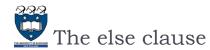


COMPSCI 105 S1 2017 Principles of Computer Science

Exceptions 2





- ▶ Executed only if the try clause completes with no errors
 - It is useful for code that must be executed if the try clause does not raise an exception.

```
try:
    statement block here
except:
    more statements here (undo operations)
else:
    more statements here (close operations)
```

2 COMPSCI 105 Lecture08



 ${\tt Example07.py}$

```
try:
    age = int(input("Please enter your age: "))
except ValueError:
    print("Hey, that wasn't a number!")
else:
    print("I see that you are %d years old." % age)
```

Please enter your age: 4
I see that you are 4
years old.

Please enter your age: a Hey, that wasn't a



Exercise 1

▶ What is the output of the following code fragment?

```
try:
    my_list = [1, 2, 3]
    num = int(input('Enter an index: '))
    value = my_list[num]
except IndexError:
    print("Invalid index!")
else:
    print(value)
print("DONE")
```

- Cases:
 - ▶ Enter an index: I
 - ▶ Enter an index: 6

COMPSCI 105 Lecture08

COMPSCI 105 Lecture08





- ▶ The finally block is optional, and is not frequently used
- Executed after the try and except blocks, but before the entire tryexcept ends
- ➤ Code within a finally block is <u>guaranteed</u> to be executed if any part of the associated try block is executed regardless of an exception being thrown or not
 - It allows for <u>cleanup</u> of actions that occurred in the try block but may remain undone if an exception is caught
 - Often used with files to close the file

```
try:
    statement block here
except:
    more statements here (undo operations)
finally:
    more statements here (close operations)
```

5 COMPSCI 105 Lecture08

x = divide(2, 1)
print(x)

result is 2.0
finally clause
2.0

Example

6 COMPSCI 105 Lecture08



Example08.py

```
def divide(a, b):
    try:
        result = a / k
    except ZeroDivisionError:
        result = 'Divided by zero'
    else:
        print("result is", result)
    finally/
        print("finally clause")
    return result
```

Case 2:

Divided by zero

```
x = divide(2, 0)
print(x)
```

finally clause Divided by zero



Example08.py

```
def divide(a, b):
    try:
        result = a / b

    except ZeroDivisionError:
        result = 'Divided by zero'
    else:
        print("result is", result)
    finally:
        print("finally clause")
    return result
```

```
Case 3:
```

▶ Case I:

No error

Other error

```
x = divide('2', '1')
print(x)
```

```
finally clause
Traceback (most ...
TypeError: unsupported operand type(s) ...
```

COMPSCI 105 Lecture08

COMPSCI 105 Lecture08



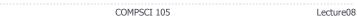
▶ What is the output of the following code fragment?

```
try:
    age = int(input("Please enter your age: "))
except ValueError:
   print("Hey, that wasn't a number!")
    print("I see that you are %d years old." % age)
finally:
   print("It was really nice talking to you. Goodbye!")
```

Cases:

- Please enter your age: a
- ▶ Please enter your age: I
- ▶ Please enter your age: 4

9





FileNotFoundError & IOError

Raised when an input/output operation fails, such as the print statement or the open function when trying to open a file that does not exist.

```
Example: input_file = open ("numbers1.txt", "r")
             print ("Reading from file numbers.txt")
             one line = input file.readline()
             print(one line)
             print ("Completed reading of file input.txt")
             input file.close()
```

▶ FileNotFoundError: ..No such file or directory: 'numbers I.txt''

10 COMPSCI 105 Lecture08

Handling With Exceptions for FileIO

▶ Basic structure of handling exceptions

```
trv:
    Attempt something where exception error may happen
     (i.e. open a file and read the content)
except IOERROR
    React to the error
else:
    What to do if no error is encountered
    (i.e. close the file)
finally:
    Actions that must always be performed
```



Exceptions: File Example

Consider the following code:

```
inputFileName = input("Enter name of input file: ")
       input file = open (inputFileName, "r")
       one line = input file.readline()
except IOError:
      print("File", inputFileName, "could not be opened")
else:
       print(one line)
       input file.close()
       print ("Closed file", inputFileName)
```

Enter name of input file: numbers.txt 43 34 Closed file numbers.txt

Case I:

Case 2: Enter name of input file: test.txt File test.txt could not be opened

12 COMPSCI 105 Lecture08



🚾 Raising an exception:

You can create an exception by using the raise statement

```
raise Error('Error message goes here')
```

- ▶ The program stops immediately after the raise statement; and any subsequent statements are not executed.
- It is normally used in testing and debugging purpose
- Example:

COMPSCI 105

Handling Exceptions

 Put code that might create a runtime error is enclosed in a try block

Example09.pv

Lecture08

```
def checkLevel(level):
    try:
        if level < 1:
            raise ValueError('Invalid level!')
        else:
            print (level)
            print ('This print statement will not be reached.')
        except ValueError as x:
            print ('Problem: {0}'.format(x))

def checkLevel(level):
        try:
        if level < 1:
            raise ValueError('Invalid level!')
        ...
        except ValueError as x:
        pass</pre>
```

COMPSCI 105

THE UNIVERSITY OF ALCELAND

13

Using Exceptions

- When to use try catch blocks?
 - If you are executing statements that you know are unsafe and you want the code to continue running anyway.
- ▶ When to raise an exception?
 - When there is a problem that you can't deal with at that point in the code, and you want to "pass the buck" so the problem can be dealt with elsewhere.



Exercise 3

▶ Modify the following function that calculates the mean value of a list of numbers to ensure that the function generates an informative exception when input is unexpected

```
def mean(data):
    sum = 0
    for element in data:
        sum += element
    mean = sum / len(data)
    return mean
```

Lecture08 16 COMPSCI 105 Lecture08

COMPSCI 105

L5

Lecture08



- ▶ Exceptions alter the flow of control
 - When an exception is raised, execution stops
 - When the exception is caught, execution starts again
- try... except blocks are used to handle problem code
 - Can ensure that code fails gracefully
 - ▶ Can ensure input is acceptable
- finally

17

▶ Executes code after the exception handling code





- ▶ TypeErrors are caused by combining the wrong type of objects, or calling a function with the wrong type of object.
 - ▶ This happens when someone tries to do an operation with different kinds of incompatible data types. A common example is to do addition of Integers and a string.
 - print (I + "a")
- ▶ A ValueError is used when a function receives a value that has the right type but an invalid value
 - value = int('a')
- value = float ('a')

18 COMPSCI 105 Lecture08