Did You Catch That?

This exercise will give you practice following the flow of control when a program throws (or raises) an exception.

The type of an exception provides information about the kind of error that has occurred—e.g. a NameError indicates that the program has used a name that is not in scope; an IndexError indicates that the program tried to access a list element using an index that is too large or too small; a Key Error indicates that the program tried to access a value in a dictionary using a key that does not exist in the dictionary; etc.

Python raises many exceptions automatically. But it also provides a raise instruction, which you can use in your program to cause the exception to be raised. For example,

```python
raise TypeError
```

will throw a TypeError when it is executed.

Python also allows you to define new exception types by subclassing the Exception class. For example:

```python
class GreenError (Exception):
  pass
```

defines a new exception type, which your program can then throw using a raise statement.

Recall how Python executes a try/except statement:

- It will “try” to execute the try-suite.
- If no error occurs then it skips over all the subsequent except blocks.
- However, if an exception is thrown, Python stops executing the try-suite and looks for an except block to handle the exception.
  - If the exception thrown is an instance of an exception type that appears in the header of an except block, Python executes the associated suite and skips over any remaining except blocks.
  - Otherwise, Python stops executing the suite that contains the try/except statement, and (re-)throws the exception at the point where this suite was called.

This re-throwing of the exception at the call site is called propagating the exception. If the exception propagates all the way up to the top-level frame, Python stops execution of the whole program and prints the exception.

To give you a better understanding of these concepts, you’ll act out the code on the next page. To get you started, your Team Leader will give “control” to someone in your group and charge the student with simulating (acting out) the call to main. Simulation of the program proceeds as follows.
To simulate (act out) execution of a function or method:

- Upon receiving control, stand up and start simulating execution of the body.
- For a print statement, just call out what is printed.
- To call a method, hand control to a new student and charge this student with executing an activation of the method; BUT REMAIN STANDING (since you are still activated) and REMEMBER WHAT METHOD CALL YOU ARE EXECUTING.
- When control returns from a method call, continue executing the body from the site of the call.
- To return control normally, give control back to the student simulating the function or method that activated you and sit down.
- In the try-suite of an activation of \texttt{fraz}, your Team Leader will throw one of three possible errors: \texttt{GreenError}, \texttt{BlueError}, or \texttt{RedError}. The student acting out execution of the try-suite will either catch the exception or propagate it.
- To propagate an exception, give control back to the student simulating the function or method that activated you, throw the exception to that person, and sit down.
- If an exception propagates up to \texttt{main} and \texttt{main} does not catch it, drop the exception on the floor.

Repeat this exercise a few times. Your Team Leader will choose different exceptions for \texttt{fraz} to throw and different students to act out the simulations.
class GreenError (Exception):
    pass

class BlueError (Exception):
    pass

class RedError (Exception):
    pass

def foo():
    print("starting foo")
    try:
        bar()
    except BlueError:
        print("foo caught BlueError")
    bar()
    print("returning from foo")

def bar():
    print("starting bar")
    try:
        fraz()
    except GreenError:
        print("bar caught a GreenError")
    print("returning from bar")

def fraz():
    print("starting fraz")
    try:
        # raise ???Error -- to be determined by the peer leader
        print("after exception in fraz")
    except RedError:
        print("fraz caught a RedError")
    print("returning from fraz")

def main():
    try:
        foo()
        bar()
        fraz()
    except BlueError:
        print("main caught a BlueError")
    except RedError:
        print("main caught a RedError")
    print("returning from main")

main()