Foundation

Advanced Python

Weeks 2 & 3

Module 1 Exercises

Version	Date	Author	Comments
1.0	01 / 09 / 21	Nikola Ignjatovic	First draft

Module 1A: Lambda Functions & Comprehensions

1. Create a lambda function named `salary_with_bonus` that calculates employee's salary with the bonus,

    a) where both employee's salary and bonus are passed in as parameters

    b) where both employee's salary and bonus are passed in as parameters but bonus is fixed as keyword argument ("kwarg")

    c) where only employee's salary is passed in as a parameter and bonus is fixed (hard-coded)

    Assume bonus is 5%.
    Example: for the salary of `25000` the lambda function should return `26250.0`

2. Use the lambda function from Question 1 as a lambda function:

    a) passed as an argument to an ordinary higher order function

    b) passed as an argument to a lambda higher order function

    c) returned from an ordinary custom-built higher order function

    d) returned from a lambda custom-built higher order function

    to calculate the salary with the bonus for a given employee's salary.

    Create two solutions for questions a) and b): the first where the lambda function accepts two changeable parameters: salary & bonus, and the second where the lambda function accepts two parameters: salary & bonus, one of which (bonus) is fixed (keyword argument) - making it effectively a lambda function that accepts one changeable parameter.

    In questions c) and d) pass only the bonus value as argument to the higher order function.
    Example: for the salary of `25000` and the bonus of 5%, tall solutions should return `26250.0`

3. Write the function `filter_valid_salaries` that removes any non-positive salary from the list of salaries (leaving only positive salaries) using:

    a) list comprehension

    b) generator comprehension

    c) the filter() built-in function

    Example: for the list of salaries
    `[29000, 48000, 0, 31213, 15875, -21500, 17450, 50245, 75801, -100000]`,
    the function should return the list
    `[29000, 48000, 31213, 15875, 17450, 50245, 75801]`, as negative numbers and 0 should be ignored.

4. Create a function `salaries_with_bonus` that returns the list of salaries with the bonus for a list of employees' salaries passed to it as parameter using:

    a) list comprehension

    b) generator comprehension

    c) the map() built-in function

    Assume bonus is 5%.

    Example: for the list of salaries
    `[29000, 48000, 31213, 15875, 17450, 50245, 75801]`
    the function should return the list
    `[30450.0, 50400.0, 32773.65, 16668.75, 18322.5, 52757.25, 79591.05]`

5. Create a function `salaries_with_bonus` that returns the list of salaries with the bonus for a list of employees' salaries passed to it as parameter using:

    a) lambda function passed as an argument to an ordinary higher order function featuring

        i. list function

ii. list comprehension
iii. generator comprehension

b) lambda function passed as argument to a lambda higher order function featuring
   i. list function
   ii. list comprehension
   iii. generator comprehension

c) lambda function returned from an ordinary higher order function featuring
   i. list function
   ii. list comprehension
   iii. generator comprehension

d) lambda function returned from a lambda higher order function featuring
   i. list function
   ii. list comprehension
   iii. generator comprehension

Assume bonus is 5%.
Example: for the list of salaries

```
[29000, 48000, 31213, 15875, 17450, 50245, 75801]
```

the function should return the list

```
[30450.0, 50400.0, 32773.65, 16668.75, 18322.5,
 52757.25, 79591.05]
```

6. The input list contains integer values that should represent employee's salaries; however some of its values may be erroneous: some values may be 0, while others may be negative. Assuming that bonus is 5%, write a function `valid_salaries_with_bonus` that returns the list of salaries with the bonus for a list of employees' salaries passed to it as parameter, while ignoring any erroneous values using:

   a) list comprehension

b) generator comprehension

c) map() and filter() built-in functions

Example: for the list of salaries:

```
[29000, 48000, 0, 31213, 15875, -21500, 17450, 50245,
75801, -100000]
```

the function should return the list

```
[30450.0, 50400.0, 32773.65, 16668.75, 18322.5,
 52757.25, 79591.05]
```

, as negative numbers and 0 should be ignored.

7. Write a Python program to sort a list of tuples using lambda function. Sort the courses by their achievement in descending order.
   Example:
   Original list of tuples:

```
[('SQL', 88), ('Excel', 90), ('Python', 97), ('Unix',
82), ('Web Apps', 78), ('Java', 75)]
```

   Sorted list of tuples:

```
[('Python', 97), ('Excel', 90), ('SQL', 88), ('Unix',
82), ('Web Apps', 78), ('Java', 75)]
```

   Tip: pass the lambda function as the value to the key parameter of sorted() function or sort() method.

8. Write a Python program to sort a list of dictionaries using lambda function. Sort the mobile phone models by their colour in alphabetical order.
   Example:
   Original list of dictionaries:

```
[{'make': 'Nokia', 'model': 216, 'colour': 'Black'},
 {'make': 'Mi Max', 'model': '2', 'colour': 'Gold'},
 {'make': 'Samsung', 'model': 7, 'colour': 'Blue'}]
```

   Sorted list of dictionaries:

```
[{'make': 'Nokia', 'model': 216, 'colour': 'Black'},
 {'make': 'Samsung', 'model': 7, 'colour': 'Blue'},
 {'make': 'Mi Max', 'model': '2', 'colour': 'Gold'}]
```

9. Write a lambda function named `starts_with` that finds if a given string starts with a character 'P' (returns True if it does; False if it doesn't).
   Tip: use the `dir` command to list all string methods and try to find one that could help with this task.

10. Change the lambda function from previous question to find if a given string starts with a given character passed to it as a parameter.

11. Write a Python program to find intersection of two given lists using lambda function.
    Example: for two given lists:
    ```
    list_nums1 = [1, 2, 3, 5, 7, 8, 9, 10]
    list_nums2 = [1, 2, 4, 8, 9]
    ```
    the program should return the list `[1, 2, 8, 9]`
    Tip: pass the lambda function as argument to the `filter()` function

12. Write a Python program to find the elements of a given list of strings that contain specific substring using lambda function.
    Example: for the following two inputs:
    ```
    list_strings = ['January', 'February', 'March',
    'April', 'May', 'June', 'July', 'August', 'September',
    'October', 'November', 'December']
    substring = 'emb'
    ```
    the program should return the list
    ```
    ['September', 'November', 'December']
    ```
    Tip: pass the lambda function as argument to the `filter()` function

13. Write a Python program to find the values of length 5 in a given list using lambda function.
    Example: for the following list:
    ```
    months = ['January', 'February', 'March', 'April',
    'May', 'June', 'July', 'August', 'September',
    'October', 'November', 'December']
    ```
    the program should return just two values: `March April`

14. Change the lambda function from previous question to find values of any given length in a given list.

15. Write a Python program to find palindromes in a given list of strings using lambda function.
Example:
Original list of strings:
```
['!', 'php', 'w3r', 'Python', 'ada', 'Java', 'abcd',
'aaa', 'aoxomoxoa', 'BaroccoraB']
```
List of palindromes:
```
['!', 'php', 'ada', 'aaa', 'aoxomoxoa', 'BaroccoraB']
```

16. Write a Python program that multiplies each number of a given list with a given number using lambda function. Print the result as a string of numbers.
Example:
Original list: `[2, 4, 6, 9, 11]`
Given number: `2`
Result:
```
4 8 12 18 22
```

17. Write a Python function to multiply all the numbers in a given list using lambda.
Example:
Original list:
```
[4, 3, 2, 2, -1, 18]
```
Result: `-864`
as 4 * 3 * 2 * 2 * -1 * 18 = -864
Tip: import the `functools` module and find help on the function `reduce()`

18. Write a Python function that finds the maximum of all the numbers in a given list using lambda.

19. Write a Python function that finds the minimum of all the numbers in a given list using lambda.

20. Write a Python function that finds the average of all the numbers in a given list using lambda.

21. Write a Python function to find the list with maximum length in a list of lists using lambda function. The function should return a tuple consisting of

two elements:
1) the max length
2) the list having the max length
Example:
Original list:
```
[[0], [1, 3], [5, 7], [9, 11], [13, 15, 17]]
```
Returned value:
```
(3, [13, 15, 17])
```

22. Write a Python function to find the list with minimum length in a list of lists using lambda function. The function should return a tuple consisting of two elements:
1) the min length
2) the list having the min length
Example:
Original list:
```
[[0], [1, 3], [5, 7], [9, 11], [13, 15, 17]]
```
List with minimum length of lists:
```
(1, [0])
```

23. a) Write an ordinary function to create the biggest number by rearranging the digits of a given number.
Example: for the input 213 the function should return 321

b) Use method chaining to create a one-line version of the above function.

c) Create a lambda function from the one-line version of the above function

24. Write a Python program to count the occurrences of the items in a given list using lambda.
Tip: return the dictionary where the keys are the different (unique) items and the values are their corresponding frequency. Use the count() list method to count the occurrences of different list items.
Example:
Original list:
```
[3, 4, 5, 8, 0, 3, 8, 5, 0, 3, 1, 5, 2, 3, 4, 2]
```
Count the occurrences of the items in the said list:

```
{3: 4, 4: 2, 5: 3, 8: 2, 0: 2, 1: 1, 2: 2}
```

25. Write a lambda function named `cap_string` that capitalises the first letter of a string.

26. a) Write an ordinary function that accepts two parameters: a string and a boolean key variable upper_rest. If upper_rest=True is passed as 2nd argument, lambda function capitalises the first letter and capitalises the rest of the string. If upper_rest=False is passed as 2nd argument, lambda function capitalises the first letter and leaves the rest of the string as the original string. Tip: use the expression from the lambda function created in Question 25 as starting point to produce a one-line solution by including the if statement to it.

    b) Write a lambda function that performs the task given in Question 26a.

    When testing the ordinary function and the lambda function, call them with upper_rest=True, upper_rest=False and by omitting the parameter upper_rest altogether.

Module 1B: Handling Exceptions

Note: when in doubt what exception to use to protect the code, let the code produce the error and look at the error message. The error message always begins with the error name followed by the colon character (:).
Use the error name that appears before the colon as the exception name.

1. The following code is prone to errors:

```
employee_list = ['Jane', 'Claire', 'Van', 'Sarah',
  'Tomasz', 'Alisha']
emp_code = input('Enter an employee code between 0 and
  ' + str(len(employee_list)-1) + ' to retrieve the
  corresponding employee name: ')
print("The corresponding employee is: %s" %
  employee_list[int(emp_code)])
```

Run it and enter an inexistent list index to cause the code to throw an error and find out the name of the specific exception from the traceback message. Then use that exception name to change the code to make it safe:
a) using the try-except block
b) using the try-except-else block
In both cases prompt the user to enter the employee code until it is valid.

2. The following code is prone to errors:

```
employee_dict = {'id': 425137, 'name': 'Ali Rahmani',
  'job-title':'Developer', 'emp_date': '15-04-2018'}
attr = input('Enter an employee attribute (id, name,
  job-title, emp-date): ')
print('The value for the employee attribute', attr,
  'is:', employee_dict[attr])
```

Run it and enter an inexistent dictionary key to cause the code to throw an error and find out the name of the specific exception from the traceback message. Then use that exception name to change the code to make it safe:
a) using the try-except block

b) using the try-except-else block
In both cases prompt the user to enter the employee attribute (dictionary key) until it is valid.

3. The following code throws an error when the user enters a non-numeric value.

```
number = float(input('Enter a number:'))
print("You entered the number: ", number)
```

Find the name of the exception and use it to change the code to make it safe using the `try-except-else` block.

4. The solutions in question 1 will still throw an error if user enters a non-integer value. Amend the code from questions 1a and 1b to make them safe.

5. The following code throws an error if the user enters a file name that does not exist in the current working directory.

```
file_name = input('Enter a file name:')
f = open(file_name, 'r')
print(f.read())
```

Rectify the code to make it safe using the try-except block (without using the 'with' statement).

6. The safe code you wrote in question 5 will still throw an error if the file entered by the user includes some unreadable characters. Rectify the code to ensure it does not throw an error if the file is not found and if it is found but is unreadable. Use the input files `existingReadableFile.txt` and `existingUnreadableFile` included with solutions to test your code.

7. Two lists are given:
`lst_monthly_incomes`, which lists 3 freelancer's potential monthly incomes after tax (for each month in a quarter);
`lst_weeks`, which lists the number of weeks freelancer worked in each of the 3 months for that quarter (0-5)

Create an ordinary function `safe_weekly_income()` that takes the two lists as parameters and returns the third list, lst_weekly_incomes, listing weekly amounts each salary produces rounded to 2 decimal places.

Examples:

If the list of monthly incomes is:

```
lst_monthly_incomes = [12345.67, 10646.77, 11734.34]
```

a) `lst_weeks = [4, 3, 2]` produces `lst_weekly_incomes = [3086.42, 3548.92, 5867.17]`

b) `lst_weeks = [0, 3, 2]` produces `lst_weekly_incomes = []`

c) `lst_weeks = [5, 0, 1]` produces `lst_weekly_incomes = [2469.13]`

d) `lst_weeks = [4, 1, 0]` produces `lst_weekly_incomes = [3086.42, 10646.77]`

e) `lst_weeks = [4, 0, 0]` produces `lst_weekly_incomes = [3086.42]`

f) `lst_weeks = [0, 0, 0]` produces `lst_weekly_incomes = []`

Tip: remember that if the finally clause executes a return statement, the saved exception is discarded.

8. Create a function `improved_safe_weekly_income()` to improve the function `safe_weekly_income()` from question 7 by including a user-friendly message informing the user that the weekly income was not calculated in case(s) 0 number of weeks is entered in the second list.

9. Create a function `further_improved_safe_weekly_income()` to further improve the function `improved_safe_weekly_income()` from question 8 by including a user-friendly message in case the second list (lst_weeks) has more elements than the first (lst_monthly_incomes).

10. Create a function `final_safe_weekly_income()` to improve further still the function `further_improved_safe_weekly_income()` from question 9 by including a user-friendly message in case any of the two lists do not have 3 elements.

    Tip: raise an exception to direct dealing with this error within the same place where `IndexError` exception is dealt with and change the error message accordingly to relate to both errors.