File Input/Output and Exceptions

Introduction to Programming

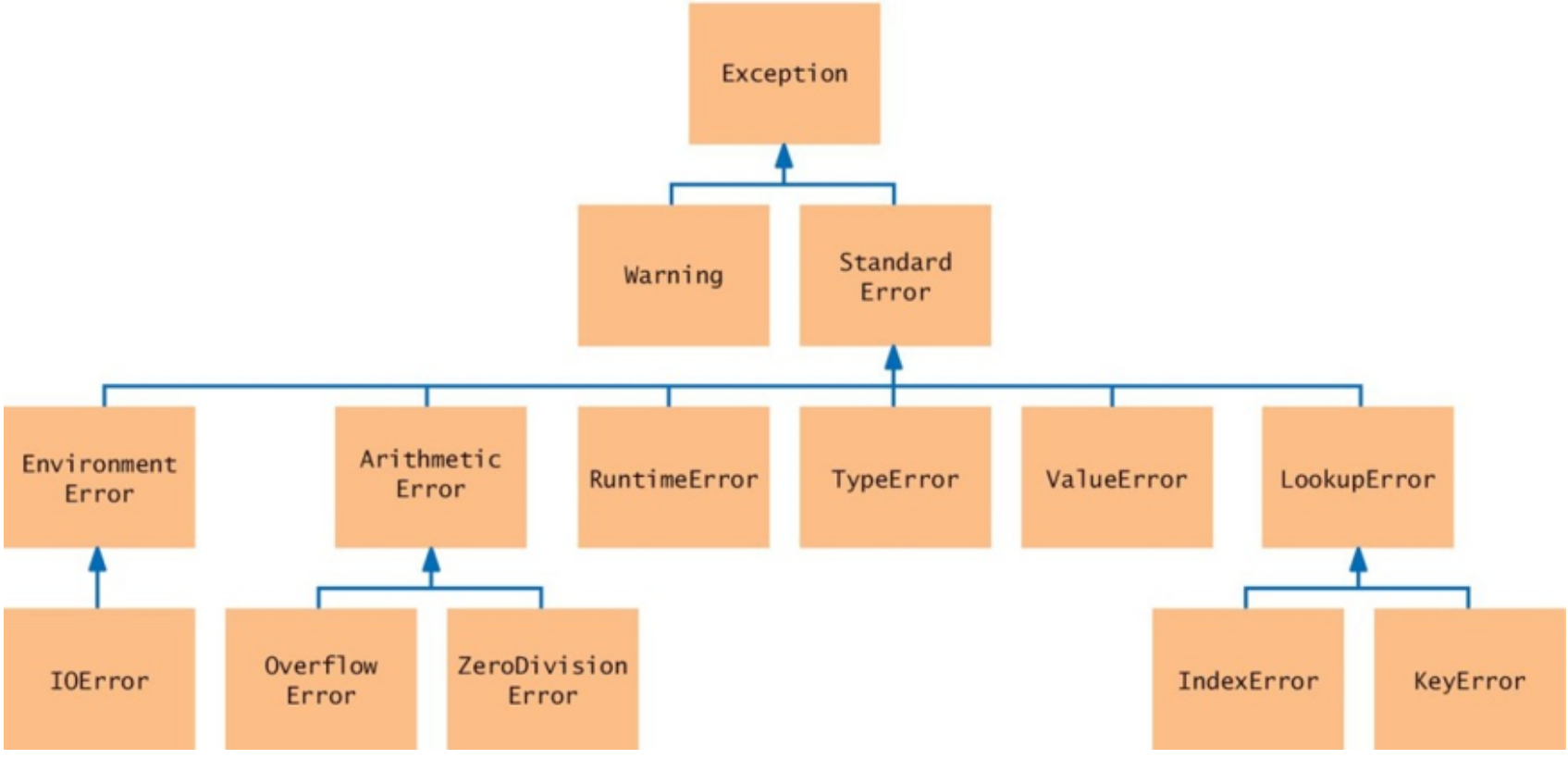
Rae Harbird, r.harbird@ucl.ac.uk

When do exceptions happen?

- When the Python interpreter encounters a problem that it cannot deal with, it will throw an exception and you will see the errors in the console.

```
>>> int(input("Enter a number:" ))
Enter a number:a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
>>>
```

Python exceptions



From: Python for Everyone, Horstmann and Necaise

Examples

- ZeroDivisionError

```
num_prizes = int(input("Enter the number of prizes: "))
num_students = int(input("Enter the number of students: "))
prizes_per_student = num_prizes / num_students
print("Number of prizes per student: ", prizes_per_student)
```

- Output:

```
Traceback (most recent call last):
  File "C:/Users/Rae/.PyCharmEdu2019.2/config/scratches/scratch_13.py",
    line 3, in <module>
    prizes_per_student = num_prizes / num_students
ZeroDivisionError: division by zero
```

Examples (continued)

- IndexError

```
lis = ["cat", "dog", "hamster", "dromedary"]

for i in range(1, len(lis) + 1):
    print(lis[i])
```

- Output:

```
dog
Traceback (most recent call last):
  File "C:/Users/Rae/.PyCharmEdu2019.2/config/scratches/scratch_14.
  line 4, in <module>
hamster
    print(lis[i])
IndexError: list index out of range
dromedary
```

Examples (continued)

- KeyError

```
dict = {"cat": "meeow!", "dog": "woof!", "hamster": "eek!",
        "dromedary": "hurrumph!"}

dict.pop("pig")
```

- Output:

```
Traceback (most recent call last):
  File "C:/Users/Rae/.PyCharmEdu2019.2/config/scratches/scratch_15.py",
    line 2, in <module>

    dict.pop("pig")
KeyError: 'pig'
```

Catching an exception

- Surround the code that might fail with a `try ... catch` block. Here is an example showing how your code could work using the python REPL:

```
>>> try :
...   int(input("Enter a number: "))
... except ValueError :
...   print("That is not a number.")
...
Enter a number: g
That is not a number.
>>>
```

Catching an exception (cont.)

- In a program, your code could look like this:

```
##  Checks whether the string passed as a parameter is an integer.
#
#   @return True or False.
#
def isInteger(number) :
    try:
        int(number)
        return True
    except ValueError:
        return False
```

- The point here is that you don't have to terminate the program, you can choose what action to take if a value is invalid.

Catching more than one exception

You should write code that allows for code to catch multiple exceptions. Look at src/analyzedata.py

```
done = False
while not done :
try :
    filename = input("Please enter the file name: ")
    data = readFile(filename)

    # As an example for processing the data, we compute the sum.
    ...

except IOError :
    print("Error: file not found.")

except ValueError :
    print("Error: file contents invalid.")

except RuntimeError as error :
    print("Error:", str(error))
```

Code that will generate an exception

Following on from the previous page, the code that will generate and exception looks like this:

```
def readData(inFile) :
    line = inFile.readline()
    numberOfValues = int(line)  # May raise a ValueError exception.
    data = []

    for i in range(numberOfValues) :
        line = inFile.readline()
        value = int(line)       # May raise a ValueError exception.
        data.append(value)

    # Make sure there are no more values in the file.
    line = inFile.readline()
    if line != "" :
        raise RuntimeError("End of file expected.")

    return data
```

. . . and finally

- There may be some actions that you want to perform whether your code generated an exeption or not. This is the purpose of the `finally` clause in a `try` block:

```
## Opens a file and reads a data set.
#  @param filename the name of the file holding the data
#  @return a list containing the data in the file
#
def readFile(filename) :
    inFile = open(filename, "r")
    try :
      return readData(inFile)
    finally :
      inFile.close()
```

- In this case we **always**, **always** want to close a file before ending the program.

Raising an exception

- You can raise your own exceptions. You might want to do this when your program encounters a value which fails your validation tests:
- In `analyzedata.py`, an exception is raised when there is unaccounted for data at the end of the file:

```
# Make sure there are no more values in the file.
line = inFile.readline()
if line != "" :
   raise RuntimeError("End of file expected.")

if bankBalance - withdrawal < 0:
  raise ValueError("Sorry, you cannot go overdrawn on
                    this account")
```