SC7

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?

> 💡 Logical errors (Using undefined variables)

2. What kind of exception do you get when you try z = 10 / 0?

> 💡 Logical errors (Divide by 0)

3. What is the same between ZeroDivisionError and OverflowError?

> 💡 Exception related to numbers (Both are logical errors.)

4. What kind of exception do you get when you run L = "hello world" print(L['5']) ? Why?

> 💡 TypeError: list indices must be integers or slices, not str

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] exception, IndexError
- L[-2]
- L[-5]
- L[-7] exception, IndexError

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] KeyError
- 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?

FileNotFoundError

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 FileNotFoundError:
    print('Cannot open file')
else:
    # file was successfully opened, continue with rest of program
    # for example, you can read the contents of the file or write to it here
    fh.close()
```

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'?

```
try:
    filenames = ['alpha', 'beta', 'gamma']
    filenum = input('select a file by typing 1, 2, or 3:')
    i = int(filenum)
    fh = open(filenames[i], 'r')
except:
    print('An error has occurred')
else:
    # file was successfully opened, continue with rest of program
    # for example, you can read the contents of the file or write to it here
    fh.close()
```

- How do you rewrite the code to **check each type of exception** and print an error message for each specific exception?

```
try:
    filenames = ['alpha', 'beta', 'gamma']
    filenum = input('select a file by typing 1, 2, or 3:')
    i = int(filenum)
    fh = open(filenames[i], 'r')
except ValueError:
    print('Invalid input. Please enter an integer between 1 and 3.')
except IndexError:
    print('Invalid index. Please enter an integer between 1 and 3.')
except FileNotFoundError:
```

```
    print('File not found. Please check the filename and try again.')
except:
    print('An error has occurred')
else:
    # file was successfully opened, continue with rest of program
    # for example, you can read the contents of the file or write to it here
    fh.close()
```

11. Given the following program

```
try:
  x = int(input('enter num1:'))
  y = int(input('enter num2:'))
  z = x / y
except ValueError:
  z = 0
except ZeroDivisionError:
  z = x
print(z)
```

- If an exception occurs on line 2, what lines of code are executed next?

If an exception occurs on line 2, the program will jump straight to the `ValueError` `except` block on line 5.

- If an exception occurs on line 3, what lines of code are executed next?

If an exception occurs on line 3, the program will jump straight to the `ValueError` `except` block on line 5.

- If an exception occurs on line 4, what lines of code are executed next?

If an exception occurs on line 4, the program will jump straight to the `ZeroDivisionError` `except` block on line 7.

- If line 6 is execute, is it possible that lines 7-8 are also executed immediately after?

💡 No, if line 6 is executed, it means that a `ValueError` occurred on line 2, and the program will jump straight to the `except` block on line 5. Line 7 and 8 will not be executed in this case.

- If either line 6 or line 8 is executed, does line 9 also get executed next?

💡 If either line 6 or line 8 is executed, line 9 will also be executed next. The value of `z` will be printed to the standard output, regardless of how it was computed.

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?

💡 Line 3 can cause a KeyError if the input value is not one of the keys in the dictionary greek.
Line 4 can cause a ValueError if the user inputs a non-integer value. Line 5 can cause a ZeroDivisionError if the input value of y is 0.

- 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 a zero division error occurs on line 5, line 13 will not be executed because the `ZeroDivisionError` `except` block on line 10 will be triggered instead. The exception is handled by the `except` block on line 10.

- If there is no exception by the end of line 5, which print statement or statements are executed next?

> 💡 If there is no exception by the end of line 5, line 12 will be executed next, which will print the value of `z`.

- 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?

> 💡 except (KeyError, ValueError):
> 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?

💡 print(f’error: {str(OSError)}’)

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 stderr.

```
1 try:
2   import sys
3   rps = input('Rock, Paper, Scissors, or Quit? [rpsq]')
4   if rps == 'q':
5      sys.exit(0)
6   if rps in {'r', 'p', 's'}:
7      play the game(rps)
8 except ValueError as err:
    sys.stderr.write(f'{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,
- 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 stderr also.

```
import sys

try:
    rps = input('rock, paper, scissors, or quit? [rpsq]')
    assert rps in 'rpsq', 'Invalid input'
    if rps == 'q':
        sys.exit(0)
    else:
        play_game(rps)
except AssertionError as err:
    sys.stderr.write(str(err))
```

2. Definitions and short answers - files

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

💡 If you omit the second parameter, Python will assume a default value of `'r'`, which means the file is opened in read mode.

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')?

💡 overwritten / append

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()
    d. read all the lines as a list of str: fh.readlines()
    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 know you have reached the **end of the file**?

💡 The `read()` method returns an empty string (`''`) when it reaches the end of the file. So you can check whether the return value of `read()` is an empty string to know whether 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?

💡 fh.write('hello')

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?

> 💡 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()
```

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?

> 💡 fh.seek(0) # move the file pointer back to the beginning of 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?

> 💡 fh.tell() # get the current position of the file pointer

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?
    a. $ grep return *.py > result
    b. $ grep return *.py >> resfile
14. In a Unix-like shell, what is the difference between the commands $ wc -w filename and $ wc -w < filename ?

💡 The difference between these two commands is that the first command reads the file directly, while the second command redirects the contents of the file to `wc`'s standard input. The effect of this difference is generally negligible for small files, but may become significant for very large files, where the first command may cause the operating system to swap memory, while the second command avoids this. Additionally, the second command may be useful when the input is not a regular file but a pipe or a process substitution.

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)?

💡 `bytes` type

```
# Open binary file for reading
with open('myfile.bin', 'rb') as bh:
    data = bh.read()  # returns a bytes object

# Open binary file for writing
with open('myfile.bin', 'wb') as bh:
    bh.write(b'hello')  # writes a bytes object to the file
```

18. How do you express the **literal** for a bytes data object consisting of ASCII characters 'h', 'e', 'l', 'l', 'o'?

💡 my_bytes = 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?

> 💡 When you try to convert a bytes object to a string using `str(b'world')`, you will get a string with the bytes representation of the object, not the string value you expect. This is because bytes and strings are fundamentally different data types in Python, and they use different encodings.

```
my_bytes = b'world'
my_string = my_bytes.decode('utf-8')
```

20. How do you convert from a str object denoted by textstring into bytes type?

> 💡 To convert a `str` object into a `bytes` object, you can use the `encode()` method, which encodes the string using a specified encoding and returns a bytes object. For example, to encode a string using the UTF-8 encoding, you can do the following:

```
my_string = "hello world"
my_bytes = my_string.encode('utf-8')
```

21. What is the meaning of the bytes literal b'\xe4\xbd\xa0\xe5\xa5\xbd'?

> 💡 The bytes literal `b'\xe4\xbd\xa0\xe5\xa5\xbd'` represents a sequence of bytes in the UTF-8 encoding that corresponds to the Chinese phrase "你好"

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?

💡 In Python, you can use the `os` module and the `os.getcwd()` function to get the path to the current working directory.

```
import os

cwd = os.getcwd()
print("Current working directory:", cwd)
```