**1 Lists**

(Chapter 11 in zyBooks and Chapter 10 in *How to Think like a Computer Scientist*)

* Describe what is a list
  + A sequence of times with one variable name
  + A list is considered a data structure
    - A data structure is a way of storing and organizing information; a composite of related data items
  + Lists can have items of different types
  + Lists are mutable
* Declare and apply single and 2-dimensional lists (nested lists)
  + Use [ ] (hard brackets) to declare lists
  + list()
* Define what is an index
  + Recall list indexing in C starts at 0
* Lists and loops
  + Use a loop to initialize each item in a list
  + Use loops to traverse through lists
* Write functions which accept lists (single and 2-dimensional) as parameters
* What happens when a list is passed to a function?
  + The list reference is copied and passed into the function
  + Via the reference, the function can modify the list
* Return lists from functions
* How to get the number of items in a list?
* Understand and apply list operators (e.g. concatenation, repetition, slice, etc.)
* Understand and apply list methods (e.g. append(), extend(), etc.)
* Understand and apply string methods that return lists (e.g. join(), split(), etc.)
* Remove items in a list
* Describe what is aliasing
  + When two or more references refer to the same object
  + Difference between how Python creates objects for mutable types (e.g. lists) and immutable types (e.g. strings)

**2 Tuples**

(Chapter 10 in *How to Think like a Computer Scientist*)

* Describe what is a tuple
  + An immutable list
* Declare and apply single and 2-dimensional tuples (nested tuples)
  + Use () and , (parentheses and/or commas) to declare tuples
  + tuple()
* How to get the number of items in a tuple?
* Tuple indexing
* Tuple slicing

**3 Dictionaries**

(Chapter 12 in zyBooks and Chapter 12 in *How to Think like a Computer Scientist*)

* Describe what are keys
* Describe what are values
* Describe what is a mapping
* Describe what are key-value pairs
* Describe what is a dictionary
  + A list with keys as indices
  + Dictionaries are mutable
  + Valid data types for keys and values
* Declare and apply single and 2-dimensional dictionaries (nested dictionaries)
  + Use { } (curly brackets) to declare dictionaries
  + dict()
* Convert tuples to dictionaries and dictionaries to tuples
* Dictionary indexing
* Dictionaries and loops
* Adding key-value pairs
* Testing existence of a key
* How to get the number of key-value pairs in a dictionary?
* Understand and apply dictionary methods (e.g. items(), etc.)

**Other Topics**

* Working with APIs
* 8x8 light board hardware/software
  + RGB values
* time module
* Converting decimal to binary
* Converting binary to decimal

**Recommended Strategy for Preparing for the Exam**

The exam questions will require you to solve problems in a variety of formats: defining key terms, evaluating expressions, writing code, determining the output of code, etc.

I recommend that you use the following activities and materials to prepare for the exam:

* Review micro assignments, lab exercises, programming assignments, and problems solved in class
  + These may well be your best resources. An excellent learning activity would be to re-solve the micro assignments, lab exercises, in class exercises, and programming assignments.
* Lesson notes and example code
* Read the textbook
  + Read or re-read zyBooks chapters 11-12 and How to Think Like a Computer Scientist chapters 10 and 12.

**Extra Practice Problems**

**1**

Write a program to roll a 6-sided die 1,000 times. Use a dictionary to store the count of each die face to determine the percentage of each outcome. Are the percentages what you would expect them to be?

Repeat the above problem for rolling two 6-sided dice (i.e. the sum of both dice). Are the percentages what you would expect them to be?

In [104]:

**import** **random**

**def** roll\_die\_n\_times(n):

*'''*

*'''*

results = {}

**for** i **in** range(n):

random\_roll = random.randint(1, 6)

**if** random\_roll **in** results:

results[random\_roll] += 1

**else**: *# random\_roll is not a key in results dict*

results[random\_roll] = 1

**return** results

roll\_dict = roll\_die\_n\_times(100000)

print(roll\_dict)

**for** roll **in** roll\_dict:

count = roll\_dict[roll]

print("**%d** rolled **%.2f%%**" %(roll, count / 100000 \* 100))

{1: 16799, 2: 16663, 3: 16547, 4: 16778, 5: 16543, 6: 16670}

1 rolled 16.80%

2 rolled 16.66%

3 rolled 16.55%

4 rolled 16.78%

5 rolled 16.54%

6 rolled 16.67%

**2**

Write a function called my\_str\_concatenate(str1, str2) that implements the functionality of str1 + str2 (string concatenation). **Do not use the string concatenation operator + in your solution.**

In [103]:

**def** my\_str\_concatenate(str1, str2):

*'''*

*'''*

str1\_list = list(str1)

str2\_list = list(str2)

str1\_list.extend(str2\_list)

merged\_string = "".join(str1\_list)

**return** merged\_string

print("hello" + "goodbye")

new\_string = my\_str\_concatenate("hello", "goodbye")

print(new\_string)

hellogoodbye

hellogoodbye

**3**

Write a function called my\_split(astring, delimiter) that implements the functionality of the string method split(). The parameter delimiter is a string of *one or more characters* to break a string astring into pieces. This function should return a list of strings in astring that are separated by delimiter. **Do not use the string method split() in your solution.**

Example: my\_split("hello??how??are??you", "??") returns the list ["hello", "how", "are", "you"]

In [102]:

**def** my\_split(astring, delim):

*'''*

*'''*

accum = ""

pieces = []

**for** i, char **in** enumerate(astring):

**if** char == delim:

pieces.append(accum)

accum = ""

**else**:

accum += char

**if** len(accum) > 0:

pieces.append(accum)

**return** pieces

**def** my\_split\_long\_delim(astring, delim):

*'''*

*'''*

dl = len(delim)

pieces = []

start\_index = 0

i = 0

**while** i < len(astring) - dl:

**if** astring[i:i+dl] == delim:

pieces.append(astring[start\_index:i])

i = start\_index = i + dl

**else**:

i += 1

**if** start\_index < i:

pieces.append(astring[start\_index:])

**return** pieces

words = my\_split("hello how are you", " ")

print(words)

words2 = my\_split\_long\_delim("hello??how??are??you", "??")

print(words2)

['hello', 'how', 'are', 'you']

['hello', 'how', 'are', 'you']

**4**

Write a program that prompts the user for a *single letter*. If the user enters a valid letter (upper or lower case), tell them a *random* word that *starts with that letter* from [words.txt](http://thinkpython2.com/code/words.txt). If the user enters a string of more than one letter, tell them their response is invalid and re-prompt until a valid letter is entered.

Define functions where appropriate!

In [105]:

**import** **random**

**def** get\_letter():

*'''*

*'''*

letter = ""

**while** letter == "":

letter = input("Please enter a letter: ")

**if** len(letter) > 1:

print("INVALID!!!")

letter = ""

**return** letter

**def** get\_random\_word(infile, letter):

word = ""

start = -1

end = -1

lines = infile.readlines()

**for** i **in** range(len(lines)):

line = lines[i].strip()

**if** start == -1 **and** line[0] == letter:

start = i

**if** start != -1 **and** end == -1 **and** line[0] != letter:

end = i *# end is exclusive*

rand\_index = random.randrange(start, end)

word = lines[rand\_index].strip()

**return** word

letter = get\_letter()

infile = open("words.txt", "r")

word = get\_random\_word(infile, letter)

print(word)

infile.close()

Please enter a letter: b

buggering

**5**

Download [titanic.txt](https://raw.githubusercontent.com/gsprint23/cpts111/master/lessons/files/titanic.txt). This dataset is from this [site](https://rstudio-pubs-static.s3.amazonaws.com/108515_e5d253e6997545e881759eb458b6ba61.html).

Each row in this file is a comma-separated list of strings representing attributes of passengers aboard the Titanic:

* class: 0 = crew, 1 = first class, 2 = second class, 3 = third class
* age: 1 = adult, 0 = child
* sex: 1 = male, 0 = female
* survived: 1 = yes, 0 = no

Note: the first line in the file is the header describing the order of the attributes. Each line after the header represents a single passenger's attributes.

Based on this data, answer the following questions:

1. How many people on board (according to this dataset)?
2. How many survived? Of the total number of people, what percentage survived?
3. How many children survived? Of the total number of children, what percentage survived?
4. How many male crew members survived? Of the total male crew members, what percentage survived?
   * What about the female crew members?
   * What are your observations about these findings?

Write your solution such that it handles any order of the attributes in the file. The header will tell you this order! This means, do not hard code the indices of the attributes.

In [48]:

**def** titanic\_main():

*'''*

*'''*

infile = open("titanic.txt", "r")

labels = infile.readline().strip()

labels = labels.split(",")

age\_index = labels.index("age")

surv\_index = labels.index("survived")

class\_index = labels.index("class")

sex\_index = labels.index("sex")

total = 0

total\_survived = 0

total\_children = 0

total\_children\_survived = 0

total\_male\_crew = 0

total\_female\_crew = 0

total\_male\_crew\_survived = 0

total\_female\_crew\_survived = 0

**for** line **in** infile.readlines():

line = line.strip().split(",")

**if** line[surv\_index] == "1":

total\_survived += 1

**if** line[age\_index] == "0":

total\_children\_survived += 1

**if** line[class\_index] == "0" **and** line[sex\_index] == "1":

total\_male\_crew\_survived += 1

**if** line[class\_index] == "0" **and** line[sex\_index] == "0":

total\_female\_crew\_survived += 1

**if** line[age\_index] == "0":

total\_children += 1

**if** line[class\_index] == "0" **and** line[sex\_index] == "1":

total\_male\_crew += 1

**if** line[class\_index] == "0" **and** line[sex\_index] == "0":

total\_female\_crew += 1

total += 1

infile.close()

print("1) Total people on board: **%d**" %(total))

print("2) Total survivors: **%d** (**%.2f%%**)" %(total\_survived, total\_survived / total \* 100))

print("3) Total children survivors: **%d** (**%.2f%%** of all children)" %(total\_children\_survived, total\_children\_survived / total\_children \* 100))

print("4) Total male crew member survivors: **%d** (**%.2f%%** of all male crew members)" %(total\_male\_crew\_survived, total\_male\_crew\_survived / total\_male\_crew \* 100))

print("4\*) Total female crew member survivors: **%d** (**%.2f%%** of all female crew members)" %(total\_female\_crew\_survived, total\_female\_crew\_survived / total\_female\_crew \* 100))

titanic\_main()

1) Total people on board: 2201

2) Total survivors: 711 (32.30%)

3) Total children survivors: 57 (52.29% of all children)

4) Total male crew member survivors: 192 (22.27% of all male crew members)

4\*) Total female crew member survivors: 20 (86.96% of all female crew members)

**6**

By hand:

1. Convert 11001 to decimal
2. Convert 132 to binary
3. How many bits are needed to store the decimal number 15? A bit is a single 0 or 1 digit.
4. Convert the following message to English using the [unicode](http://dev.networkerror.org/utf8/?start=33&end=133&cols=4&search=&show_uni_int=on&show_uni_hex=on&show_html_ent=on&show_raw_hex=on&show_raw_bin=on) table of values

01001000 01100101 01101100 01101100 01101111 00101100 00100000

01110111 01101111 01110010 01101100 01100100

**7**

Write a function called build\_jagged2d() that accepts an integer value greater than 1 as an argument. The function creates a nested list of integers where the first row has one zero, the second row has two zeroes, and so on. The function returns the two dimensional list.

Example: build\_jagged2d(3) returns a two dimensional list of three rows, the first row having one zero, the second having two zeros, and the third having three zeroes. Logically, this looks like the following:

0

0 0

0 0 0

Programmatically, this looks like: [[0], [0, 0], [0, 0, 0]]

**8**

Write a function called compute\_mins\_seconds(ms) that accepts a duration in milliseconds and returns a tuple of minutes and seconds. For example, compute\_min\_seconds(125000) would return the tuple (2, 5) corresponding to 2 minutes and 5 seconds in 125000 milliseconds.

**9**

For the following problems, we will need to download a file: [words.txt](http://thinkpython2.com/code/words.txt). This file contains 113,809 official crossword words, one per line. Using words.txt, write a program with the following functionality:

1. count\_words(): accepts the file object as an argument and returns the number of words in the file.
2. avg\_word\_length(): accepts the file object as an argument and returns the average number of characters per word.
3. write\_word\_lengths(): accepts the file object as an argument and for each word in words.txt, writes the number of characters in the word to a file (lengths.txt), one number per line.

You may choose to define/call additional functions if you wish.

**10**

Given the following table of data:

| **ID Number** | **Last name** | **First name** |
| --- | --- | --- |
| 28905 | Smith | Jane |
| 34590 | Johnson | Jane |
| 19485 | Smith | John |
| 28450 | Smith | John |
| 17834 | Anderson | John |

Write code to perform the following:

1. Populate a dictionary called students with the student information in the table. Keys in the dictionary should be ID Numbers and values in the dictionary should be two-item tuples containing first and last names.
2. Display each key-value pair in students in the following form: 28905: Smith, Jane
3. Display the first letter of each last name in students, all on one line separated by commas. Example: S, J, S, S, A
   * Now display the list of last name first letters in sorted order. Example: A, J, S, S, S
4. Add another student, 19654 Janet Smithy, to students
5. Remove ID number 28450 from students
6. Convert students to a list of tuples and print the list

**11**

Define and test the following two functions which operate on 2D lists of integers:

1. reverse\_rows(list\_ints): reverses the items in list\_ints by rows
2. reverse\_cols(list\_ints): reverse the items in list\_ints by columns

For example, if list\_ints =[[1, 2, 3], [4, 5, 6]], reverse\_rows(list\_ints) would modify list\_ints to be [[3, 2, 1], [6, 5, 4]] and reverse\_cols(list\_ints) would modify list\_ints to be [[4, 5, 6], [1, 2, 3]]

**12**

Write a function called geometric\_sum() that accepts one positive integer argument, n. The function returns the sum of the numbers:

For example, geometric\_sum(4) returns 1.875000

**13**

Download [mobydick.txt](https://raw.githubusercontent.com/gsprint23/cpts111/master/lessons/files/mobydick.txt) which is a text file that contains all of the text from the novel Moby Dick. Determine the 5 most common words in Moby Dick.

Hint: use a dictionary!

**14**

Write a function called count\_down() that accepts a parameter called start and returns a sequence of numbers starting at start and counting down to zero in increments of two. For example: count\_down(7) returns [7, 5, 3, 1]

**15**

In class we wrote a function that removed all instances of a number from a list of sorted numbers. We solved this by sorting the list of numbers, finding the first occurrence of the number to remove in the list, counted the number of occurrences, and deleted the slice: del nums[index:index + count]. This solution works because the list is sorted.

Write a new function that removes all instances of a number from a list of *unsorted* numbers. Solve this without sorting the list and without declaring/using a second list.

Hint: while there are instances of the number to remove in the list, delete one instance in the list.