Exception Handling

SECTION 7.4

Exception Handling

- There are two aspects to dealing with run-time program errors:
  1) Detecting Errors
  2) Handling Errors
- The open function can detect an attempt to read from a non-existent file
  - The open function cannot handle the error
  - There are multiple alternatives, the function does not know which is the correct choice
  - The function reports the error to another part of the program to be handled
- Exception handling provides a flexible mechanism for passing control from the point of the error to a **handler** that can deal with the error

Detecting Errors

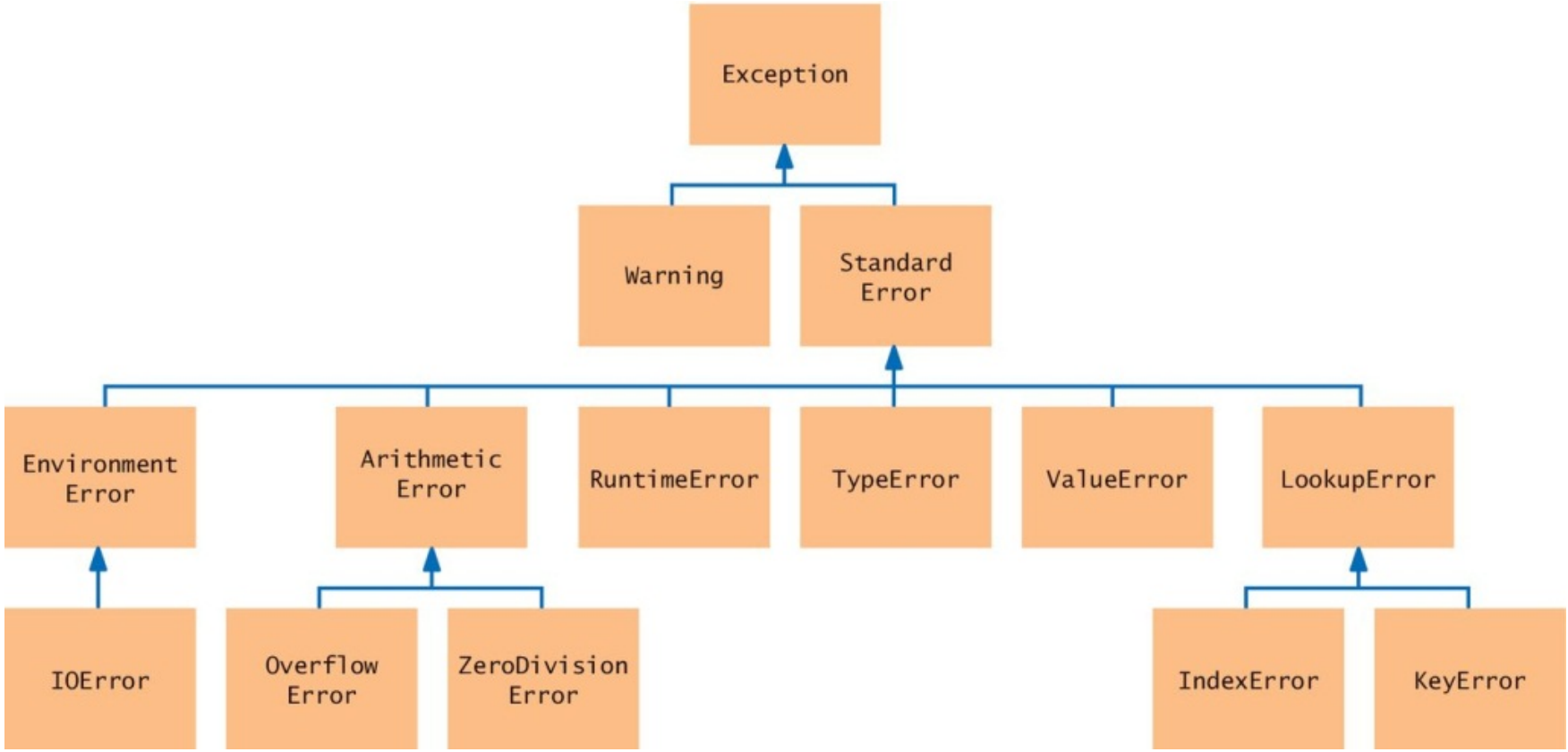
- What do you do if someone tries to withdraw too much money from a bank account?
- You can **raise** an exception
- When you **raise** an exception, execution does not continue with the next statement
  - It transfers to the **exception handler**

Use the raise statement to signal an exception

```
if amount > balance :
    raise ValueError("Amount exceeds balance")
```

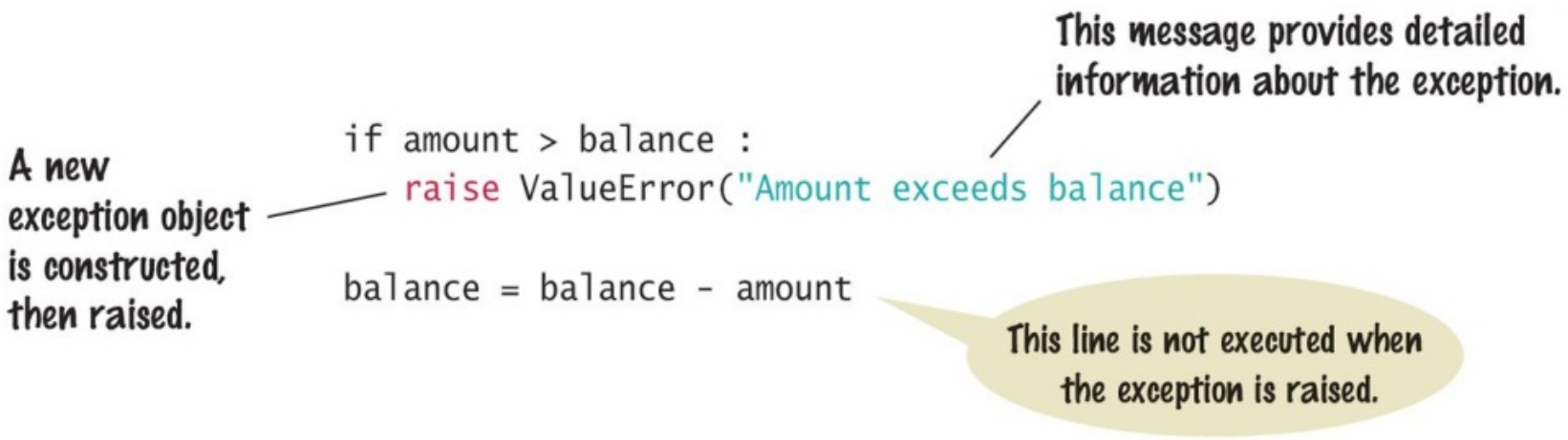
Exception Classes (a subset)

- Look for an appropriate exception



Syntax: Raising an Exception

Syntax `raise` *exceptionObject*



Handling Exceptions

- Every exception should be handled somewhere in the program
- This is a complex problem
  - You need to handle each possible exception and react to it appropriately
- Handling recoverable errors can be done:
  - Simply: exit the program
  - User-friendly: Ask the user to correct the error

Handling Exceptions: Try-Except

- You handle exceptions with the try/except statement
- Place the statement into a location of your program that knows how to handle a particular exception
- The try block contains one or more statements that may cause an exception of the kind that you are willing to handle
- Each except clause contains the handler for an exception type

Syntax: Try-Except

Syntax

```
try :
   statement
   statement
   . . .
except ExceptionType :
   statement
   statement
   . . .
except ExceptionType as varName :
   statement
   statement
   . . .
```

This function can raise an IOError exception.

```
try :
   infile = open("input.txt", "r")

   line = inFile.readline()
   process(line)

except IOError :
   print("Could not open input file.")

except Exception as exceptObj :
   print("Error:", str(exceptObj))
```

When an IOError is raised, execution resumes here.

Additional except clauses can appear here. Place more specific exceptions before more general ones.

This is the exception object that was raised.

Try-Except: An Example

```
try :
    filename = input("Enter filename: ")
    infile = open(filename, "r")
    line = infile.readline()
    value = int(line)
. . .
except IOError :
    print("Error: file not found.")
except ValueError as exception :
    print("Error:", str(exception))
```

open() can raise an IOError exception

int() can raise a ValueError exception

Execution transfers here if file cannot be opened

Execution transfers here if the string cannot be converted to an int

If either of these exceptions is raised, the rest of the instructions in the try block are skipped

Example

- If an IOError exception is raised, the **except** clause for the IOError exception is executed
- If a ValueError exception occurs, then second **except** clause is executed
- If any other exception is raised it will not be handled by any of the **except** blocks

Output Messages

- When the body of this handler is executed, it prints the message included with the exception

```
except ValueError as exception :
    print("Error:", str(exception))
```

- For example, if the string passed to the int() function was "35x2", then the message included with the exception will be:

  invalid literal for int() with base 10: '35x2'

Output Messages (2)

- To obtain the message, we must have access to the exception object itself
- You can store the exception object in a variable with the as syntax:

```
except ValueError as exception :
```

- When the handler for ValueError is executed, **exception** is set to the exception object. In our code, we then obtain the message string by calling **str(exception)**

Source of Output Messages

- When you raise an exception, you can provide your own message string. For example, when you call

```
raise ValueError("Amount exceeds balance")
```

- The message of the exception, `"Amount exceeds balance"`, is the string provided as the argument to the constructor

The finally Clause

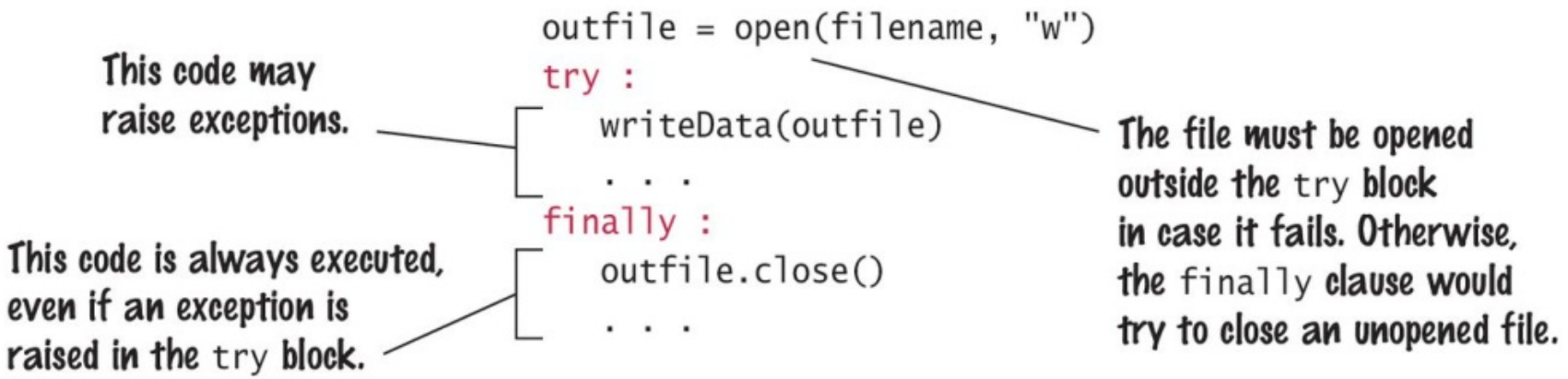
- The finally clause is used when you need to take some action whether or not an exception is raised
- Here is a typical situation
  - It is important to always close an output file whether or not an exception was raised (to ensure that all output is written to the file)
  - Place the call to close() inside a finally clause:

```
outfile = open(filename, "w")
try :
    writeData(outfile)
finally :
    outfile.close()
```

Syntax: The Finally Clause

Syntax

```
try :
   statement
   statement
   . . .
finally :
   statement
   statement
   . . .
```

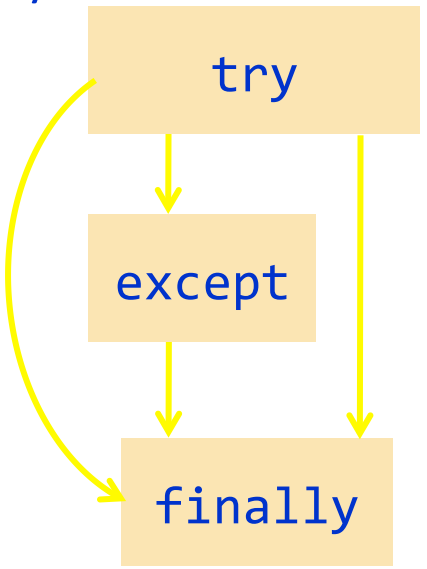


Programming Tip

- Throw exceptions early
  - When a method detects a problem that it cannot solve, it is better to throw an exception rather than try to come up with an imperfect fix
- Catch exceptions late
  - Conversely, a method should only catch an exception if it can really remedy the situation
  - Otherwise, the best remedy is simply to have the exception propagate to its caller, allowing it to be caught by a competent handler

Programming Tip

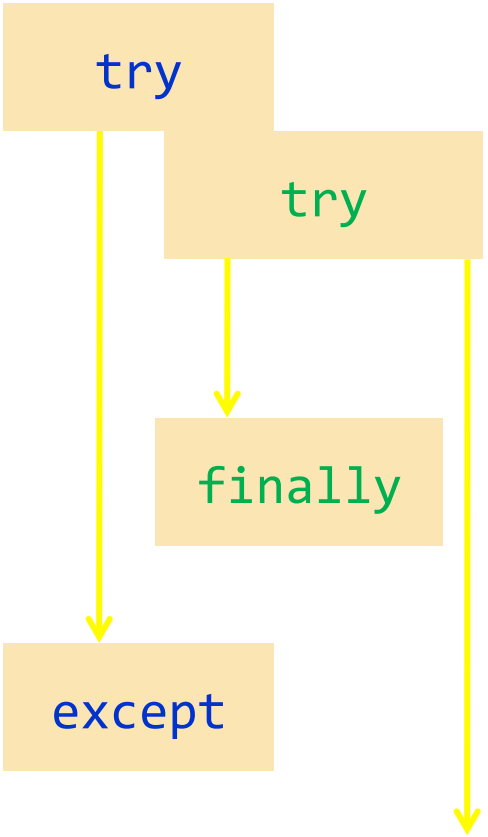
- Do not use except and finally in the same try block
  - The finally clause is executed whenever the try block is exited in any of three ways:
    1. After completing the last statement of the try block
    2. After completing the last statement of a except clause, if this try block caught an exception
    3. When an exception was raised in the try block and not handled



Programming Tip

- It is better to use two (nested) `try` clauses to control the flow

```
try :
  outfile = open(filename, "w")
  try :
    # Write output to outfile
  finally :
    out.close()  # Close resources
except IOError :
  # Handle exception
```



The `With` Statement

- Because a try/finally statement for opening and closing files is so common, Python has a special shortcut:

```
with open(filename, "w") as outfile :
    Write output to outfile
```

- This with statement opens the file with the given name, sets outfile to the file object, and closes the file object when the end of the statement has been reached or an exception is raised

Handling Input Errors

SECTION 7.6

Handling Input Errors

- File Reading Application Example
  - Goal: Read a file of data values
    - First line is the count of values
    - Remaining lines have values
  - Risks:
    - The file may not exist
      - The open() function will raise an exception when the file does not exist
    - The file might have data in the wrong format
      - When there are fewer data items than expected, or when the file doesn't start with the count of values, the program will raise a ValueError exception
      - Finally, when there are more inputs than expected, a RuntimeError exception should be raised

```
3
1.45
-2.1
0.05
```

Handling Input Errors: main()

- Outline for method with all exception handling

```
done = False
while not done :
  try:
    # Prompt user for file name
    data = readFile(filename)  # May raise exceptions
    # Process data
    done = true;
  except IOError:
      print("File not found.")
  except ValueError :
      print("File contents invalid.")
  except RuntimeError as error:
      print("Error:", str(error))
```

Handling Input Errors: readFile()

- Creates the file object and calls the readData() function
- No exception handling (no except clauses)
- finally clause closes file in all cases (exception or not)

```
def readFile(filename) :
    inFile = open(filename, "r") # May throw exceptions
    try
      return readData(inFile)
    finally
      in.close()
```

Handling Input Errors: `readData()`

- No exception handling (no try or except clauses)
- raise creates an ValueError exception and exits
- RuntimeError exception can occur

```
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()
    # Extra data in file
    if line != "" :
        raise RuntimeError("End of file expected.")
    return data
```

One Scenario

1. main calls readFile
2. readFile calls readData
3. readData calls int
4. There is no integer in the input, and int raises a ValueError exception
5. readData has no except clause; it terminates immediately
6. readFile has no except clause; it terminates immediately after executing the finally clause and closing the file
7. The IOError except clause is skipped
8. The ValueError except clause is executed

Example Code

- Open the file analyzedata.py

Summary: File Input/Output

- When opening a file, you supply the name of the file stored on disk and the mode in which the file is to be opened
- Close all files when you are done processing them
- Use the readline() method to obtain lines of text from a file
- Write to a file using the write() method or the print() function

Summary: Processing Text Files

- You can iterate over a file object to read the lines of text in the file
- Use the rstrip() method to remove the newline character from a line of text
- Use the split() method to split a string into individual words
- Read one or more characters with the read() method

Command Line Arguments

- Programs that start from the command line receive the command line arguments in the argv list defined in the sys module

Summary: Exceptions (1)

- To signal an exceptional condition, use the raise statement to raise an exception object
- When you raise an exception, processing continues in an exception handler
- Place the statements that can cause an exception inside a try block, and the handler inside an except clause
- Once a try block is entered, the statements in a finally clause are guaranteed to be executed, whether or not an exception is raised

Summary: Exceptions (2)

- Raise an exception as soon as a problem is detected
  - Handle it only when the problem can be handled
- When designing a program, ask yourself what kinds of exceptions can occur
- For each exception, you need to decide which part of your program can competently handle it