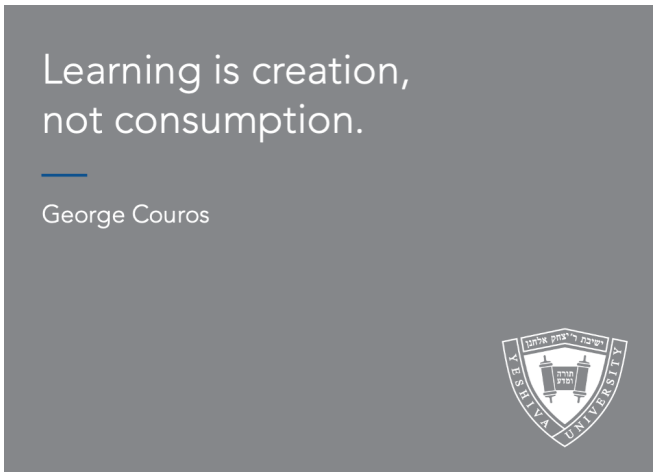
M8 AIM 5001 Midterm Exam

Due Mar 12, 2021 at 4pm **Points** 100 **Questions** 7
Available Mar 11, 2021 at 4pm - Mar 12, 2021 at 4pm 1 day **Time Limit** None

Instructions

Please answer the questions below providing the appropriate commentary or code. Do not consult with your fellow classmates.

This exam can only be taken once and must be submitted prior to March 12 4pm deadline. Please review the grading rubric.



This quiz was locked Mar 12, 2021 at 4pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	208 minutes	100 out of 100

ⓘ Correct answers are hidden.

Score for this quiz: **100** out of 100
Submitted Mar 12, 2021 at 3:34pm
This attempt took 208 minutes.

Question 1 — 10 / 10 pts

Using the Chinook database schema, write a query that shows the number of unique TrackId's associated within each customer's set of Chinook invoices.

Your Answer:

```
select "Cu"."CustomerId",count("Tr"."TrackId") from
(("Customer" as "Cu" left join "Invoice" as "In"
on "Cu"."CustomerId" = "In"."CustomerId") left join "InvoiceLine" as "Il"
on "In"."InvoiceId" = "Il"."InvoiceId") left join "Track" as "Tr"
on "Il"."TrackId" = "Tr"."TrackId" group by "Cu"."CustomerId"
order by "Cu"."CustomerId";
```

Question 2

10 / 10 pts

Using the Chinook database schema, write a query that shows the names of the distinct genres from which each customer has purchased individual Tracks.

HINT: No genre name should appear within any customer's list of genres more than once.

Your Answer:

```
select "Cu"."CustomerId", "Ge"."Name" from
((("Customer" as "Cu" left join "Invoice" as "In"
on "Cu"."CustomerId" = "In"."CustomerId") left join "InvoiceLine" as "Il"
on "In"."InvoiceId" = "Il"."InvoiceId") left join "Track" as "Tr"
on "Il"."TrackId" = "Tr"."TrackId" ) left join "Genre" as "Ge"
on "Tr"."GenreId" = "Ge"."GenreId"
group by "Cu"."CustomerId", "Ge"."Name"
order by "Cu"."CustomerId";
```

Question 3

20 / 20 pts

Suppose you work for a governmental agency responsible for keeping track of all individuals who have ever resided within your geographic area. Your agency keeps track of the following data on every such individual:

- Resident Registration Number (a unique ID) (can NEVER be edited)
- Name (Can be edited, e.g., due to marriage or divorce or legal change of name)
- Date of Birth (fixed: cannot be edited)
- Country of Birth (fixed: cannot be edited)
- Town of Birth (fixed: cannot be edited)
- Start date of residency (fixed: cannot be edited)
- Addresses Lived At (a list of addresses within the geographic area where an individual has resided, sorted in reverse chronological order (i.e., the most recent address appears first in the list and the oldest address appears last). Since addresses can change over time, this list is considered to be “dynamic” and may need to be updated at any time).

The agency has historically been storing this information on index cards but now intends to move all data to a Python-based system, and they have sufficient cloud-based computing resources to manage all such data within the context of a Python data structure on an ongoing basis rather than rely on the use of some sort of database system.

You have been instructed to design the required Python data structure, and you are told you **MUST** limit your design to the use of Python lists, tuples, sets, and dictionaries (or some combination thereof) for purposes of storing (and, when necessary, updating) the specified data. Your design must prohibit the editing/updating of any data item that is labeled as “cannot be edited”.

Describe your proposed data structure and explain your justification for your design.

Your Answer:

Resident Registration Number (a unique ID) (can NEVER be edited)

> I will use Tuple to store this data. If I need to add a new registration number, I will change it to set. After adding a new record, I will use

len() function to calculate the length of the set. If the len() result isn't changed, it means that we insert repeat data and it violent the rule of "unique ID'. Else, it, means the data is unique and can be inserted. Finally, I will change it back to "Tuple" because it can prevent the data from changing and suit the rule "can NEVER be edited". Changing to set() can only happen on inserting the new data and close for all other operatives.

Name (Can be edited, e.g., due to marriage or divorce or legal change of name)

For name, I will use dict+list to store it. The Registration Number is as the key and the list of names is as the value. If we need to change, I need to check the Registration Number is exists in the above Tuple firstly. Then I will append or remove the name from the list in the dict after we use the Registration Number as a key to get the value.eg, {'Resident Registration Number': [name1, name2....]}

Date of Birth (fixed: cannot be edited), Country of Birth (fixed: cannot be edited), Town of Birth (fixed: cannot be edited), the Start date of residency (fixed: cannot be edited)

For these four data, I will use dict+tuple to store them. As mention above, I will use the Resident Registration Number as the key and store these four types of data into Tuple which is as the value. eg: dict_resident = {'Resident Registration Number':(Date of Birth, Country of Birth, Town of Birth, Start date of residency)}. If we have the Resident Registration Number, we can get these four types of data and they can be changed.

Addresses Lived At.

For this type, I will use dict+list to store it. We also use Resident Registration Number as the key and Addresses Lived as the value. Before we add the list to dict, I will use the reverse() of list's function. If we need to add a new address to the list, we should use the reverse() again before we add the data into it. eg,{'Resident Registration Number':[address4, addree3, addreee2, address1]}

Finally, we put these four data into a big list named resident_data for Question 4. Their position in this list is like the order above.

Question 4

20 / 20 pts

Given the data structure you defined for the previous question, construct a user-defined function that can be used to update the content of the “Addresses Lived At” attribute for a given “Name” value. Remember, the “Addresses Lived At” attribute is a list of addresses, and its reverse-chronological order must be maintained when the updated address value is added to it. Your function MUST accept the data structure, the resident’s name, and their newly updated address as input parameters and must return the updated data structure upon exit, e.g.:

```
def your_function(resident_data, resident_name, new_address):
/* insert your logic here */

return (resident_data)
```

Provide the Python code for your function within the text box shown below. Make sure you include a doc string explaining the functionality of your function and be sure to include clear, concise explanatory comments within your Python code.

Your Answer:

```
def your_function(resident_data, resident_name, new_address):
    """This function need three parameters.
resident_data is a list including
Resident Registration Number (Tuple),
Name (dict),
(Date of Birth, Country of Birth , Town of Birth , the Start date of residency )(dict),
Addresses Lived(dict)

resident_name is a string

new_address
-------------------------------------------
This function will insert new_address into the Addresses Lived(dict) of resident_data
whose Resident Registration Number in Name's name is including resident_name.
```

Finally, this function will return resident_data including the new address. But is the data isn't exist, function will return NULL and print "This resident isn't exist"
rather than insert the new name into Name list and update the Addresses because it also require the Resident Registration Number.
"""

```
    Name = resident_data[1]#Name is a dict like {'Resident Registration Number': [name1, name2....]}
    Addresses = resident_data[3] #addredd is dict like {'Resident Registration Number':[address4, addree3, addreee2, address1]}
    key = ''
    for k,v in name: # find the key whose name is resident_name
        for i in v:
            if i == 'resident_name':
                key = k
    if key == '':#if the name can't find, we need to return NULL instead of update the data.
        print("This resident isn't exist")
        return
    oldaddress = Addresses[key]#oldaddress is a list which is reverse.
    oldaddress.reverse()# get the normal oeder
    oldaddress.append(new_address)
    oldaddress.reverse()#get the reverse
    Addresses[key] = oldaddress
    return (resident_data)
```

Question 5

20 / 20 pts

Suppose you are given the following block of text data:

“Server email list: Please review.

Jay Harmishton jay.harmishton@yu.DAV.edu **(mailto:jay.harmishton@yu.DAV.edu)**

Adele Chintsworth adele.chintsworth@yu.AIM.edu **(mailto:adele.chintsworth@yu.AIM.edu)**

Renfrew Swill renfrew.swll@yu.QE.edu **(mailto:renfrew.swll@yu.QE.edu)**

Kanafra Soapdi kanafra.soapdi@yu.MAT.edu **(mailto:kanafra.soapdi@yu.MAT.edu)**

Xiaoli Kzhang xiaoli.kzhang@yu.CYB.edu **(mailto:xiaoli.kzhang@yu.CYB.edu)**

END OF SERVER EMAIL LIST”

Write a regular expression that will extract **only** the email addresses contained within the nine lines of text.

Your Answer:

'[A-Za-z]+\.+[A-Za-z]+\@[A-Za-z]+\.[A-Za-z]+\.[A-Za-z]{3}'

Question 6

10 / 10 pts

You are given the following two arrays:

a = np.array([27, 5, 12, 5, 7, 15, 3, 1, 8, 0, 17])

b = np.array([11, 0, 14, 6, 3, 11, 19, 12, 4, 5])

Write a Python/Numpy code block that creates an array of **distinct** data values containing only those items that **DO NOT** appear within **both** of the arrays.

- The array your code generates should not contain any repeated occurrences of any of the data values you extract from the two arrays.

Provide the Python code within the text box shown below.

Your Answer:

```
c = np.unique(np.setdiff1d(a, b))
d = np.unique(np.setdiff1d(b, a))
e = np.unique(np.hstack((c, d)))
print(e)
```

Question 7

10 / 10 pts

Write a Python/Numpy code block that creates the following array using knowledge you have of Python's / NumPy's sequencing functionality so that you do not need to explicitly key in every integer value.

15	10	5
14	9	4
13	8	3
12	7	2
11	6	1

Provide the Python code within the text box shown below.

Your Answer:

```
import numpy as np

array = np.arange(1, 16)
arrayb = array[::-1].reshape(3, 5).T
print(arrayb)
```

Quiz Score: **100** out of 100