SCIT, University of Wollongong

CSIT110/CSIT810

SIM Session 2 2021

Assignment 3 (20%) due on Saturday 29 May 2021 at 00:00AM

Objectives

- Able to write clear code with comments and follow coding convention
- Able to use variables with meaningful names and correct data types
- Able to define functions and class objects
- Able to raise exceptions and handle different errors

Marking criteria:

- Total marks: 20. 1 mark deducted for each day late.
- More than 3 days late will result in zero marks.
- Code must be able to run with no errors: 0 marks for the whole assignment if an error is thrown.
- Correct file format (.py extension): 0 marks for the whole assignment if file submission is not in correct format.
- Use submission template for file submission.

Question 1	correctness, completeness and consistency with the assignment specification	4 marks
Question 2	correctness, completeness and consistency with the assignment specification	4 marks
Question 3	correctness, completeness and consistency with the assignment specification	4 marks
Question 4	correctness, completeness and consistency with the assignment specification	4 marks
Question 5	correctness, completeness and consistency with the assignment specification	4 marks

Overall	comments include name, student number, subject code; clear code and follow coding convention; use variables with meaningful names and correct data types	Deduct up to 1 mark

Submission Instruction: Assignment 3 submission is on Moodle. Put all your python code into a single python file (<name>_<std no>_a3.py) and submit it.

Assignment questions: there are 5 assignment questions.

Write clear code with **comments** and follow **coding conventions**. Script should include **your name**, **student number** and **subject code** on top of your code. Please also add this information to the variables as stated in the template Your code must work **exactly** like the provided examples given the input in the examples.

```
name = 'John Snow'
student_num = '1234567'  # Student number
subject_code = 'CSIT110'  # CSIT110 or SP121
```

Question 0.

Look at the submission template. Understand what `example()` in the main scope is doing.

Question 1.

Write a function that meets the following criteria.

<table>
<tr><td>Function name</td><td><code>myClass_get_int_unit_test</code></td></tr>
<tr><td>Parameter</td><td>1. a class reference</td></tr>
<tr><td>Return value</td><td>1. str or int</td></tr>
<tr><td>Detailed information</td><td>In the function, using a try and except blocks, instantiate an instance of the input class.

Next, call the instance method <code>get_integer()</code>.
The function should return the corresponding values in the table below.
<table>
<tr><th>Condition</th><th>Return value</th></tr>
<tr><td>AttributeError was raised.
An error raised when a method or variable of an instance which was referenced did not exist</td><td>A str object with the uppercase letter 'A'</td></tr>
<tr><td>ValueError was raised.
This occurs when an argument that has the right type but an inappropriate value</td><td>A str object with the uppercase letter 'V'</td></tr>
<tr><td>All other errors</td><td>A str object with the uppercase letter 'O'</td></tr>
<tr><td>If no error was raised</td><td>Return the integer which the <code>get_integer()</code> method returns</td></tr>
</table>
Hint: modify example() in question 0 to test your code.</td></tr>
</table>

Question 2.

A merchant has a collection of goods. Help him write the following function.

Function name	`compute_unit_prices`
Parameters	1. `Dict[str, List[float, int]]` A dictionary with `str` typed objects as keys and a list of two numbers as value. The keys correspond to the names of the goods. Each list has two numbers - the first, of type `float`, is the bulk cost of goods, - the second number, of type `int`, is the bulk quantity of the goods. 2. `List[str]` A list of `str` objects. Compute the unit prices of the goods in this list.
Return value	1. `Dict[str,float]` The dictionary shall contain the unit prices with the name of goods as keys and the unit prices as values. The unit price is defined by the bulk price divided by the bulk quantity.
Detailed information	Using try and except blocks, should the names in the second input cannot be found in the first, the good's unit price should be a None object, should the unit price cannot be obtained due to a `ZeroDivisionError`, the unit price should be -1. Using if else blocks will result in 0 marks for this question. If any other errors are raised, the function should return an empty dictionary.

An example of the first input

```
{
    "Popsicle": [120.0, 100],
    "Lollipop": [950, 1000],
    "Chips": [850,1100],
    "TIME Magazine": [1050, 0]
}
```

An example of the second input

```
["Popsicle", "TIME Magazine", "Newspaper"]
```

Return value of with the above examples as input

```
{
    "Popsicle": 1.20,
    "TIME Magazine": -1,
    "Newspaper": None
}
```

Question 3a.

Define an Exception class that meets the following specifications.

Class name	`AssessmentNotFoundError` This class inherits the `Exception` class.
Constructor parameters	1. `name_assessment: str` 2. `name_student: str` Assign these values to instance attributes of the same names.

Question 3b.

Define a dunder method for the above class that meets the following specifications.

Method name	`__str__`
Method parameter	-
Return value	Returns the string "`____ cannot be found in ____'s results`". Fill the first blank with the instance attribute `name_assessment`, and the second blank with the instance attribute `name_student`.

Question 3c.

1. the `Student` class is defined in the template.
2. Modify the instance method `get_weighted_results()` to raise a custom exception named `AssessmentNotFoundError` if the assessment_name is not the keys of the dictionary parameter, `weights`.

Question 4a.

Define a class that meets the following specifications.

Class name	`InvalidDepthError` This class inherits the `Exception` class.
Class constructor parameter	-
Instance method	A `__str__` dunder method for this class to return a string `"Invalid Depth"`.

Question 4b.

Define a class that meets the following specifications.

Class name	`WaterBody`
Class constructor parameter	1. `int/float` Assign this number to the instance attribute `volume`
Class attribute	 The class has class attributes `RHO = 997` and `G = 9.81`.

Question 4c.

Define a **class** method that meets the following specifications.

Method name	`get_hydrostatic_pressure`
Method parameter	1. `depth: float`
Return value	1. `float`
Detailed information	 Using the float argument, `depth`. calculate and return the hydrostatic pressure. Hydrostatic pressure given depth = `RHO*G*depth` If the depth is less than 0, the class method should raise an `InvalidDepthError`. This should be defined in question 4a.

Question 4d.

Define an **instance** method that meets the following specifications.

Method name	`get_water_mass`
Method parameter	-
Return value	1. `Float`
Detailed information	 This method should return the `mass` of the `waterbody` given that `mass = RHO* volume`.

Question 4e.

Define three **static** methods that meet the following specifications.

Method names	`is_large`
	`is_medium`
	`is_small`

Method parameter	1. `float` the volume of the water body in km^3
Return value	1. `bool`
Detailed information	Return a Boolean according to the criteria – is_small returns True if volume is less than 50 km^3. is_medium returns True if the volume is between and inclusive of 50 km^3 to 100km^3 is_large returns True if the volume is greater than 100km^3

Question 4f.

Define a **class** method that meets the following specifications.

Method names	`spawn`
Method parameter	–
Return value	1. a `WaterBody` instance
Detailed information	Return an instance of `WaterBody` with a volume that is randomly generated from the random module. Note that volume must be a positive value. (>0)

You can use the following lines of code to verify part of your code.

```
pool = WaterBody(10)
print(pool.get_hydrostatic_pressure(1)) # prints 9780.57
print(pool.get_water_mass())  # prints 9970
water_body = WaterBody.spawn()
print(f"water_body is a WaterBody object:{isinstance(water_body,Waterbody)}")
try:
    pool.get_hydrostatic_pressure(-1)
except Exception as e:
    print(e)  # prints Invalid Depth
```

Question 5

Create a class `SingaporeNumbers`.

Part 1

A typical vehicle registration number comes in the format **xxx #### y**:

- **x** – prefixes
- #### – Numerical series (from 1 to 9999, without leading zeroes)
- **y** – Checksum
  - The checksum letter is calculated by converting the letters into numbers, *i.e.*, where A=1 and Z=26, potentially giving seven individual numbers from each registration plate. However, only two letters of the prefix are used in the checksum. For a three-letter prefix, only the last two letters are used; for a two-letter prefix, both letters are used; for a single letter prefix, the single letter corresponds to the second position, with the first position as 0. For numerals less than four digits, additional zeroes are added in front as placeholders, for example "1" is "0001". SBS 3229 would therefore give 2, 19, 3, 2, 2 and 9 (note that "S" is discarded); E 12 would give 0, 5, 0, 0, 1 and 2. SS 108 would be given as 19, 19, 0, 1, 0, 8.
  - Each individual number is then multiplied by 6 fixed numbers (9, 4, 5, 4, 3, 2). These are added up, then divided by 19. The remainder corresponds to one of the 19 letters used (A, Z, Y, X, U, T, S, R, P, M, L, K, J, H, G, E, D, C, B), with "A" corresponding to a remainder of 0, "Z" corresponding to 1, "Y" corresponding to 2 and so on. In the case of SBS 3229, the final letter should be a P; for E 23, the final letter should be a H. SS 11 back letter should be a T. The letters F, I, N, O, Q, V and W are not used as checksum letters.

Question 5a

Define a **static** method that meets the following specifications.

Method name	`car_plate_checksum`
Parameter	1. str This str contains the prefixes and numerical series mentioned in the description above.
Return value	1. str An upper case letter
Detailed description	Compute and return the checksum from the string parameter. The computation logic is described in the section titled 'Part 1'. The checksum is one character in length. The return value is case insensitive. You should use the try and except blocks to find out is a character in a string is an integer or not. The input string may contain 1-3 letters for prefixes while there can be 1 to 4 digits for the numerical series that follows.

Part 2

The checksum letter of a magic 7 digits number is calculated as such:

d = $[(i_1\ i_2\ i_3\ i_4\ i_5\ i_6\ i_7) \bullet (2\ 7\ 6\ 5\ 4\ 3\ 2)]$ mod 11

= $(\ 2i_1 + 7i_2 + 6i_3 + 5i_4 + 4i_5 + 3i_6 + 2i_7\ )$ mod 11

Where i_x is the 1st to last of the 7 digits of the numbers and (2,7,6,5,4,3,2) are the weights.

Write a static method `magic_num_checksum` that Return the letter which corresponds to the number d as shown in the look-up table below

d	10	9	8	7	6	5	4	3	2	1	0
Check digit	A	B	C	D	E	F	G	H	I	Z	J

Question 5b

Define a **static** method that meets the following specifications.

Method name	`magic_num_checksum`
Parameter	1. str This str contains the numerical series mentioned in the description above.
Return value	1. str An upper case letter
Detailed description	From the given string of 7 numbers, computer the number d as described in the section 'Part 2'. Return the letter which corresponds to the number d as shown in the look-up table below *d* 10 9 8 7 6 5 4 3 2 1 0 Check digit A B C D E F G H I Z J