


Assessment 2: Programming assignment 2 instructions and submission link

CSE4IP: Introduction to Programming

Assessment 2: Programming assignment 2

Assignment type	Assignment
Weighting	25%
Word count / length	1000 words (equivalent)
SILOs	1, 2, 4

 *For current Melbourne time, please check information under the Assessment tile in the LMS of this subject.

Topic overview

The purpose of this assessment is to put into practice the programming skills that you have learnt via the topic content and lab sessions. You will be given the opportunity to demonstrate your ability to analyse problems, apply basic programming constructs to design computational solutions, and implement executable code in the Python programming language.

Guidelines

This programming assignment focuses on content from Week 1 to Week 5. You are expected to use the knowledge learnt from those topics to design and implement working Python solutions to programming problems.

Instructions

This assessment is structured similarly to the labs, and as such the instructions are similar.

- Read each question thoroughly multiple times to ensure that you understand the requirements.
- Once you understand the question, code your solution in the associated code cell titled 'Your solution'.
- Test your solution multiple times, with a variety of inputs – not just those found in the example runs.
- Unless explicitly asked in the question, you are not required to validate user input (i.e. you can assume that the user will enter a valid input value).
- Keep in mind that partial attempts at solutions will likely still be worth some marks, so be sure to have a go at writing code for every task.

When you have satisfactorily attempted all problems, confirm that your solution works by resetting the notebook runtime and running your code again – instructions can be found at the bottom of this notebook.

The submission for this assignment is the .ipynb notebook file. You should export this file and upload it via the link in the LMS – instructions can be found on the assessment submission page. Do not submit your solutions in any other format as they will not be graded.

Programming tasks

This assessment consists of four programming tasks, with a total of 100 marks allocated between them according to their difficulty.

Take your time to read and re-read the instructions before attempting to write your solution – you can save a lot of trouble by simply having a clear understanding of the problem before touching any code.

Task 1 (20 marks)

For this task, you are to write code that manages the scores for a computer game and prints the leaderboard.



Instructions

The scoreboard program found below is currently missing two important pieces of functionality; it doesn't record scores, nor is it able to print the leaderboard.

Provided for you is the `ScoreEntry` class, which contains the player's name and score of each entry on the scoreboard, and the `Scoreboard` class, which is responsible for maintaining the list of score entries. Your task is to implement this missing functionality by adding two methods to the `Scoreboard` class as described below.

`add_score`

This method is to take a player's name and their score as arguments, create a new `ScoreEntry` object and then append it to the appropriate list of scores.

`print_leaderboard`

This method is to take no arguments and print the name and score of the top three players in descending order of score.

Hint: If you find your solution printing the bottom three scores, you may need to look at [the documentation](#), and in particular the 'reverse' named argument.

Requirements



Instructions

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- You **must** use the sort method (hint: the provided function `get_score` will come in handy).
- You **must** use list slicing as part of your solution. You **must not** use list indexing to retrieve individual list items.
- You **must not** use `break`, `continue` or `return` statements in `print_leaderboard`.

Example Runs

Run 1

```
Bob: 12103
Charlie: 8762
Alice: 7821
```

Your code should execute as closely as possible to the example runs above. To check for correctness, ensure that your program gives the same outputs as in the examples, as well as trying it with other inputs.

Your Solution

```
[ ] class ScoreEntry:

    def __init__(self,name,score):

        self.name = name

        self.score = score

    def get_score(socre_entry):

        return score_enentry.score

class Scoreboard:

    def __init_ (self):

        self.score = [ ]

    # Write your methods here

    scoreboard = Scoreboard ( )

    scoreboard.add_score('Alice', 7821)

    scoreboard.add_score('Bob', 12103)

    scoreboard.add_score('Charlie', 8762)

    scoreboard.add_score('Denise', 6753)

    Scoreboard.print_leaderboard ( )
```

Task 2 (20 marks)

For this task, you are to write a simple program to track time spent on miscellaneous chores.

Instructions



Instructions

Your program should be capable of keeping a sum of the total hours spent on any number of chores. You are to add the missing methods to the ChoreTracker class as described below.

add_hours

This method takes the name of a chore and the number of hours spent on this chore and adds it to the total hours for that chore. When this method is called, you will need to consider two cases:

1. No hours have been logged for the chore yet.
2. The chore already has some hours logged.

print_summary

This method takes no arguments and prints the name and total hours spent on each chore, as well as the total number of hours spent on **all** chores. Hours are to be displayed with two decimal places of precision.

Hint: if you don't remember how to iterate over the items in a dictionary, have a look at the relevant workbook.

Requirements

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- You **must** use f-strings to format the outputs (do not use string concatenation).
- You **must** ensure that the hours are printed with two decimal places of precision.
- The order in which the projects are printed is **not** important.
- You **must** use a single loop to both print individual chore hours and aggregate the total number of hours spent on all chores.

Example

Run 1

```
sweeping: 0.75 hours  
laundry: 1.50 hours  
working: 11.50 hours  
mopping: 0.50 hours  
TOTAL: 14.25 hours
```

Your code should execute as closely as possible to the example runs above. To check for correctness, ensure that your program gives the same outputs as in the examples, as well as trying it with other inputs.

Task 3 (30 marks)

For this task, you are to write code that handles the transfer of funds from one bank account to another. This task is divided into three parts to help with solving it.

After you have completed and tested each part, you are to copy and paste your code into the next part's code cell and continue. Thus, you are required to complete the parts in order. Please do not complete all parts in the Part A solution code cell.

Task 3, Part A (10 marks)

Instructions



Instructions

As this task relates to transferring funds between bank accounts, a BankAccount class has been provided, which supports depositing and withdrawing funds. A simple interactive loop has already been implemented, which allows the user to repeatedly transfer funds from one account to another.

You are to write a function called `transfer_funds` with three parameters: the amount to transfer, the account the funds are coming from and the account the funds are going to. It should use the appropriate instance methods on each of the accounts to update their balances as required.

Test your code thoroughly to ensure its correctness, as it will be used as the basis for Part B of this task. When you are happy with your solution, copy and paste the code **in its entirety** where indicated in Part B.

Requirements

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- The `transfer_funds` function **must not** directly access the balance instance variable of either bank account.

Example

Run 1

```
== Account balance ==
Alice : $100.00
Bob : $100.00
Enter transfer amount ($): 30
== Account balance ==
Alice : $70.00
Bob : $130.00
Perform another transfer? (y/n): y
Enter transfer amount ($): 15.55
== Account balance ==
Alice : $54.45
Bob : $145.55
Perform another transfer? (y/n): n
```

Your code should execute as closely as possible to the example runs above. To check for correctness, ensure that your program gives the same outputs as in the examples, as well as trying it with other inputs.

Your Solution

```
[ ] class BankAccount:

    def __init__(self,name,initial_balance):

        self.name = name

        self.balance = initial_balance

    def deposit (self, amount):

        self.balance+= self.balance = amount

    def withdraw(self, amount):

        # Part B: Raise an exception as appropriate

        self.balance = self.balance - amount

    def print balances (account a, account b):

        print('== Account balances ==')

        print(f' {account_a.name}: ${account_a.balance: .2f}')

        print(f' {account_b.name}: ${account_b.balance:.2f}')
```



Instructions

```
# Part A. Write your transfer funds function here

account a = BankAccount ("Alice", 100)

account_b = BankAccount (*Bob', 100)

print _balances (account a, account b)

another_transfer = 'y'

while another_transfer == 'y':

    amount = float (input("Enter transfer amount ($):"))

# Part C. Print an appropriate message if an exception is encountered

transfer_funds (amount, account_a, account_b)

print balances (account_a, account_b)

another_transfer = input ('Perform another transfer? (y/n):')
```

Task 3, Part B (10 marks)

Instructions

Before commencing this part of the task, ensure that you have completed Part A and copied your solution into the code cell below where indicated.

If you tested your solution thoroughly in the previous exercise, you probably came across a logic error in the program — there's nothing to stop us from transferring funds from an account, even if the account balance becomes negative! You are to fix this problem by raising an exception and preventing the account balance from becoming negative. You should do this by raising a `ValueError` in the appropriate place in the `BankAccount` class.

It would be good to also check that the amount being deducted is a positive value, but that's beyond the scope of this task (and hence your solution does not need to do this).

Hint: the order in which the withdrawal and deposit occur in `transfer_funds` matters — the wrong order will result in someone receiving free money!

Requirements



Instructions

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- You **must** raise an exception from within the BankAccount class if a withdrawal would cause an account balance to become negative.
- You **must not** modify the balance of *any* accounts involved in a transaction before the exception is raised.

Example Runs

Run 1

```
== Account balances ==
Alice: $100.00
Bob: $100.00
Enter transfer amount ($): 35.65
== Account balances ==
Alice: $64.35
Bob: $135.65
Perform another transfer? (y/n): y
Enter transfer amount ($): 22.90
== Account balances ==
Alice: $41.45
Bob: $158.55

Perform another transfer? (y/n): y
Enter transfer amount ($): 50

ValueError Traceback (most recent call last)
  ... stacktrace omitted ...
ValueError: Balance would be negative after withdrawal
```

Your code should execute as closely as possible to the example runs above. To check for correctness, ensure that your program gives the same outputs as in the examples, as well as trying it with other inputs.

Task 3, Part C (10 marks)

Instructions

Before commencing this part of the task, ensure that you have completed Part B and copied your solution into the code cell below where indicated.

At this point, the program prevents a transfer occurring if there aren't enough funds in the 'from' account. However, simply crashing a program isn't a very nice user experience. You are to modify your program so that it handles the ValueError and displays '<< Error transferring funds >>' to the user. The program should otherwise continue as normal, with the user being asked whether they would like to perform another transaction. Recall from Part B that a failed transaction should not result in either account balance changing.

Hint: a little rusty at exception handling? Take a look at the relevant notebook for examples.

Requirements



To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- You **must** specify the appropriate exception type to handle in the except block — marks will be deducted for failing to do so.
- The transfer_funds function call **must** be the only thing in your try block.
- A failed transaction **must not** result in either account balance changing.

Example Runs

Run 1

```
== Account balances ==
Alice: $100.00
Bob: $100.00
Enter transfer amount ($): 95
== Account balances ==
Alice: $5.00
Bob: $195.00

Perform another transfer? (y/n): y
Enter transfer amount ($): 7.50
<< Error transferring Funds >>
== Account balances ==
Alice: $5.00
Bob: $195.00
Perform another transfer? (y/n) : y
Enter transfer amount ($): 4.95
== Account balances ==
Alice: $ 0.05
Bob: $199. 95
Perform another transfer? (y/n): n
```

Your code should execute as closely as possible to the example runs above. To check for correctness, ensure that your program gives the same outputs as in the examples, as well as trying it with other inputs.

Task 4 (30 marks)

For this task, you are to graph some physical characteristics for three species of penguin using Pandas and Matplotlib. This task has been divided into three parts; complete the steps in order as usual.

Task 4, Part A (5 marks)

Instructions



Instructions

This task uses a dataset containing some physical measurements of three penguin species: Adelie, Chinstrap and Gentoo. You can view the dataset online [by clicking here](#). Run the following code cell to load the dataset into a Pandas dataframe and look at the first few rows. Follow the code before executing it.

```
import pandas as pd
URL=
'https://gist.github.com/anibali/c2abc8cab4a2f7b0a6518d11a67c693c/raw/3b1bb5264736bb762584104c9e7a828bef0f6ec8/penguins.csv'
df = pd.read_csv (URL)
print (df.head ( ))
```

	species	island	bill length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	36.7	19.3	103.0	3450.0	FEMALE
4	Adelie	Torgersen	39.3	20.6	190.0	3650.0	MALE

As you can see, the dataset consists of seven columns. To get an idea of how these values differ between species, in the next cell you will calculate the mean value of each column, when grouped by 'species' — this only takes a single line of code.

This should produce a table which contains the mean value of each numerical column for each of the three species. There is no need to print the result — Google Colab will automatically display the result (like an interactive interpreter session does). Take note of how the values vary between species; we'll visualise it in Part B.

Hint: you can refer to the relevant workbook for an example of per-group aggregation.

Requirements

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must also meet the following requirements:

- You **must** use Pandas aggregation on the df object.
- You **must** present your solution as a single line of code.

Task 4, Part B (15 marks)

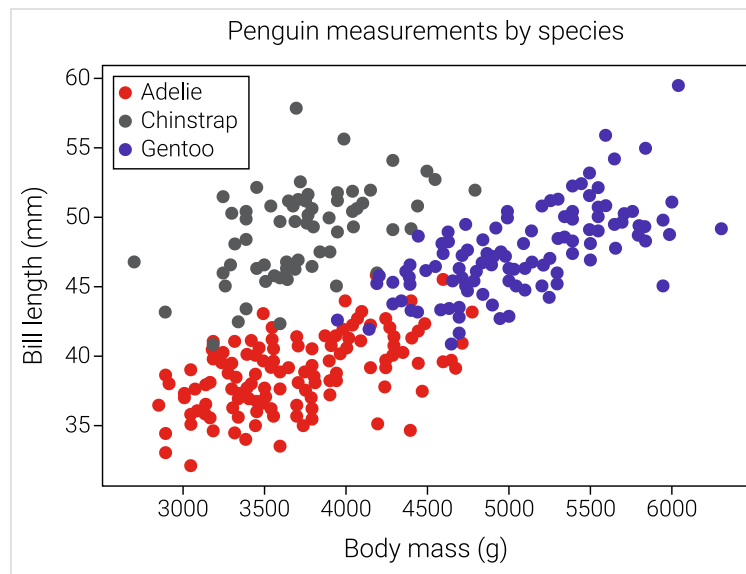
Instructions



Before commencing this part of the task, ensure that you have run the code in Part A, which loads the dataframe.

For this part, you are to produce a scatter plot which relates penguin mass and bill length. A successful solution should be identical to the example below, which clearly shows how these measurements are a good indicator of penguin species.

The image below shows what the plot created by your solution code should look like:



Hint: every step in this task has a corresponding example in the workbook.

Requirements

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- You **must** create a different dataframe for each species by *filtering* the data appropriately. (*Hint: you should end up with three new dataframes.*)
- You **must** plot each of these dataframes in the same graph, giving each data series an appropriate label.
- You **must** give the plot a title, axis labels and a legend.

Task 4, Part C (10 marks)

Instructions

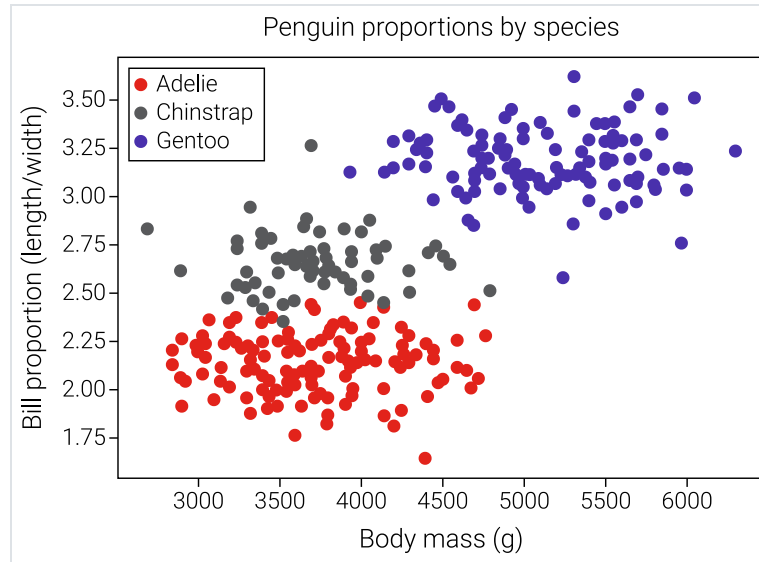


Instructions

The graph from Part B shows a nice separation between the species when plotting penguin weight and bill length. To improve upon this, you are to add another column to the dataset, and plot this instead of the penguin bill length.

In the next cell, write code which adds another column to the dataframe `df` called `bill_proportion`, which is the penguins' bill length divided by their bill depth. This new column represents the ratio between bill length and depth, and serves as an indicator of bill shape.

Copy and paste your code from Part B into the below cell, and add the new column — you can call `df.head()` to check. Then, update your graphing code to use this new column; don't forget to update the labels and title! A successful solution should be identical to the example below, which does an even better job of separating penguin species than Part B.



Although beyond the scope of this subject, we would now be able to guess a penguin's species using just their weight and bill proportion, by plotting it on this graph and seeing which other points it lands near!

Requirements

To achieve full marks for this task, you must follow the instructions above when writing your solution. Additionally, your solution must adhere to the following requirements:

- The new column **must** be created using Pandas.

As your solution is based upon your code from Part B, the same requirements apply:

- You **must** create a different dataframe for each species by *filtering* the data appropriately. (*Hint: you should end up with three new dataframes.*)
- You **must** plot each of these dataframes in the same graph, giving each data series an appropriate label.
- You **must** give the plot a title, axis labels and a legend.

