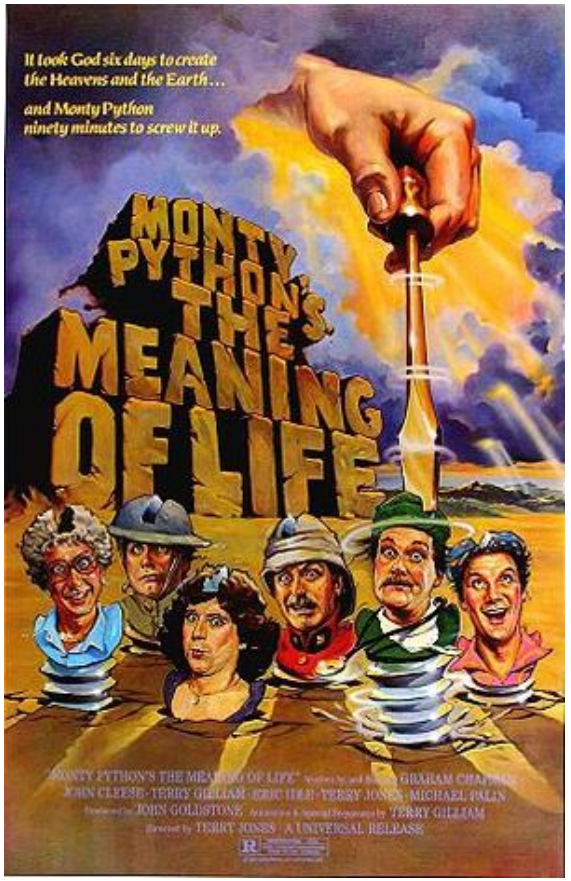
Python Files and Lists



Files

- Chapter 9 actually introduces you to opening up files for reading
- Chapter 14 has more on file I/O
- Python can read a plain text files using the built-in function open
  - Takes the name of the file as a parameter and returns a file object

```
>>>infile_name = input ('Enter input file name : ')
>>>fin = open(infile_name)
<open file 'words.txt', mode 'r' at 0xb7f4b380>
```

File Methods for reading

- Several methods for reading

```
>>>fin.readline()
'aa\r\n'
```

- If white space bothers you, use method strip

```
>>>line = fin.readline()
>>>s = line.strip()          #or line.rstrip()
>>>print (word)
aa
```

Read lines from a file

- Can use a file object as part of a for loop

```
fname = input('Enter file name: ')
fin = open(fname)
for line in fin:
     s= line.rstrip()
```

- Close the file when you are done reading

```
fin.close()
```

Using try, except and open

```
fname = input ('Enter file name: ')
try:
    fin = open (fname)
except:
    print(fname + ' file cannot be opened')
    exit()

#rest of code
```

File Output

- Default mode when opening file is read, to write …include a second parameter

```
>>>fout = open('myresults.txt', 'w')
>>>print (fout)
<open file 'myresults.txt', mode 'w' at 0xb7eb2410>
```

- If the file already exists, erases file and overwrites it
- If not, creates a new one

Writing to a file

- Use the write method

```
>>>line1 = "This here's a result.\n"
>>>fout.write(line1)
This here's a result.
```

- Close the file when you are done

```
>>>fout.close
```

Format operator

- The argument to write has to be a string
  - Easiest way to do this is with `str()`

```
>>>x = 15
>>>fout.writeline(str(x))
```

Or

```
>>>fout.writeline('{}'.format(x))
```

- Another way to do it is with the format operator %
  - % with integers is modulus, but when the first operand is a string it is a format operator
  - The first operand is the format string containing one or more format sequences, which specify how the second operand is to be printed

`'%d'` means the second operand is an `integer`

```
>>>n = 14
>>>'%d' % 14
'14'
```

Format operator

- The format operator can appear anywhere in a string

```
>>>print ('There are %d apples' % n)
'There are 14 apples'
```

- If there is more than more than one format sequence in a string, then the second argument has to be a tuple

```
>>>print ('There are %g %s left in the
half.' % (3.14, 'minutes'))
'There are 3.14 minutes left in the half.'
```

Format operator

- Number of arguments and types of arguments have to match

```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: illegal argument type for built-in operation
```

Filenames and paths

- Files are organized into *directories* (or *folders)*
- Can get access use `os` module

```
>>>import os
>>>cwd = os.getcwd()
>>>print (cwd)
home/stringfellow
```

- Files we've opened so far have been in the current working directory
  - A relative path starts from the current working directory
  - An absolute path starts from the topmost directory

os methods

```
>>>os.path.abspath('data.txt')
home/stringfellow/programs/prog1/data.txt
>>>os.path.exists('data.txt')
True
>>>os.path.exists('home/stringfellow')
True
>>>os.path.isdir('data.txt')
False
>>>os.path.isfile('data.txt'
True
>>>os.listdir(cwd)
['prog1.py', 'data.txt']
```

Traverse directory example

- Following walks through a directory, prints the names of all the files or recursively calls itself on all its directories

```
def walk(dirname):
    for name in os.listdir(dirname):
        #join the directory name with the filename
        path = os.path.join(dirname, name)
        if os.path.isfile(path):
            print (path)
        else:
            walk(path)
```

DEMO

Writing modules

- Any file that contains Python code can be imported as a module
- Suppose you wrote some code in a file called mylib.py

```
def linecount(fname):
    count = 0
    fin = open(fname)
    for line in fin:
        count = count + 1
    return count
```

- Now import it and call it

```
>>>import mylib
>>>print(mylib.linecount('mylib.py'))
6
```

Lists

- A String is a sequence of characters
- Access characters one at a time with a bracket operator and an offset index

```
>>>fruit = 'banana'
>>>letter = fruit[1]
>>>print (letter)
a
```

  - Index must be an integer

```
>>>letter = fruit[1.5]
TypeError: string indices must be integers, not float
```

More lists

```
>>>cheeses = [‘Edam’, ‘Swiss’, ‘American’]
>>>numbers = [17, 21]
>>>empty = [ ]
>>>nested = [‘abc’, ‘def’, ‘ghi’, [0, 1]]
>>>print (cheeses, numbers, empty, nested)
[‘Edam’, ‘Swiss’, ‘American’] [17, 21] [ ] [‘abc’, ‘def’, ‘ghi’, [0, 1]]
```

Accessing elements in a list

- Access element in a list similar to a string

```
>>>print (cheeses[0])
'Edam'
```

- Unlike strings, lists are ***mutable***, so elements can be changed

```
>>>cheeses[0] = 'brie'
>>>print (cheeses[0])
'brie'
```

- List indices must be an integer expression
- If you access out of bounds → `IndexError`
- If index is negative, counts back from end of list

```
>>>print (cheeses[-1])
'American'
```

In operator

- The `in` operator works on lists too.

```
>>>cheeses = ['Edam', 'Swiss', 'American']
>>>'Edam' in cheeses
True
>>>'Havarti' in cheeses
False
```

Traversing a list

- for each loop

```
for cheese in cheeses:
    print (cheese)
```

- Need index to access/change element in list

```
for i in range(len(numbers)):
    numbers[i] = numbers[i]*numbers[i]
```

- A loop that never executes

```
for item in []:
    print ('This never happens')
```

List operations

- + operator concatenates

```
>>>list1 = [‘a’, ‘b’, ‘c’]
>>>list2 = [‘d’, ‘e’, ‘f’]
>>>list3 = list1 + list2
>>>print (list3)
[‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
```

- * repeats list certain number of times

```
>>>print ([1,2,3]*2)
[1, 2, 3, 1, 2, 3]
```

List slices

- Lists also have slices

```
>>>t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>t[1:3]
['b', 'c']
>>>t[:4]
['a', 'b', 'c', 'd']
>>>t[3:]
['d', 'e', 'f']
>>>t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

- Because lists are mutable

```
>>>t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>t[1:3] = ['x', 'y']
>>>print (t)
['a', 'x', 'y', 'd', 'e', 'f']
```

List methods

- Append adds a new element to the end of the list

```
>>> t= [1, 2, 3]
>>> t.append(4)
>>> print (t)
[1, 2, 3, 4]
```

- Extend adds a list of elements to the end of the list

```
>>> t2 = [5, 6]
>>> t.extend(t2)
>>> print (t)
[1, 2, 3, 4, 5, 6]
```

sort

- Arranges elements in a list from low to high

```
>>> t= [5, 3, 1, 2, 4]
>>> t.sort()
>>> print (t)
[1, 2, 3, 4, 5]
```

- NOTE: Most list methods return `void` so

```
>>> t = t.sort()
```

will disappoint, t will be set to `None`

Deleting elements from a list

- If you know index, use `pop`. If you don't provide an index, it pops the last element.

```
>>>t = ['a', 'b', 'c']
>>>x = t.pop(1)
>>>print ('list: ' + t + ' top: ' + x)
list: ['a', 'c'] top: 'b'
```

- If you know index, but don't need the removed element use `del`

```
>>>t = ['a', 'b', 'c']
>>>del t[1]
>>>print (t)
['a', 'c']
```

Deleting elements from a list

- If you know the element to remove, but not the index, use remove

```
>>>t = ['a', 'b', 'c']
>>>x = t.remove('b')
>>>print (t)
['a', 'c']
```

- Can delete a slice

```
>>>t = ['a', 'b', 'c', 'd', 'e', 'f']
>>>del t[1:3]
>>>print (t)
['a', 'd', 'e', 'f']
```

Lists and functions

- Number of list functions that let you ask certain questions about lists

```
>>>nums = [31, 72, 23, 14, 5, 36]
>>>print (len(nums))
6
>>>print (max(nums))
72
>>>print (min(nums))
5
>>>print (sum(nums))
181
```

- `max` and `min` only works if elements are comparable
- `sum` only works if elements are numbers

Example

- Find the average of a list of numbers

```
numlist = list()
s = input ('Enter a number: ')
while s != 'done'
    x = float (s)
    numlist.append(x)
    s = input ('Enter a number: ')

if len(numslist) > 0:
    average = sum(numlist)/len(numlist)
    print ('Average is ' + average)
else:
    print ('No numbers to average')
```

Lists and strings

- A string is a sequence of characters and a list is a sequence of values
  - But a list of characters is not a string
  - You can, however, convert a string to a list with the `list` function

```
>>>s = 'spam'
>>>t = list(s)
print (t)
['s', 'p', 'a', 'm']
```

- You can convert a list to a string with the `join` function
  - See later

split

- `split` breaks a string into a list of tokens or words

```
>>>s = 'The dog chased the cat'
>>>tokens = s.split()
>>>print (tokens)
['The', 'dog', 'chased', 'the', 'cat']
```

Can call `split` with a delimiter

```
>>>s = "11:14:15"
>>>tokens = s.split(':')
>>>print (tokens)
[11, 14, 15]
```

join

- `join` is the opposite of `split`
- It takes a list of strings and concatenates them
- It is a string method, so invoke it on the delimiter string and pass it a list argument

```
>>>t = [‘The’, ‘cow’, ‘jumped’, ‘over’, ‘the’, ‘moon’]
>>>delimiter = ‘ ‘
>>>delimiter.join(t)
>>>print (t)
‘The cow jumped over the moon’
```

- To concatenate without spaces, use empty string ‘’

```
>>>t= ['e', 'g', 'g', 's']
>>>s = ''.join(t)
>>>print (s)
eggs
```

Objects and values

- Bindings are different for strings and lists

```
>>>a = 'banana'
>>>b = 'banana'
```

- Question: do they refer to the same *object?*
  - *Answer: Use* `is` *operator*

```
>>>a is b
True
```

Figure 10.2: State diagram.

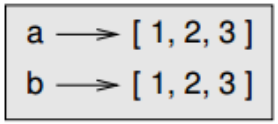


Figure 10.3: State diagram.

- Lists will refer to different objects.

```
>>>list1 = [1, 2, 3]
>>>list2 = [1, 2, 3]
>>>list1 is list2
False
```

- a and b are *identical,* while list1 and list 2 are *equivalent*

Aliasing

- If you assign one object to another, you create an alias

```
>>>a = [1, 2, 3]
>>>a = b
>>>a is b
True
```

- If an object is mutable, you will affect the aliased referenced object

```
>>>a[0] = 0
>>>print (b)
[0, 2, 3]
```

List Arguments

- When you pass a list as an argument to a function, you are passing a *reference* to the list!
  - If the function modifies the object, the caller sees an effect

```
def delete_head(t)
   del t[0]

list1 = [1, 2, 3, 4]
delete_head (list1)
print (list1)
```

  - Will give [2, 3, 4]

List arguments

- Important to distinguish between functions that modify lists and functions that create new lists
  - `append` modifies a list
  - `+` creates a new list

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print t1
[1, 2, 3]
>>> print t2
None

>>> t3 = t1 + [3]
>>> print t3
[1, 2, 3]
>>> t2 is t3
False
```

CAREFUL!!!

- Note: `list` is a function, do not use it for a variable name!

List as a queue

- Demo testqueue.py