FINALE

Brief Recap

- Iterators
- File Handling
- Exceptions

Menu for Today!

- Modules
- (A very brief) Introduction to `numpy` and `pandas`

MODULES

Modules

- Structuring programs makes them more readable and easier to understand / debug
  - *Functions*
  - *Classes*
- It is common to split a program into multiple files
  - *Makes programs easier to understand*
  - *Makes code re-use easier*
  - *Makes it easier to organize and manage code*

Using Code from Different Files

- You can use the code from a different file in the current file using `import`
- Functions from the imported file can be accessed as `file.function`

```
------- myMath.py ----------
def myAdd(x,y): return (x + y)
def mySub(x,y): return (x - y)
….
--------main.py--------------
import myMath
print(myMath.myAdd(2,3))
```

Simplifying `import`

- `file.function` naming format can be cumbersome
  - Simpler names for files

```
import myMath as m
x = m.myAdd(2,3)
```

Controlling Import: `from`

- A single file may define several function; from allows us to control what we import
  - Note that file.function is no longer needed, just function is enough

```
from myMath import myAdd
print(myAdd(3,2))
print(mySub(3,2))              # Error! name mySub not defined
print(myMath.mySub(3,2))       # Error! name myMath not defined
```

Controlling Import: `from`

- *We can also import all functions from a file using from*
  - *Not recommended as you begin to loose visibility of where a function was defined*

```
from myMath import *
print(myAdd(3,2))
print(mySub(3,2))
```

Built-In Modules

- `math`
  - *defines many mathematical functions like* `ceil`, `floor`, `sqrt` *etc*
- `statistics`
  - *defines many statistical functions like* `median`, `mean`, `mode` *etc*
- `random`
  - *Used to generate random numbers*
  - *Different functions such as* `randrange`, `random` *and* `randint` *can be used to control the distribution of the number generated*

Libraries in Python

- Python gets its power from the availability of a very large number of libraries
  - `numpy:` supports multi-dimensional arrays (tensors) in Python; foundation of many other libraries
  - `scipy`: provides highly optimized implementations for several scientic computation problems; built on top of `numpy`
  - `pandas:` providing high-performance data manipulation and analysis tool using its powerful data structures; supports a powerful data structure called `dataframe` ; built on top of `numpy`
  - Many AI libraries such as `keras, tensorflow and pyTorch`

NUMPY

Arrays

- Two key advantage of numpy
  - *Extremely fast implementation of* `array`
  - *Efficient mechanism for manipulating (numerical) data in the* `array`
- `Array`
  - *Faster, more compact than* `list`
  - *Data elements are homogenous*
  - *A grid of values*
  - *Contains information about the raw data*
    - how to locate an element (indexing)
    - how to interpret an element.

Indexing Arrays

- Arrays can be multidimensional
  - *indexed by a tuple of nonnegative integers, by booleans, or by another array*
  - *the number of dimensions of the array is known as its rank*
  - *The dimensions of the array are called axes*
  - `shape` *of the array is the size of the array along each dimension.*
  - *All elements of the array are of the same type* `dtype`

```
import numpy as np
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a.dtype
a.shape
```

Using Arrays

- Creating arrays

```
np.array(), np.zeros(), np.ones(), np.empty(), np.arange(),
np.linspace()
```

- Sorting arrays

```
a = np.array([[1,4], [3,2]])
a.sort(0)
a.sort(1)
```

Manipulating Arrays

```
a = np.arange(6)
ones = np.ones(6, dtype=int)
data = a.reshape(2,3)
ones = ones.reshape(2,3)
newData = data + ones
newData = newData * 1.6
print(newData * data)
```

RECAP

What Did We Learn

- Basics Of Python
  - *Objects and expressions*
  - *Variables and Assignment*
- Strings and Input / Output
- Conditionals
- Iteration: for and while loops
- Functions: Abstractions and arguments, scoping and global / local variables, recursion
- Introduction to Object Oriented programming
- Structured types: Tuples, Lists and Dictionaries
- Iterables and Iterators
- Higher order functions: functions as objects; functional programming
- Exceptions and Assertions
- Modules and files
- Libraries

Final Words

- All the best for your end-sems!

Questions?

```
class my_class:
     def method1(self):
         self.num1=5
     def method2(self):
         print(self.method1.num1)
```