EECS 1210 Introduction to Programming in Python
Spring Semester 2024
Prof. Pai H. Chou

Self-Check 7

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?
   NameError
2. What kind of exception do you get when you try `z = 10 / 0`?
   ZeroDivisionError
3. What is the same between ZeroDivisionError and OverflowError?
   they are both ArithmeticError
4. What kind of exception do you get when you run
   ```
   L = "hello world"
   print(L['5'])
   ```
   ? Why?
   TypeError, because index to the string must be an int, but '5' is the wrong type (a str not an int)
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.
   int('0x25', base=16)
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]

IndexError for 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

   KeyError: D[2], D['Thu']
8. When you try to open a file by fh = open('filename', 'r') but cannot, what kind of exception do you get?

   OSError
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?

```
try:
    fh = open('filename', 'r')
except:
    print('Cannot open file')
```

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**?

      line 3: ValueError if filenum is not an integer literal

      line 4: IndexError if i (converted from filenum) is not a valid index from 0 to 2

      line 4: OSError if the file cannot be opened
    - How do you rewrite the code to check and **handle all types of exception the same way** by printing 'An error has occurred'?

```
try:
    # paste code lines 1-4 here
except:
    print('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?

```
try:
    # paste code lines 1-4 here
except ValueError:
    print('invalid int')
except IndexError:
    print('index out of range')
except OSError:
```

```
        print('cannot open file')
```

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? 5 6 9
- If an exception occurs on line 3, what lines of code are executed next? 5 6 9
- If an exception occurs on line 4, what lines of code are executed next? 7 8 9
- If line 6 is execute, is it possible that lines 7-8 are also executed immediately after?
  no, multiple except clauses are like elif so if one exception clause matches then the subsequent ones in the same try statement are skipped.
- If either line 6 or line 8 is executed, does line 9 also get executed next?
  yes, because if an exception is handled (without re-raising or causing another exception, which is the case here), then the try-statement completes normally and the next statement (line 9) is e xecuted.

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 con ditions?
  line 5 can cause three kinds of errors: (1) KeyError, if the first input string is not a key in gree k dict. (2) converting int(y) can be a ValueError if y is not a decimal literal, (3) ZeroDivisionE rror if y happens to be 0.

- In the case of zero division, is line 13 executed? Why or why not? If not, is ZeroDivision Error handled and by which line(s)?
  no, line 11 is never executed, because ZeroDivisionError is a kind of ArithmeticError and would be caught every time on line 8, so line 9 handles it by printing 'arithmetic error' instead. Because Python exception clauses are tested in order and are related like elif, as soon as it is caught it skips the rest of the exception clauses associated with the same try statement.
- If there is no exception by the end of line 5, which print statement or statements are executed next?
  lines 13 and 15, because else is for no error, and finally is for everybody (caught or uncaught) leaving the try.
- If the user does not input 'alpha', 'beta', or 'gamma' for the variable key on line 3,
  i. which statements are executed next?
  ii. Which statement causes an exception of which type?
  iii. What statements are executed after the exception?
  iv. Is the exception handled by any of the statements here?
  v. What happens to the exception after the entire code above is finished?
  lines 4 next; it starts executing line 5 but gets KeyError when attempting to evaluate greek[key]. The rest of line 5 is skipped. Because none of the except clauses matches KeyError, the exception is not handled. It does not match else clause either, because else is for no-error case. Then, all cases go through finally path on lines 14-15 and print 'last action before leaving try'. The KeyError propagates beyond lines 1-15 to the code at the outer level.
- 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?
  replace lines 6-7 with
  except (ValueError, KeyError):
  print('invalid input')

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?

```
try:
    fh = open(filename, mode)
except OSError as err:
    print(err)
```

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?

line 11 doesn't - because else gets executed only if no error!! It does not catch any uncaught er ror.

line 13 does, because finally is executed for all exit paths.

line 14 doesn't - because it is outside the nested try-statement and the exception uncaught in in ner try-statement has already been propagated out to the outer try.

line 16 does because it catches the OSError from the inner try.

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.

```
import sys
try:
    rps = input('rock, paper, scissors, or quit? [rpsq]')
    if rps == 'q':
        sys.exit(0)
    elif rps in 'rps':
        print(rps)
    else:
        raise ValueError(f'invalid input: {rps}')
except ValueError as err:
```

```
    sys.stderr.write(str(err))
```

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,

      ```
      try:
          assert rps in 'rps', 'input must be r,p,s'
          play_game(rps)
      except AssertionError as err:
          sys.stderr.write(str(err))
      ```
    - enclose the code in a try-except construct but catch the assertion type of exception (what i s it?) instead of ValueError. Handle it by writing the error message to sys.stderr also.

      AssertionError

2. Definitions and short answers - files

1. When opening a file using fh = open('filename'), why is it ok to omit the second parameter?

   because the by default the mode is 'r'
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')?

   'w' means to create if the file does not exist, or overwrite if it exists; 'a' means to open and append (add to the end) of a file, instead of overwriting it.
3. Once you opened a file as file handle named fh, how do you
   a. read one character at a time as a str fh.read(1)
   b. read 10 characters as a str fh.read(10)
   c. read one line as a str fh.readline() # singular
   d. read all the lines as a list of str fh.readlines() # plural
   e. read the entire file as one str fh.read()
4. When reading a file either one line at a time or a number of characters at a time, when do you kno w you have reached the **end of the file**?

   when the return value is '' (empty string)
5. To open a file named 'myfile' **for writing** (or overwrite it completely if already exists), how should you open the file?

   fh = open('myfile', 'w')
6. Once a file has been opened as file handle fh for writing, how should you **write** a string 'hello' to i t?

   fh.write('hello')
7. What is the difference between print(s) and sys.stdout.write(s)?

   print() can take arguments of different types and convert them to string if not already, and can take mult iple arguments separated by comma; it also adds a newline by default. On the other hand, sys.stdout.wri te() does not add the newline at the end of the line and requires its argument to be a single str, rather tha n converting non-str into str. write() takes only one argument.
8. When you are done with a file referenced by file handle fh, how do you **close** it?

fh.close()

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()
```

```
with open('filename', 'r') as fh:
    print('totally %d lines in file' % len(fh.readlines()))
```

The advantage of with-construct is the file is closed automatically when you exit the with statement. It defines the scope of the statements that uses the file handle.

10. If you finish reading a file (whose handle is fh) but want to start from beginning again, what shoul d you do without closing and reopening the file?

fh.seek(0)

11. After you have opened a file whose file handle is fh for reading or writing for a while, how do yo u find the **current position** in the file?

fh.tell()

12. What is a difference between input('') and sys.stdin.readline()?

input() removes the newline character whereas readline() keeps it

13. In a Unix-like shell such as bash (not Python shell), what do the following do?

    a. $ grep return *.py > result

    redirect the result of the 'grep return *.py' command into the file named result, by overwriting i t if it exists or creating it if it doesn't.

    b. $ grep return *.py >> resfile

    redirect the standard output of the command 'grep return *.py' into the file named resfile, by ap pending output to the end of resfile instead of overwriting it.

14. In a Unix-like shell, what is the difference between the commands

    $ wc -w filename

    and

    $ wc -w < filename

    ?

    the former passes filename as a command-line argument to the wc program and the wc program opens t he file.

    In the latter command, the shell redirects the content of filename into the wc program, which does not s ee the filename but just reads from standard-input (sys.stdin), but it does not know the shell redirects th e file from the file.

15. What does the following Unix shell command do?

    $ grep return *.py | wc

    the shell runs the command 'grep return *.py' to display those lines in all the .py files (in the current dir ectory) that match the string "return". the standard output of the grep command is then redirected to the standard input of the wc program to count words.

16. Why should you write to sys.stderr instead of sys.stdout to display a text-based error message, eve n though both appear on the same text terminal?

because if you need to redirect the standard output to a file or to a pipe then sys.stderr will not get redirected but still get displayed on the 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)?
    the name of the data type is bytes
18. How do you express the **literal** for a bytes data object consisting of ASCII characters 'h', 'e', 'l', 'l', 'o'?
    b'hello'
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?
    you need to say str(b'world', 'UTF8') because str works with Unicode, which can be encoded in different ways, but raw data bytes could be any one of those encodings. To properly convert raw data into str, the conversion function needs to know what encoding is used. By default, the most popular is 'UTF8' and is actually used by Python3 as well.
20. How do you convert from a str object denoted by textstring into bytes type?
    bytes(textstring, 'UTF8')
21. What is the meaning of the bytes literal b'\xe4\xbd\xa0\xe5\xa5\xbd'?
    it denotes five raw data bytes whose values are hex E4, BD, A0, E5, A5, BD.
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?
    module named os, function named os.getcwd(). code looks like
    ---
    import os
    print(os.getcwd())

3. Programming Exercise

1. 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 value

s 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:

a. You should **open** the text file for writing or overwriting.
b. You have several options to **write** the products to file. Be sure each line (including the last) ends on a newline.
c. 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.
d. 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.

Answer

```
import sys

if __name__ == '__main__':
    try:
        M_cand = int(sys.argv[1])
        M_plier = int(sys.argv[2])
        file_name = sys.argv[3]
        with open(file_name, 'w') as fh:
            for c in range(1,(M_cand + 1)):
                product_list = [ str(c*s) for s in range(1, (M_plier + 1)) ]
                fh.writelines(' '.join(product_list))
                fh.write('\n')
    except ValueError as err:
        sys.stderr.write(str(err))
    except OSError as err:
        sys.stderr.write(str(err))
```

2. 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

a. 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.
b. 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.
c. You need to keep a variable for error count. When printing the final message, check if err or is 1. If so, don't add the 's'; otherwise, add 's' to make it plural.

Answer

```
import sys

if __name__ == '__main__':
    Error_Count = 0
    try:
        M_cand = int(sys.argv[1])
        M_plier = int(sys.argv[2])
        file_name = sys.argv[3]
        with open(file_name, 'r') as fh:
            for c in range(1,(M_cand + 1)):
                line = fh.readline()
                p_list = line.split(" ")
                for i,s in enumerate(p_list, 1):
                    try:
                        s_num = int(s)
                        if (s_num != c * i):
                            sys.stderr.write(str(f'{c} x {i} = {s} is incorrect; should be {c*i}\n'))
```

```
                    Error_Count += 1
                except ValueError:
                    sys.stderr.write(str(f'{c} x {i} = {s} is badly formatted; should be {c*i}\n
'))

                    Error_Count += 1

        except ValueError as err:
            sys.stderr.write(str(err))
        except OSError as err:
            sys.stderr.write(str(err))

        if (Error_Count == 0):
            print("Multiplication table in file 5x7.txt is correct.")
        elif (Error_Count == 1):
            print("Multiplication table in file checkmul.py contains 1 error.")
        else :
            print(f"Multiplication table in file checkmul.py contains {Error_Count} errors.")
```

1. 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 (0.475 → 0.48).
   - 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!
$ _
```

Answer

```
import sys

if __name__ == '__main__':
        while True :

                exp_str = input("Enter an expression: ")
                if exp_str == "quit" :
                        print("Goodbye!")
                        break
                try:
                        exp_list = exp_str.split(" ")
                        var1 = int(exp_list[0])
                        var2 = int(exp_list[2])
                        op = exp_list[1]

                        if op == '+' :
                                print(f"Answer: {var1 + var2}")
                        elif op == '-' :
                                print(f"Answer: {var1 - var2}")
                        elif op == '*' :
                                print(f"Answer: {var1 * var2}")
                        elif op == '/' :
                                print("Answer: %.2f" % (var1 / var2))
                        else :
                                raise ValueError

                except ValueError,IndexError :
                        sys.stderr.write("Error: Invalid expression\n")
```

```
except ZeroDivisionError:
        sys.stderr.write("Error: Cannot divide by 0\n")
```