Python 8

Data Structures

Sometimes a *simple* list or dictionary just doesn't do what you want. Sometimes you need to organize data in a more *complex* way. You can nest any data type inside any other type. This lets you build multidimensional data tables easily.

List of lists

List of lists, often called a matrix are important for organizing and accessing data

Here's a way to make a 3 x 3 table of values.

```
>>> M = [[1,2,3], [4,5,6],[7,8,9]]
>>> M[1] # second row (starts with index 0)
[4,5,6]
>>>M[1][2] # second row, third element
6
```

Here's a way to store sequence alignment data:

Four sequences aligned:

```
AT-TG
AATAG
T-TTG
AA-TA
```

The alignment in a list of lists.

```
aln = [
['A', 'T', '-', 'T', 'G'],
['A', 'A', 'T', 'A', 'G'],
['T', '-', 'T', 'T', 'G'],
['A', 'A', '-', 'T', 'A']
]
```

Get the full length of one sequence:

```
>>> seq = aln[2]
>>> seq
['T', '-', 'T', 'T', 'G']
```

> Use the outermost index to access each sequence

Retrieve the nucleotide at a particular position in a sequence.

```
>>> nt = aln[2][3]
>>> nt
'T'
```

> Use the outermost index to access the sequence of interest and the inner most index to access the position

Get every nucleotide in a single column:

```
>>> col = [seq[3] for seq in aln]
>>> col
['T', 'A', 'T', 'T']
```

> Retrieve each sequence from the aln list then the 4th column for each sequence.

Lists of dictionaries

You can nest dictionaries in lists as well:

```
>>> records = [
... {'seq' : 'actgctagt', 'accession' : 'ABC123', 'genetic_code' : 1},
... {'seq' : 'ttaggttta', 'accession' : 'XYZ456', 'genetic_code' : 1},
... {'seq' : 'cgcgatcgt', 'accession' : 'HIJ789', 'genetic_code' : 5}
... ]
>>> records[0]['seq']
'actgctagt'
>>> records[0]['accession']
'ABC123'
>>> records[0]['genetic_code']
1
```

> Here you can retrieve the accession of one record at a time by using a combination of the outer index and the key 'accession'

Dictionaries of lists

And, if you haven't guessed, you can nest lists in dictionaries

Here is a dictionary of kmers. The key is the kmer and its values is a list of postions

```
>>> kmers = {'ggaa': [4, 10], 'aatt': [0, 6, 12], 'gaat': [5, 11], 'tgga':
... [3, 9], 'attg': [1, 7, 13], 'ttgg': [2, 8]}
>>> kmers
{'tgga': [3, 9], 'ttgg': [2, 8], 'aatt': [0, 6, 12], 'attg': [1, 7, 13], 'ggaa': [4,
10], 'gaat': [5, 11]}
>>>
>>> kmers['ggaa']
[4, 10]
>>> len(kmers['ggaa'])
2
```

> Here we can get a list of the positions of a kmer by using the kmer as the key. We can also do things to the returned list, like determining its length. The length will be the total count of this kmers.

You can also use the `get()` method to retrieve records.

```
>>> kmers['ggaa']
[4, 10]
>>> kmers.get('ggaa')
[4, 10]
```

> These two statements return the same results, but if the key does not exist you will get nothing and not an error.

Dictionaries of dictionaries

Dictionaries of dictionaries is my favorite!! You can do so many useful things with this data structure. Here we are storing a gene name and some different types of information about that gene, such as its, sequence, length, description, nucleotide composition and length.

```
>>> genes = {
... 'gene1' : {
...     'seq' : "TATGCC",
...    'desc' : 'something',
...     'len' : 6,
... 'nt_comp' : {
...             'A' : 1,
...             'T' : 2,
...             'G' : 1,
...             'C' : 2,
...            }
...   },
...
... 'gene2' : {
...     'seq' : "CAAATG",
...    'desc' : 'something',
...     'len' : 6,
... 'nt_comp' : {
...            'A' : 3,
...            'T' : 1,
...            'G' : 1,
...            'C' : 1,
...            }
...       }
... }
>>> genes
{'gene1': {'nt_comp': {'C': 2, 'G': 1, 'A': 1, 'T': 2}, 'desc': 'something', 'len': 6,
'seq': 'TATGCC'}, 'gene2': {'nt_comp': {'C': 1, 'G': 1, 'A': 3, 'T': 1}, 'desc':
'something', 'len': 6, 'seq': 'CAAATG'}}
>>> genes['gene2']['nt_comp']
{'C': 1, 'G': 1, 'A': 3, 'T': 1}
```

> Here we store a gene name as the outermost key, with a second level of keys for qualities of the gene, like sequence, length, nucleotide composition. We can retrieve a quality by using the gene name and quality in conjunction.

To retrieve just one gene's nucleotide composition

```
>>> genes['gene1']['nt_comp']
{'C': 2, 'G': 1, 'A': 1, 'T': 2}
```

Alter one gene's nucleotide count with `=` assignment operator:

```
>>> genes['gene1']['nt_comp']
{'C': 2, 'G': 1, 'A': 1, 'T': 2}
>>>
>>> genes['gene1']['nt_comp']['T']=6
>>> genes['gene1']['nt_comp']
{'C': 2, 'G': 1, 'A': 1, 'T': 6}
```

Alter one gene's nucleotide count with `+=` assignment operator:

```
>>> genes['gene1']['nt_comp']
{'C': 2, 'G': 1, 'A': 1, 'T': 6}
>>>
>>> genes['gene1']['nt_comp']['A']+=1
>>>
>>> genes['gene1']['nt_comp']
{'C': 2, 'G': 1, 'A': 2, 'T': 6}
```

To retrieve the A composition of every gene use a for loop.

```
>>> for gene in sorted(genes):
...    A_comp = genes[gene]['nt_comp']['A']
...    print(gene+":","As=", A_comp)
...
gene1: As= 2
gene2: As= 3
```

Building Complex Datastructures

Below is an example of building a list with a mixed collection of value types. Remember that all elements inside a list or dictionary should be the same type. In other words, the values in a list should all be lists or dictonaries or scalar values. This allows you to loop over the data structure.

This is a list with lists and a dictionary. The dictionary has a key with a value that is a dictionary.

```
[
   [1, 2, 3],
   [4, 5, 6],
   {
       'key': 'value',
       'key2':
            {
                'something_new': 'Yay'
            }
    }
 ]
```

Building this data structure in the interpreter:

```
>>> new_data = []
>>> new_data
[]
```

```
>>> new_data.append([1,2,3])
>>> new_data
[[1, 2, 3]]
>>> new_data[0]
[1, 2, 3]
>>> new_data.append([4,5,6])
>>> new_data
[[1, 2, 3], [4, 5, 6]]
>>> new_data[1]
[4, 5, 6]
>>> new_data[1][2]
6
>>> new_data.append({})
>>> new_data
[[1, 2, 3], [4, 5, 6], {}]
>>> new_data[2]['key']='value'
>>> new_data
[[1, 2, 3], [4, 5, 6], {'key': 'value'}]
>>> new_data[2]['key2']={}
>>> new_data
[[1, 2, 3], [4, 5, 6], {'key2': {}, 'key': 'value'}]
>>> new_data[2]['key2']['something_new']='Yay'
>>> new_data
[[1, 2, 3], [4, 5, 6], {'key2': {'something_new': 'Yay'}, 'key': 'value'}]
>>>
```

Same example in a script file: Building Complex Datastructures

Course T-shirt Organization and Counting

We have a spreadsheet of everyone's style, size, color. We want to know how many of each unique combination of style-size-color we need to order

shirts.txt

```
mens  small heather seafoam
womens  medium  Heather Purple
womens  medium  berry
mens  medium  heather coral silk
womens  Small Kiwi
Mens  large Graphite Heather
mens  large sport grey
mens  small Carolina Blue
```

We want something like this:

```
womens  small antique heliconia   2
womens  xs      heather orange      1
womens  medium  kiwi                2
womens  medium  royal heather       1
```

shirts.py

```
#!/usr/bin/env python3

shirts = {}
with open("shirts.txt","r") as file_object:
  for line in file_object:
    line = line.rstrip()
    [style, size, color] = line.split("\t")
    style = style.lower()
    size = size.lower()
    color = color.lower()
    if style not in shirts:
        shirts[style] = {}
    if size not in shirts[style]:
        shirts[style][size] = {}
    if color not in shirts[style][size]:
        shirts[style][size][color] = 0

    shirts[style][size][color] += 1

for style in shirts:
  for size in shirts[style]:
    for color in shirts[style][size]:
      count = shirts[style][size][color]
      print(style,size,color,count,sep="\t")
```

Output:

```
sro$ python3 shirts.py
mens  small heather maroon      1
mens  small royal blue          1
mens  small olive               1
mens  large graphite heather    1
womens  medium  heather purple      3
womens  medium  berry               2
womens  medium  royal heather       1
womens  medium  kiwi                2
...
```

This is what the data structure we just built looks likes

```
{
  'mens':
    {
      'small':
        {
           'heather seafoam': 1,
           'carolina blue': 1,
           'cornsilk': 1,
           'dark heather': 1,
           'heather maroon': 1,
           'royal blue': 1,
           'olive': 1
        },
      'large':
        {
          'graphite heather': 1,
          'sport grey': 1,
          'heather purple': 1,
          'heather coral silk': 1,
          'heather irish': 1,
          'heather royal': 1,
          'carolina blue': 1
        },
      'medium':
        {
          'heather coral silk': 1,
          'heather royal': 2,
          'heather galapagos blue': 1,
          'heather forest': 1,
          'gold': 1,
          'heather military green': 1,
          'dark heather': 1,
          'carolina blue': 1,
          'iris': 1
        },
      'xs':
        {
          'white': 1
        },
      'xl':
        {
          'heather cardinal': 1,
          'indigo blue': 1
        }
          },
  'womens':
    {
      'medium':
        {
          'heather purple': 3,
          'berry': 2,
          'royal heather': 1,
          'kiwi': 2,
          'carolina blue': 1
        },
      'small':
        {
          'kiwi': 1,
          'berry': 1,
          'antique heliconia': 2
        },
      'large':
        {
          'kiwi': 1
        },
      'xs':
        {
          'heather orange': 1
        }
    },
  'child':
    {
      '4t':
        {
          'green': 2
        },
        '3t':
          {
            'pink': 1
          },
        '2t':
          {
            'orange': 1
          },
        '6t':
          {
            'pink': 1
          }
    }
}
```

There are also specific data table and frame handling libraries like Pandas.

Here is a intro to data structures in Panda.

Here is a very nice interactive tutorial

Link to Python 8 Problem Set