Programmer-Defined (Abstract) Types and Classes

Python

- Python has a number of built-in composite types we've already seen
  - Lists
  - Dictionaries
  - Sets
- One we haven't seen:
  - Structured array (NumPy)

NumPy Structured Array

- A record array allows access to its elements by a name rather than by an index.
- Frequently used with spreadsheet data
  - Column headings are the fields
- Example: name the dimensions “space” and “time” rather than i,j
  - x[‘space’,’time’]
- Example: image data color x height x width

img = array([[(0,0,0), (1,0,0)], [(0,1,0), (0,0,1)]], [('r',float32),('g',float32)('b',float32)])

However, the indices really represent primitive types, we just give them more convenient names.

NumPy Record Array

- A record array is a structured array that adds the ability to access the field as an attribute.
- arr[‘x’], arr[‘y]’ : structured array
- arr.x ; arry.y : record array

```
x = np.array([(1.0, 2), (3.0, 4)], dtype=[('x',
                float), ('y', int)])
x = x.view(np.recarray)
```

- Create a new recarray:

np.recarray((2,), ... dtype=[('x', int), ('y', float), ('z', int)])

NumPy recfromcsv

- Reads in a csv file and uses the header values as the fields.
- Example (out there on the Intertubes:)

```
import numpy as np
test = np.recfromcsv(data, missing='N/A',
                                names=True,
   case_sensitive=True)
```

http://www.scipy.org/Cookbook/InputOutput

Defined Types: Fortran

- In Fortran these are called *defined types*.
- Syntax is extremely simple (ptype stands for a primitive type)

```
type mytype
   <ptype> var1
   <ptype> var2
   <ptype>, dimension(:)), allocatable :: var3
   !F2003 but most newish compilers support
   type(anothertype) :: var4
end type mytype
```

Fortran Defined Types

- We nearly always put defined types into modules that also define functions that operate on the type
- The module must not have the same name as the defined type (this is somewhat inconvenient).
- If you need to allocate memory to create a variable of a given type (because a member is an allocatable array or another type that needs allocation) this *will not* happen automatically. You must write a **constructor** to allocate the memory.

Fortran: Declaring Types and Accessing Fields

```
type(mytype)                          ::  thevar
type(mytype), dimension(100) :: var22
type(mytype), dimension(:), allocatable ::var11
```

To access the fields of the type use the name of the type, the percent sign as a separator, and the name of the field

```
thevar%var2
var11(12)%var1
var22(1)%var4%varx
```

Public and Private

- In Fortran you can declare module symbols (variables and names of routines) to be private. You may also explicitly declare them public but that is the default.
- Private variables are not accessible by program units that use the module.
- Example:

```
real, private   :: x, y, z
private         :: r_fun,d_fun
```

Public and Not-So-Private

- In Python there are no explicitly private symbols in the module.
- If you start a name with an underscore (_) it is not loaded automatically when you import your module with `from *`
- It is not really private since it can be probed explicitly. However, it's considered impolite to do so.

Python: Class

- Python makes no distinction between a class and a structure/record type

```
class myclass:
   var1=0
   var2=''
   var3=0.0
```

- Just as in Fortran, a class will be in a module (even more so in Python). Some versions of Python may not particularly like having the module name be the same as the class name.

Declaring Variables (Instantiating) a Class

```
import theclass
avar=theclass.myclass()
```

When we get to classes with member constructors we will usually pass values through the parameter list.

Accessing class fields:

```
theclass.avar.itsvar
```