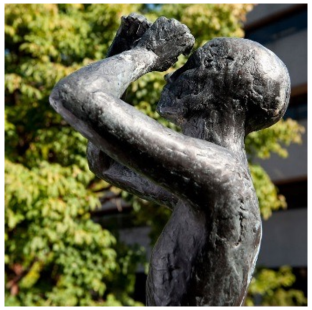

Lecture 11 – Exceptions and Shelve

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

Exceptions

Shelve

Remember to Record!

Class Test

- When: Thursday 9th November 2pm
- Where: Probably the lab G.20 and 24 but we will clarify this.
- Topics – Everything up to November 2nd Lecture but predominantly OOP.

- DSS Supports – contact to arrange eligible supports. Also let me know so we can have arrangements made on CS side.

- Sample paper posted, but we will do revision before the test that will be more focused on this year.

try-except

- We can simplify our code using some other exception constructs

```
try:
    val = int(input(“Enter a positive number: ”))
except ValueError:
    print(“An invalid value was entered – enter numbers”)
except Exception:
    print(“Some unanticipated event occured”)
```

- Either the try block executes
- Otherwise, an exception is raised in the try block and the exception is handled by the exception handlers, or propagated onto a higher level handler

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Try – Except - Finally

- We can also add actions that are always executed whether or not there is an exception

```
try:
    #Attempt to open a network connection and write data to it
except TimeoutError:
    print("Network connection has timed out")
finally:
    #Close the network connection
```

- In this case, we would have wrapped our network interactions with a try/raise block. It is good practice in either case to tidy up network resources.
- In this case, finally executes both when the try is successful and if an exception is raised (even if the exception is propagated to higher level exceptions)

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Raising Exceptions

- We can raise an exception in our code when we want to communicate that something unusual has occurred

```
raise ValueError()
```

- We can also customise the message we pass back which can be useful between instances of the same exception

```
raise ValueError(“An incorrect number was entered”)
```

Customizing Exceptions

- An exception is written as a class. We can create custom exceptions as follows:

```
class MyNewException(Exception): pass
```

- This effectively allows us to create exceptions with custom names to allow us to differentiate between types of events
  - We could also override parts of the implementation but that's beyond the scope of our course.
  - Remember when we define a new class we are defining a new type.
  - We shouldn't define new exceptions if an appropriate exception is already defined
  - The approach can be useful, we will generally tend towards using existing exceptions provided by Python.

Exceptions and Classes

- One of the goals of writing classes is that we can initialise or set the instance variables in a reliable way.
- Our classes are used programmatically – writing print statements won't give sufficient feedback – so we use exceptions to guide correct use of our classes.
- While we might handle exceptions within our classes in some circumstances, when considering users of our class, we need only let them know that something is wrong – we don't need to erroneous usage of our class.

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Adding Exceptions to Our Cxtrs

- The prime area to handle exceptions are at construction and when values are taken into the class. We should, for instance, check that they are in a valid range of values and of the expected type.
- Similarly, if we are creating network connections, dealing with files, etc, we should ensure that their operation is correct.
    - We are guided by looking at our code and wondering ‘what can go wrong here’, ‘how can this value break my code further down stream’
    - There are times when we must judge between handling potentially erroneous state with normal python constructs or using exceptions
    - Typically:
        - If a potential erroneous state could arise and be handled WITHIN the class and is not unexpected (it can be expressed as a conditional) then we would use regular features of Python
        - If we must communicate with some external code and the result is something other than the expected result, for example, some code that has imported our class for use, then we should use exceptions
        - If the event cannot be predicted or is obscure then we should use exceptions

Object Persistence

- There are many ways to persist data from your program
  - You can write to file
  - You can store data in database
  - You can serialise your objects and write them to disk - This final option is called shelving

Object Persistence

- Pickles allow us to serialise (convert to a string of bytes and write to a file) any object or collection of objects.
- Shelves allow us to store pickled objects as key/value pairs
  - In effect a persist-able dictionary
  - It provides the usual dictionary functions such as len() etc.
  - But in addition to what we are used to, we must open and close them (read and write from disk).
  - Pickling is done ’under the hood’ when we shelve an object; we don’t have to explicitly do this action.

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Shelve Example

- Person Class - see person.py examples.

- It is a standard class with state that includes a social security number, name and salary. The class has methods for getters/setters, givePayrise() and a method to represent the object as a string (__str__).

Shelve Example

```
import shelve #1
from person import Person

john = Person("SN12345", "John Doe", 32000)
#2

db = shelve.open(”persondb") #3
db[john.ssn] = john #4
db.close() #5
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Writing an object to a Shelve

#1 - we import the class from its module and also import the shelve functionality.

#2 - we create an instance of the class

#3 - we open a shelve by identifying the file where our objects and their keys will be written to .

#4 - we assign the object to the shelve (this causes it to be written to file. We assign as we would to a dictionary providing a key and the object as a value. The key must be unique. If the key is already in the shelve, then the original pickle will be overwritten

#5 - we close the shelve (causes the shelve to be synchronised)

Reading from a Shelve

- Access through key

```
import shelve #1
from person import Person

db = shelve.open(“persondb") #2
john = db[“SN12345"] #3
print(john)
db.close() #4
```

Reading an object from a Shelve

#1 - import the shelve and class to be deserialized

#2 - open the shelve and label it

#3 - access the shelve using a known key. This approach works when you know the shelve keys/ Once deserialised the object can be used as before

#4 - close the shelve

Updating a Shelve - for an individual object

```
import shelve #1
from person import Person

db = shelve.open(”persondb") #2
john = db[“SN12345"] #3
print(john)
john.givePayRaise(10) #4
print(john)
db[john.ssn] = john #5
db.close() #6
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Updating a Shelve - for an individual object

#1 - import required objects

#2 - open the shelve

#3 - read the object from the shelve (and label it)

#4 - apply a pay raise to the object

#5 - rewrite object to shelve

#6 - close shelve

Writing a Collection one by one

```
import shelve #1
from person import Person

john = Person("SN12345", "John Doe", 32000) #2
mary = Person("SN12346", "Mary Doe", 36000)
joe = Person("SN12347", "Joe Doe", 24000)

personList = [john, mary, joe] #3
db = shelve.open(“persondb") #4
for person in personList: #5
    db[person.ssn] = person
db.close() #6
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Writing a collection - write entire list

#1 - import required classes

#2 - create a series of objects

#3 - create a list of employee objects

#4 - open the shelve

#5 - write each individual object to the shelve. Each is indexed in the shelve

#6 - save the shelve

Writing a collection - write entire list

```
import shelve #1
from person import Person

john = Person("SN12345", "John Doe", 32000) #2
mary = Person("SN12346", "Mary Doe", 36000)
joe = Person("SN12347", "Joe Doe", 24000)

personList = [john, mary, joe] #3
db = shelve.open(“personwholelist") #4
db["store"] = personList #5
db.close() #6
```

Updating a Shelve - for an individual object

#1 - import the required libraries

#2 - create a series of objects

#3 - add the new objects to a list

#4 - open the shelve

#5 - write a single object (the list containing objects)

#6 - close the shelve

Reading and updating Collection where objects are individually indexed

```
import shelve #1
from person import Person

db = shelve.open(“persondb") #2
for key in db.keys(): #3
        person = db[key] #4
        person.givePayRaise(10) #5
        db[key] = person #6
db.close() #7
```

Reading and updating Collection where objects are individually indexed

#1 - import required classes

#2 - open the shelve

#3 - access each individual key in the shelve

#4 - access the individual element

#5 - access the object and update it

#6 - rewrite the object into the shelve

#7 - close the shelve

Reading a Collection when the entire collection was written

```
import shelve #1
from person import Person

db = shelve.open("employeedbwholelist") #2
theList = db["store"] #3
db.close() #4
for personobj in theList: #5
    print(personobj)
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Reading and updating Collection where objects are individually indexed

#1 - import the required files

#2 - open the shelve

#3 - read the data stored in the shelve

#4 - close the shelve

#5 - use the persisted data

Next Time:

Document our Examples