THE UNIVERSITY OF SYDNEY

INFO1910 S2 2023 Week 10 Tutorial

Classes

Classes

We have seen different ways of structuring data, from concrete types such as int and float to more abstract types such as lists, tuples and dictionaries but we have the ability to build our own types through classes.

What is a class

A class allows the programmer to create a `type` which can bundle both data and functionality. The `str` class contains an internal list of characters as well as functions to operate on its own data.

`__init__(self, ...)` constructor and `self`

Part of constructing and initialising instances of classes requires implementing the __init__ method. This method is invoked when an object of the type is initialised. You can define properties to be passed to the object through the constructor arguments.

It is also important to note the use of the `self` variable used in the constructor. The self variable is a binding to the instance of the variable and when writing instance methods, we will need to specify and use the self variable.

```
class Person:
  def __init__(self, name, age):
    self.age = age
    self.name = name

  def get_name(self):
    return self.name
```

To create an instance of a class:

```
p = Person('Jim Moriarty', 28)
#To use the name method
p.get_name()
#However, we can refer to the variable itself
p.name
```

Instance properties, getters and setters

We can extend our class from the previous example by adding an occupation property in the constructor. When creating a class we need to consider how we should expose the data.

- If instance properties should be read after initialisation then you should add a getter method. Denoted with a `get_` prefix. Example: `get_name`, `get_age`
- If instance properties should be changed after initialisation then your class should add a setter method. Denoted with a `set_` prefix. Example: `set_occupation`

```
class Person:

  def __init__(self, name, age, job):
    self.age = age
    self.name = name
    self.occupation = job

  def get_name(self):
    return self.name

  def get_age(self):
    return self.age

  def get_occupation(self):
    return self.occupation

  def set_occupation(self, job):
    self.occupation = job
```

Private attributes

Other programming languages have the notion of encapsulation (or how data is exposed). However python does not have such a mechanism. Simple case of "private attributes do not exist" but typically to demonstrate to other developers that someone shouldn't use or manipulate a variable is denoted by __ prefix. Example:

```
class Person:
  def __init__(self, name, age):
```

```
    self.age = age
    self.name = name
    self.__times_name_called = 0

  def get_name(self):
    self.__times_name_called += 1
    return self.name
```

Question 1: Components and variables

Consider the following class definition:

```
class Book:
  def __init__(self, title, author, year, url):
    self.title = title
    self.author = author
    self.year = year
    self.url = url
```

- What do each of the components/keywords of the this class mean?
- How can we access the data in the class?
- What type is `self`?
- What does it mean to create a new `Book` in our code?

Question 2: Attributes

- Have we already used classes prior to this tutorial? What type and what data do you think they have stored?
- What are the advantages to creating classes instead of using what is already there?
- With the given type, describe what kind of attributes we could give it
  - Polygon
  - Song (music)
  - Album
  - Employee
  - Table (furniture)
  - Company

For each example, consider the following questions:

- What data, incudling data type, should be stored in each object?
- How should the data be accessed?
- Should someone be able to read/write the data?

Question 3: Player and Highscore

You are given a class called Player. Each player has a highscore they have achieved from the game. After all player highscores are entered.

- You will need to define the properties associated with a player
- You will need to define a **class method** that will take a list of players and return which player has the highest score.

Player Class Scaffold:

```
class Player:

  def __init__(self, name, score):
    pass

  def highscore(self):
    pass

  def highest_score(players):
    pass
```

Usage:

```
p = Player('Example', 200)

player_list = [p] #Add other players and test your results
hp = Player.highest_score(player_list)
print("Highest Score: {} with {} points".format(hp.name, hp.score))
```

Attributes and Dispatch Methods

Elements of `__builtins__` are dispatch methods on attribute functions associated with objects in Python. For example the `__str__` method is implemented by:

```
def str(obj):
  return obj.__str__()
```

To make an object 'printable' in Python we can simply create a class that implements the 'Representation' method `repr`. As this is probably the same as casting the object to a string using `str` we can implement that method too by implementing `__str__`.

```
class C():
  def__init__(self, a):
    self.a = a
  def __repr__(self):
    return self.__str__()
  def __str__(self):
    return "I contain: " + a.__str__()
```

Our ability to manipulate objects now depends on the number of these special methods which we understand. For example if I wish to define the addition operator + over my object then I would simply implement the `__add__` operator.

```
class C():
  def__init__(self, a):
    self.a = a
  def __add__(self, other):
    return C((self.a, other.a))
  def __str__(self):
    return "I contain: " + a.__str__()
```

Question 4: Class Methods

For each of the following functions or operators find the appropriate method. Implement each method on your own class object. You may want to consult the `dir` method on some common Python types.

The following question is **not** exhaustive. But these should be useful.

- `a + b` using `__add__`
- `a - b` using `__sub__`
- `a << b` using `__lshift__`
- `a >> b` using `__rshift__`
- `a * b`
- `a / b`
- `a // b`
- `a % b`

- `a[b]` using `__getitem__`, `__setitem__` and `__delitem__`
- `a => b`, `a > b`, `a == b`, `a < b`, `a <= b`
- `abs(a)`
- `ceil(a)`
- `floor(a)`
- `hash(a)`
- `len(a)`
- `help(a)` using `__doc__`
- `iter(a)`
- `next(a)`

Question 5: Circular List

Write a class that implements a 'circular' list, that is that if the nth element exceeds the number of elements in the list then it should wrap back around again. Similarly if the $-n$th element exceeds the number of elements, then it should also wrap around.

```
class CircularList():
  def__init__(self):
    self.lst = []
  def __getitem__(self):
    pass
  def __setitem__(self):
    pass
  def __delitem__(self):
    pass
  def __add__(self, other):
    pass
  def __str__(self):
    pass
  def append(self):
    pass
  def pop(self):
    pass
  def remove(self):
    pass
  def __len__(self):
    pass
```

Inheritance

We saw with our vtables in C earlier that we could create a struct that contains another struct. By placing the other struct at the first position of the struct it would be aliased and we could access the elements of that struct by simply casting. If the vtable was correctly constructed then this would permit us to exactly inherit all the methods and functions of the first struct, and better yet build this to the same location using a constructor.

```c
struct object {
  void* associative_vtable;
  int property;
}
struct another_object {
  struct object obj;
  int other_properties;
}
```

Some elements of the vtable could then be overwritten in the constructor of the higher object, sharing properties where needed and removing properties where no longer needed for the new object.

This concept is known as 'inheritance' and is better defined within Python's runtime tagged type system.

```python
class A():
  def __init__(self):
    self.a = 5

class B(A): # A is the parent class
  def __init__(self):
    self.b = 6
    super().__init__() # Call the initialiser of the parent class
```

The previous structure we saw for the circular list, and for our C struct is 'composition' rather than inheritance.

Question 6: Circular Linked List Redux

Build a circular linked list using inheritance rather than composition.

```python
class CircularList(list):
  pass
```

Using the `import` and `from` keyword

Typically a good practice is to write classes in separate files and `import` them when needed. This allows for better organisation for project and gives you flexibility of use.

The from keyword allows you to specify the file and then import specific variables, functions and types that can be used in that file.

We can import another file into our current file like so:

Format:

```
import <file>
```

Example:

```
import person
```

Or if we want a specific variable or type from a file.

```
from <file> import <component1>, [<component2>], ...
```

Example:

```
from person import Person
```

The later examples allows the usage of very specific components of module to be used within your own code. They will also be namespaced to your module (you do not need to use person.Person to use the Person class).

Question 7: Testing your pet class

We will be introducing a unit testing framework called pyunit. Last week you wrote test cases for your simple calc program. Now we are going to transform those unit test cases to py unit test cases. Since we have just covered classes we will be creating a class that will inherit from the unit testing framework. Like so:

Format:

```
import circular_list
import unittest
class CircularListTest(unittest.TestCase):
```

After creating this class we will create methods that that start with `test_` (these methods will be ran by test runner, other methods are ignored).

```
import CircularList
import unittest
```

```
class CircularListTest(unittest.TestCase):

  def test_circular_list_len(self):
    #Use self.assertTrue(condition)
    # Your code here

  def test_circular_list_add(self, element):
    #Use self.assertEqual(actual, expected)
    # Your code here

  ...
```

To execute a pyunit test case, we will run it similarly to last week's pdb.

```
python -m unittest <your test file>

python -m unittest circular_list_test.py
```

Separation of concerns

Classes play an important role within the python ecosystem. Your builtin types such as `int`, `str` and `bool` are classes. However when creating a class it is important to separate the class into its own file.

The benefits of this approach are:

- Resuability with other code segments
- Smaller file size
- Clear idea of type

However, it is sometimes not necessary to strictly limit it to only one class per a file.

Question 8: Moduled Classes

Given the following, discuss with your class and tutor how you would construct your application.

- Employees in a company that will have different jobs which all have a different duration and name
- Listing of art galleries that contain individual works that relate an artist and movement
- An online subscription streaming service where users pay for access to different channels
- A bookstore that sells dvds, cds, books and tapes

Creating submodules

When importing a module Python first checks the local directory for the appropriate file and then searches the Python path. The Python path is simply a list of directories which Python will search for modules, you can see it using `sys.path`.

Installing a new python package is them simply a matter of compiling the package to Python bytecode (and egg), and adding its location to the Python path.

When constructing your application we will need to identify and translate the requirements into code. Using the first item from the previous section, we can translate the requirements into 4 different files where each file (except main.py) will include class that will suit the description.

Lets take this example of a project's directory

```
main.py #imports all 3 files and uses their classes
employee.py #Contains an Employee class
company.py #Contains a Company class
job.py #Contains a Job class
```

We are able to import objects from a module using the `import` keyword or a combination of `from` and `import` where we want to exclusively pick certain fields from a file.

`__init__.py` and `__all__`

Using the previous example with the files `company.py`, `employee.py` and `job.py`, we will move these into a separate folder named `model`

As best practice it is best to get into the habit of categorising your files into folders. Depending on your project you may be able to provide something more descriptive that represents those files. For example, if you have multiple test files, it would be appropriate to organise them into a test folder.

If we were to organise our company entities into a folder called `model` but still keep `main.py` in the parent folder. `main.py` needs some way of referring to these files.

```
main.py #imports all 3 files and uses their classes
model/employee.py
model/company.py
model/job.py
```

This is where `__init__.py` and `__all__` come in. To expose the files within the model folder and consider the model folder to be a submodule, your project will need to contain a `__init__.py` file.

```
model/__init__.py
```

During import resolution, if python detects that the path is a folder, it will implicitly search for `__init__.py` file that will then dictate other files import.

After creating `__init__.py` within the model folder, you will simply need to indicate to Python what is in this module:

```
from . import employee, company, job
```

Alternatively:

```
__all__ = ['employee', 'company', 'job']
```

You can observe that the `__all__` variable is assigned to a list of strings which are filenames (without the extension) of the files within the same folder.

Your `main.py` can now import files within the model folder with the following statements.

```
import model #imports everything in model
import model.employee #imports the employee file
import model.job #imports the job file

#imports the Employee class in model/employee
from model.employee import Employee
```

Question 9: Computer Store (Part 1)

Using your knowledge of classes, you are to write a system that will model a computer store. The following list are just a couple of classes that assist with the program's logic.

- Part, which has the following:
  - name
  - price
- ShoppingBag, which has the following:
  - Can contain a list of parts
  - Total value of the parts

Your program should start from `main.py` which will contain logic for handling items.

Example scenario:

```
Options:
LIST - Show parts list
ADD - Add part to shopping list
CHECKOUT -  Checkout and pay

LIST
0 - Intel6620U for $250
1 - Ryzen1700 for $365
2 - RaspberryPi for $35
3 - Odroid C2 for $65

Options:
LIST - Show parts list
ADD - Add part to shopping list
CHECKOUT -  Checkout and pay

ADD 0
Intel6620U added

ADD 2
RaspberryPi added

CHECKOUT
Your order comes to the total of $285
Enter C to cancel and keep shopping or enter in an amount greater than
or equal to $285
290

You received $5 change
<Program Ends>
```

Question 10: Loading parts (Part 2)

After you have constructed a computer store and checkout functionality, you will need to incorporate some way of loading a parts list into your program. Discuss with your tutor where you plan on writing your parts loader code and rationalise your decision.

The parts list is given in the format:

```
name,price
```

Use can use the following list to help test your loader:

```
Intel6620U,250
Macbook13,3000
Ryzen1700,365
Corsair8GB,80
```

Question 11: Adding quantity (Part 3)

Discuss with your tutor and class members an appropriate place for `quantity`. The parts list from the previous question will now have a 3rd column which specifies quantity of a part.

Updated format:

```
name,price,quantity
```

Without writing any code what would be the appropriate place for quantity. Would other segments of your code need to change with this addition or could you simply change how other functions interact with the code.

Creating a Python Package

Given our Python module we can turn it into a package using setuptools:

In the top level of your repository create a `setup.py` file containing the following:

```
from setuptools import setup, find_packages

setup(
        name="demo",
        package_dir={"":"src"},
        packages=find_packages(where="src"),
        )
```

Your modules should be placed in an src directory. We can run this script using `python setup.py install`.

This will create a new package named 'demo' and associate all the modules in src with this package.

If the package itself depends on other Python packages it is common to include a 'requirements.txt' file containing the appropriate packages and versions. This can be installed using pip:

```
pip install -r requirements.txt
```

Question 12: Packaging

Turn your Python programs from this tutorial into packages.