



Data structures pt I: Lists

Lesson 4 – 07/16/19

Slide by Sarah Middleton and edited by Sammy Klasfeld

(Please sign-in on the counter near the back door)

Today's TAs



Apexa



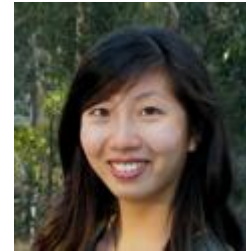
Sammy



Ben



Alex



Alex



Marianne



Kathy



Dr.
Shweta
Ramdas

Lab3 Review

- Please give your variables descriptive names
- Avoid dividing by integers!

How do we fix this code?

```
1  def gc(seq):  
2      x=0  
3      for i in range(len(seq)):  
4          if (seq[i]=='G') or (seq[i]=='C'):  
5              x = x + 1  
6      return round(gcCount/len(seq),2)
```

How do we fix this code?

```
1 def gc(seq):  
2     x=0  
3     for i in range(len(seq)):  
4         if (seq[i]=='G') or (seq[i]=='C'):  
5             x = x + 1  
6     return round(x/len(seq),2)  
7
```

ANSWER:

1. We give `x` a better variable name like “gc_count”
2. We convert x into a float either using the float function or by adding 1.0 instead of 1.

Lab3 Review

- Please give your variables descriptive names
- Avoid dividing by integers!
- You can increment loops with numbers other than 1

How do we fix this code?

(C) (1pt) Print the positive integers less than 500 that are multiples of 13.

```
1 i = 1
2 while i < 500:
3     if i%13 == 0:
4         print (i)
5     i = i + 1
```

How can we fix this code?

(C) (1pt) Print the positive integers less than 500 that are multiples of 13.

```
1 i = 1
2 while i < 500:
3     if i%13 == 0:
4         print (i)
5     i = i + 1
```

ANSWER: INCREMENT BY 13

```
1 i = 13
2 while i < 500:
3     print (i)
4     i = i + 13 # increments by 13
```


How can we fix this code?

(C) (1pt) Print the positive integers less than 500 that are multiples of 13.

```
for i in range(1,500):  
    if i%13 == 0:  
        print (i)
```

How can we fix this code?

(C) (1pt) Print the positive integers less than 500 that are multiples of 13.

```
for i in range(1,500):  
    if i%13 == 0:  
        print (i)
```

ANSWER: INCREMENT BY 13

```
for i in range(13,500,13):  
    print (i)
```

Lab3 Review

- Please give your variables descriptive names
- Avoid dividing by integers!
- You can increment loops with numbers other than 1
- Python Execution Order with respect to custom functions

Question 6A

Refer to the code below to answer the following questions:

```
: 1  def fancy_calc(a, b, c):
   2      x1 = basic_calc(a,b)
   3      x2 = basic_calc(b,c)
   4      x3 = basic_calc(c,a)
   5      z = x1 * x2 * x3
   6      return z
   7
   8  def basic_calc(x, y):
   9      result = x + y
  10      return result
  11
  12  x = 1
  13  y = 2
  14  z = 3
  15  result = fancy_calc(x, y, z)
```

(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.


Question 6A

Refer to the code below to answer the following questions:

```
: 1 def fancy_calc(a, b, c):
  2     x1 = basic_calc(a,b)
  3     x2 = basic_calc(b,c)
  4     x3 = basic_calc(c,a)
  5     z = x1 * x2 * x3
  6     return z
  7
  8 def basic_calc(x, y):
  9     result = x + y
10     return result
11
12 x = 1
13 y = 2
14 z = 3
15 result = fancy_calc(x, y, z)
```

Python executes code **top to bottom**, but it **skips definitions of custom functions**. It only runs functions when called in the main code.

So this line is executed **first** in the cell

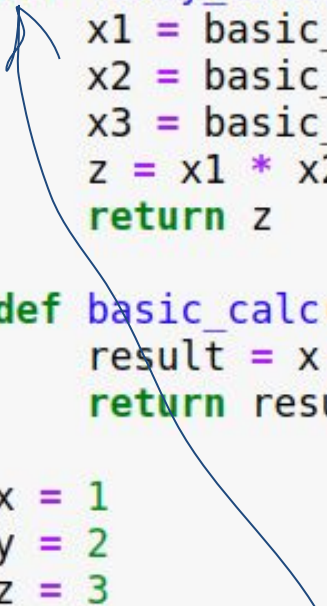


(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

```
: 1 def fancy_calc(a, b, c):  
2     x1 = basic_calc(a,b)  
3     x2 = basic_calc(b,c)  
4     x3 = basic_calc(c,a)  
5     z = x1 * x2 * x3  
6     return z  
7  
8 def basic_calc(x, y):  
9     result = x + y  
10    return result  
11  
12 x = 1  
13 y = 2  
14 z = 3  
15 result = fancy_calc(x, y, z)
```



(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

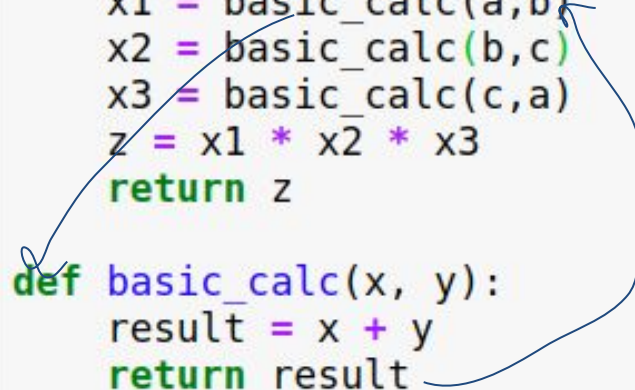
```
: 1  def fancy_calc(a, b, c):  
2      x1 = basic_calc(a,b)  
3      x2 = basic_calc(b,c)  
4      x3 = basic_calc(c,a)  
5      z = x1 * x2 * x3  
6      return z  
7  
8  def basic_calc(x, y):  
9      result = x + y  
10     return result  
11  
12  x = 1  
13  y = 2  
14  z = 3  
15  result = fancy_calc(x, y, z)
```

(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

```
: 1  def fancy_calc(a, b, c):  
2      x1 = basic_calc(a,b)  
3      x2 = basic_calc(b,c)  
4      x3 = basic_calc(c,a)  
5      z = x1 * x2 * x3  
6      return z  
7  
8  def basic_calc(x, y):  
9      result = x + y  
10     return result  
11  
12  x = 1  
13  y = 2  
14  z = 3  
15  result = fancy_calc(x, y, z)
```



(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

```
: 1 def fancy_calc(a, b, c):  
2     x1 = basic_calc(a,b)  
3     x2 = basic_calc(b,c)  
4     x3 = basic_calc(c,a)  
5     z = x1 * x2 * x3  
6     return z  
7  
8 def basic_calc(x, y):  
9     result = x + y  
10    return result  
11  
12 x = 1  
13 y = 2  
14 z = 3  
15 result = fancy_calc(x, y, z)
```

(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

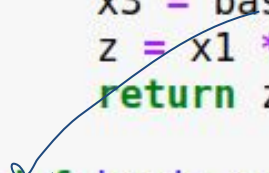
```
: 1 def fancy_calc(a, b, c):  
2     x1 = basic_calc(a,b)  
3     x2 = basic_calc(b,c)  
4     x3 = basic_calc(c,a)  
5     z = x1 * x2 * x3  
6     return z  
7  
8 def basic_calc(x, y):  
9     result = x + y  
10    return result  
11  
12 x = 1  
13 y = 2  
14 z = 3  
15 result = fancy_calc(x, y, z)
```

(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

```
: 1  def fancy_calc(a, b, c):  
2      x1 = basic_calc(a,b)  
3      x2 = basic_calc(b,c)  
4      x3 = basic_calc(c,a)  
5      z = x1 * x2 * x3  
6      return z  
7  
8  def basic_calc(x, y):  
9      result = x + y  
10     return result  
11  
12  x = 1  
13  y = 2  
14  z = 3  
15  result = fancy_calc(x, y, z)
```



(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

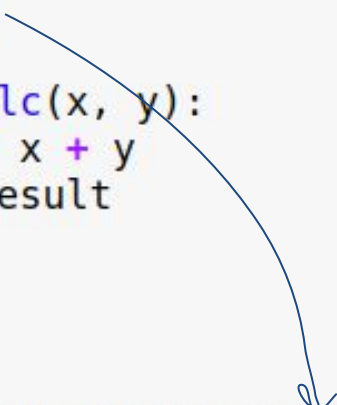
```
: 1 def fancy_calc(a, b, c):  
2     x1 = basic_calc(a,b)  
3     x2 = basic_calc(b,c)  
4     x3 = basic_calc(c,a)  
5     z = x1 * x2 * x3  
6     return z  
7  
8 def basic_calc(x, y):  
9     result = x + y  
10    return result  
11  
12 x = 1  
13 y = 2  
14 z = 3  
15 result = fancy_calc(x, y, z)
```

(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Question 6A

Refer to the code below to answer the following questions:

```
: 1  def fancy_calc(a, b, c):
   2      x1 = basic_calc(a,b)
   3      x2 = basic_calc(b,c)
   4      x3 = basic_calc(c,a)
   5      z = x1 * x2 * x3
   6      return z
   7
   8  def basic_calc(x, y):
   9      result = x + y
  10      return result
  11
  12  x = 1
  13  y = 2
  14  z = 3
  15  result = fancy_calc(x, y, z)
```



(A) List the line numbers of the code above in the order that they will be **executed**. If a line will be executed more than once, list it each time.

Lab3 Review

- Please give your variables descriptive names
- Avoid dividing by integers!
- You can increment loops with numbers other than 1
- Python Execution Order with respect to custom functions
- Use print statements to help you debug

Family simulation

Use a simulation to determine the average number of children in a family if **all families had children until they have a boy AND girl** (and then stopped). Assume equal probability of girls and boys and use a random generator to simulate each birth. **Simulate 10,000 families** and output the average number of children they had. Your answer should be close to 3.

Incorrect Answer

Where could we insert print statements to debug our code? What is wrong here?

```
1  # Incorrect Answer
2  numFamilies=10000
3  totalChildren=0
4  numGirls=0
5  numBoys=0
6  numChildren=0
7  for i in range(numFamilies):
8      gender = random.randint(0,1)
9      while numGirls < 1 or numBoys < 1:
10         #girls are 1 and boys are 0
11         gender = random.randint(0,1)
12         if gender:
13             numGirls = numGirls + 1
14         else:
15             numBoys = numBoys + 1
16             numChildren = numChildren + 1
17         totalChildren = totalChildren + numChildren
18 print(totalChildren/numFamilies)
```


Correct Answer

```
1 # Correct Answer
2 numFamilies=10000
3 totalChildren=0
4 for i in range(numFamilies):
5     numGirls=0
6     numBoys=0
7     numChildren=0
8     gender = random.randint(0,1)
9     while numGirls < 1 or numBoys < 1:
10         #girls are 1 and boys are 0
11         gender = random.randint(0,1)
12         if gender:
13             numGirls = numGirls + 1
14         else:
15             numBoys = numBoys + 1
16             numChildren = numChildren + 1
17     totalChildren = totalChildren + numChildren
18 print(float(totalChildren)/numFamilies)
```

We moved the counting variables inside the loop

2.9968

We don't divide two ints together

Today's schedule

1. Lists
2. File parsing with `.split()`
3. Nested Lists
4. List comprehension (Advanced)

* These slides are extra

1. Lists

What is a list?

- A **list** is a built-in data structure in Python (along with sets, tuples, and dictionaries)
- **What's a data structure?** It is basically a way of storing large amounts of data (numbers, strings, etc) in an organized manner, making storage and retrieval easier

What is a list?

Last time we used `range()` to create `for` loops. In fact, we can use any list to create `for` loops!

```
for i in [1, 20, 3, 19, 6]:  
    print(i)
```

Output:

```
1  
20  
3  
19  
6
```

Practice with lists

What will the following code print?

```
for i in ["cat", "dog", "mouse", "human"]:  
    print("I am a", i)
```

Practice with lists

What will the following code print?

```
for i in ["cat", "dog", "mouse", "human"]:  
    print("I am a", i)
```

Result:

```
I am a cat  
I am a dog  
I am a mouse  
I am a human
```

Practice with lists

What will the following code print?

```
myStuff = ["cat", 2, True, 99.5]
for i in myStuff:
    print(i)
```


Practice with lists

What will the following code print?

```
myStuff = ["cat", 2, True, 99.5]
for i in myStuff:
    print(i)
```

Result:

```
cat
2
True
99.5
```

How lists work

The lists we've seen so far look like this:

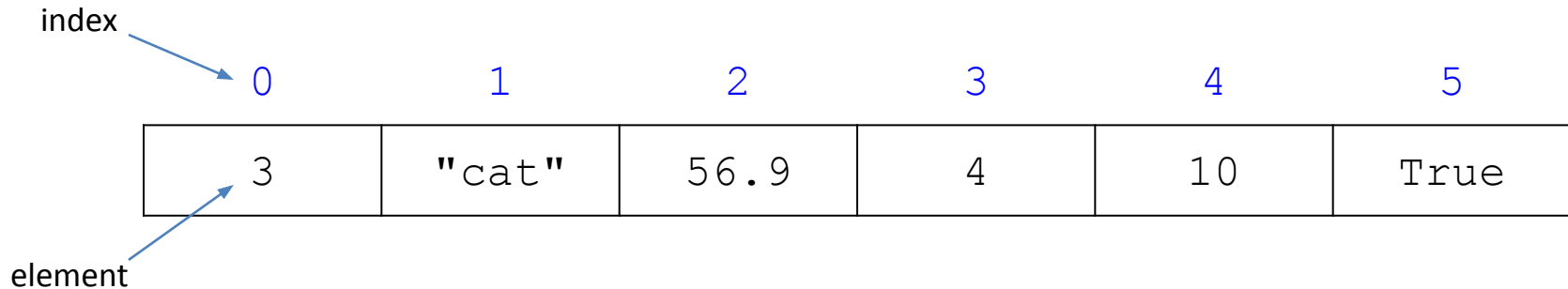
```
[3, "cat", 56.9, 4, 10, True]
```

How lists work

The lists we've seen so far look like this:

```
[3, "cat", 56.9, 4, 10, True]
```

However, it may be more helpful to think of it like this:



where each **element** is given an **index**, starting at 0.

Accessing elements in a list

We use only one variable name to refer to the whole list. For example:

```
myList = [3, "cat", 56.9, 4, 10, True]
```

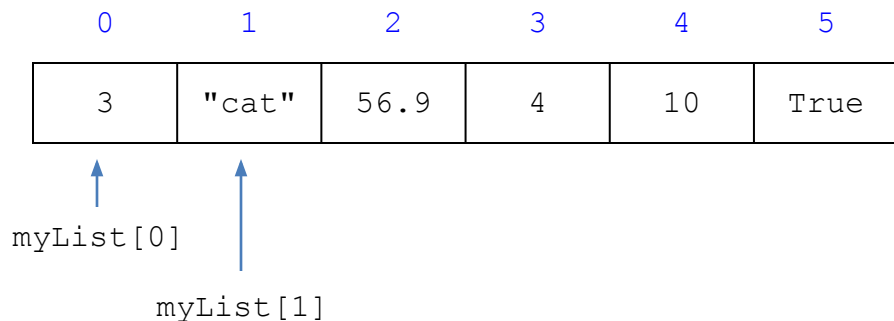
To access a specific element in the list, we use the following syntax: *listName[index]*

```
>>> myList[0]
```

```
3
```

```
>>> myList[1]
```

```
cat
```



Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
print(myList[1])
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
print(myList[1])
```

Result:

cat

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]  
print(myList[4])
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]  
print(myList[4])
```

Result:

10

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]  
print(myList[6])
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
print(myList[6])
```

Result:

```
Traceback (most recent call last):
  File "L5_test.py", line 2, in <module>
    print(myList[6])
IndexError: list index out of range
```

This is an "index out of bounds" error. You cannot access an index that does not exist!

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]  
print(myList[-1])
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
print(myList[-1])
```

Result:

True

Yep, this works! This comes in handy when you know you want the last element, but you don't know what the index of the last element is.

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
print(myList[-2])
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]  
print(myList[-2])
```

Result:

10

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

How would you get the third element?

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

How would you get the third element?

Answer:

```
myList[2]
```


Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
myList[0] = "dog"
print(myList)
```

Practice with list indexing

0	1	2	3	4	5
3	"cat"	56.9	4	10	True

What will this code print?

```
myList = [3, "cat", 56.9, 4, 10, True]
myList[0] = "dog"
print(myList)
```

Result:

```
['dog', 'cat', 56.9, 4, 10, True]
```

This is an easy way to
overwrite list elements

Creating a list

Create an empty list:

```
myList = []
```

Create a list with elements:

```
myList = [2, 7, 8]
```

Automatically create a list of numbers:

```
myList = range(5, 50, 10)
```

Adding to a list

After creating a list, you can add additional elements to the **end** using `.append()`.

Syntax:

```
list.append(newElement)
```

Example:

```
>>> myList = [2, 4, 6, 8]
>>> myList.append(10)
>>> print(myList)
[2, 4, 6, 8, 10]
```

Important to note:

Most of the functions we've seen so far do not modify variables directly -- they simply "return" a value. (e.g. `line.rstrip('\n')` does nothing to the original string -- it just returns a modified version. You have to say `line = line.rstrip('\n')` to actually change `line`.)

`.append()` is different. When you say `mylist.append()`, you are directly modifying `mylist`. We'll see several examples of this type of function today.

Removing from list

After creating a list, you can remove elements from it using `.pop()`.

Syntax:

```
list.pop(index)  
list.pop()
```

This in-place function removes the element at the specified index, or if no index is given, removes the last item.

It also returns the removed item.

Example:

```
>>> myList = [22, 44, 66, 88]  
>>> myList.pop(2)  
>>> print(myList)  
[22, 44, 88]
```

Elements that come after will be moved up one index, so that there are no empty spaces in the list.

Practice with lists

	0	1	2	3	4	5
myList	'a'	'b'	'c'	'd'	'e'	'f'

How do I add an 'g' to the end?

Practice with lists

	0	1	2	3	4	5
myList	'a'	'b'	'c'	'd'	'e'	'f'

How do I add an 'g' to the end?

Answer:

```
myList.append('g')
```

Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.pop(4)  
print(myList)
```


Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.pop(4)  
print(myList)
```

Answer:

```
[1, 2, 3, 4, 6]
```

Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.pop()  
print(myList)
```

Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.pop()  
print(myList)
```

Answer:

```
[1, 2, 3, 4, 5]
```

Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
item = myList.pop()  
print(item)
```

Practice with lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
item = myList.pop()  
print(item)
```

Answer:

6

Checking if something is in the list

To check if a particular element is in a list, you can just use `in`, as we've seen before:

```
myList = [22, 44, 66, 88]
if (66 in myList):
    print("found it!")
```

Code output:

```
found it!
```

Iterating through a list

Again, we've actually already done this. It's as simple as using a `for` loop:

```
myList = ["Joe", "Sally", "George", "Mike"]  
for name in myList:  
    print("Hello", name)
```

Code output:

```
Hello Joe  
Hello Sally  
Hello George  
Hello Mike
```

List slicing

Sometimes you may want to extract a certain subset of a list.

	0	1	2	3	4	5
myList	'a'	'b'	'c'	'd'	'f'	'g'

Syntax: `list[begin:end]` returns from index `begin` to `end-1`

```
>>> myList = ['a', 'b', 'c', 'd', 'f', 'g']
>>> myList[2:] #get from 2 to the end
['c', 'd', 'f', 'g']
>>> myList[:4] #get from the beginning to 3
['a', 'b', 'c', 'd']
>>> myList[2:4] #get from 2 to 3
['c', 'd']
```


Side note: indexing strings like lists

Strings are NOT lists. But we can index into strings like we do lists:

```
>>> seq = "ATGAC"
```

```
>>> seq[0]
```

```
'A'
```

```
>>> seq[-1]
```

```
'C'
```

```
>>> seq[1:4]
```

```
'TGA'
```

Side note: indexing strings like lists

Strings are immutable (cannot be changed), so none of these operations are allowed:

```
>>> seq = "ATGAC"
```

```
>>> seq[0] = "T"
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'str' object does not support item assignment
```

```
>>> seq.append("A")
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
AttributeError: 'str' object has no attribute 'append'
```

Strategies for modifying strings

```
>>> seq = "ATGAC"
```

```
>>> seq = name.replace("G", "T")
```

```
>>> print(seq)
```

```
ATTAC
```

```
>>> seq += "A"
```

```
>>> print(seq)
```

```
ATGACA
```

Strategies for modifying strings

- You can also turn the string to a list

```
>>> seq = "ATGAC"
```

```
>>> l_seq = list(seq)
```

```
>>> print(l_seq)
["A", "T", "G", "A", "C"]
```

```
>>> l_seq[0] = "T"
```

```
>>> l_seq.append("A")
```

```
>>> print("".join(l_seq)
TTGACA
```

Useful list functions

Lists come with several other helpful functions:

- `.sort()` - sorts **in place** (overwrites the list). Can sort both strings and numerical data.
- `.reverse()` - reverses order of items, **in place**
- `.index(element)` - returns index of the first occurrence of the specified element
- `.remove(element)` - Removes the first occurrence of the specified element. Elements that come after will shift down one index.
- `.insert(index, value)` - insert the value at the specified index. Elements that come after that index will shift up one index.
- `.count(element)` - returns the number of times the specified element occurs in the list

Functions that work on lists

There are also several built-in Python functions that work on lists:

- `len(list)` - returns the total number of elements in the list
- `max(list)` - returns the element in the list with the largest value
- `min(list)` - returns the element in the list with the smallest value
- `sum(list)` - returns the sum of the elements of the list

2. File parsing with `.split()`

The situation

- You have a file with multiple columns separated by tabs, commas, etc
- You want to extract certain columns of data to analyze.
- How can you do this in Python?

`.split()`

- This function splits a string into a list based on a delimiter.
- The delimiter can be anything you want, but usually it'll be a tab, space, or comma.
- This effectively lets you chop up a file into columns!

.split()

Purpose:

Splits a string every time it encounters the specified delimiter. If no delimiter is given, splits on whitespace (spaces, tabs, and newlines). The delimiter is not included in the output. If *maxsplit* is given, splits no more than *maxsplit* times. Returns a list.

Syntax:

```
result = string.split([delimiter[,maxsplit]])
```

Example:

```
>>> sentence = "Hello, how are you today?"
>>> sentence.split()
['Hello,', 'how', 'are', 'you', 'today?']
```

} Notice that the spaces
are removed!

More examples

```
>>> sentence = "Hello, how are you today?"
```

```
>>> sentence.split(',')  
['Hello', ' how are you today?']
```

Notice that now the comma is removed, but spaces are not!

```
>>> sentence.split(None, 2)  
['Hello,', 'how', 'are you today?']
```

maxsplit must always be the second parameter. So if we don't want to specify a delimiter, we can put `None` instead as a placeholder.

Why is `.split()` important?

This is perhaps the single most useful tool for parsing a text file (for what I do, anyway).

Let's take a look at a real-life example.

A more realistic example: parsing a data file

A data file organized in rows and columns (data "table") can be easily parsed using a combination of a `for` loop and `.split()`.

A more realistic example: parsing a data file

A data file organized in rows and columns (data "table") can be easily parsed using a combination of a `for` loop and `.split()`.

Example data file:

knownGene	Gene	InitCodon	DistCDS	Frame	InitContext	CDSLen	PeakSt	PeakWidth	#Reads	PeakScore	Codon	Product
uc007afd.1	Mrpl15	248	79	1	AATATGG 15	247	2	368	2.61	aug	internal-out-of-frame	
uc007afh.1	Lypla1	36	5	0	AACATGT 225	34	4	783	3.27	aug	n-term-trunc	
uc007afi.1	Tcea1	28	-24	0	GGCTTGT 325	27	3	446	1.43	nearcog	n-term-ext	
uc007afi.1	Tcea1	100	0	0	GCCATGG 301	99	3	3852	3.79	aug	canonical	
uc007afn.1	Atp6v1h	100	-13	-1	GCTATCC 10	99	3	728	0.77	nearcog	uorf	
uc007afn.1	Atp6v1h	149	3	0	AAGATGG 480	147	3	1407	1.36	aug	n-term-trunc	
uc007agb.1	Pcmt1	120	-97	-1	GCGCTGG 45	119	3	65	0.75	nearcog	uorf	
uc007agb.1	Pcmt1	265	-49	0	GCGCTGC 42	264	3	133	0.86	nearcog	uorf	
uc007agb.1	Pcmt1	412	0	0	GTCATGG 357	411	3	246	1.60	aug	canonical	
uc007agb.1	Pcmt1	737	108	1	ATCATGG 44	735	3	93	2.37	aug	internal-out-of-frame	
uc007agb.1	Pcmt1	890	159	1	AGTATGA 17	889	2	87	1.32	aug	internal-out-of-frame	
uc007agk.1	Rrs1	25	-19	0	GTAGTGG 10	25	1	927	1.52	nearcog	uorf	

A more realistic example: parsing a data file

Let's say I just want to extract the 6th column of each row (in this case, the initiation context for each start site).

Code:

```
inFile = "init_sites.txt"
input = open(inFile, 'r')
input.readline() #skip header

for line in input:
    line = line.rstrip('\n')
    data = line.split() #splits line on tabs
    print(data[5]) #6th column = index 5

input.close()
```

A more realistic example: parsing a data file

Let's say I just want to extract the 6th column of each row (in this case, the initiation context for each start site).

Code:

```
inFile = "init_sites.txt"
input = open(inFile, 'r')
input.readline() #skip header

for line in input:
    line = line.rstrip('\n')
    data = line.split() #splits line on whitespace
    print(data[5]) #6th column = index 5

input.close()
```

Output:

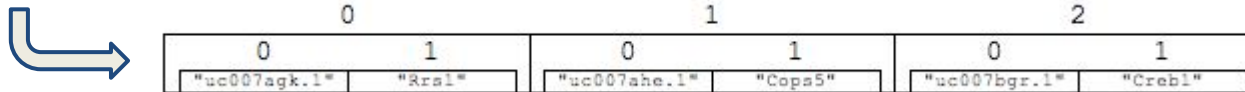
```
AATATGG
AACATGT
GGCTTGT
GCCATGG
GCTATCC
AAGATGG
GCGCTGG
GCGCTGC
GTCATGG
ATCATGG
AGTATGA
GTAGTGG
```


3. Nested Lists

Nested lists

A list can hold pretty much anything, including other lists:

```
>>> geneList = [["uc007agk.1", "Rrs1"], ["uc007ahe.1", "Cops5"],  
["uc007bgr.1", "Creb1"]]
```



```
>>> geneList[1]  
['uc007ahe.1', 'Cops5']  
>>> geneList[1][0]  
'uc007ahe.1'
```

You can access individual items in a list of lists using double indexing: `list[index][subindex]`

Practice with nested lists

What will this code print?

```
# Nested List
```

```
n_list = ["Happy", [2,0,1,5]]
```

```
print(n_list[1][2])
```

Practice with nested lists

What will this code print?

```
# Nested List  
n_list = ["Happy", [2,0,1,5]]  
  
print(n_list[1][2])
```

Answer:

1

Practice with nested lists

What will this code print?

```
# Nested List
```

```
n_list = ["Happy", [2,0,1,5]]
```

```
print(n_list[0][4])
```

Practice with nested lists

What will this code print?

```
# Nested List  
n_list = ["Happy", [2,0,1,5]]  
  
print(n_list[0][4])
```

Answer:

y

4. List Comprehensions (advanced)

List comprehensions (advanced)

A list comprehension is just a quick, concise way of performing operations on the elements of a list. Returns a new list with the modified elements.

Syntax:

```
newList = [expression for item in list if condition]
```

Example:

```
>>> myList = [1, 2, 3, 4, 5]
>>> newList = [i * 2 for i in myList]
>>> newList
[2, 4, 6, 8, 10]
>>> newList = [i * 2 for i in myList if i > 3]
>>> newList
[8, 10]
```


List comprehensions (advanced)

Almost any function can be used as the *expression* part:

```
>>> myList = ["Joe", "Sally", "George", "Mike"]
>>> [len(i) for i in myList]
[3, 5, 6, 4]
>>>
>>> [i.upper() for i in myList]
['JOE', 'SALLY', 'GEORGE', 'MIKE']
>>>
>>> [(i == "George") for i in myList]
[False, False, True, False]
>>>
>>> [print(i) for i in myList]
File "<stdin>", line 1
    [print(i) for i in myList]
      ^
```

SyntaxError: invalid syntax



Why doesn't this work?

The *expression* must return something that can be assigned to a list, which `print` (does not do).

Practice with a list comprehension (advanced)

What will this code print?

```
f_list = ["samples", "experiments", "runs"]  
print ([x + ".txt" for x in f_list])
```

Practice with a list comprehension (advanced)

What will this code print?

```
f_list = ["samples", "experiments", "runs"]  
print ([x + ".txt" for x in f_list])
```

Answer:

```
['samples.txt', 'experiments.txt', 'runs.txt']
```

Appendix

Examples of `.insert()` and `.remove()`

Inserting into a list: `.insert()`

Purpose:

Insert new element at specified index. All elements after will be pushed back one index.

Syntax:

```
list.insert(index, newElement)
```

Example:

```
>>> myList = [2, 4, 6, 8]
>>> myList.insert(1, "hi!")
>>> print(myList)
[2, 'hi!', 4, 6, 8]
```

Practice with adding to lists

	0	1	2	3	4	5
myList	'a'	'b'	'c'	'd'	'f'	'g'

How do I insert an 'e' between the 'd' and 'f'?

Practice with adding to lists

	0	1	2	3	4	5
myList	'a'	'b'	'c'	'd'	'f'	'g'

How do I insert an 'e' between the 'd' and 'f'?

Answer:

```
myList.insert(4, 'e')
```

Remove element from list: `.remove()`

Purpose:

Removes the first occurrence of the specified element. Elements that come after will be moved up one index.

Syntax:

```
list.remove(element)
```

Example:

```
>>> myList = [22, 44, 66, 88]
>>> myList.remove(44)
>>> print(myList)
[22, 66, 88]
```


Practice with removing from lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.remove(4)  
print(myList)
```

Practice with removing from lists

	0	1	2	3	4	5
myList	1	2	3	4	5	6

What will this code print?

```
myList.remove(4)
print(myList)
```

Answer:

```
[1, 2, 3, 5, 6]
```