THE UNIVERSITY OF SYDNEY

INFO1110 / COMP9001 Week 12 Tutorial

Program Structure and Numpy

Structuring your programs

You would have imported modules into your program from previous tutorials. However this section will focus on structuring your program. During development of a program the requirements may change and you may need to change the code or add additional features.

During the construction of your project you should be creating testcases and test each file.

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 1: 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 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 just need to add the following lines:

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

You can observe that the `__all__` variable is assigned to alist 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 2: 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 e
290

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

Question 3: 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 4: 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.

Arrays and DTypes

You may have previously noticed that all collections that are native to Python can store objects of any type. However, the type of each element of the collection can be determined in advance, and this can be used for error checking, better performance and for integration with other programming languages.

An array is similar to a list, however it should be of a single pre-defined type. Arrays are implemented in the numpy package and are initialised from an existing Python list.

```
import numpy as np
my_arr = np.array([1,2,3,4,5])
print(my_arr)
```

We can specify a type for an array using a 'dtype' object, such as an integer number, a string or a floating point number,along with specifying how many bits each element should be allocated. These types are hard coded into Python (and into numpy) but you can create your own custom types if it is necessary.

```
my_arr_of_integers = np.array([1,2,3,4,5], dtype=np.int32)
my_arr_of_floats = np.array([1,2,3,4,5], dtype=np.float64)
```

New dtypes can also be composed as a collection of existing dtypes as a single object. They are defined by an array of tuples of strings and dtypes. The strings form keys that access a value of the specified type when a variable of that type is used. In all it can be thought of as a rigid implementation of a dictionary collection.

When properly implemented, dtypes can allow for very descriptive and readable Python.

```
# Our dtype is defined as a list of tuples containing a string as a key,
# a standard numpy dtype and a length as an optional argument
dt = np.dtype([('name', np.unicode_, 24), ('age', np.int32)])

# We can initialise our array with a certain dtype by specifing
# it as a keyword argument
students = np.array([('alice', 19), ('bob', 18), ('eve', 25)], dtype=dt)

# We can now use the keys specified in the definition of our
# dtype to access elements by name
print(students[0]['name'])
print(students['name'])
print(students[:]['age'])
```

Question 5: Dtype Builder

Write a python program to read the following file and to build an appropriate dtype and store all the relevant data.

```
sid, name    , grade
1  , alice   , 75
2  , bob     , 60
3  , cathy   , 80
4  , douglas , 85
```

Once you have read from the file and added it to your 'database', find a Pythonic way to sort the students by their grade.

Linear Algebra

The main advantage to arrays over other collections is its applications in linear algebra and mathematics. It would be hard to suggest how to multiply two lists or tuples containing elements of arbitrary typing. This allows us to perform vector and matrix multiplication along with other mathematically useful transformations that might be poorly defined when applied to a collection such as a dictionary.

Question 6: Vectorisation

Once we can deal with variables as single mathematical objects rather than collections, it becomes possible to 'vectorise' our code to perform single operations on a variable rather than looping over each element of the collection. However, this requires that our code is written to expect both array and single values.

Vectorise the following functions such that they can still take a single value, or they can perform the operation on an array.

```
def greater_than_two(qwop):
        if qwop > 2:
                return qwop + 1
        else:
                return qwop - 3

# When vectorised, this function should return an array of
# the maximum value from each of the arrays in a 2D array
def array_max(arr):
        return max(arr)

# Currently this function takes a coordinate and the map
# and returns the updated coordinate. You should
```

```
# Remove the dependance on the coordinates and simply update the
# Map and return it.
# The input to this function should be a two dimensional array
def limited_game_of_life(game_map, x, y):
        # Boarders are fixed at zero to prevent attempting to access
        new_value = 0
        # Make sure we are not on the boarder
        if (0 < x * y
        and len(game_map) - 1 > y
        and len(game_map[y]) - 1 > x):
                new_value = (game_map[y, x]
                        + game_map[y + 1, x] + game_map[y - 1, x]
                        + game_map[y, x + 1] + game_map[y, x - 1])
        # Return the new value
        return new_value
```

One approach is to simply use numpy's own 'vectorize' function, however this is essentially a for loop and grants none of the improved performance that vectorising your own functions does.

Question 7: Matrix Multiplication

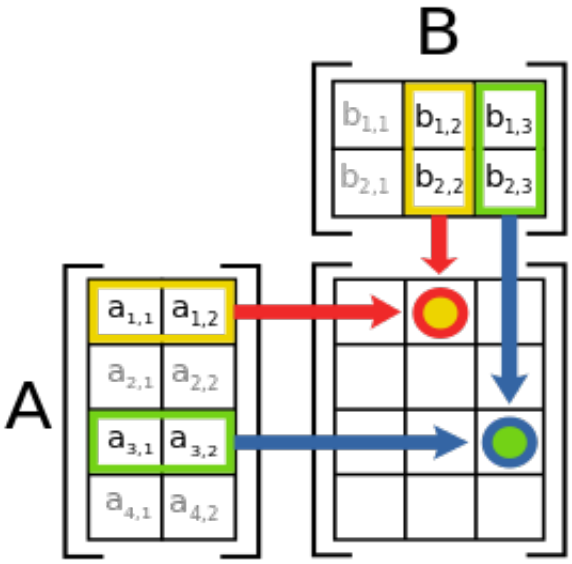
Matrix multiplication is a mathematical procedure that produces a resultant matrix from two input matrices. Given the following function prototype:

```
def matrix_mult(m1, m2)
```

Mathematically, the multiplication of $A \times B$ can be expressed as:

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \,.$$

where A is an $n \times m$ matrix and B is an $m \times p$ matrix. Graphically, this can be seen as:



Implement your own matrix multiplication function without using numpy. After implementing your own, use the numpy's `numpy.matmult` function while creating arrays using `np.array([1,2,3,4,5], dtype=np.int32)`.

```
matrix_mult([[1,2],[3,4]], [[2,0],[1,2]]) #[[4,4], [10,8]]
matrix_mult([[1, 2, 3], [4, 5, 6]], [[9, 8, 7]]) # [[46, 118]]
```

Extension Create a function call `matrix_same(m1, m2)` that will check if two matrices are the same. Afterwards use numpy's `array_equal` function to compare the two matrices.

Question 8: Inner Products, Outer Products and Transposes

For the next section, and for 3D plotting in general, it is very useful to be able to generate an array of coordinates. Write a Python function that takes two arrays as arguments that represent the range of the X and Y coordinates and returns two, two dimensional arrays of X and Y coordinates.

You should then find a way to easily 'flip' these coordinates so that you can view your three dimensional map from another angle.

For example:

```
x = [1,2,3,4,5]
y = [10, 12, 14, 16]
```

should return

```
x_map = array(
           [[1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5]])
y_map = array(
           [[10, 10, 10, 10, 10],
        [12, 12, 12, 12, 12],
        [14, 14, 14, 14, 14],
        [16, 16, 16, 16, 16]])
```

Numpy's outer product function, transpose function and ones function will be useful for doing this efficiently.