Week 1, Day 3: Wednesday

Agenda:

- Review HW
- Lists
- Tuples, sets, dictionaries
- If time: algorithmic thinking with sorting lists
- If time: brief intro to recursive functions

- Big Picture Application #1: Astrochemistry!

Homework Review & Discussion Prep

Lists

- A list is a collection of objects
- Denoted by square brackets
- List properties:
  - Ordered
  - Can contain arbitrary objects
  - Accessed by index
  - Can be nested to arbitrary depth
  - Mutable
  - Dynamic

Lists are ordered

- The order of the objects in a list is a characteristic of the list and is remembered by the computer!

- Lists that have the same elements but in a different order are not the same list

    a = [‘cosmology’, ‘astrophysics’]

    b = [‘astrophysics’, ‘cosmology’]

    a == b returns False.

Lists can contain arbitrary objects

- Lists can contain any number of objects (incl. 0)
- Objects don't need to be unique
- a = [4, 3, 4, 4, 6]
- a = [4, 'boogity!', -42.0, 'blah!']

In [6]:

```
1  import math
2
3  def test_function():
4      print("I am a function!")
5
6  a = ['2', 'boogity', math, test_function]
7
8  a
```

Out[6]:

```
['2',
 'boogity',
 <module 'math' from '/usr/local/Cellar/python/3.7.2_1/Frameworks/Python.framework/Versions/
3.7/lib/python3.7/lib-dynload/math.cpython-37m-darwin.so'>,
 <function __main__.test_function()>]
```

Indexing Lists

- Individual elements in a list can be retrieved by using an index in square brackets.
- Lists in Python index **starting at 0!**

```
1  my_list = ['a', 'b', 'c', 'd', 'e']
2
3  print("Element at 0th index is", my_list[0])
4  print("Element at 1st index is", my_list[1])
5  print("Element at 2nd index is", my_list[2])
6  print("Element at 3rd index is", my_list[3])
7  print("Element at 4th index is", my_list[4])
```

```
Element at 0th index is a
Element at 1st index is b
Element at 2nd index is c
Element at 3rd index is d
Element at 4th index is e
```

Indexing Lists

- You can also index lists backwards using *negative indices*:

```
1  my_list = ['a', 'b', 'c', 'd', 'e']
2
3  print("Element at -1st index is", my_list[-1])
4  print("Element at -2nd index is", my_list[-2])
5  print("Element at -3rd index is", my_list[-3])
6  print("Element at -4th index is", my_list[-4])
7  print("Element at -5th index is", my_list[-5])
```

```
Element at -1st index is e
Element at -2nd index is d
Element at -3rd index is c
Element at -4th index is b
Element at -5th index is a
```

List Slicing

- You can get parts of a list using a loop, but there's a faster, cleaner way!

```
1  my_list = ['a', 'b', 'c', 'd', 'e']
2
3  for i in range(1, 4):
4      print("Element at index", i, "is", my_list[i])
```

```
Element at index 1 is b
Element at index 2 is c
Element at index 3 is d
```

```
1  my_list[1:4]
```

```
['b', 'c', 'd']
```

List Slicing

- my_list[a:b] gives the part of list my_list starting at index a and up through but not including index b
- Can use positive or negative indices

```
my_list = ['a', 'b', 'c', 'd', 'e']
```

```
In [19]:  1  my_list[1:4]
```

```
Out[19]: ['b', 'c', 'd']
```

```
In [23]:  1  my_list[-4:-1]
```

```
Out[23]: ['b', 'c', 'd']
```

```
In [24]:  1  my_list[-4:-1] == my_list[1:4]
```

```
Out[24]: True
```

List Slicing

- You can specify a stride in the slice, positive or negative: my_list[*a, b, stride*]

```
my_list = ['a', 'b', 'c', 'd', 'e']
```

```
In [25]:  1  my_list[1:4:2]
```

```
Out[25]: ['b', 'd']
```

```
In [29]:  1  my_list[3:0:-2]
```

```
Out[29]: ['d', 'b']
```

List Slicing

You can do all kinds of black magic…

```
1  my_list = ['a', 'b', 'c', 'd', 'e']
2
3  my_list[::-1]
```

```
['e', 'd', 'c', 'b', 'a']
```

And my_list[:] returns a copy of the list.

List operators

- *in*, *not in*

```
1  my_list = ['astrophysics', 'cosmology', 'genomics', 'artifical intelligence']
2
3  print('cosmology' in my_list)
4  print('economics' in my_list)
```

```
True
False
```

- *concatenation* (+) and *replication* (*)

```
1  my_list = ['astrophysics', 'genomics', 'artifical intelligence']
2  new_speaker = ['cosmology']
3
4  my_list += new_speaker
5  print(my_list)
6
7  my_list *= 2
8  print(my_list)
```

```
['astrophysics', 'genomics', 'artifical intelligence', 'cosmology']
['astrophysics', 'genomics', 'artifical intelligence', 'cosmology', 'astrophysics', 'genomic
s', 'artifical intelligence', 'cosmology']
```

List operators

- len(), min(), max() functions

```
1  particles_per_dim_in_sims = [256, 2660, 7000, 512, 32]
2  print("Length of list is", len(particles_per_dim_in_sims))
3  print("Max value in list is", max(particles_per_dim_in_sims))
4  print("Min value in list is", min(particles_per_dim_in_sims))
```

```
Length of list is 5
Max value in list is 7000
Min value in list is 32
```

Lists can be nested

- An element of a list can be anything (incl. a list!)

```
1  subjects = ['Physics', 'Math', ['English', 'Spanish', 'Mandarin'], ['Choir', 'Orchestra']]
2
3  print(subjects[0])
4  print(subjects[2])
```

```
Physics
['English', 'Spanish', 'Mandarin']
```

- Access items in sublists with a 2nd index:

```
1  subjects = ['Physics', 'Math', ['English', 'Spanish', 'Mandarin'], ['Choir', 'Orchestra']]
2
3  print(subjects[0])
4  print(subjects[2])
5  print(subjects[2][0])
```

```
Physics
['English', 'Spanish', 'Mandarin']
English
```

Lists can be nested

- You can nest a list as many levels deep as you like. All levels can be accessed using an extra index corresponding to that level.
- List slicing notation applies to sublists.
- However, len(my_list) only applies to the level at which it is called:

```
1  subjects =( ['Physics', 'Math', ['Mandarin', 'Spanish', 'English'],
2              [['Choir', 'Orchestra'], ['Painting', 'Drawing', 'Pottery']]] )
3
4  print('Level 0 of list:', len(subjects))
5  print('Level 1 of list at index 2:', len(subjects[2]))
6  print('Level 2 of list at indices 3, 0:', len(subjects[3][0]))
```

```
Level 0 of list: 4
Level 1 of list at index 2: 3
Level 2 of list at indices 3, 0: 2
```

Lists can be nested

- Similarly:

```
1  subjects =( ['Physics', 'Math', ['Mandarin', 'Spanish', 'English'],
2              [['Choir', 'Orchestra'], ['Painting', 'Drawing', 'Pottery']]] )
3
4  print('Physics' in subjects)
5  print('Choir' in subjects)
6  print(['Choir', 'Orchestra'] in subjects)
7  print(['Choir', 'Orchestra'] in subjects[3])
8
```

```
True
False
False
True
```

- max() and min() can only compare elements of the same type.

Lists are mutable

- The elements in a list can be changed

Lists are dynamic

- Lists can change their size, i.e., with *append*, *pop*, etc.

Activity: List Operations

Tuples

- Tuples are just like lists, except that:
  - They're denoted with parentheses: ()
  - They are immutable

```
1  t = ('first part of tuple', 'second part of tuple', 'third part')
2  print(t[1])
```

```
second part of tuple
```

```
1  t[1] = 'I want to change this value!'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-5bf5f9e3ee8b> in <module>
----> 1 t[1] = 'I want to change this value!'

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

Why use tuples instead of lists?

- Programs run a bit faster when manipulating tuples rather than lists, but this won't be noticeable for short tuples / lists

- Sometimes you don't want certain data to be modified. Putting it in a tuple will protect against accidental modification.

Tuple packing and unpacking

```
1  def three_musketeers():
2      return 'Athos', 'Porthos', 'Aramis'
3
4  n1, n2, n3 = 'Athos', 'Porthos', 'Aramis'
5  m1, m2, m3 = three_musketeers()
6
7  print(n1, n2, n3)
8  print(m1, m2, m3)
9
```

```
Athos Porthos Aramis
Athos Porthos Aramis
```

Number of variables on left hand side must match number of variables on right hand side!

Tuples

```
1  a,b = 3,4
2
3  a, b = b, a
4
5  print(a, b)
```

```
4 3
```

Returning tuples from functions

```
1  def meaning_of_life():
2      return "42", "???"
3
4  answer, question = meaning_of_life()
5
6  print("The answer is ", answer, " but the question was ", question)
```

```
The answer is  42  but the question was  ???
```

Activity 2: Tuples

Sets

- A set is a collection of unique, unordered objects. *Syntax:* enclosed in braces {}

- The set() function takes as input any iterable (such as a list) and returns a set containing all the unique elements of the iterable
  - In other words, set([1, 1, 2, 1, 3]) would return {1, 2, 3}.
  - set([“Hello”, “Hi”, “Hello”, “Bonjour”]) would return {“Bonjour”, “Hello”, “Hi”}

Dictionaries

- Collection of objects
- Like lists, dicts are mutable, dynamic, can be nested
- Unlike lists, order doesn't matter. Lists elements are accessed using indices; dictionary elements are accessed using keys

Activity 3: Sets

Activity 4: Dictionaries

Recursive functions:

```
1  def recursive_function(k):
2      print("entering recursive function with k =", k)
3      if(k>0):
4          result = k + recursive_function(k-1)
5      else:
6          result = 0
7      return result
8
9  print("\n\nRecursion Example Results")
10 recursive_function(6)
11
```

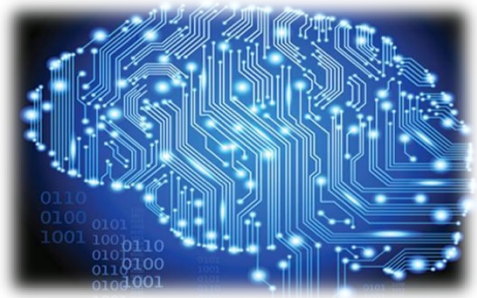
```
Recursion Example Results
entering recursive function with k = 6
entering recursive function with k = 5
entering recursive function with k = 4
entering recursive function with k = 3
entering recursive function with k = 2
entering recursive function with k = 1
entering recursive function with k = 0

21
```

Activity:

Write a recursive function that takes as input a positive integer N and calculates N!

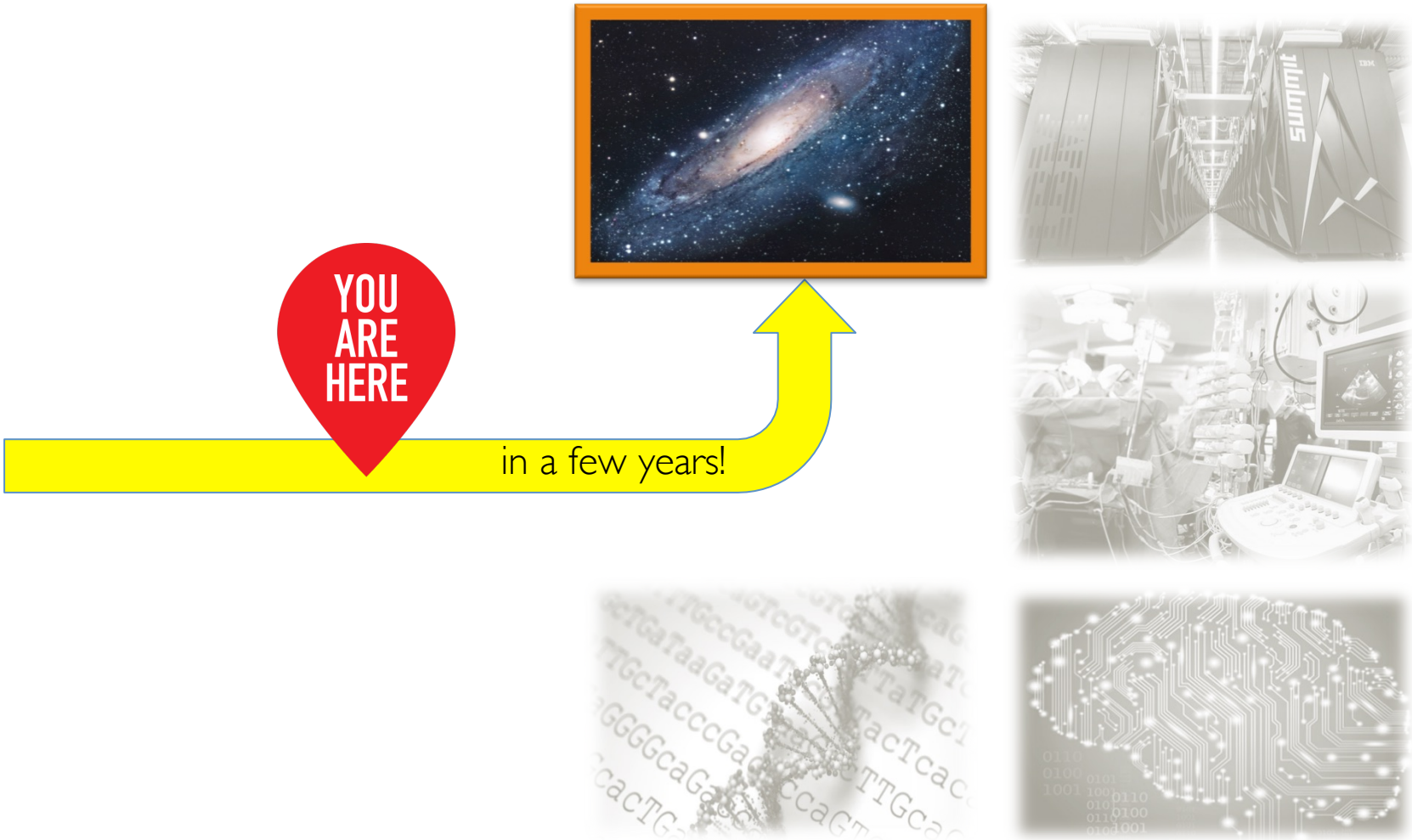
Goal #3: oh, the places you'll go!

Gain exposure to some of the big science questions you could one day work on if you continue down this path.



Goal #3: oh, the places you'll go!

Gain exposure to some of the big science questions you could one day work on if you continue down this path.



Reflection

- How did today go for you? What is one thing you could be doing to get more out of your learning? (e.g., attending office hours, emailing instructors with questions, etc.)

- What was the most interesting thing you learned today about astrochemistry?

- What is one follow-up question you'd like to ask?