ITSE 1359 – Lab 9 Assignment (Refer to Ch. 9 as needed):

NOTE: Lab 9 requires you to work on and submit two solutions (3 files total). Read this document carefully and comply with submission and grading criteria. Your first problem (Student Information) is more prescriptive (use video provided), while the second program (Guess Number) is more open ended (solve as specified).

Program 1 - Student Information:

Submit 2 files: student_class.py and student_client.py

Problem Definition: Create a Python program that utilizes basic OOP concepts to create two student instances from a student class. The purpose of this application is to determine the type of student (level of education) based on student age.

This application must consist of two files. The student_class.py file will contain the Student class and the student_client.py file which will contain the student instances of the Student class.

Coding: After student data is printed, data is mutated (age), and printed again. During the mutation attempt of age, data validation will not allow for negative or zero updates.

Output: Screenshot of output is provided to match up against, as well as pseudocode and code examples.

Description of student_class.py:

First create a file called student_class.py and build the Student class in it. This class must utilize the __init__ method which initializes name and age attributes for a student.

In this class definition for a student, in addition to the _init__ method, you should also have FOUR special methods:

1.) Create a setter method for age that will validate age by performing a test to accept updates only if age > 0, else do not allow for update and provide feedback either way.

2.) Create a setter method for name that will validate name by performing a test to accept updates only if not the empty string. Feedback provided either way.

3.) Create a method that describes a student. This method should print the name, age, and type of student (level of education).

4.) Create a method that returns the “type of student” based on educational level. See comments for logic of this method.

Description of student_client.py:

Then create your driver program that will use the Student class in the student_class.py file. In your student_client.py file, after the import statement giving you access to the Student class, build the logic that will instantiate two students, show the data of these two students, attempt to mutate age to test data validation scheme, and then show updates. Refer to screen shot to match up against. Use the test data below and the mutation directives to match up with program.

Both files described above should be in the same folder and when you run you should be in student_client.py. Build student_class.py first and then student_client.py.

Test Data used in student_client to creates instances of Student class:

Student #	Name	Age
1	bob benavides	65
2	melissa gonzalez	17

Note: You can use different names and ages when initializing objects so long as age does not violate validation scheme. Use lower case for names because title case will be applied when object created. Use the mutation directives below to match up with screen shot of program.

Mutations:

Change Student 1's age to -10. (This will test data validation.)
Change Student 2's age to 13.
Change Student 2's name directly without using the setter method.

Special Topic - The __init__ Method:

The __init__ method is a special method that initializes objects of your class. It is like the "constructor" in Java and other C based languages. You must use two underlines before and after the word **init**.

The **self** key word must be the first parameter in the __init__ method and receives a reference to the object that gets created. The self key word is sort of like "This" in Java.

Special Topic - Public Data and Python:

This will make sense to those of you who know C++ or Java, if not it will make sense later as you learn more about programming languages

In Python, there is no private or protected. It's all public. So really no need for setter method since you can access the attributes directly. However, you can simulate private attributes. Later in your studies, you will learn about **name mangling** which will give you limited support for private variables, etc. Bet you can't wait to learn that!

Screenshot of Output (Student Information):

```
Report - Student Information:

Name:   Bob Benavides
Age:    65
Type:   College

Name:   Melissa Gonzalez
Age:    17
Type:   High School

*** Attempt to change Bob Benavides's age to -10.
>>> Age for Bob Benavides can not be <= 0.  NO update made.

*** Attempt to change Melissa Gonzalez's age to 13.
>>> Age was updated for Melissa Gonzalez.

Name:   Bob Benavides
Age:    65
Type:   College

Name:   melissa cantu
Age:    13
Type:   Middle School

[Finished in 0.2s]
```

- Note that student 2 is shown at bottom in lower case because it was updated directly rather than by using a setter method.

Specific Comments – Student class (student_class.py):

Class definition for the student class::

__init__ method initializes student – receives name, age

setter method for age

    # data validation logic: if age > 0, update allowed, else not

setter method for name

    # data validation logic: if name != empty string allow update, else not

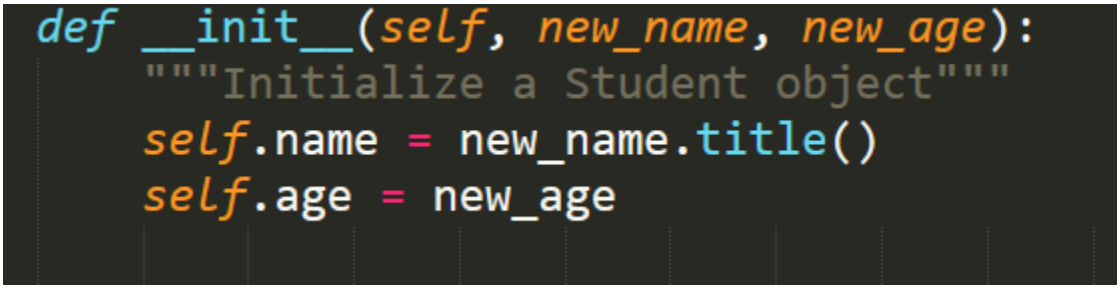
method that determines type of student based on age

    # if age >= 0 and age <=4, then type = Preschool
    # else if age == 5, then type = Kindergarten
    # else if age >= 6 and age <= 10, then type = Elementary School
    # else if age >= 11 and age <= 13, then type = Middle School
    # else if age >= 14 and age <= 17, then type = High School
    # else if age >= 18, they type = College
    # else type = “”

method that describes student – prints name, age, type of student

Code Examples (student_class.py):

How do you create the initializer method?



How do you provide for data validation of the age attribute?

```
def set_age(self, new_age):
    """validate - allow update of age if > 0. Provide feedback eitherway."""
    if new_age > 0:
        self.age = new_age
        print(f"\n*** Attempt to change {self.name}'s age to {new_age}.")
        print(f">>> Age was updated for {self.name}.")
    else:
        print(f"\n*** Attempt to change {self.name}'s age to {new_age}.")
        print(f">>> Age for {self.name} can not be <= 0.  NO update made.")
```

How do you provide for data validation of the name attribute?

```
def set_name(self, new_name):
    """Validate - allow update to name if update is not empty string"""
    if new_name != "":
        print(f"\n*** Attempt to change name for {self.name}.")
        self.name = new_name.title()
        print(f">>> Name was updated to {self.name}.")
    else:
        print(f"\n*** Attempt to change {self.name} to empty string.")
        print(f">>> Name for {self.name} was not changed.")
```

How do you build method that determine type of student (education level)?

```
def determine_type(self):
    """ return the probably level of schooling given age """
    if self.age >= 0 and self.age <= 4:
        type = "Preschool"
    elif self.age == 5:
        type = "Kindergarden"
    elif self.age >= 6 and self.age <= 10:
        type = "Elementary School"
    elif self.age >= 11 and self.age <= 13:
        type = "Middle School"
    elif self.age >= 14 and self.age <= 17:
        type = "High School"
    elif self.age >= 18:
        type = "College"
    else:
        type = ""

    return type
```

How to build function that describes student?

```
def describe_student(self):
    """display a summary of student information"""
    print(f"\nName: \t{self.name}")
    print(f"Age: \t{self.age}")
    print(f"Type: \t{self.determine_type()}")
```

Specific Comments – Student Client (student_client.py):

Client Program (contains instances of the Student class):

import Student class

print program title – Student Info

instantiate two students - pass name and age

call method to print student information for student 1 object
call method to print student information for student 2 object

mutate age of student1 by passing -10 (test data) using setter method
mutate age of student2 by passing 13 (test data) using setter method
mutate name for student2 (pass “melissa cantu” – lowercase) directly

call method to print student information for student 1 object
call method to print student information for student 2 object

Code Examples (student_client.py):

How do you instantiate an instance of the Student class?

```
s1 = Student("bob benavides", 65)
```

- Use same syntax to instantiate student 2.

How do you call method to print student information?

```
s1.describe_student()
```

- Use same syntax to describe student 2.

How to update age with call to setter method?

```
s1.set_age(-10)
```

- Use same syntax to set age to 13 for student 2.

How do you update (mutate) name – DIRECTLY?

```
#update name of student 2 directly without using the setter method
s2.name = "melissa cantu"
#s2.set_name("")
#s2.set_name("melissa cantu")
```

- The first line updates name for student 2 directly without going through a setter method.
- The second line below the first which is commented out, uses a setter method to set name to an empty string.
- The third line below the second, also commented out, uses a setter method to pass a name change.
- The first line is the only one you need to turn in for this lab, but you should try the other two to see how they differ. Can you tell how they differ?

Program 2 – Guess Number (guess_number.py):

Problem Statement: Code the first program the author of your book did when he was a kid. See the Introduction chapter for elaboration. You will have to generate a random number and match his interface. Save your file as: guess_number.py. Don't forget general and specific comments.

Coding: I am making two minor changes to his program. First, you should ask for a number between 1 and 10. Second, to make testing easier, print the random number to the screen before initial prompt.

Resources: Refer to chapter 7 for a review of user input and the while statement. Refer to chapter 9 for a review of the random module (p. 180). Also, our book's online website has extra material on the random module you may want to read: [Random Functions - Python Crash Course, 2nd Edition (ehmatthes.github.io)]

Output: Here is the screen shot from the introduction chapter:

```
I'm thinking of a number! Try to guess the number I'm thinking of: 25
Too low! Guess again: 50
Too high! Guess again: 42
That's it! Would you like to play again? (yes/no) no
Thanks for playing!
```

My version: (see random number printed at top to help testing)

```
rand # is 4
I'm thinking of a number between 1 and 10! Try to guess the number I'm thinking of: 3
Too low! Guess again: 5
Too high! Guess again: 4
That's it! Would you like to play again? (yes/no) yes

rand # is 2
I'm thinking of a number between 1 and 10! Try to guess the number I'm thinking of: 1
Too low! Guess again: 3
Too high! Guess again: 2
That's it! Would you like to play again? (yes/no) no

Thanks for playing!

PS C:\Users\Presentations>
```

Specific Comments / Pseudocode (Guess Number):

Note: There are many ways of doing this. This is my version. You are free to develop your own as long as you satisfy the problem definition and your program runs like my output example.

```
# import python random module
# set repeat flag to True
# set guessing flag to True
# while repeat
    # generate random number from 1 to 10 and assign to variable
    # print out random number for debugging purposes
    # input a guess number from the user
    # convert guess to int
    # while guessing
        # if guess == random number
            # prompt to play again
            # if again == no, set repeat flag to false and guessing flag to false
            # else if again == yes, print blank line and break
        # else if guess > random number say too high guess again
        # else if guess < random number say too low guess again
# print thanks for playing!
```

Example code (Guess Number):

```
while guessing:
    if guess_num == rand_num:
        again = input("That's it! Would you like to play again? (yes/no) ")
        if again == 'no':
            repeat = False
            guessing = False
        elif again == 'yes':
            print("")
            break
    elif guess_num > rand_num:
        guess_num = int(input("Too high! Guess again: "))
    elif guess_num <rand_num:
        guess_num = int(input("Too low! Guess again: "))
```

Comment: To print a blank line all you need is the print statement and you do not have to pass the empty string. So **print()** is sufficient.

Discussion of Break Statement:

Link: How To Use Break Statement In Python - Python Pool

You are free to use the break statement if you like, but there are some programmers that prefer not to use the break statement. You should be able to accomplish the same effect with other coding structures. For example, in the guess number program, instead of using the break statement, you can do the following:

1.) Remove the guessing flag that appears before the while repeat loop and place it before the while guessing embedded loop:

```
# set guessing flag to True
guessing = True

# while guessing
while guessing:
```

2.) Remove the break statement from the embedded if testing for again and replace it with the guessing flag:

```
if again == 'no':
    repeat = False
    guessing = False
elif again == 'yes':
    print()
    # break
    guessing = False
```

Comment: If you do make changes to your program, make sure you test your program thoroughly to make sure it's working right.

Link: Bad practice to use break in a for loop? - Stack Overflow. Some programmers disagree on whether you should use a break statement. For example, one said: "Professional Code" that is riddled with breaks doesn't really sound like professional code to me. It sounds like lazy coding ;)

Submit your lab assignment:

VIP: Since this lab assignment involves multiple files, create a folder called "lab9_username", where username is your college username. Put your three files this folder (student_class.py, student_client.py, guess_number.py). Zip your folder. Using the Canvas assignment tool, upload your zipped file to the lab 9 assignment. After you have uploaded your zipped file, submit your lab assignment.

See last page of this document for directions on how to create a folder and zip a folder. If you do not submit right, you may get points off or no grade at all.

All lab assignments must be submitted using the CANVAS assignments tool. Lab Assignments will not be accepted any other way**.** Make sure you submit your work to the right lab assignment number otherwise you will not get credit.

Grading Criteria:

- ✓ You must follow the Student Information video.
- ✓ Don't forget general comments.
- ✓ Specific comments are optional.
- ✓ Use white space to make your code easy to read.
- ✓ Comply with PEP-8 conventions for variable names, file names, etc.
- ✓ Satisfy the problem definition and other grading standards.
- ✓ Your work should not have syntax errors.
- ✓ Your work must be your own.
- ✓ Match your output screen to screenshot provided.
- ✓ Any deviations from lab specifications will result in points off.
- ✓ If you want to vary, do so on your own.

VIP Videos:

Student Information Lab (step-by-step): Lab 9 YouTube Video

Lectures and other videos: VIEW MY PYTHON PLAYLIST

Closing:

If you have questions about this lab send me a message using canvas inbox or attend zoom office hours. See Unit 0 for link for my office hours. Programming tutors are also available and listed in your Canvas class as announcements.

By-the-way, a great way to get ready for your lab assignments (and exams) is do the Try It Yourself problems in your book. Most of the solutions are on the authors website and I also discuss them in my YouTube video lectures.

Another way to get ready for your lab assignments is to review my lecture notes for each chapter. In Unit 0, find the link for my chapter notes.

Warning: Labs will increase in complexity with each lab assignment. A beginner may need 5 to 15 hours to complete each lab. Please start early because there will be no extensions.

Consider doing this lab over a three-day period:

- ✓ Day 1 – Get confused and run out of time.
- ✓ Day 2 – Research and debug errors.
- ✓ Day 3 – Polish, double-check everything, and submit.

Have an exception free day!

Prof. Benavides

Resources – Creating a Folder and zipping it:

Win 10 - How do I create a folder?

1.) Right-mouse click on the Desktop.
2.) Point to “New” and then click on “Folder”.
3.) Type in the name of your folder and press Enter.

Win 10 - How do I zip a folder? (Do it this way please.)

1.) Right-mouse click on the folder.
2.) Point to “Send to” and then click “Compressed (zipped”) folder”.
3.) Now you see zipped file by same name as folder with a zipper on it.
4.) Do not send a .rar file or any other file type other than .zip.

macOS - How do I create a folder?

1.) Right-mouse click on the Desktop.
2.) Select “New Folder”.
3.) Type in the name of your folder and press Enter.

macOS - How do I zip a folder? (Do it this way please.)

1.) Right-mouse click on the folder.
2.) Select “Compress”. Name of folder appears after word Compress.
3.) Now you see zipped file by same name as folder with a zipper on it.
4.) Do not send a .rar file or any other file type other than .zip.