SCIT, University of Wollongong

CSIT110/CSIT810

Autumn Session 2020

Examination (60%) due on Friday 4th December 2020 at 01:00PM

Marking criteria:

- Total marks is 60.
- No errors should be raised unless otherwise specified.
- Correct file format (.py extension):5 marks penalty if file submission is not in correct format.
- Names of variables, functions and classes that do not meet specification will result in 0 marks for the question. Spelling errors in attributes, variables, functions, class names which trigger errors in testing the question will be penalised.
- Use submission template for file submission.

Question 1	Correctness, completeness and consistency with the exam specification	20 marks
Question 2	Correctness, completeness and consistency with the exam specification	5 marks (bonus, optional)
Question 3	Correctness, completeness and consistency with the exam specification	12 marks
Question 4	Correctness, completeness and consistency with the exam specification	14 marks
Question 5	Correctness, completeness and consistency with the exam specification	14 marks
Overall	Comments include name, student number, subject code; clear code and follow coding convention; use variables with meaningful names and correct data types. No calling of functions outside of main() block. Use the template given.	Deduct up to 1 mark for each issue Deduct up to 5 marks

Submission Instruction: Submission is on Moodle. Put all your python code into a single python file (<name>_<uow std no>_exam.py) and submit it.
Exam questions: there are 5 examination questions.
Write clear code with **comments** and follow **coding conventions**. Comments 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.
You may assume that the parameters used for testing are of the right data type specified in each question, unless stated otherwise.

```
"""Examination

Name: John Snow
Student number: 1234567
Subject code: CSIT110
"""

name = "John Snow"
student_uow_id = "1234567"  # UOW student number
course_code = "CSIT110"  # CSIT110 or SP420
```

Question 1: There are 10 parts to this question. Write the following **functions.**

Q1a.

Function name	`question_1a`
Parameter(s)/ argument(s)	In this order, 1. a list 2. a list
Return value	A list
Detailed description	Add the contents of the second list to the back of the first list. The contents of the lists can be of any data type.
Example usage	`print(question_1a([1,"asd",234],[["ewq",32],True]))`
Example usage output	`[1,"asd",234,["ewq",32],True]`

Q1b.

Function name	`question_1b`
Parameter(s)/ argument(s)	In this order, 1. a list 2. a str or an integer, `item`
Return value	1 integer
Detailed description	If the second parameter exists in the first list parameter, return the index of the first occurrence of the `item` found in the given list. If it does not, return -1
Example usage	`print(question_1b([1,4,"abc",3,2,9,-1,4],4))`

Example usage output	`1`

Q1c.

Function name	`question_1c`
Parameter(s)/ argument(s)	In this order, 1. an integer, `x` 2. an integer, `N`
Return value	A list of integers
Detailed description	return a list of `N` number of `x`
Example usage	`question_1c(15,3)`
Example's return value	`[15,15,15]`

Q1d.

Function name	`question_1d`
Parameter(s)/ argument(s)	In this order, 1. a list of integers 2. an integer
Return value	An integer
Detailed description	return the number of times the given integer appears in the given list

Example usage	`print(question_1d([1,5,2,-1,99,5,0],-1))`
Example usage output	`1`

Q1e.

Function name	`dot_product`
Parameter(s)/ argument(s)	In this order, 1. a list of N integers 2. a list of N integers
Return value	An integer

Detailed description	Given:
	`list1 = [a₁, a₂, a₃, a₄, ...]`
	`list2 = [b₁, b₂, b₃, b₄, ...]`
	A dot product is defined as such
	$a_1*b_1 + a_2*b_2 + a_3*b_3$ ...
	Return dot product of the two lists.
	You can assume the list parameters are of the same length and N > 0.

Example usage	`print(dot_product([1,2,3,4,5],[9,8,7,6,-5]))`

Example usage output	`45`
	Because
	`1*9+2*8+3*7+4*6+5*-5 = 45`

Q1f.

Function name	`root_mean_square`
Parameter(s)/ argument(s)	1. A list of integers
Return value	1. A float
Detailed description	Given a list: `list1 = [a1, a2, a3, a4, ..., aN]` The root mean square of a list of numbers is defined as such $\sqrt{\frac{1}{N}({a_1}^2 + {a_2}^2 + {a_3}^2 + {a_4}^2 + \cdots + {a_N}^2)}$ Return the root mean square of the list of integers. There is no need to format the number of decimal places in a float.
Example usage	`print(root_mean_square([1,2,3,4,5,6,7]))`
Example usage output	`4.47213595499958`

Q1g.

Function name	`list_to_dict`
Parameter(s)/ argument(s)	1. A list of lists, each containing two elements. `e.g. [[a0, a1],[b0,b1], ...]`
Return value	A dictionary
Detailed description	Using the first elements of each nested list as keys and the second elements as values, return a dictionary that contains all the key-value pairs.
Example usage	`print(list_to_dict([["eg1",123],["eg2",234],["eg3",345]]))`
Example usage output	`{"eg1":123,"eg2":234,"eg3",345}`

Q1h.

Function name	`question_1h`
Parameter(s)/ argument(s)	1. a `str`
Error handling	Function should raise an exception. The type of exception raised is described below.
Return value	No return values.
Detailed description	Using the `str` parameter as the error message, raise a `ValueError`.

Q1i.

Function name	`question_1i`
Parameter(s)/ argument(s)	- No parameters for this function
Return value	A `str`
Detailed description	Get a user input with the prompt `"Enter size: "` You may assume the user input will be a **positive numeric value, but not limited to integers** Base on the value, `x`, obtained from the user input, return the following character(s) as stated in the table below:

`x < 8`	`"XS"`
`8 <= x < 10`	`"S"`
`10 <= x =< 14`	`"M"`
`x > 14`	`"L"`

Example usage (user input is indicated in bold)	`x = question_1i()`
	`Enter size: 7`
	`print(x)`

Example usage output	`XS`

Q1j.

Function name	`question_1j`
Parameter(s)/ argument(s)	1. An integer, `N`
Return value	A dictionary with two key-value pairs: 1. Key: `"qns"` Value: A str of the N randomly generated integers separated by `" x "`, that is a space, a lower-cased letter `"x"`, and another space. 2. Key: `"ans"` Value: The product of the `N` integers
Detailed description	Given that the integer in the parameter is N, generate N integers between 1 and 100, return a dictionary with two key value pairs as described. You may assume that `N>=2`
Example usage	`print(question_1j(6))`
Example usage output	`{"qns": "60 x 17 x 83 x 71 x 75 x 72", "ans": 32458644000}`

Q2a.
Write a **class** that fulfils the following specifications.

Class name	`Product`
Parameter(s)/ argument(s) for the constructor __init__	In this order, 1. name - a `str` 2. price - a `float`
Detailed description	1. Assign the parameters to instance attributes **of the same names, i.e. `name` and `price`**

Example usage	`soap = Product("Johnson Baby Bath",12.34)`
	`print(soap.name)`
	`print(soap.price)`

Example usage output	`Johnson Baby Bath` `12.34`

Q2b.
Write a **class method** for the `Product` class that fulfils the following specification

Method name	`from_dict`
Parameter(s)/ argument(s)	A dictionary with the following key value pairs 1. Key - `"name"`, value - a `str` 2. Key - `"price"`, value - a `float`
Return value	An instance of the `Product` class
Detailed description	Return an instance of the Product class using the values of the keys `"name"` and `"price"` in the dictionary.

Example usage	`new_product = Product.from_dict({"name":`
	`"hand sanitizer", "price":1.75})`
	`print(new_product.name)`
	`print(new_product.price)`

Example usage output	`hand sanitizer` `1.75`

Q3:
Write the following **function**:

Function name	`get_class_statistics`
Parameter(s)/ argument(s)	1. List of Student instances. `i.e. type list[Student]` The definition of the student class is shown in your submission template
Return value	Dictionary with str for keys and integer for values `i.e. type dict[str, int]` There should be 3 key-value pairs: 3. Key: `"pass_count"` Value: Number of passing students (total grade greater than or equal to 50) 4. Key: `"fail_count"` Value: Number of failing students (total grade less than 50) 5. Key: `"invalid_count"`, Value: Number of students with no grades (`grades == None`)
Error handling	The function must handle `NoGradesError`. The definition of the `Error` is provided in the template. The function must not raise any exceptions.

Detailed description	Notes:
	● You may assume the function will be called with a valid argument.
	● The function should not write to stdout i.e. do not use `print()`.
	● The `Student` class definition is given below and in the submission template.
	● You may assume the grades are non-negative.
	● The student's total grade is the sum of the student's grades.
	● **You HAVE to use the `Student` instance method** `get_grades()` to access the `Student` instance attribute `self.__grades`.
	
	`class NoGradesError(Exception):`
	`  pass`
	
	`class Student():`
	`  def __init__(self, name= "", grades=None):`
	`  self.__grades = grades`
	
	`  def get_grades(self):`

	`        if self.__grades == None:`
	`            raise NoGradesError()`
	`        return self.__grades`

Example usage	`students = [`
	`        Student("StudentA"),                 # Invalid`
	`        Student("StudentB", [10, 2, 3]),     # Fail`
	`        Student("StudentC", [11, 4, 5]),     # Fail`
	`        Student("StudentD", [20, 30, 40]),   # Pass`
	`        Student("StudentE", [30, 40, 50]),   # Pass`
	`        Student("StudentF", [10, 15, 25]),   # Pass`
	`]`
	`print(get_class_statistics(students))`

Example usage output	`{"pass_count": 3, "fail_count": 2, "invalid_count": 1}`

Q4a
Write a class that fulfils the following specifications.

Class name	`Staff`
Parameter(s)/ argument(s) for the constructor __init__	In this order 1. A `str, name` 2. A `str, staff_num` 3. A `float, salary` 4. A `bool, staff_benefits` `staff_benefits` is an optional parameter with a default value of `True`
Detailed description	Using the parameters, initialise instance attributes of the same names.

Example usage	`jane = Staff("Jane Doe", "EB4034", 2500)`
	`tom = Staff("Tom Harris", "663D5E", 2950, False)`
	`print(jane.staff_benefits)`
	`print(tom.staff_benefits)`

Example usage output	`True` `False`

Q4b.
Define an **instance method** for the `Staff` class as follows.

Method name	`get_paycheck`
Parameter(s)/ argument(s)	- No parameter for this method
Return value	`A float`
Detailed description	Return the instance attribute salary

Example usage	`jane = Staff("Jane Doe", "EB4034", 2500)`
	`tom = Staff("Tom Harris", "663D5E", 2950, False)`
	`print(jane.get_paycheck())`
	`print(tom.get_paycheck())`

Example usage output	`2500.0` `2950.0`

Q4c.

Create a class that fulfils the following specifications

Class name	`ContractStaff`
Parameter(s)/ argument(s)	In this order 1. A `str, name` 2. A `str, staff_num` 3. A float, `hourly_rate`
Description	This class inherits the class `Staff`. 1. Call the parent class constructor using the appropriate parameters. 2. The `salary` for a `ContractStaff` instance is `None` 3. The `staff_benefits` is `False`. 4. The `contractStaff` instance will also have the instance attribute, `hours_worked`. The value of `hours_worked` is `0` when the instance is created. 5. Assign the parameter hourly_rate to the instance attribute of the same name.
Example usage	`ashlyn = ContractStaff("Ashlyn Hennessy", "9423c4", 8.5)` `print(ashlyn.staff_benefits)`
Example usage output	`False`

Q4d.

Define an **instance method** for the `ContractStaff` class

Method name	`add_hours_worked`
Parameter(s)/ argument(s)	1. A float
Return value	- No return value for this method
Description	Increment the instance attribute `hours_worked` by the number provided in the parameter
Example usage	`ashlyn.add_hours_worked(8.5)`

	`print(ashlyn.hours_worked)`
Example usage output	`8.5`

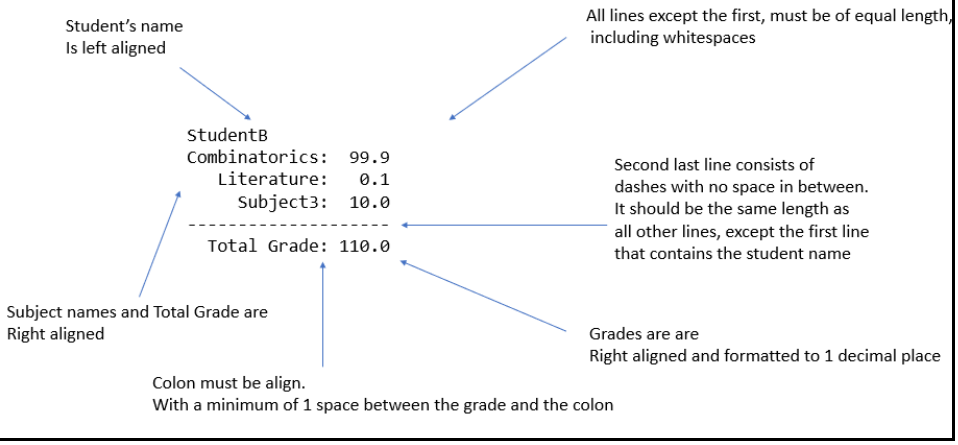
Q4e.

Override the `get_paycheck` **instance method**

Method name	`get_paycheck`
Parameter(s)/ argument(s)	- No parameters for this method
Return value	A `float`
Description	Return the product of instance attributes `worked_hours` and `hourly_rate`
Example usage	`print(ashlyn.get_paycheck())`
Example usage output	`72.25`

Q5.

Write the following **function**:

Function name	`get_student_report`
Parameter(s)/ argument(s)	1. An instance of `HighSchoolStudent`. Refer to the submission template for its class definition.
Return value	A `str`, formatted as described below
Error handling	None. The function must not raise any exceptions.
Detail Description	See next page. 

Detailed description	Notes: ● You may assume the function will be called with a valid argument. ● The function should not write to stdout i.e. **do not use `print()`.** ● The `Student` class definition is given in the submission template. ● You **must** use the instance method `get_name()` to get the name of the student. You may assume the HighSchoolStudent method get_name will return a valid name. No error will be raised. ● You **must** use the instance method `get_grades()` to get the grades dictionary. You may assume the HighSchoolStudent method `get_grades()` will return a valid dictionary of subject names and grades. ● You may assume the grades are non-negative and have at most 1 decimal place. The student's total grade is the sum of the student's grades. The **contents** of the returned `str` are as follows: 1. First line shows the student's name 2. For each entry in the student's grades dictionary: 1. Add a line showing the subject name, followed by the separator `": "` (1 colon, 1 space), and grade 3. The penultimate (second-to-last) line is a divider consisting a series of dashes (`-`) 4. The last line shows `"Total Grade"`, followed by the separator `": "` (1 colon, 1 space) and the student's total grade The **formatting** of the returned `str` is as follows: 1. The colons should be aligned i.e. the colons should be in the same position in each line 2. The subject names should be right-justified 3. The grades should be formatted to one decimal place and right-justified 4. On the final line, `"Total Grade"` is right justified 5. On the final line, the total grade is formatted to one decimal place and right-justified 6. All lines, except the first line, should have the same length (including whitespaces) 7. The penultimate (second-to-last) line should be the same length as all other lines, excluding the first line. 8. Do not insert any unnecessary whitespace 9. Your `str` must include newlines where appropriate.

CLUE:	Use nested curly brackets with a variable in it to set the minimum number of characters of a string with a variable.
	
	Given:
	`num1 = 123`
	`num2 = 53`
	
	`str1`, `str2` and `str3` will be identical.
	
	`w1 = 6  # column width`
	`w2 = 4  # column width`
	
	`str1 = "!{0:<6}!{1:^4}!".format(num1, num2)`
	
	`str2 = "!{0:<{var1}}!{1:^{var2}}!".format(`
	`num1, num2, var1=w1, var2=w2)`
	
	`str3 = f"!{num1:<{w1}}!{num2:^{w2}}!"`
	
	`print(str1)`
	`print(str2)`
	`print(str2)`

Output of usage in clue	`!123   ! 53 !` `!123   ! 53 !` `!123   ! 53 !`

Example usage 1	`student = HighSchoolStudent(`
	`"StudentA",`
	`{`
	`"Art History": 1.2,`
	`"Spanish": 23.4,`
	`"Computer Science": 45,`
	`}`
	`)`
	`print(get_student_report(student))`

Example usage 1 output	`StudentA`
	`Art History:  1.2`
	`Spanish: 23.4`
	`Computer Science: 45.0`
	`----------`
	`Total Grade: 69.6`

Example usage 2	`student = HighSchoolStudent(`
	`"StudentB",`
	`{`
	`"Combinatorics": 99.91,`
	`"Literature": 0.1,`
	`}`
	`)`
	`print(get_student_report(student))`

Example usage 2 output	`StudentB`
	`Combinatorics:  99.9`
	`Literature:  0.1`
	`--------------------`
	`Total Grade: 100.0`

Example usage 3	`student = HighSchoolStudent(`
	`"StudentC",`
	`{`
	`"Environmental Management - 5014": 2000.5,`
	`"French": 10.1,`
	`}`
	`)`
	`print(get_student_report(student))`

Example usage 3 output	`StudentC`
	`Environmental Management - 5014: 2000.5`
	`French:  10.1`
	`--------------------------------------`
	`Total Grade: 2010.6`

----------End of Paper----------