File I/O Reference Sheet

- Opening a file
  - <variable>=open(<“filename”>,<“mode”>)
  - Three modes:
    - “r”; read
    - “w”; write
    - “a”; append
  - Ex.: myFile=open(“myFile.txt”, “r”)
- Reading from files:
  - Three functions:
    - .read()/.read(int)
      - .read() returns a string containing all the characters in the file.
      - .read(int) returns a string containing the first <int> characters in the file.
    - .readline()
      - returns a string containing the next line in the file up to the next newline character.
    - .readlines()
      - returns a list of strings of all the lines in the file.
- Writing to files:
  - When a file is opened for writing (ie. open(<filename>, “w”)), a new file is created in your current working directory. If that file exists, everything in the file is erased.
  - One function:
    - .write(<“text”>)
      - writes the string to the file
      - To write non-stings to file cast them as strings then write.
        - .write(str(non-string))
- Appending to files[1]:
  - When a file is opened for appending(ie. open(<filename>, “a”)), the buffer for writing begins at the end of the file
  - One function:
    - .write(<“text”>)
      - writes the string to the file

1 Appending not covered in class, for your information.

CS 1301

Functional Programming Reference Sheet*

- Advantages of functional programming
  - Can take advantage of parallel processing
    - Allows computer processors to divide the data needed to be processed and execute code on multiple cores/CPU's at the same time.
  - 3 python functions
  - map(<function name>,<sequence>)
    - Applies the function to every item in the sequence.
    - Ex. There's a function called addOne(num) that takes in a number and adds one to it.
      - map(addOne,[1,2,3,4]) → [2,3,4,5]
    - Ex. There's a function called add(num1,num2) that takes in two numbers and adds them together.
      - map(add,[1,2,3,4],[1,2,3,4]) → [2,4,6,8]
      - In this case map uses the first list as the first parameter and the second, as the second parameter.
  - filter(<function name>,<sequence>)
    - Applies the function to every item in the sequence. If the function returns TRUE (or something that evaluates to TRUE, such as any non-zero value), filter will add that item to the list of values to return. Any items for which the function returns FALSE (or zero) will not be included in the returned list.
      - Ex. There's a function called firstS(string) that takes in a string and returns True, if the first letter of the string is an S it returns True, otherwise it returns False.
      - filter(firstS,[“apple”, “sand”, “ferst”, “shallow”]) → [“sand”, “shallow”]
  - reduce(<function name>,<sequence>)
    - Applies the function to every item in the list to in order to reduce it to one item.
      - Ex. There's a function called multiply(num1, num2) that multiplies the first number by the second number and returns the result. Reduce would multiply the first two numbers, then multiply the result by the third, the the next result by the fourth, etc. and return the final answer.
      - reduce(multiply,[5,4,3,2,1]) → 120
- Lambda Functions
  - Lambda functions allow a programmer to quickly write short functions in one line rather than having to define whole functions to do a job.
  - Syntax:
    - lambda [parameters]:[expression]
      - Expressions cannot contain any block code, so no loops, if statements, etc. Lambda functions are mostly just used for expressions.
    - Ex:
      ```
      def add(x,y):
          return x+y
      ```
    - Is exactly the same as writing:
      ```
      add=lambda x,y:x+y
      ```

* Functions addOne, add, multiply, firstS; are not built into python, they are examples of code that could be written and used.