EECS 1210 Introduction to Programming in Python

Fall Semester 2022

Prof. Pai H. Chou

Self-Check 7

Due Date: --

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests.

# 1. Definitions and Short Answers - exceptions

1. If your program tries to print a variable that has not been defined, what kind of **exception** do you get?
2. What kind of exception do you get when you try z = 10 / 0?
3. What is the same between ZeroDivisionError and OverflowError?
4. What kind of exception do you get when you run  
   L = "hello world"  
   print(L['5'])  
   ? Why?
5. Consider the following interactive session:  
   >>> int('25')  
   25  
   >>> int('0x25')  
   Traceback (most recent call last):  
    File "<stdin>", line 1, in <module>  
   ValueError: invalid literal for int() with base 10: '0x25'  
   Even though 0x25 is a valid hex literal in Python, why do you still get ValueError? How do you correctly convert '0x25' into 37? Hint: type help(int) to get documentation on different ways of using the int() function.
6. Which of the following expressions cause exceptions (and of what kind), assuming   
   L = "hello"?
   * L[4]
   * L[0]
   * L[5]
   * L[-2]
   * L[-5]
   * L[-7]
7. Suppose D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}, which of the following expressions or assignment statements cause exceptions and of what kind?
   * D['Sun']
   * D[2]
   * D['Thu']
   * D['Fri'] = 5
8. When you try to open a file by fh = open('filename', 'r') but cannot, what kind of exception do you get?
9. When trying to open a file as in the previous question, how can your program **check if an exception has occurred** and inform the user by printing   
   'Cannot open file' to the standard output and continue running the rest of the program as usual?
10. Suppose you are trying to execute the following sequence of statements  
     1 filenames = ['alpha', 'beta', 'gamma']  
     2 filenum = input('select a file by typing 1, 2, or 3:')  
     3 i = int(filenum)  
     4 fh = open(filenames[i], 'r')
    * On which **lines** can exceptions occur and what **types**?
    * How do you rewrite the code to check and **handle all types of exception the same way** by printing 'An error has occurred'?
    * How do you rewrite the code to **check each type of exception** and print an error message for each specific exception?
11. Given the following program   
     1 try:  
     2 x = int(input('enter num1:'))  
     3 y = int(input('enter num2:'))  
     4 z = x / y  
     5 except ValueError:  
     6 z = 0  
     7 except ZeroDivisionError:  
     8 z = x  
     9 print(z)
    * If an exception occurs on line 2, what lines of code are executed next?
    * If an exception occurs on line 3, what lines of code are executed next?
    * If an exception occurs on line 4, what lines of code are executed next?
    * If line 6 is execute, is it possible that lines 7-8 are also executed immediately after?
    * If either line 6 or line 8 is executed, does line 9 also get executed next?
12. Given the following program  
     1 greek = {'alpha': 0, 'beta': 1, 'gamma': 2}  
     2 try:  
     3 key = input('enter key alpha, beta, or gamma:')  
     4 y = input('enter integer: ')  
     5 z = greek[key] / int(y)  
     6 except ValueError:  
     7 print('invalid int')  
     8 except ArithmeticError:  
     9 print('arithmetic error')  
    10 except ZeroDivisionError:  
    11 print('zero division error')  
    12 else:  
    13 print('the value of z is', z)  
    14 finally:  
    15 print('last action before leaving try')
    * Which line or lines can cause one or more exceptions, of which types, and under what conditions?
    * In the case of zero division, is line 13 executed? Why or why not? If not, is ZeroDivisionError handled and by which line(s)?
    * If there is no exception by the end of line 5, which print statement or statements are executed next?
    * If the user does not input 'alpha', 'beta', or 'gamma' for the variable key on line 3,
      1. which statements are executed next?
      2. Which statement causes an exception of which type?
      3. What statements are executed after the exception?
      4. Is the exception handled by any of the statements here?
      5. What happens to the exception after the entire code above is finished?
    * Suppose you want to modify the code above by handling both KeyError and ValueError exactly the same way by the same print('invalid input') statement, which lines do you modify into what code?
13. Failure to open a file causes an OSError, but how can you find out more information about the **specific reason** why the file cannot be opened?
14. Given the following code  
     1 import sys  
     2 try:  
     3 try:  
     4 fh = open('myfile')  
     5 A = int(fh.read())  
     6 B = int(fh.read())  
     7 quotient = A / B  
     8 except (OverflowError, ZeroDivisionError):  
     9 quotient = 0.0  
    10 else:  
    11 quotient = 1.0  
    12 finally:  
    13 print('exiting inner try')  
    14 print('quotient = %f' % quotient)  
    15 except OSError as err:  
    16 sys.stderr.write(str(err))  
    If an OSError occurs on line 4, do the following lines get executed?
    * line 11?
    * line 13?
    * line 14?
    * line 16?
15. Suppose you are writing a rock, paper, scissors game as follows:  
     1 import sys  
     2 rps = input('rock, paper, scissors, or quit? [rpsq]')  
     3 if rps == 'q':  
     4 sys.exit(0)  
     5 elif rps in 'rps':  
     6 play\_game(rps)  
     7 else:  
     8 # report error in the form of a ValueError exception  
    Rewrite the code so that
    * line 8 reports error in the form of a ValueError exception with an error message,
    * enclose lines 1-8 in a try-except construct to catch the exception, and
    * handle the exception by writing the error message to sys.stderr.
16. Rewrite the code in the previous problem by using **assertion** instead. This means
    * replace lines 5-8 with an assert condition and the error message,
    * enclose the code in a try-except construct but catch the assertion type of exception (what is it?) instead of ValueError. Handle it by writing the error message to sys.stderr also.

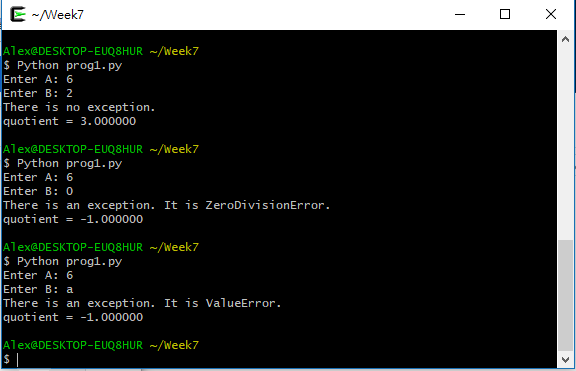
# 2. Definitions and short answers - files

1. When opening a file using fh = open('filename'), why is it ok to omit the second parameter?
2. What is the difference between opening a file with 'w' mode vs 'a' mode, as in   
   fh = open('filename', 'w') vs. fh = open('filename', 'a')?
3. Once you opened a file as file handle named fh, how do you
   1. read one character at a time as a str
   2. read 10 characters as a str
   3. read one line as a str
   4. read all the lines as a list of str
   5. read the entire file as one str
4. When reading a file either one line at a time or a number of characters at a time, when do you know you have reached the **end of the file**?
5. To open a file named 'myfile' **for writing** (or overwrite it completely if already exists), how should you open the file?
6. Once a file has been opened as file handle fh for writing, how should you **write** a string 'hello' to it?
7. What is the difference between print(s) and sys.stdout.write(s)?
8. When you are done with a file referenced by file handle fh, how do you **close** it?
9. Convert the following code into one that uses the with construct. What are some advantages?  
    1 fh = open('filename', 'r')  
    2 print('totally %d lines in file' % len(fh.readlines()))  
    3 fh.close()
10. If you finish reading a file (whose handle is fh) but want to start from beginning again, what should you do without closing and reopening the file?
11. After you have opened a file whose file handle is fh for reading or writing for a while, how do you find the **current position** in the file?
12. What is a difference between input('') and sys.stdin.readline()?
13. In a Unix-like shell such as bash (not Python shell), what do the following do?
    1. $ grep return \*.py > result
    2. $ grep return \*.py >> resfile
14. In a Unix-like shell, what is the difference between the commands  
    $ wc -w filename  
    and  
    $ wc -w < filename  
    ?
15. What does the following Unix shell command do?  
    $ grep return \*.py | wc
16. Why should you write to sys.stderr instead of sys.stdout to display a text-based error message, even though both appear on the same text terminal?
17. If you open a **text file** fh, the data object returned by fh.read() and the parameter s passed to fh.write(s) are of str type. If you open a **binary file** bh, what is the **data type** of the data object returned by bh.read() and parameter s passed to bh.write(s)?
18. How do you express the **literal** for a bytes data object consisting of ASCII characters 'h', 'e', 'l', 'l', 'o'?
19. If you want to convert a bytes literal b'world' into a str type object, why can't you just do str(b'world') even though that is how you would convert other types of objects into str, such as str(23), str(['a', 'b', 'c'])? What is the proper way?
20. How do you convert from a str object denoted by textstring into bytes type?
21. What is the meaning of the bytes literal b'\xe4\xbd\xa0\xe5\xa5\xbd'?
22. In Python, what **module** and what **function** can be called to get the **path** to the **current working directory**? How would you write a short program to print it?

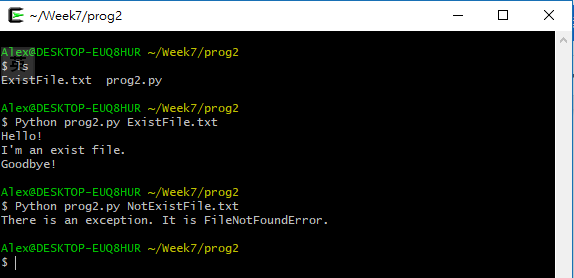
# 3. Programming Exercise

1. (Difficulty: ★☆☆☆☆) Write a command-line program named **prog1.py** to print **A / B**.

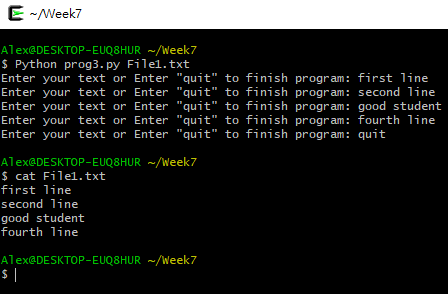
It needs to handle at least two exceptions: **ZeroDivisionError** , **ValueError**  . When exception occurs, quotient should be set to **-1 .** The program should display as follows:



1. (Difficulty: ★☆☆☆☆) Write a command-line program named **prog2.py** to read a file and print it out. It needs to handheld the exception: **FileNotFoundError .** The program should display as follows:



1. (Difficulty: ★☆☆☆☆) Write a command-line program named **prog3.py** to write a file and print it out. It needs to handheld the exception: **FileNotFoundError .** The program should display as follows:



1. (Difficulty: ★★☆☆☆) Write a command-line program named **genmul.py** to generate a multiplication table into a text file. It takes three command-line arguments and use them to generate a multiplication table by writing the text into a file.

$ **python3 genmul.py 5 7 5x7.txt**  
$ **cat 5x7.txt**  
1 2 3 4 5 6 7  
2 4 6 8 10 12 14  
3 6 9 12 15 18 21  
4 8 12 16 20 24 28  
5 10 15 20 25 30 35  
$ \_  
  
So, essentially the program reads the command-line arguments by import sys and reading the values of sys.argv list to get the two numbers for the multiplication table and the file name. If the arguments are not valid int literals then the program should report error to sys.stderr. The file name is the command-line argument after the two numbers to multiply, and in this example it is '5x7.txt'. If opening a file fails, then the error message should be reported to sys.stderr also.  
  
Hints:

* 1. You should **open** the text file for writing or overwriting.
  2. You have several options to **write** the products to file. Be sure each line (including the last) ends on a newline.
  3. Be sure you **close** the file after you finish writing. You may call close method on the file handle explicitly or you may use the with construct to automatically close the file when leaving the suite.
  4. You may use a nested loop to calculate each product, but it may be easier if you use list comprehension to calculate the list of products and then print.

1. (Difficulty: ★★★☆☆) Write a command-line program named **checkmul.py** to read the multiplication table for correctness. It takes the same three command-line arguments as **genmult.py** but instead of writing to the file, it reads the file and checks if the multiplication table is correct. If incorrect, it prints the incorrect entry and the correct answer. For example, if you have a file generated by genmul.py above, then  
   $ **python3 checkmul.py 5 7 5x7.txt**  
   Multiplication table in file 5x7.txt is correct.  
   $ \_  
   However, if you use another file named **5x7wrong.txt** whose content is as follows  
   1 2 3 4 5 6 7  
   2 4 6 8 13 12 14  
   3 6 9 12 15 18 21  
   4 8 12 17 20 24 28  
   5 10 15 z0 25 30 35  
     
   (where the incorrect entries are highlighted), then the **checkmul.py** program should display as follows:  
   $ **python3 checkmul.py 5 7 5x7wrong.txt**  
   2 x 5 = 13 is incorrect; should be 10  
   4 x 4 = 17 is incorrect; should be 16  
   5 x 4 = z0 is badly formatted; should be 20  
   Multiplication table in file checkmul.py contains 3 errors.  
   $ \_  
     
   Hints
   1. You should read one line at a time and use the split() method to convert them into a list of strings. Then, convert each one into an integer. If the string is not a properly formatted integer (decimal) literal, then report it as an incorrect result.
   2. Your program is likely to be a nested-loop structure: the outer loop iterates over the multiplier and the inner loop iterates over the multiplicand.
   3. You need to keep a variable for error count. When printing the final message, check if error is 1. If so, don't add the 's'; otherwise, add 's' to make it plural.
2. (Difficulty: ★★☆☆☆) Write a simple calculator that supports addition (+), subtraction (-), multiplication (\*), and division (/).  
     
   - The program should prompt the user to enter an expression.  
   - The program only handles one operator with two operands at a time. The operands and the operator are separated by a space.  
   - After calculating, print the answer to the user and prompt again.  
   - If the user enter 'quit' instead of an expression, say goodbye to the user and end the program.  
   - Consider only integer operands. The result of an division should be rounded to the nearest hundredth (0.666 → 0.67).  
   - Report error if dividing by 0.  
   - Report error for incomplete expressions, non-integer operands and not supported operators.  
   - You should at least raise or catch ZeroDivisionError and ValueError in your code.  
     
   $ **python3 calc.py**  
   Enter an expression: **3 + 5**Answer: 8  
   Enter an expression: **4 \* 3**Answer: 12  
   Enter an expression: **2 - 14**Answer: -12  
   Enter an expression: **5 / 3**Answer: 1.67  
   Enter an expression: **8 / 0**Error: Cannot divide by 0  
   Enter an expression: **2 +**Error: Invalid expression  
   Enter an expression: **a + 3**Error: Invalid expression  
   Enter an expression: **2 ^ 3**Error: Invalid expression  
   Enter an expression: **quit**Goodbye!  
   $ \_

# 5. Small Programming Project (won't be in quiz)

1. (Difficulty: ★★★★☆) Tiki Topple(中譯:推倒提基) is a famous board game. Players change the order of 9 Tikis to win the game. In this problem, two files named **Tiki\_line.txt** and **Operation\_list.txt** whose content is as follows:

**Tiki\_line.txt**

#1 red

#2 yellow

#3 blue

#4 green

#5 purple

#6 pink

#7 orange

#8 brown

#9 grey

**Operation\_list.txt**

# operation list:

# help - check the operation list.

# look - print Tiki line in the screen.

# Tiki up 1 - move a tiki up 1 space in the line.

# Tiki up 2 - move a tiki up 2 space in the line.

# Tiki up 3 - move a tiki up 3 space in the line.

# Tiki Topple - move a tiki to the bottom of the line.

# Tiki Toast - remove the tiki at the bottom of the line.

# Reset - Reset Tiki line.

# quit - quit the game and say goodbye.

Write a program named **mini\_Tiki\_Topple.py**,

[NOTE]

* The program should prompt the user to enter an operation in the **Operation\_list.txt**.
* When user input **help** :
* The program will open **Operation\_list.txt** and print its content.
* When user input **look** :
* The program will open **Tiki\_line.txt** and print its content.
* When user input **Tiki up n** (n = 1 ~ 3):
* The program asks the user to select a target color(target tiki name).
* The program will open **Tiki\_line.txt** and read its content.
* The program will move the target tiki up **n** space in the tiki line.
* The program will write the result to the **Tiki\_line.txt** .
* If user enter a target color against the rule, the program will except and report the error message (**Error: Invalid target**) to sys.stderr
* For example, the user enters **Tiki up 3** and selects **blue** being the target color. Because **blue** is in the #3 position, it cannot be moved up 3 space. So an exception has occurred, and the program don’t change any tikis in the **Tiki\_line.txt** .
* When user input **Tiki Topple** :
* The program asks the user to select a target color(target tiki name).
* The program will open **Tiki\_line.txt** and read its content.
* The program will move the target tiki to the bottom of the tiki line.
* The program will write the result to the **Tiki\_line.txt** .
* For example, **blue** is in the #3 position. The user enters **Tiki Topple** and selects **blue** being the target color. After this operation, all tiki under the **blue** tiki will move up 1 space. The program will put **blue** to the bottom of the tiki line, and write the result to **Tiki\_line.txt**
* When user input **Tiki Toast** :
* The program will open **Tiki\_line.txt** and read its content.
* The program will remove the tiki at the bottom of the tiki line.
* The program will write the result to the **Tiki\_line.txt** .
* For example, **grey** is in the #9 position. The user enters **Tiki Toast**. After this operation, The program will remove **grey** from the tiki line, and write the result to **Tiki\_line.txt .** Note that **Tiki\_line.txt** will have only 8 tikis .
* ps: There must be at least three tikis in the tiki line, if the user enters **Tiki Toast**, but it only three tikis in the line. An exception occurs(**Error: You can't do this operation.**), and the program won’t change any tikis in the **Tiki\_line.txt** .
* When user input **quit** :
* The program will print(“Goodbye !”) and quit the program.
* When user input **Reset** :
* The program will open **Tiki\_line.txt** .
* The program will write new content to **Tiki\_line.txt**
* New content is a list whose content as follows :

**"#1 red\n#2 yellow\n#3 blue\n#4 green\n#5 purple\n#6 pink\n#7 orange\n#8 brown\n#9 grey\n"**

* **(optional)** New content may be assigned by user
* If user enter an operation not occurs in **Operation\_list.txt :**
* An exception occurs(**Error: Invalid operation.**), and the program won’t change any tikis in the **Tiki\_line.txt** .
* You should consider exceptions as more as possiple,for example : **ValueError**,**KeyError**,**FileNotFoundError**, etc.

The program should display as follows:

$ **python3 mini\_Tiki\_Topple.py**

What's your operation? **help**# operation list:

# help - check the operation list.

# look - print Tiki line in the screen.

# Tiki up 1 - move a tiki up 1 space in the line.

# Tiki up 2 - move a tiki up 2 space in the line.

# Tiki up 3 - move a tiki up 3 space in the line.

# Tiki Topple - move a tiki to the bottom of the line.

# Tiki Toast - remove the tiki at the bottom of the line.

# Reset - Reset Tiki line.

# quit - quit the game and say goodbye.

What's your operation? **look**

#1 red

#2 yellow

#3 blue

#4 green

#5 purple

#6 pink

#7 orange

#8 brown

#9 grey

What's your operation? **Tiki up 1**

Select your target color: **purple**

What's your operation? **Tiki up 2**

Select your target color: **pink**

What's your operation? **look**

#1 red

#2 yellow

#3 blue

#4 pink

#5 purple

#6 green

#7 orange

#8 brown

#9 grey

What's your operation? **Tiki up 3**

Select your target color: **blue**

Error: You can't do this operation.

What's your operation? **Tiki up 3**

Select your target color: **white**

Error: Invalid target

What's your operation? **Tiki up 3**

Select your target color: **pink**

What's your operation? **Tiki Topple**

Select your target color: **green**

What's your operation? **look**

#1 pink

#2 red

#3 yellow

#4 blue

#5 purple

#6 orange

#7 brown

#8 grey

#9 green

What's your operation? **Tiki Toast**

What's your operation? **Tiki Toast**

What's your operation? **look**

#1 pink

#2 red

#3 yellow

#4 blue

#5 purple

#6 orange

#7 brown

What's your operation? **Reset**

What's your operation? **look**

#1 red

#2 yellow

#3 blue

#4 green

#5 purple

#6 pink

#7 orange

#8 brown

#9 grey

What's your operation? **quit**

Goodbye!

ps:

If you feel interesting about this project, you can extend your program to real game.

Play your creativity !

Reference:

[Tiki Topple boardgame rules](https://www.fgbradleys.com/rules/rules4/Tiki%20Topple%20-%20rules.pdf)