print('IAPI Analyser')
maxAPI = minAPI = prev = None
countH = countU = countWorse = 0
sum = 0
period = int(input('Enter number of days of the IAPI data collection period: '))
day = 1
while day <= period:
    print('Enter the daily IAPI of day ', day, ' : ', sep='', end='')
    iapi = float(input())
    if iapi < 0 or iapi > 20:
        print('Input Error: IAPI should be between 0 and 20.0')
        continue
    sum += iapi
    if maxAPI is None or iapi > maxAPI:
        maxAPI = iapi
    if minAPI is None or iapi < minAPI:
        minAPI = iapi
    if iapi > 10.0:
        countU += 1
    if iapi <= 2.0:
        countH += 1
    if prev is not None and iapi > prev:
        countWorse += 1
    prev = iapi
    day += 1

print('Average daily IAPI in the period:', sum / period)
print('Maximum daily IAPI is', maxAPI)
print('Minimum daily IAPI is', minAPI)
print('Number of days with Very Unhealthy IAPI is', countU)
print('Number of days with Healthy IAPI is', countH)
print('Number of days with IAPI higher than previous day is', countWorse)

```

## Operator Evaluations

### Question 2(g)

You should find the table of precedence applicable to Question 2(g). What you needed was to get the table and note the precedence of the operators. The one with the highest precedence will be executed first.

This is just the example

```

x + y + x * 2
Order of execution is:
1.    x * 2
2.    x + y
3.    x + y + x * 2

Final value is: 15

(i)

```

```

Assume a = 2 and b = 3
a - 4 * b
Order of execution is:
1. 4 * b
2. a - 4 * b

```

```

Final value is: - 10

```

```

(ii)
Assume a = 2 and b = 3
a + b / 2 == b + 4
Order of execution is:
1. b / 2
2. a + b / 2
3. b + 4
4. a + b / 2 == b + 4

```

```

Final value is: False

```

```

(iii)
Assume a = 2 and b = 3
a**2 > b and 3 < 4
Order of execution is:
1. a**2
2. a**2 > b
3. 3 < 4
4. a**2 > b and 3 < 4

```

```

Final value is: True

```

```

(iv)
Assume a = 2 and b = 3
a = b = not(a > 3)
Order of execution is:
1. a > 3
2. not(a > 3)
3. b = not(a > 3)
4. a = b = not(a > 3)


```

```

Final value is: True

```

A precedence table for common Python operators

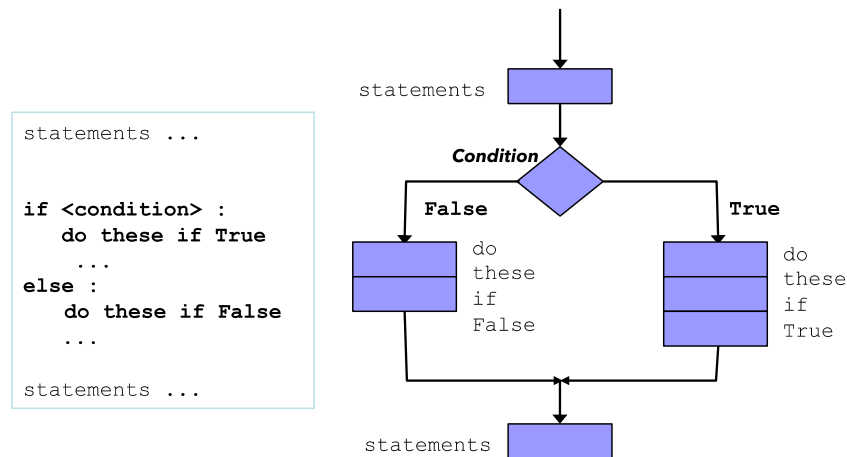
Increasing precedence 	Operators	Associativity	Remarks
	( )	left to right	Bracket
	f(...)		Function call
	**	left to right	Exponential
	+x, -x		Positive, negative
	* / % //	left to right	Multiplication, division, modulus, integer division
	+ -	left to right	Addition, subtraction
	==, !=, >, >=, <, <=, is, is not, in, not in	left to right	Comparison, identity, membership
	not	left to right	Logical
	and	left to right	Logical
	or	left to right	Logical
	=	right to left	Assignment

## Appendix. Two-way Comparison: if and if-else structures

Python provides support for implementation of **two-way selection**.

- If **True**, do A.
- If **False**, do B. An optional **else** statement block can be added to the end of an if structure.

The if-else structure can contain both **if** statements and **else** statements. The following diagram shows how the two-way selection is implemented using an if-else structure.



The following shows an example of using if-else structure to implement the dosage calculation program that was discussed before. Note that there is a colon (:) after **else**.

Example: *dosage-cal-else.py*

```

age = int(input("Enter your age: "))
if (age <= 12) :
    dosage = 1    # execute this if True
else :
    dosage = 3    # execute this if False
print("The dosage is", dosage, "tablet(s)")

```

Note that two-way decision could be implemented correctly with all the following methods.

Method A: if-else	Method B: if with default value	Method C: two if structures
<pre> if (age &lt;= 12) :     dosage = 1 else :     dosage = 3 </pre>	<pre> dosage = 3 ... if (age &lt;= 12) :     dosage = 1 </pre>	<pre> if (age &lt;= 12) :     dosage = 1 if (age &gt; 12) :     dosage = 3 </pre>
<ul style="list-style-type: none"> <li>• The two ways' outcomes are found in one place, making them visible to readers. (Positive)</li> <li>• One comparison is needed, so faster execution speed. (Positive)</li> </ul>	<ul style="list-style-type: none"> <li>• The False outcome is not clear. (Negative)</li> <li>• One comparison is needed, so faster execution speed. (Positive)</li> </ul>	<ul style="list-style-type: none"> <li>• The two ways' outcomes are found in one place, making them visible to readers. (Positive)</li> <li>• Two comparisons are needed, so slower execution speed. (Negative)</li> </ul>

We considered two factors in deciding which is the better method:

- Readability: the condition and the two outcomes should be highly visible to readers.
- Speed: the time taken to execute the code is dependent on the number of comparisons.

The if-else method is preferred if readability and speed is important. The condition and the two outcomes are in the same structure (the if part and the else part), which makes it is highly visible. The condition is also executed once.