Python data types and structures

Data types

Like any scripting language, `Python`, has multiple data types. Unlike many other languages, `Python` does not require designation of a variable's data mode prior to assignment. This is convenient, but also can cause problems when `Python` assigns an unintended data mode to a variable. The common data types in `Python` include:

- integer - number with the decimal place at the end
- float - number with the decimal place at any location
- complex - imaginary numbers
- string - word or text values
- boolean - True or False; `Python` also treates these as 1 and 0, respectively

Data structures

Data of any of the modes described above is stored in one of `Python`'s data structures. `Python` and its packages have a number of different data structures, including string, list, tuple, dictionary, set, array, dataframe. We will primarily use four of these across the semester: list, numpy array, pandas Dataframe, and dictionary.

1. List:

A list is a one-dimensional `Python` data structure that contains an ordered set of values.

a. Creating a list

- `list()`: converts a string into a list with each character in its own element

  ```
  myList=list("apple")
  ```

- `[]` : use square brackets to define a list with elements separated by commas; one can also define an empty list by putting no elements in the square brackets

  ```
  myList=[1,5,9,10]
  ```

- `range()`: generates an ordered sequence of integers incremented by the specified value

  ```
  myList=range(0,10,1)
  ```

- `numpy.linspace()`: generates an ordered sequence of floats incremented by the specified value

  ```
  myList=range(0,10,30)
  ```

- `[]*n`: generates a list with n repeats of the elements contained in the square brackets

  ```
  myList=[1]*10
  ```

- `.split()`: generates a list from a string with elements separated by the indicated delimiter

  ```
  myList=myList.split(",")
  ```

b. Useful list methods/functions

Python and its packages have a number of functions that work on lists, but data structures in Python are something called classes and classes have "functions" specific to them called methods. Functions are used in the manner we've previously discussed `function_name(arguments)`. Methods are used with the syntax `variable_name.method(arguments)`. Below you can distinguish functions from methods by whether the name begins with `.` or not. For methods that make some change to the list (e.g. append or insert) you do not need to do any sort of assignment. You can simply call the method and the change will be stored in the object.

- `len()`: returns the length of the list
- `max()`: returns the maximum value contained in the list
- `min()`: returns the minimum value contained in the list
- `.append()`: appends arguments to the list
- `.extend()`: adds the elements of a list passed as an argument to the end of the list
- `.count()`: returns the count of how many times the argument occurs in the list
- `.index()`: returns the lowest index in the list that the argument appears
- `.insert()`: inserts the second argument at the index specified in the first argument
- `.pop()`: removes and returns the last object from the list
- `.remove()`: removes the a single occurence of the argument from the list
- `.reverse()`: reverses objects contained in the list
- `.sort()`: sorts the contents of the list

c. List indexing and subsetting

Because lists are an ordered sequence, a single element or subset of elements can be referred to using square brackets, [ ], and a numerical index. Note that indexing begins with zero!

```
v=[1,3,5,9,13]
print(v[0])
print(v[4])
```

```
## 1
## 13
```

2. numpy array:

A numpy array (from here on "array") is an *n*-dimensional structure that holds a single data type. A 1D array is a lot like a list, a 2D array is like a matrix and is analogous to an Excel worksheet. A 3D array is analogous to an Excel workbook. As with `Python` lists, arrays have both functions and methods that can be used to gather information about an array or modify their contents and shape. Because this data structure is defined in the numpy package, we must `import numpy` prior to its use.

a. Creating arrays

- `numpy.array()`: creates an array from a list

`A=numpy.array([1,2,3,4])` makes a 1x4 array

`B=numpy.array([(1,2,3),(4,5,6)])` makes a 2x3 array

- `numpy.zeros()`: creates an array with specified dimensions that is filled with zeroes

  `B=numpy.zeros((2,3))` makes a 2x3 array filled with zeroes
- `numpy.ones()`: creates an array with specified dimensions that is filled with ones

  `B=numpy.ones((2,3,2))` makes a 2x3x2 array filled with ones

b. Useful numpy array functions/methods

- `.ndim`: returns the number of dimensions (rank) of the array; this is an attribute so no arguments or parentheses are included when used
- `.shape`: returns the number of rows in the array; attribute
- `.size`: returns the total number of elements in the array; attribute
- `.dtype`: returns the data type of elements in the array
- mathematical operators: +, - , *, /; these are all applied element-wise
- logical operators: <, >, <=, >=, ==, !=; these are all applied element-wise
- `.tolist()`: convert an array to a list; similar methods to convert to other `Python` data structures
- `.tofile()`: write an array to a text file
- `.fill()`: fill an array with a scalar value provided as an argument
- `.reshape()`: change the dimensions of an array, but the size must remain the same
- `.resize()`: change the shape and size of an array
- `.transpose()`: transpose an array
- `.min`: returns the minimum of an array along the axis provided as an argument
- `.sum()`: returns the sum of an array along the axis provided as an argument
- `.mean()`: returns the mean of an array along the axis provided as an argument
- `numpy.vstack()`: combines two arrays (arguments) by stacking them vertically
- `numpy.hstack()`: combines two arrays (arguments) by stacking them horizontally)
- `numpy.column_stack()`: stacks 1D arrays side by side into a 2D array

c. Array indexing/subsetting

Just like lists, elements or subsets of arrays can be indexed using square brackets. We can use a numerical index along each dimension of an array, separated by a comma, to index a particular element or subset of elements.

```
import numpy

A=numpy.arange(4).reshape(2,2)
print(A)
print(A[1,0])
print(A[1,1])
print(A[:,1])
```

```
## [[0 1]
##  [2 3]]
## 2
## 3
## [1 3]
```

Logical values work for indexing with matrices too.

```
import numpy

A=numpy.arange(15).reshape(3,5)
print(A)
print(A<8)
B=A[A<8]
print(B)
```

```
## [[ 0  1  2  3  4]
##  [ 5  6  7  8  9]
##  [10 11 12 13 14]]
## [[ True  True  True  True  True]
##  [ True  True  True False False]
##  [False False False False False]]
## [0 1 2 3 4 5 6 7]
```

You can also subset a matrix based upon the content of a particular row or column.

```
import numpy

A=numpy.arange(15).reshape(5,3)
print(A)
print(A[:,0]<6)
B=A[A[:,0]<6,:]
print(B)
```

```
## [[ 0  1  2]
##  [ 3  4  5]
##  [ 6  7  8]
##  [ 9 10 11]
##  [12 13 14]]
## [ True  True False False False]
## [[0 1 2]
##  [3 4 5]]
```

3. pandas Dataframe:

The pandas package for `Python` provides us with the dataframe data structure. Dataframes are exclusively two-dimensional, but can hold different data types as columns. This data structure was modeled after the dataframe in `R`. As with `Python` lists, arrays have both functions and methods that can be used to gather information about a dataframe or modify their contents and shape. Because this data structure is defined in the pandas package, we must `import pandas` prior to its use.

a. Creating dataframes

The easiest ways to create a dataframe are to convert a numpy array or a dictionary

```
import numpy
import pandas

A=numpy.arange(15).reshape(5,3)
D=pandas.DataFrame(A,columns=['data1','data2','data3'])
print(D)
```

```
##    data1  data2  data3
## 0      0      1      2
## 1      3      4      5
## 2      6      7      8
## 3      9     10     11
## 4     12     13     14
```

b. Useful dataframe functions/methods

- `.T`: returns the transpose of the dataframe; this is an attribute so no arguments or parentheses are included when used
- `.shape`: returns the number of rows in the array; attribute
- `.size`: returns the total number of elements in the array; attribute
- `.dtypes`: returns the data type of elements in the array; attribute
- mathematical operators: +, - , *, /; these are all applied element-wise
- logical operators: <, >, <=, >=, ==, !=; these are all applied element-wise
- `.apply()`: apply a function to the specified axis
- `.applymap()`: apply a function element-wise to a dataframe
- `.assign()`: add a column to a dataframe
- `.drop()`: remove a column from a dataframe
- `.drop_duplicates()`: remove duplicate rows from the dataframe
- `.sort_values()`: sort the dataframe by specified columns
- `.set_value()`: set a value specified by the arguments
- `.filter()`: subset rows or columns of dataframe according to laels in the specified axis
- `.insert()`: insert column into dataframe at specified location
- `.melt()`: "unpivots" a dataframe; converts form a wide to long format
- `.merge()`: merge dataframe objects by columns
- `.pivot()`: converts from a long to a wide format
- `.replace()`: replace values givein in "to_replace" with "value"
- `.round()`: round dataframe entries
- `.sample()`: take random selections from rows or columns of a dataframe
- `.head()`: returns the top number of rows that are requested as an argument
- `.tail()`: returns the bottom number of rows that are requested as an argument
- `.to_csv()`: send a dataframe to a text file; there are a large number of `.to_something()` functions to choose from

- `.min`: returns the minimum of a dataframe along the axis provided as an argument
- `.sum()`: returns the sum of a dataframe along the axis provided as an argument
- `.mean()`: returns the mean of a dataframe along the axis provided as an argument
- `.max()`: returns the max of a dataframe along the axis provided as an argument

c. Indexing dataframes

There are a number of equivalent ways to access columns of a dataframe. Square brackets can be used like with arrays if the `.iloc` attribute is used. Alternatively column names can be used to access a particular column.

```
import numpy
import pandas

A=numpy.arange(15).reshape(5,3)
D=pandas.DataFrame(A,columns=['data1','data2','data3'])
print(D)
print(D.data1)
print(D.iloc[:,0])
```

```
##    data1  data2  data3
## 0      0      1      2
## 1      3      4      5
## 2      6      7      8
## 3      9     10     11
## 4     12     13     14
## 0     0
## 1     3
## 2     6
## 3     9
## 4    12
## Name: data1, dtype: int64
## 0     0
## 1     3
## 2     6
## 3     9
## 4    12
## Name: data1, dtype: int64
```

4. Dictionary

A dictionary is a unordered group that holds multiple values/objects. Values or objects are accessed from the dictionary using a *key*. The key-value pairs are what defines a dictionary. Duplicate keys are not allowed in a dictionary, but values can be repeated.

a. Creating dictionaries

There are a multitude of ways to create a dictionary in `Python`.

- braces are used to define a series of key-value pairs, but an empty pair of braces can also be used to define an empty dictionary

```
D={"hello":"hola","goodbye":"adios"}
```

- two lists can be “zipped” to create a dictionary

```
Keys=["hello","goodbye"]
Values={hola","adios"]
D=dict(zip(Keys,Values))
```

- key-value pairs can be added to an existing dictionary using square brackets

```
D["thanks"]="gracias"
```

- removing entris from a dictionary is accomplished with `del`

```
del D["thanks"]
```

b. Useful dictionary functions/methods

- square brackets and the key value can be used to retrieve a value (e.g. `D["thanks"]`)
- `.get()`: returns a value from a dictionary with the key passed as an argument
- `.keys()`: return a list of the keys for a dictionary
- `.clear()`: removes all entries from a dictionary
- `.has_key()`: returns true or false depending on whether a key is in a dictionary
- `len()`: return the number of key-value pairs
- `apply()`: applies a function that works on a vector to each row or column of a matrix
- `merge()`: joins two data frames together using a shared column as an index
- `lapply()`: analogous to `apply()`, but operates on lists and returns a list
- `sapply()`: the same functionality as `lapply()`, but returns a matrix or vector