

Severance Chapter 14 Coding Assignment

General Instructions

My expectations for your work on coding assignment exercises will grow as we progress through the course. In addition to applying any new programming techniques that have been covered in the current chapter, I will be expecting you to follow all of the good programming practices that we have adopted in the preceding weeks. Here is a quick summary of good practices that we have covered so far:

- Include a Python Docstring that describes the intent of the program.
- Place your highest-level code in a function named *main*.
- Include a final line of code in the program that executes the *main* function.
- Follow all PEP-8 Python coding style guidelines enforced by the PyCharm Editor. For example, place two blank lines between the code making up a function and the code surrounding that function.
- Choose names for your variables that are properly descriptive.
- Define `CONSTANT_VALUES` and use them in place of *magic numbers*.
- Always use f-strings for string interpolation and number formatting.
- When processing items from Python lists and tuples, unpack the values into variables with meaningful variable names to avoid using indexed expressions in your code.
- Close all files before the conclusion of the program.
- Remember that your program should behave reasonably when it is not given any input. This might be the result of the user pressing enter at a console prompt. Or, it might be the result of the user providing an input file that is empty.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Make sure that your test input/output matches the sample provided.
- Create a sub-directory named *data* within your PyCharm project to hold data files.
- Remember to submit all data files with your PyCharm project – including the files that were provided as starter files to this assignment.
- All functions that are not *main()* should have descriptive, action-oriented names.
- All functions should be of reasonable size.
- All functions should have high *cohesion*, and low *coupling*.
- Remember to test your program thoroughly before submitting your work.
- Your code must pass all relevant test cases. Make sure that it passes tests at the boundaries created by *if*, *else*, and *elif* conditions in your program (boundary value tests).
- Use of the *break* statement is allowed but not encouraged.
- Use of the *continue* statement is forbidden.
- Regular expression patterns should be expressed as Python *raw strings*
- Your finished code must be refactored to meet all good program design practices covered in this course.

- Custom Python classes should be created using Python Dataclasses and follow all practices demonstrated in our course.

Important Note: Do NOT use the Python *numpy* or *pandas* packages

It is possible that you are already aware of the Python *numpy* and *pandas* packages. In this course, we do NOT cover either of these packages. Instead, we are exploring how to create program solutions without them. So, please do NOT include *numpy* or *pandas* in your solutions to these exercises. When grading this assignment, we will be making point deductions for solutions that use either of these packages.

Exercise 1 (Regular)

Create a program named *my_land_mammals.py*. It should be modeled after the program that I demonstrated in *Beyond the Textbook* lecture for this chapter (*my_states.py*). Your program should be different in the following respects:

1. Your program will implement the *LandMammal* class that holds data facts regarding the world's largest land mammals.
2. The *LandMammal* class should implement the following instance variables:
 - a. `name` (str)
 - b. `minimum_mass_in_pounds` (int)
 - c. `maximum_mass_in_pounds` (int)
3. You will also need to implement the following method:
 - a. `calculate_range_of_mass_in_pounds()` returns the maximum value minus the minimum value as an int.
4. Unit testing code should be placed in the *main()* function and should follow the approach demonstrated in the *Beyond the Textbook* lecture.

When running the unit tests, you should expect the following output on your console:

```
Unit testing output follows...
```

```
Test Case: Test constructor
```

```
Passed
```

```
Passed
```

```
Passed
```

```
Test Case: Test calculate_range_of_mass_in_pounds method
```

```
Passed
```

Exercise 2 (Regular)

Create a program named *create_land_mammal_mass_reports.py*. It should be modeled after the program that I demonstrated in *Beyond the Textbook* lecture for this chapter (*create_state_area_reports.py*). Your program should be different in the following respects:

1. Your program will create a report of LandMammal data facts in two different sort orders:
 - a. By Land Mammal Name
 - b. By Descending Range of Mass in Pounds
2. Your program should give expected results when run with the following input files provided as starter files:
 - a. *empty_file.txt*
 - b. *land_mammals.txt*
3. The importing of the *my_land_mammals.py* module into your program should NOT cause the unit test code in that program to be executed.

When running a test with the empty input file, you should expect the following input/output on your console:

```
Please enter input file name: empty_file.txt
```

BY LAND MAMMAL NAME

Land Mammal Name	Minimum Mass in Pounds	Maximum Mass in Pounds	Range of Mass in Pounds
---------------------	---------------------------	---------------------------	----------------------------

BY DESCENDING RANGE OF MASS IN POUNDS

Land Mammal Name	Minimum Mass in Pounds	Maximum Mass in Pounds	Range of Mass in Pounds
---------------------	---------------------------	---------------------------	----------------------------

When running a test with the populated input file, you should expect the following input/output on your console:

Please enter input file name: land_mammals.txt

BY LAND MAMMAL NAME

Land Mammal Name	Minimum Mass in Pounds	Maximum Mass in Pounds	Range of Mass in Pounds
African elephant	10,000	24,000	14,000
American bison	700	2,200	1,500
Asian elephant	8,000	17,640	9,640
Black rhinoceros	1,500	4,000	2,500
Cape buffalo	1,100	2,200	1,100
Gaur	1,000	3,000	2,000
Giraffe	1,544	4,255	2,711
Hippopotamus	2,500	8,820	6,320
Water buffalo	660	2,200	1,540
White rhinoceros	3,000	9,920	6,920

BY DESCENDING RANGE OF MASS IN POUNDS

Land Mammal Name	Minimum Mass in Pounds	Maximum Mass in Pounds	Range of Mass in Pounds
African elephant	10,000	24,000	14,000
Asian elephant	8,000	17,640	9,640
White rhinoceros	3,000	9,920	6,920
Hippopotamus	2,500	8,820	6,320
Giraffe	1,544	4,255	2,711
Black rhinoceros	1,500	4,000	2,500
Gaur	1,000	3,000	2,000
Water buffalo	660	2,200	1,540
American bison	700	2,200	1,500
Cape buffalo	1,100	2,200	1,100

Exercise 3 (Regular)

Create a program named *my_vehicles.py*. It should be a new version of the program that I demonstrated in the *Beyond the Textbook* lecture for this chapter (*my_vehicles_starter.py*). Start by copying the program from the starter files for this assignment and renaming it.

Your program should be different in the following respects:

1. In addition to the *Car* and *Truck* subclasses, your program will also implement the *Motorcycle* subclass.
2. The *Motorcycle* subclass will provide the following distinguishing instance variable:
 - a. *displacement_in_ccs* (int)
3. The *Motorcycle* subclass will provide an implementation for the following method:
 - a. *determine_annual_registration_fee()* returns float.

If *displacement_in_ccs* is less than 1,000, then the annual fee is 75.00.
Otherwise, the annual fee is 150.00.

4. The *Motorcycle* subclass should provide an implementation for the following method:
 - a. *get_invoice_content()* returns a tuple that includes the following:
 - i. A string containing the title of the invoice
 - ii. A list of strings that contain the name and address for the invoice
 - iii. A list of tuples that contain the detail for the invoiceRemember to use the implementation of this method in the other subclasses as a guide when writing the code for this subclass.
5. Unit testing code should be placed in the *main()* function and should follow the approach demonstrated in the *Beyond the Textbook* lecture.

When running the unit tests, you should expect the following output on your console:

Unit testing output follows...

Test Case: Test Vehicle constructor

Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Vehicle get_invoice_content method

Passed
Passed
Passed

Test Case: Test Car constructor

Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Car get_invoice_content method

Passed
Passed
Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Electric

Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Hybrid
Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Fossil
Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Plutonium
Passed

Test Case: Test Truck constructor
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Truck get_invoice_content method
Passed
Passed
Passed

Test Case: Test Truck determine_annual_registration_fee method, gross_weight = 14000
Passed

Test Case: Test Truck determine_annual_registration_fee method, gross_weight = 14001
Passed

Test Case: Test Motorcycle constructor
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Passed
Passed
Passed
Passed
Passed

Test Case: Test Motorcycle get_invoice_content method
Passed
Passed
Passed

Test Case: Test Motorcycle determine_annual_registration_fee,
displacement_in_ccs = 999
Passed

Test Case: Test Motorcycle determine_annual_registration_fee,
displacement_in_ccs = 1000
Passed

Exercise 4 (Regular)

Create a program named `create_vehicle_registration_invoices.py`. It should be a new version of the program that I demonstrated in the *Beyond the Textbook* lecture for this chapter (`create_vehicle_registration_invoices_starter.py`). Start by copying the program from the tutorial into your project and renaming it.

Your program should be different in the following respects:

1. In addition to creating registration invoices for instances of the *Car* and *Truck* subclasses, your program should also create registration invoices for the *Motorcycle* class.
2. Your program should give expected results when run with the following input files provided as starter files:
 - a. `empty_file.txt`
 - b. `car_truck_and_motorcycle_records.txt`
3. The importing of the `my_vehicles.py` module into your program should NOT cause the unit test code in that program to be executed.

When running a test with the empty input file, you should expect the following input/output on your console:

```
Please enter the input filename: empty_file.txt
```

```
-----
```

```
0 invoices have been printed.
```

When running a test with the populated input file, you should expect the following input/output on your console:

Please enter the input filename: car_truck_and_motorcycle_records.txt

CAR REGISTRATION RENEWAL INVOICE

Bella Baker
100 West End Street
Champaign, IL 62609

Make: Tesla
Model: Model 3
Year: 2022
Color: Blue
Vehicle ID: CAR4489679911
Fuel Type: Electric

AMOUNT DUE: \$ 100.00

CAR REGISTRATION RENEWAL INVOICE

John Howard
600 Pleasant Circle
Apt A
Champaign, IL 60577

Make: Toyota
Model: Camry
Year: 2021
Color: White
Vehicle ID: CAR1074521368
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

CAR REGISTRATION RENEWAL INVOICE

Faith Langdon

335 River Circle
Champaign, IL 61256

Make: Toyota
Model: Corolla
Year: 2021
Color: Red
Vehicle ID: CAR2927528306
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

TRUCK REGISTRATION RENEWAL INVOICE

Joshua Lewis
801 River Court
Apt B
Champaign, IL 62030

Make: Nissan
Model: Titan XD
Year: 2021
Color: Black
Vehicle ID: TRK6602773660
Gross Weight: 11,000

AMOUNT DUE: \$ 400.00

TRUCK REGISTRATION RENEWAL INVOICE

Sebastian Lewis
100 Potter Way
Champaign, IL 60143

Make: Ford
Model: Super Duty F-350
Year: 2021
Color: Grey
Vehicle ID: TRK3575913453
Gross Weight: 12,000

AMOUNT DUE: \$ 400.00

CAR REGISTRATION RENEWAL INVOICE

Carol Metcalfe
1000 Pleasant Court
Apt C
Champaign, IL 60883

Make: Nissan
Model: Altima
Year: 2021
Color: Grey
Vehicle ID: CAR8804836953
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

TRUCK REGISTRATION RENEWAL INVOICE

Michael North
1000 Main Way
Apt C
Champaign, IL 62220

Make: Ford
Model: Super Duty F-350
Year: 2021
Color: White
Vehicle ID: TRK5168323404
Gross Weight: 12,000

AMOUNT DUE: \$ 400.00

MOTORCYCLE REGISTRATION RENEWAL INVOICE

Dylan Paige
800 Center Blvd
Unit D
Champaign, IL 60214

Make: BMW

Model: R1250 GS
Year: 2021
Color: White
Vehicle ID: MCY8266162579
Displacement (CCs): 1,254

AMOUNT DUE: \$ 150.00

***** A Large Number of Invoices Have Been Omitted to Save Space *****

MOTORCYCLE REGISTRATION RENEWAL INVOICE

Dominic Mackay
750 Center Blvd
Waukegan, IL 62374

Make: Royal Enfield
Model: Meteor 350
Year: 2021
Color: Grey
Vehicle ID: MCY5807211506
Displacement (CCs): 349

AMOUNT DUE: \$ 75.00

CAR REGISTRATION RENEWAL INVOICE

Tracey Peake
555 High Court
Waukegan, IL 61926

Make: Nissan
Model: Altima
Year: 2021
Color: White
Vehicle ID: CAR2412599457
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

CAR REGISTRATION RENEWAL INVOICE

Wanda Underwood
702 Center Way
Waukegan, IL 61636

Make:	Honda
Model:	Civic
Year:	2022
Color:	White
Vehicle ID:	CAR2407296694
Fuel Type:	Fossil

AMOUNT DUE: \$ 300.00

86 invoices have been printed.

Exercise 5 (Challenge)

Please note that there are two parts to this exercise. Be sure to complete both parts.

Create a program named *my_vehicles_challenge.py*. It should be a new version of the program created in Exercise 3 (*my_vehicles.py*). Start by copying the program and renaming it.

Your program should be different in the following respects:

1. In addition to the *Car*, *Truck*, and *Motorcycle* subclasses, your program will also implement the *Snowmobile* subclass.

2. The *Snowmobile* subclass will NOT provide a distinguishing instance variable.

3. The *Snowmobile* subclass will provide an implementation for the following method:

b. *determine_annual_registration_fee()* returns float.

The annual fee is always 45.00.

7. The *Snowmobile* subclass should provide an implementation for the following method:
 - a. *get_invoice_content()* returns a tuple that includes the following:
 - i. A string containing the title of the invoice
 - ii. A list of strings that contain the name and address for the invoice
 - iii. A list of tuples that contain the detail for the invoiceRemember to use the implementation of this method in the other subclasses as a guide when writing the code for this subclass.
4. Unit testing code should be placed in the *main()* function and should follow the approach demonstrated in the tutorial.

When running the unit tests, you should expect the following output on your console:

Unit testing output follows...

Test Case: Test Vehicle constructor

Passed

Passed

Passed

Passed

Passed

Passed

Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Vehicle get_invoice_content method

Passed
Passed
Passed

Test Case: Test Car constructor

Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Car get_invoice_content method

Passed
Passed
Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Electric

Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Hybrid

Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Fossil

Passed

Test Case: Test Car determine_annual_registration_fee method, fuel_type = Plutonium

Passed

Test Case: Test Truck constructor

Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Truck get_invoice_content method

Passed
Passed
Passed

Test Case: Test Truck determine_annual_registration_fee method,
gross_weight = 14000

Passed

Test Case: Test Truck determine_annual_registration_fee method,
gross_weight = 14001

Passed

Test Case: Test Motorcycle constructor

Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed
Passed

Test Case: Test Motorcycle get_invoice_content method

Passed
Passed
Passed

Test Case: Test Motorcycle determine_annual_registration_fee,
displacement_in_ccs = 999

Passed

Test Case: Test Motorcycle determine_annual_registration_fee,
displacement_in_ccs = 1000

Passed

Test Case: Test Snowmobile constructor

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Passed

Test Case: Test Snowmobile get_invoice_content method

Passed

Passed

Passed

Test Case: Test Snowmobile determine_annual_registration_fee method

Passed

Create a program named `create_vehicle_registration_invoices_challenge.py`. It should be a new version of the program created in Exercise 4 (`create_vehicle_registration_invoices.py`). Start by copying the program and renaming it.

Your program should be different in the following respects:

1. In addition to creating registration invoices for instances of the *Car*, *Truck*, and *Motorcycle* subclasses, your program should also create registration invoices for the *Snowmobile* class.
2. Your program should give expected results when run with the following input files provided as starter files:
 - a. `empty_file.txt`
 - b. `car_truck_motorcycle_and_snowmobile_records.txt`
3. The importing of the `my_vehicles_challenge.py` module into your program should NOT cause the unit test code in that program to be executed.

When running a test with the empty input file, you should expect the following input/output on your console:

```
Please enter the input filename: empty_file.txt
```

```
-----
```

```
0 invoices have been printed.
```

When running a test with the populated input file, you should expect the following input/output on your console:

Please enter the input filename:
car_truck_motorcycle_and_snowmobile_records.txt

CAR REGISTRATION RENEWAL INVOICE

Bella Baker
100 West End Street
Champaign, IL 62609

Make: Tesla
Model: Model 3
Year: 2022
Color: Blue
Vehicle ID: CAR4489679911
Fuel Type: Electric

AMOUNT DUE: \$ 100.00

CAR REGISTRATION RENEWAL INVOICE

John Howard
600 Pleasant Circle
Apt A
Champaign, IL 60577

Make: Toyota
Model: Camry
Year: 2021
Color: White
Vehicle ID: CAR1074521368
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

SNOWMOBILE REGISTRATION RENEWAL INVOICE

Colin King
800 Brook Circle
Unit C
Champaign, IL 61461

Make: Yamaha
Model: Sidewinder L-TX GT
Year: 2022
Color: White
Vehicle ID: SNW2387865728

AMOUNT DUE: \$ 45.00

CAR REGISTRATION RENEWAL INVOICE

Faith Langdon
335 River Circle
Champaign, IL 61256

Make: Toyota
Model: Corolla
Year: 2021
Color: Red
Vehicle ID: CAR2927528306
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

TRUCK REGISTRATION RENEWAL INVOICE

Joshua Lewis
801 River Court
Apt B
Champaign, IL 62030

Make: Nissan
Model: Titan XD
Year: 2021
Color: Black
Vehicle ID: TRK6602773660
Gross Weight: 11,000

AMOUNT DUE: \$ 400.00

TRUCK REGISTRATION RENEWAL INVOICE

Sebastian Lewis
100 Potter Way
Champaign, IL 60143

Make: Ford
Model: Super Duty F-350
Year: 2021
Color: Grey
Vehicle ID: TRK3575913453
Gross Weight: 12,000

AMOUNT DUE: \$ 400.00

SNOWMOBILE REGISTRATION RENEWAL INVOICE

Boris Marshall
103 High Circle
Apt A
Champaign, IL 60700

Make: Ski-Doo
Model: Summit Edge 850 E-TEC 165
Year: 2022
Color: Grey
Vehicle ID: SNW6504064609

AMOUNT DUE: \$ 45.00

***** A Large Number of Invoices Have Been Omitted to Save Space *****

MOTORCYCLE REGISTRATION RENEWAL INVOICE

Oliver Cameron
555 Pleasant Circle
Waukegan, IL 61303

Make: Triumph
Model: Trident 660
Year: 2021
Color: Red
Vehicle ID: MCY1042465955
Displacement (CCs): 660

AMOUNT DUE: \$ 75.00

MOTORCYCLE REGISTRATION RENEWAL INVOICE

Dominic Mackay
750 Center Blvd
Waukegan, IL 62374

Make: Royal Enfield
Model: Meteor 350
Year: 2021
Color: Grey
Vehicle ID: MCY5807211506
Displacement (CCs): 349

AMOUNT DUE: \$ 75.00

SNOWMOBILE REGISTRATION RENEWAL INVOICE

John May
888 Main Blvd
Waukegan, IL 61261

Make: Arctic Cat
Model: ZR 9000 Thundercat
Year: 2021
Color: Blue
Vehicle ID: SNW9112403883

AMOUNT DUE: \$ 45.00

SNOWMOBILE REGISTRATION RENEWAL INVOICE

Richard Metcalfe
611 West End Street
Apt B
Waukegan, IL 60838

Make: Polaris
Model: Pro RMK Matryx Slash Patriot Boost 163
Year: 2021
Color: Black
Vehicle ID: SNW5667579989

AMOUNT DUE: \$ 45.00

CAR REGISTRATION RENEWAL INVOICE

Tracey Peake
555 High Court
Waukegan, IL 61926

Make: Nissan
Model: Altima
Year: 2021
Color: White
Vehicle ID: CAR2412599457
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

CAR REGISTRATION RENEWAL INVOICE

Wanda Underwood
702 Center Way
Waukegan, IL 61636

Make: Honda
Model: Civic
Year: 2022

Color: White
Vehicle ID: CAR2407296694
Fuel Type: Fossil

AMOUNT DUE: \$ 300.00

100 invoices have been printed.

Tools

Use PyCharm to create and test all Python programs.

Submission Method

Follow the process that I demonstrated in the tutorial video on submitting your work.

This involves:

- Locating the properly named directory associated with your project in the file system.
- Compressing that directory into a single .ZIP file using a utility program.
- Submitting the properly named zip file to the submission activity for this assignment.

File and Directory Naming

Please name your Python program files as instructed in each exercise. Please use the following naming scheme for naming your PyCharm project:

surname_givenname_exercises_severance_chapter_14

If this were my own project, I would name my PyCharm project as follows:

trainor_kevin_exercises_severance_chapter_14

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

surname_givenname_exercises_severance_chapter_14.zip

If this were my own project, I would name the zip file as follows:

trainor_kevin_exercises_severance_chapter_14.zip

Due By

Please submit this assignment by the date and time shown in the Weekly Schedule.

Last Revised

2023-03-22