**Discussion 1**

What is Abstraction? Why do we need it? What are the two aspects of Abstraction?

**Discussion 2**

When choosing a password for online accounts, there are typically certain requirements for the strength of the password. Develop a Python program for testing if a string satisfies some appropriate criteria for a strong password. It’s up to you to define the requirements.

**Discussion 3**

Consider a system for storing anonymous grades of each lab class. Define a data structure, which can identify individuals in each lab group by an ID number 1-40 (inclusive). To identify the person in the entire class you would also need the group name, e,g., ‘FE2’. Each corresponding person should have a number between 1-100 (inclusive) to define grade.

**Discussion 4**

Given two lists of grades (list of integers) from two classes, write a Python program that will check which class has the highest average score and the highest maximum score.

**Discussion 5**

Write a Python program, in the fewest number of lines possible, which creates a list of all the square numbers: x2 (where 1<=x<=100) that are divisible by 3.

Guide for Instructor

**Discussion 1**

* Abstraction is a process to identify what is important without worrying too much about the detail so that we can focus on the important part.
* Since Abstraction provides a means to distil what is essential, it can help us to come up with manageable approach to create computational solutions. In addition, Abstraction also help us to view things at different degrees of detail.
* Abstraction can be performed in two main aspects: data and algorithm, which form the two bases for programming: data structures and functions. This session is focus on the first part, data structures.

**Discussion 2**

* The students can decide their criteria for strong passwords. Some common criteria are as follows:
  + At least one upper case letter
  + At least one lower case letter
  + At least one digit
  + The length of the password is more than 8 characters
  + etc.
* The purpose of this question is to let the students discuss and practice common useful string functions and methods, e.g., isupper(), islower(), isdigit(), len(), etc.
* The students’ solutions should correspond to their criteria for strong passwords
* Example code, 4-1.py, is provided for reference.

##############################################################

# a password is strong if all the following conditions hold:

# - its length is more than 8

# - it has at least one upper case letter

# - it has at least one lower case letter

# - it has at least one digit

##############################################################

LENGTH = 8 # the minimum length for a strong password

password = input("Input your password: ")

upCase = False # indicating if the password has at least one upper case letter, default value: False

lowCase = False # indicating if the password has at least one lower case letter, default value: False

digit = False # indicating if the password has at least one digit, default value: False

for char in password: # iterate on each character of the password

if char.isupper(): # if the character is in upper case, set upCase to True

upCase = True

if char.islower(): # if the character is in lower case, set lowCase to True

lowCase = True

if char.isdigit(): # if the character is a digit, set digit to True

digit = True

length = len(password) # get the length of the password

strong = upCase and lowCase and digit and length > LENGTH

# strong would be True, if all the conditions hold

if strong:

print("Your password is strong enough.")

else:

print("Your password is weak.")

**Discussion 3**

* The purpose of this question is to let the students discuss the characteristics of different composite data types. Based on the built-in data structures provided by Python (strings, lists, tuples, dictionaries), students are asked to come up with “suitable” data structures to fulfil the requirements of this question.
* There is no absolute “correct” answer for this question. As long as the information is properly stored and can be easily retrieved, the solution is correct. For example, given group name and student ID, the grade of the student should be able to be retrieved.
* Notice that if students use Python dictionaries, the key has to be immutable and unique.
* Example code, 4-2.py, is provided for reference.

#####################

# solution 1: to use a dictionary

# the key is a tuple (group\_name, ID). NOTICE: the key has to be immutable and unique

# the value is the grade

grades = {

('FS1', 1) : 45,

('FS1', 2) : 75,

('FS1', 3) : 25,

('FS1', 4) : 65,

('FS2', 1) : 75,

('FS2', 2) : 40,

('FS2', 3) : 70,

('FS2', 4) : 80

}

print(grades[('FS1', 1)])

######################

# solution 2: to use a dictionary

# the key is string of group name (immutable)

# the value is a list of grades in the group

# ID = index + 1

grades2 = {

'FS1' : [45, 75, 25, 65],

'FS2' : [75, 40, 70, 80]

}

print(grades2['FS1'][1])

**Discussion 4**

* The purpose of this question is to let students discuss and practice common functions and methods of Python lists, e.g., min(), max(), len(), etc.
* Students may not know they can use min(), max(), and len() to quickly solve this question. Instead, some students may use for/while loops to obtain the maximum and minimum values. This is also a good chance to tell the students how the built-in min() and max() functions are implemented.
* Example code, 4-3.py, is provided for reference.

scores1 = [10, 11, 15, 20, 55, 76, 90, 84]

scores2 = [4, 9, 12, 98, 35, 42, 4, 5, 10]

avg1 = float(sum(scores1) / len(scores1))

avg2 = float(sum(scores2) / len(scores2))

maxNum = max(max(scores1), max(scores2))

maxAvg = max(avg1, avg2)

print("Highest Avg: ", maxAvg)

print("Hightest Score: ", maxNum)

**Discussion 5**

* The purpose of this question is to let students discuss and practice Python list comprehension.
* The same functionality can be achieved by a for loop, as shown in example code, 4-4.py.
* Some students may not know this question can be easily solved by Python list comprehension. Instructors can highlight that list comprehension is common and helpful. Make sure the students know how to use it.
* Example code, 4-4.py, is provided by reference.

########################################################

# solution 1: to generate the list by list comprehension

list1 = [x\*\*2 for x in range(1,101) if x\*\*2 % 3 == 0]

print(list1)

#########################################################

# solution 2: to generate the list by a for loop

list2 = [] # create an empty list first

for num in range(1,101):

if num \*\* 2 % 3 == 0:

list2.append(num \*\* 2) # add the number satisfying the conidtion into the list

print(list2)