PROBLEM 1:  (List Comprehensions (18 pts))

For problem Part A, give the output. For each of the remaining problems, assign the variable result to a Python expression that calculates the answer. That means if we changed the list given, your code would still calculate the correct answer.

Each of these must be written in one line and include a list comprehension.

Here is an example.

The variable result should calculate the list of words from list vehicles that have the letter 'a' in their word. Assume each string in the list is one word that is lowercase.

Using the list vehicles below, result would calculate the list:
['train', 'airplane', 'car', 'longboard']

vehicles = ['train', 'airplane', 'car', 'truck', 'bike', 'longboard']
result =

ANSWER:
result = [w for w in vehicles if 'a' in w]

PART A (2 pts)

In this part we have given you the list comprehension and you need to give the output when this code executes.

lst = [6, 8, 2, 1]
result = [n-1 for n in lst]
print(result)
The output is:

[5, 7, 1, 0]
PART B (4 pts)
The variable result should calculate the list of abbreviations for each word in the list lst, where an abbreviation is the first three letters of the word. You can assume all words in lst contain at least three letters, and the abbreviations in result are in the same order their corresponding words are in lst.
Using the list lst below, result would calculate the list: ['ros', 'cur', 'bas', 'thy', 'sag']

\[
\text{lst} = \text{['rosemary', 'curry', 'basil', 'thyme', 'sage']} \]

\[
\text{result} = \left[ w[3] \text{ for } w \text{ in lst} \right]
\]

PART C (4 pts)
The variable result should create a list of the words from the list lst that are of length 4, 5, or 6. The words in result should appear in the same order they appear in lst.
Using the list lst below, result would calculate the list: ['lime', 'maroon', 'olive']

\[
\text{lst} = \text{['lime', 'fushsia', 'maroon', 'red', 'dandelion', 'olive']} \]

\[
\text{result} = \left[ w \text{ for } w \text{ in lst if } \text{len}(w) \geq 4 \text{ and } \text{len}(w) \leq 6 \right]
\]
PART D (4 pts)
The variable result should calculate the list of numbers from the list lst that are divisible by 5. The numbers in result should appear in the same order they appeared in lst.
Using the list lst below, result would calculate the list: [40, 75, 30]

\[
\text{result} = \left[ \begin{array}{l}
\text{num for num in lst if num \% 5 == 0}
\end{array} \right]
\]

\[
\text{lst} = [83, 40, 62, 75, 30, 86, 22]
\]

PART E (4 pts)
The variable result should create a list of the words from the list lst that start with the word "blue". The words in result should appear in the same order they appear in lst.
Using the list lst below, result would calculate the list: ['bluegreen', 'bluemoon']

\[
\text{result} = \left[ \begin{array}{l}
\text{word for word in lst if word[0:4] == ''blue''}
\end{array} \right]
\]

\[
\text{lst} = ['bluegreen', 'lightblue', 'red', 'bluemoon', 'lightbluemoon']
\]
PROBLEM 2:  *(Splitting and Joining (6 pts))*

For each of the following problems, use only what is indicated to set result to a Python expression. Do not use any Python methods. Do not use concatenation or splicing.

**PART A**

Given string `str` below with its assigned value, use only join and split to make the new phrase "cars-big-boats-big-planes"

You can use temporary variable(s) if you want to write the code in more than one line.

```python
str = "cars-little-boats-little-planes"
x = str.split("little")
x = ['cars-', 'boats-', 'planes']
result = "big".join(x)
```

```
result = 'cars-big-boats-big-planes'
```

**PART B**

Given string `str` below with its assigned value, use only join and split to make the new phrase "strongbornlordbongo"

You can use temporary variable(s) if you want to write the code in more than one line.

```python
str = "stringbeanleadbingo"
x = str.split('i')
y = '0'.join(x)

result = str.split('ae')
x = ['str', 'ngbeanlead', 'ngo']
y = ['strongbeanleadbongo']
z = ['strong', 'n', 'dongo']
result = 'strongbornlوردbogo'
```
Consider the following function named total that has three parameters. One parameter is a list of integers named alist. The second and third parameters are named limit and stop, and are both an integer. The function total computes the sum of those integers in alist from left to right that are greater than the parameter limit, but if it comes across a number greater than or equal to the parameter stop then no more numbers are added.

The function total does not work correctly. It is shown below.

```python
1 def total(alist, limit, stop):
2     sum = 0
3     index = 0
4     while index < len(alist):
5         num = alist[index]
6         if num > limit:
7             sum += num
8         if num >= stop:
9             index = len(alist)
10            index += 1
11     return sum
```

For example, suppose lista is the list of numbers shown below. We show several calls to total showing the return value and the expected return value.

```python
lista = [5, 7, 64, 13, 56, 140, 80, 6]
```

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
<th>expected value</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>total(lista, 1, 50)</td>
<td>76</td>
<td>12</td>
<td>76 is from 5 + 7 + 64, 12 is from 5 + 7</td>
</tr>
<tr>
<td>total(lista, 50, 100)</td>
<td>260</td>
<td>120</td>
<td>260 is from 64 + 56 + 140, 120 is from 64 + 56</td>
</tr>
<tr>
<td>total(lista, 50, 200)</td>
<td>340</td>
<td>340</td>
<td>340 is from 64 + 56 + 140 + 80</td>
</tr>
<tr>
<td>total(lista, 150, 200)</td>
<td>0</td>
<td>0</td>
<td>no numbers greater than 150</td>
</tr>
</tbody>
</table>

**Question 1:** What are the key line number(s) in the program that make the loop stop early (not going through all the numbers). Give the line number(s) and explain how they work.

**Question 2:** Explain what the bug is in the program and how to fix it minimally (do not rewrite the whole function). Give the line number(s) of the code you want to change and what the new line(s) of code would be.
PROBLEM 4: (Selling Items (30 pts))

This problem is about data related to selling items. There are four functions to write in this part. Your functions should work for any valid data, not just the examples shown.

All four problems refer to data as a list of lists, with each inner list having the following four items: a string representing a first name, a string representing an item to sell, a float value representing the price for the item, and a list of one or more floats representing sales in which each sale is for one or more of this item.

For example, assume data is the lists of lists shown below. Note that in the first inner list the name of the seller is "claire", the item claire is selling is peach pie, the peach pie costs $5.00 to purchase, and the list shows claire has made three sales: the first sale for two peach pies (10.00), the second sale for one peach pie (5.00) and the last sale for 4 peach pies (20.00).

```
data = [['claire', 'peach pie', 5.0, [10.0, 5.0, 20.0]],
['bill', 'strawberries', 4.5, [4.5, 4.5, 4.5, 9.0]],
['chris', 'onion bread', 3.0, [3.0, 6.0, 18.0]],
['raj', 'flowers', 7.0, [7.0, 7.0]],
['yesenia', 'peaches', 9.0, [18.0, 9.0, 9.0]],
['xylo', 'eggs', 6.0, [6.0, 6.0, 6.0, 6.0, 6.0]],
['greta', 'carrots', 3.5, [3.5, 3.5, 3.5, 3.5, 10.5]]
```

Code you write should work for any value for data in the correct format.

Go to the next page to start Part A of this problem.
Part A (8 pts)
Write the function named `whatSold` that has two parameters. The first parameter is named `bigList`, which is a list of lists in the format described earlier, and the second parameter is named `person`, the first name of a person. We repeat the format of the lists of lists again here. Each inner list has four items: a string representing a first name, a string representing an item to sell, a float value representing the price for the item, and a list of one or more floats representing sales in which each sale is for one or more of this item.

This function returns the name of the item the person is selling. Assume name appears no more than once in `bigList`. If the player’s name is not in data then return ”nothing”.

For example, assume data is the list of lists shown below (this is the same list on the first page of this problem, not all items shown here). We give several examples of calls to this function.

```python
def whatSold(bigList, person):
    for lst in bigList:
        if lst[0] == person:
            return lst[1]
    return "nothing"
```

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>whatSold(data, &quot;raj&quot;)</td>
<td>&quot;flowers&quot;</td>
<td>raj sells flowers</td>
</tr>
<tr>
<td>whatSold(data, &quot;greta&quot;)</td>
<td>&quot;carrots&quot;</td>
<td>greta sells carrots</td>
</tr>
<tr>
<td>whatSold(data, &quot;marie&quot;)</td>
<td>&quot;nothing&quot;</td>
<td>marie not in list</td>
</tr>
</tbody>
</table>

\[
\text{data} = \left[ \begin{array}{c}
    \text{["claire", "peach pie", 5.0, [10.0, 5.0, 20.0]], }...
    \text{["raj", "flowers", 7.0, [7.0, 7.0]], }...
    \text{["greta", "carrots", 3.5, [3.5, 3.5, 3.5, 3.5, 3.5, 10.5]] ] }
\end{array}\right]
\]
Part B (8 pts)
Write the function named `numItemsSold` that has one parameter named `alist`, which is a list, in the format described earlier for the inner list. The parameter list has four items: a string representing a first name, a string representing an item to sell, a float value representing the price for the item, and a list of one or more floats representing sales in which each sale is for one or more of this item.

This function returns an integer representing the total number of items the person sold.

We give several examples of calls to this function. Note claire sold 7 peach pies (2 + 1 + 4).

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>numItemsSold([‘claire’, ‘peach pie’, 5.0, [10.0, 5.0, 20.0]])</td>
<td>7</td>
</tr>
<tr>
<td>numItemsSold([‘yesenia’, ‘peaches’, 9.0, [18.0, 9.0, 9.0]])</td>
<td>4</td>
</tr>
<tr>
<td>numItemsSold([‘raj’, ‘flowers’, 7.0, [7.0, 7.0]])</td>
<td>2</td>
</tr>
</tbody>
</table>

Complete the function below.

```python
def numItemsSold(alist):
    sum = 0
    price = alist[2]
    for num in alist[3]:
        sum = sum + num / price
    return int(sum)
```
Part C (8 pts)

Write the function named topSellers that has two parameters. The first parameter is named bigList, which is a list of lists in the format described on the first page of this problem, and the second parameter is a float named amount. We repeat the format of the lists of lists again here. Each inner list has four items: a string representing a first name, a string representing an item to sell, a float value representing the price for the item, and a list of one or more floats representing sales in which each sale is for one or more of this item.

This function returns a list of names of those sellers in bigList who have sold more than the monetary amount worth. Those names should appear in the same order they appear in the parameter bigList.

For example, assume data is the list of lists shown on the first page of this problem. We give several examples of calls to this function. The first example shows that claire, and yesenia are the only sellers that sold more than 30 dollars worth of their item.

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>topSellers(data, 30.00)</td>
<td>['claire', 'yesenia']</td>
</tr>
<tr>
<td>topSellers(data, 23.00)</td>
<td>['claire', 'chris', 'yesenia', 'xylo', 'greta']</td>
</tr>
<tr>
<td>topSellers(data, 40.00)</td>
<td>[]</td>
</tr>
</tbody>
</table>

In writing this function, you MUST call the function numItemsSold in a meaningful way. Complete the function below.

```python
def topSellers(bigList, amount):
    answer = []
    for lst in bigList:
        num = numItemsSold(lst)
        if num * lst[2] > amount:
            answer.append(lst[0])
    return answer
```
Part D (10 pts)

Let’s focus on converting one string of data, that represents one seller and sales for that seller’s item, into the format for the inner list for this problem.

Write the function named `processData` that has one string parameter named `line` in the following format. First there is a name (one word); followed by a semi-colon; followed by the item to sell made up of one or more words, with any two words separated by a blank; followed by a semi-colon; followed by a decimal number for the price of the item; followed by a semi-colon; followed by one or more decimal numbers, any two separated by a blank.

Your task is to return a list in the format described before. The list has four items: a string representing a first name, a string representing an item to sell, a float value representing the price for the item, and a list of one or more floats representing sales in which each sale is for one or more of this item. Here are two examples.

<table>
<thead>
<tr>
<th>call:</th>
<th>processData(&quot;claire;peach pie;5.00;10.00 5.00 20.00&quot;)</th>
<th>returns: ['claire', 'peach pie', 5.0, [10.0, 5.0, 20.0]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>call:</td>
<td>processData(&quot;yesenia;peaches;9.00;18.00 9.00 9.00&quot;)</td>
<td>returns: ['yesenia', 'peaches', 9.0, [18.0, 9.0, 9.0]]</td>
</tr>
</tbody>
</table>

Complete the function below.

```python
def processData(line):
    lst = line.split(';')
    lst[2] = float(lst[2])
    nlst = lst[3].split()
    numlist = [float(n) for n in nlst]
    lst[3] = numlist
    return lst
```
```python
def process_data(line):
    answer = []
    lst = line.split(' ', 3)
    answer.append([lst[0]])
    answer.append([lst[2]])
    answer.append([float(lst[3])])
    num_list = lst[3].split(' ,')
    f_list = []
    for item in num_list:
        f_list.append(float(item))
    answer.append(f_list)
    return answer
```
# Python Reference Sheet for Compsci 101, Exam 2, Spring 2022

YOU MUST TURN THE Reference Sheet pages in with your exam!!!!  Do not write any work to be graded on the Python Reference sheets!
Put your NETID at the top of any loose pages you turn in.

## Mathematical Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>$4 + 5 = 9$</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>$9 - 5 = 4$</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>$3*5 = 15$</td>
</tr>
</tbody>
</table>
| / and // | division     | $6/3 = 2.0$  
|        |              | $6/4 = 1.5$  
|        |              | $6//4 = 1$   |
| %      | mod/remainder | $5 \% 3 = 2$ |
| **     | exponentiation | $3**2 = 9$, $2**3 = 8$ |

## String Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>concatenation</td>
<td>&quot;ab&quot;+&quot;cd&quot;=&quot;abcd&quot;</td>
</tr>
<tr>
<td>*</td>
<td>repeat</td>
<td>&quot;xo&quot;*3 = &quot;xoxoxo&quot;</td>
</tr>
</tbody>
</table>

## Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>is equal to</td>
<td>$3 == 3$ is True</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
<td>$3 != 3$ is False</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
<td>$4 &gt;= 3$ is True</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>$4 &lt;= 3$ is False</td>
</tr>
<tr>
<td>&gt;</td>
<td>is strictly greater than</td>
<td>$4 &gt; 3$ is True</td>
</tr>
<tr>
<td>&lt;</td>
<td>is strictly less than</td>
<td>$3 &lt; 3$ is False</td>
</tr>
</tbody>
</table>

## Boolean Operators

- x=5

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>flips/negates the value of a bool</td>
<td>(not x == 5) is False</td>
</tr>
</tbody>
</table>
| and      | returns True only if both parts of it are True | (x > 3 and x < 7) is True  
|          |                                          | (x > 3 and x > 7) is False           |
| or       | returns True if at least one part of it is True | (x < 3 or x > 7) is False  
|          |                                          | (x < 3 or x < 7) is True             |

## Type Conversion Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int(x)</td>
<td>turn x into an integer value</td>
<td>int(&quot;123&quot;) == 123</td>
</tr>
<tr>
<td>float(x)</td>
<td>turn x into a float value</td>
<td>float(&quot;2.46&quot;) == 2.46</td>
</tr>
</tbody>
</table>

int can fail, e.g., int("abc") raises an error
float can fail, e.g., float("abc") raises an error
| **str(x)** | turn x into a string value | **str(432) == "432"** |
| **type(x)** | the type of x | **type(1) == int**
|  |  | **type(1.2) == float** |

### String index and splicing

\[ s = "colorful" \]

| **s[x]** | index a character | **s[0] == 'c'**
|  |  | **s[-3] == 'f'**
|  |  | **s[5] == 'f'** |

| **s[x:y]** | splice of string, substring from index x up to but not including index y | **s[2:5] == 'lor'**
|  |  | **s[5:] == 'ful'** |

### String Functions

\[ s = "colorful" \]

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Returns</strong></th>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.find(str)</strong></td>
<td>index of first occurrence</td>
<td><strong>s.find(&quot;o&quot;) == 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>s.find(&quot;e&quot;) == -1</strong></td>
</tr>
<tr>
<td><strong>.rfind(str)</strong></td>
<td>index of last occurrence</td>
<td><strong>s.rfind(&quot;o&quot;) == 3</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>s.rfind(&quot;e&quot;) == -1</strong></td>
</tr>
<tr>
<td><strong>.index(str)</strong></td>
<td>same as .find(str), error if str not in string</td>
<td><strong>s.index(&quot;o&quot;) == 3</strong></td>
</tr>
</tbody>
</table>
| **.count(str)** | number of occurrences | **s.count("o") == 2**
|  |  | **s.count("r") == 1**
|  |  | **s.count("e") == 0** |
| **.strip()** | copy with leading/trailing whitespace removed | "big ".strip() == "big" |
| **.split()** | list of "words" in s | "big bad dog".split() == ["big", "bad", "dog"] |
|  |  |  |
| **.split(,"\")** | list of "items" in s that are separated by a comma | "this,old,man".split(",")) == ["this", "old", "man"] |
|  |  |  |
| **".".join(lst)** | concatenate elements of lst, a list of strings, separated by ' ' or any string |:".join(["a","b","c"]) == "a:b:c" |
| **.startswith(str)** | boolean if starts with string | **s.startswith("color") == True** |
|  |  | **s.startswith("cool") == False** |
| **.endswith(str)** | boolean if ends with string | **s.endswith("ful") == True** |
|  |  | **s.endswith("color") == False** |
| **.upper()** | uppercase of s | **s.upper() == "COLORFUL"** |
### .lower()
Lowercase of s

"HELLO".lower() == "hello"

### .isupper()
Boolean is uppercase

'A'.isupper() == True
'a'.isupper() == False

### .islower()
Boolean is lowercase

'A'.islower() == False
'a'.islower() == True

### .isalpha()
Boolean is alphabetic character

'3'.isalpha() == False
'?).isalpha() == False
'z'.isalpha() == True

### .capitalize()
Capitalized s

s.capitalize() == "Colorful"

### .replace(str1, str2)
Replace all occurrences of str1 with str2

s.replace('o','y') == "cylorful"

### .replace(str1, str2, n)
Replace the first n occurrences of str1 with str2

s.replace('o','y',1) == "cylorful"

## Miscellaneous Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>help(x)</td>
<td>Documentation for module x</td>
<td></td>
</tr>
<tr>
<td>len(x)</td>
<td>Length of sequence x, e.g., String</td>
<td>len(&quot;duke&quot;) == 4</td>
</tr>
<tr>
<td>list(str)</td>
<td>A list of the characters from string str</td>
<td>list(&quot;cards&quot;) == ['c','a','r','d','s']</td>
</tr>
<tr>
<td>sorted(x)</td>
<td>Return list that is sorted version of sequence/iterable x, doesn't change x</td>
<td>sorted(&quot;cat&quot;) == ['a','c','t']</td>
</tr>
<tr>
<td>range(x)</td>
<td>A list of integers starting at 0 and going up to but not including x</td>
<td>range(5) == [0, 1, 2, 3, 4]</td>
</tr>
<tr>
<td>range(start, stop)</td>
<td>A list of integers starting at start and going up to but not including stop</td>
<td>range(3, 7) == [3, 4, 5, 6]</td>
</tr>
<tr>
<td>range(start, stop, inc)</td>
<td>A list of integers starting at start and going up to but not including stop with increment inc</td>
<td>range(3, 9, 2) == [3, 5, 7]</td>
</tr>
<tr>
<td>min(x, y, z)</td>
<td>Minimum value of all arguments</td>
<td>min(3, 1, 2) == 1</td>
</tr>
<tr>
<td>max(x, y, z)</td>
<td>Maximum value of all arguments</td>
<td>max(3, 1, 2) == 3</td>
</tr>
<tr>
<td>abs(x)</td>
<td>Absolute value of the int or float x</td>
<td>abs(-33) == 33</td>
</tr>
</tbody>
</table>

## List Index and Splicing

<table>
<thead>
<tr>
<th>lst = [3, 6, 8, 1, 7]</th>
<th>lst[0] == 3</th>
<th>lst[-1] == 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst[x]</td>
<td>Index an element</td>
<td></td>
</tr>
<tr>
<td>lst[x:y]</td>
<td>Splice of list, sublist from index x up to but not including index y</td>
<td>lst[1:3] == [6, 8]</td>
</tr>
</tbody>
</table>

Your Netid: ____________________
### List Functions

```python
lst = [3, 6, 8, 1, 7]
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum(lst)</code></td>
<td>returns sum of elements in list <code>lst</code></td>
<td><code>sum([1,2,4]) == 7</code></td>
</tr>
<tr>
<td><code>max(lst)</code></td>
<td>returns maximal element in <code>lst</code></td>
<td><code>max([5,3,1,7,2]) == 7</code></td>
</tr>
<tr>
<td><code>lst.append(...)</code></td>
<td>append an element to <code>lst</code>, changing <code>lst</code></td>
<td><code>[1,2,3].append(8) == [1,2,3,8]</code></td>
</tr>
<tr>
<td><code>lst.insert(pos,elt)</code></td>
<td>append <code>elt</code> to <code>lst</code> at position <code>pos</code>, changing <code>lst</code></td>
<td><code>[1,2,3].insert(1,8) == [1,8,2,3]</code></td>
</tr>
<tr>
<td><code>lst.extend(lst2)</code></td>
<td>append every element of <code>lst2</code> to <code>lst</code></td>
<td><code>[1,2,3].extend([8,9]) == [1,2,3,8,9]</code></td>
</tr>
<tr>
<td><code>lst.remove(elt)</code></td>
<td>remove first occurrence of <code>elt</code> from <code>lst</code></td>
<td><code>[1,2,3,2,3,2].remove(2) == [1,2,3,2,3,2]</code></td>
</tr>
</tbody>
</table>
| `lst.sort()` | sorts the elements of `lst` | `lst = [3,6,8,1,7]`  
`lst.sort()`  
`lst is now [1, 3, 6, 7, 8]` |
| `lst.index(elt)` | return index of `elt` in `lst`, error if `elt` not in `lst` | `[1,5,3,8].index(5) == 1` |
| `lst.count(elt)` | return number of occurrences of `elt` in `lst` | `[1,2,1,2,3].count(1) == 2` |
| `lst.pop()` | remove and return last element in `lst`, so has side-effect of altering list and returns value. | `lst = [3,6,8,1,7]`  
`x = lst.pop()`  
`x is 7, lst is [3,6,8,1]` |
| `lst.pop(index)` | remove and return element at position `index` in `lst`, so has side-effect of altering list and returns value. Default index is last value. | `lst = [3,6,8,1,7]`  
`x = lst.pop(1)`  
`x is 6, lst is [3,8,1,7]` |

### Math Functions (import `math`)  

```python
math.pi 3.1415926535897931
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>math.sqrt(num)</code></td>
<td>returns square root of <code>num</code> as float</td>
<td><code>math.sqrt(9) == 3.0</code></td>
</tr>
</tbody>
</table>

### File Functions

```python
open("filename")  
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open(&quot;filename&quot;)</code></td>
<td>opens a file, returns file object</td>
<td><code>f = open(&quot;foo.txt&quot;)</code></td>
</tr>
<tr>
<td><code>f.read()</code></td>
<td>returns the entire file as one string</td>
<td><code>s = f.read()</code></td>
</tr>
</tbody>
</table>

### Random Functions (import `random`)  

```python
random.choice(list_of_choices)
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>random.choice(list_of_choices)</code></td>
<td>returns a random element from <code>list_of_choices</code>. Gives an error if <code>list_of_choices</code> has length 0.</td>
<td></td>
</tr>
<tr>
<td><code>random.randint(start, end)</code></td>
<td>Returns a random integer between <code>start</code> and <code>end</code>. Unlike <code>range()</code> and list slicing, the largest value it can return is <code>end</code>, not <code>end-1</code>.</td>
<td></td>
</tr>
<tr>
<td><code>random.random()</code></td>
<td>Returns a random float between 0 and 1.</td>
<td></td>
</tr>
</tbody>
</table>

DO NOT WRITE any work to be graded on the Python Reference Sheet.