Learning Quiz 14: Errors and Exceptions

Due Oct 16 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Prior to completing this quiz, be sure to read:

- Section 4.4, Errors and Exceptions (p. 116-119)
- Section 7.3, Exceptional Control Flow (p. 215-223)

Please also go over Practice Problems 4.11, 7.3, and 7.4 in the textbook (solutions at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	11 minutes	1.78 out of 5

Score for this attempt: **1.78** out of 5
Submitted Oct 15 at 11:52pm
This attempt took 11 minutes.

Question 1

0 / 1 pts

Errors can be categorized into three types:

- **Syntax** - errors due to the incorrect format of a Python statement; these errors occur before the program runs and while the statement or program is being translated into machine language. In other words, Python doesn't understand what you just typed or there's something about the formatting that's unexpected. Examples include:

```
print[2]   ## needs parenthesis instead of bracket
```

```
if 4 = 5:  ## two equal signs
```

```
mytuple = (3. 4. 5)    ## needs comma instead of period
```

- **Runtime** - errors because the program gets into an erroneous state during execution; these errors occur after program has started running but prevent the program from completing. In other words, the syntax in Python is correct, but the statement execution reaches an invalid state. Examples include:

```
a = "hello" + 5   ## cannot add string and integer
```

```
pets = ['cat', 'dog', 'fish']
pets[3]
```

```
b = 5/0          ## cannot divide by 0
```

- **Logical** - errors that occur unintentionally even though the program runs to completion. These errors are the most challenging to catch because Python will not know to tell you that there's an error. Examples include:

```
def sumFun(x,y):
    return(x-y)   ## suppose the programmer intended for this to be +
```

```
pets = ['cat', 'dog', 'fish']
firstpet = pets[1]  ## suppose the programmer wanted 'cat'
```

Consider the following program:

```
def sqSum(x,y):
    'returns the sum of squares'
    print(x**2 + y**2)

z = sqSum(2,3)
```

Using the class definition of errors, what type of error best describes the program above.

You Answered

- (●) no error

Correct Answer

- ( ) logical
- ( ) syntax
- ( ) runtime

Question 2

0.78 / 1 pts

In Section 8.6, you will find that you can define your own exceptions. However, as you read in Section 4.4, Python has some built-in exceptions. Refer to Table 4.6 on page 118 for a list of common exception types.

Match the following exceptions to their corresponding definition or *potential* example (think about what kind of error you *might* get).

Correct!

KeyboardInterrupt — User stops a running progran

Correct!

OverflowError — 3.0**10000

Correct!	**ZeroDivisionError**	a = 7/0
Correct!	**IOError**	file1 = open('rster.csv')
Correct!	**IndexError**	Programmer tries to access e
Correct!	**NameError**	Programmer tries to access v
You Answered	**TypeError**	int('one')
	Correct Answer	3 + 'hello'
You Answered	**ValueError**	3 + 'hello'
	Correct Answer	int('one')
Correct!	**SyntaxError**	for = 4

Question 3

1 / 1 pts

So far we have written programs where we assume that the user will follow the rules and given input in an ideal format. Given the ideal case, our program runs without issue. However, what if the user spells out his age or enters a file that does not exist? Rather than crashing, it would be great if we could have a back-up plan for unexpected situations.

Use try and except statements to handle potential errors.

In the following code, the program will run smoothly and successfully open the file if the user enters the name of a file that exists. Rather than crashing, it will print "We can't seem to..." if the user enters an invalid file name.

```
try:
    userinput = input('Please enter the file name: ')
    file = open(userinput)
except:
    print("We can't seem to open", userinput)
```

Consider the following program:

```
try:
    userinput = eval(input('Please enter your age: '))
except:
    print('Invalid input!')
```

What user input will generate a 'Invalid input!' message?

- [ ] 2,323
- [ ] 70
- [ ] -3
- [ ] 9765,321543
- [ ] 'ten'
- [x] one — Correct!
- [ ] 12345
- [x] x — Correct!
- [ ] 42.0
- [x] refused — Correct!

Question 4

0 / 1 pts

Let's say we know what type of error might occur and we only want to catch that error. You can specify in your except statement what type of error you expect to catch. See below for an example and compare to Question 3:

```
try:
    userinput = input('Please enter the file name: ')
    file = open(userinput)
except IOError:
    print("The file", userinput, "does not exist.")
```

Suppose the user will interact with the program from Question 3 as follows:

```
Please enter your age: refused
```

Update the code below by filling in the blank with the type of error we expect to catch:

```
try:
    userinput = eval(input('Please enter your age: '))
```

```
except ValueError:
    print('Invalid input!')
```

Answer 1:

You Answered: ValueError

Correct Answer: NameError

Question 5

0 / 1 pts

Suppose you would like your program to handle different errors in different ways. You can use multiple except statements to handle the different errors.

```
try:
    ## code blah blah blah
    ..............
except OverflowError:
    print('Overflow error')
except ZeroDivisionError:
    print('Cannot divide by 0')
except IOError:
    print('File does not exist')
except:
    print('all other errors')
```

Depending on the type of error AND which error occurs first, the program will print different statements. Of course, you can modify the code so that it does more than just print.

Consider the following lines of code and user-interaction.

```
try:
    value1 = eval(input('Please enter the first number: '))
    value2 = eval(input('Please enter the second number: '))
    print("The quotient of the two numbers is: " + value2/value1)
except OverflowError:
    print('Overflow error')
except ZeroDivisionError:
    print('Cannot divide by 0')
except NameError:
    print('Please enter a number')
except TypeError:
    print('Type error')
except:
    print('Something went wrong...')

--------------------------------------

Please enter the first number: False
Please enter the second number: 5.5**100000
```

What will the program print next?

Correct Answer

- Overflow error
- The quotient of the two numbers is:
- Something went wrong...

Please enter a number

Type error

You Answered

Cannot divide by 0

Quiz Score: **1.78** out of 5