CS 88 Fall 2018
Prof. David Culler

John Yang
Friday 2-4 PM

Exceptions

Link to Discussion Slides: https://goo.gl/Rv8KAC
Link to Relevant Reading: http://composingprograms.com/pages/33-exceptions.html

1 Exceptions

1.1 Which Error?

Within each blank, fill out the type of error that would show up if the line of Python was run. If the given line does not produce an error, simply write "No Error". Assume that all variables and identifiers in each statement that have not been explicitly declared do not exist (references don't point at actual values, objects, or modules).

1. >>> 3 + "hello"
   _____
2. >>> import nonexistentmodule
   _____
3. >>> open("imaginaryfile.txt")
   _____
4. >>> print(nonexistent)
   _____
5. >>> a = "key" : "value"
   >>> a["not a key"]
   _____
6. >>> abs("hello")
   _____
7. >>> def f(): f()
   >>> f()
   _____

1.2 Error Proofing!

Given the following functions, fill in the "try-catch" blocks with the appropriate errors and operations so that the function 1. executes correctly when the inputs are appropriate 2. catches and handles the error when the inputs are inappropriate. Before implementing, think about what error could result what kinds of inputs to these functions.

1.2.1 Division

```
def division(a, b):
    try
        return _____________
    except _____________ as e:
        print(_____________ Caught! - , e)
        return 0
    else:
        return _____________
```

1.2.2 Addition or Concatenation

```
def adder(a, b):
    try:
        return _____________
    except _____________ as e:
        print(_____________ Caught!, e)
        return
    else:
        return _____________
```

1.3 Exception Handling Design

Sometimes, it's possible that the code we write could go wrong in more than one way. In other words, we could generate more than one error! The syntax for such handling would be as follows:

```
try:
    <insert code here>
catch (SpecificErrorOne, SpecificErrorTwo) as error:
    <handle error here>
```

With this in mind, let's think about how we can error proof the following section of code.!

1.3.1 Part 1: Handle All the Exceptions

Given the following code, what kinds of errors should we look out for if we place the entire block of code within a "try-catch" block? How would you recommend going about handling such errors? This problem is very high level and conceptual, don't worry too much about figuring out a formulaic pattern to getting the solution. In fact, the solution for this problem just demonstrates one of many, many possibilities, so really focus on diagnosing the nuances of each operation here!

```
def mergeDicts(a, b):
    try:
        for key in a:
```

```
        a[key], b[key] = a[key] + b[key], a[key] / b[key]
    return a, b
# List different possible errors on the following 1 line
except (_____ as error:
```

_____

_____

_____

_____

1.3.2 Part 2: Test Cases

A common practice in the software engineering world is writing tests for your code. Now that you've successfully error-proofed your code, what kinds of test cases could you write to ensure that your code either works or catches the anticipated exceptions correctly?

In this problem, the test cases for the above problem are written out for you. For each test case, determine whether the input would cause an error, and if so, which one?

a, b = mergeDicts("a" : 3, "b" : 2, "a" : 6, "b" : 4) _____

a, b = mergeDicts("a" : 3, "b" : 2, "a" : "c", "b" : "d") _____

a, b = mergeDicts("a" : 3, "b" : 2, "a" : 1, "b" : 0) _____

a, b = mergeDicts("a" : 3, "b" : 2, "c" : 1, "d" : 0) _____