Python for Astronomers

Lists, for loops, and file I/O

Exercises: String formatting

- Produce a pretty logarithmic table for the numbers 0.1, 0.2,... 10.0 which gives the logarithms for bases 2, e, and 10 to 7 digits.
- Hint: math.log lets you specify the base. (Consult math.log.__doc__.) Use "%10.7f" for string formatting.

Exercise: Solution

```
>>> import math
>>> # Define output format here:
>>> frmt = "\t%.1f" + "\t%10.7f"*3
>>> x = .1
>>> while x<10.05:
...     lb = math.log(x,2)
...     ln = math.log(x)
...     lg = math.log10(x)
...     # Use output format here:
...     print frmt % (x, lb, ln, lg)
...     x += .1
```

Lists

```
>>> # Ordered, heterogeneous container:
>>> l = ['spam', 'eggs', 42, 17.4]
>>> l

>>> l[0], l[-1]  # Indexing as for strings
>>> # Positive indices: 0 .. len(l)-1
>>> # Negative indices: -1 .. -len(l)

>>> l[1:-1]      # Slicing too
>>> # First index included, second not
```

Changing items

```
>>> l = ['spam', 'eggs', 42, 17.4]

>>> l[0] = 'ham'   # Lists are mutable!
>>> l[2] = 'circus' # Change type of item

>>> # Slices can be changed too!
>>> l[:2] = ['bacon', 'bread']

>>> # But be careful!
>>> l[:2] = ["Monty", "Python", "flying"]
```

Adding items

```
>>> l = ['spam', 'eggs', 42, 17.4]

>>> l[5] = 'boo'           # Will not work
>>> l.append('boo')        # But this will
>>> l.extend([1,2,3])      # Append list
>>> l.insert(1, 'bacon') # Insert item

>>> l + [1,2,3]           # Same as extend
>>> l + 3                 # No
>>> l * 3                 # Yes
```

Removing items

```
>>> l = range(20)

>>> del l[2]          # Remove single item
>>> del l[1:3]        # Remove slice
>>> del l[::3]        # Will this work?

>>> l[1:3] = []       # ok
>>> l[1] = []         # ???
>>> l[1:2] = []       # ok
>>> l[::3] = []       # Error
```

Some list methods

```
>>> l = [2, 5, 17, 1, 0, 1]

>>> len(l)              # Number of items
>>> 17 in l             # Test if item present

>>> l.sort()            # In place!
>>> l
>>> l.reverse()         # Also in place
>>> l

>>> m = l[:]            # Copies list
```

Exercise: More list methods

- Explore further methods by looking at the online documentation and trying things out

```
>>> # Hint:
>>> l = [1,2,3]
>>> l.<TAB>     # Gives list of methods
>>> print l.sort.__doc__ # Prints doc
```

Some special lists

```
>>> []                  # Empty list

>>> # May be used e.g. in while statement:
>>> l = [1,2,3]
>>> while l:            # Equivalent to l!=[]
...     del l[-1]
...     print l

>>> range(100)          # 0, 1, ..., 99
>>> range(13,42,3)
```

Lists are universal

```
>>> # Keep in mind:
>>> # Lists may contain really anything!

>>> import math
>>> l = [math.pi, math.sin, math]
>>> print l[0]
>>> print l[1](l[0]/4.)
>>> print l[-1].e

>>> # In particular...
```

Nested lists

```
>>> # ...lists may contain lists:
>>> l = [[1,2], [3,4,5], 'spam']

>>> l[0]        # First item: a list
>>> l[0][1]     # An item of the inner list
>>> l[-1][1]    # This works too

>>> # Slicing:
>>> l[:2]       # As expected
>>> l[1][1:]    # Again as expected
>>> l[:2][0]    # Surprise?
```

Nested lists

```
>>> # Lists may be deeply nested:
>>> l = [1,[2,3],[4,[5,6]]]
>>> l[-1]
>>> l[-1][-1]
>>> l[-1][-1][-1]

>>> # Pathological:
>>> l = [0]
>>> l[0] = l
```

Exercise: Nested lists

- Experiment:

```
>>> l = [[1,2,3], [4,5]]
>>> l = [0][1]
>>> l = [1][0]
>>> l = [0][2]
>>> l = [2][0]
>>> l = [-1]
>>> l = [-1:]
>>> l = [-1][-1]
```

The for statement

```
>>> for i in [17,4,2]:
...   # i takes all values of list
...   print i, i*2
... <RETURN>

>>> for i in range(5):
...   print i, i*2
... <RETURN>
```

The for statement

```
>>> # Also for heterogeneous lists:
>>> for i in ['spam', 42, [1,2,3]]:
...   print i, i*2
... <RETURN>

>>> # which may lead to problems:
>>> for i in ['spam', 42, [1,2,3]]:
...   print i, i**2
... <RETURN>
```

The for statement

```
>>> # The loop variable may run over other
>>> # sequences too:
>>> for c in 'spam':
...   print c, c*2
... <RETURN>
```

The for statement

```
>>> # Nested for loops:
>>> for i in range(1,5):     # Loop var: i
...   for j in range(1,i+1): # Loop var: j
...     print i, j, i*j

>>> # May be nested with while loops
```

Exercises

- Calculate the sum of a nested list:

```
>>> l = [[1,2], [3,4,5]]
>>> sum = 0
>>> for inner in l:
...     for item in inner:
...         sum += item
```

- Will this also work for:

```
>>> l = [[1,2], 3, 4, 5]
>>> l = [[1,2], []]
>>> l = []
>>> l = [[]]
```

ASCII File I/O

```
>>> f = open("/home/rschaaf/test.txt")
>>> # Read everything into single string:
>>> content = f.read()
>>> len(content)
>>> print content
>>> f.read()                # At End Of File
>>> f.close()

>>> # f.read(20) reads (at most) 20 bytes
```

ASCII File I/O

```
>>> # Use arrow keys to get cmd back!
>>> f = open("/home/rschaaf/test.txt")
>>> f.readline()  # Read single line
>>> f.readline()
>>> f.readline()  # Only \n
>>> f.readline()
>>> f.readline()  # Empty string: EOF
```

ASCII File I/O

```
>>> f.seek(0)  # Move to start of file
>>> # Read with while loop:
>>> line = f.readline()
>>> while line:
...     print line,    # No 2nd newline added
...     line = f.readline()
... <RETURN>
```

ASCII File I/O

```
>>> f.seek(0)
>>> # Shorter with readlines and for loop:
>>> lines = f.readlines() # Memory?
>>> lines
>>> for line in lines:
...   print line,
... <RETURN>
```

ASCII File I/O

```
>>> f.seek(0)
>>> # Even shorter and memory efficient:
>>> for line in f:
...     print line,
... <RETURN>
>>> f.close()
```

ASCII File I/O

```
>>> # Now write to files:
>>> # If mytest.dat exists, it will be
>>> # overwritten!
>>> f = open("mytest.dat", "w")
>>> f.write("Line 1\n")
>>> f.write("Line 2\n")
>>> f.close() # Data actually written here
```

ASCII File I/O

```
>>> # Now append data to mytest.dat:
>>> f = open("mytest.dat", "a") # Append
>>> lines = ["Line 3\n", "Line 4\n"]
>>> f.writelines(lines)
>>> f.close()
```

Exercise: More on File I/O

- Explore further methods by looking at the online documentation and trying things out

```
>>> # Hint:
>>> f = open("mytest.dat")
>>> print f.__doc__
>>> f.<TAB>     # Gives list of methods
>>> print f.flush.__doc__ # Prints doc
```

Exercises

- File /home/rschaaf/test.dat contains two columns with (random) floating point data. Create a new file that has three columns: Columns 1 and 2 are the data from /home/rschaaf/test.dat, column 3 the sum of the numbers in column 1 and 2.