Python: Tuple, Lists
Introduction

- Have seen data type such int, float, char,… scalar data.
- Compound data type:
  - Tuple
  - List
  - Array
Tuple

- an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- represented with parentheses

```
t e = ()  # empty tuple
t = (2,"mit",3)
t[0]    # evaluates to 2
(2,"mit",3) + (5,6)  # evaluates to (2,"mit",3,5,6)
t[1:2]  # slice tuple, evaluates to ("mit",)
t[1:3]  # slice tuple, evaluates to ("mit",3)
len(t)  # evaluates to 3
t[1] = 4 # gives error, can’t modify object
```

Tuple

- Conveniently used to **swap** variable values
  
  \[
  \begin{align*}
  x &= y \\
  y &= x \\
  \end{align*}
  \]

- Used to **return more than one value** from a function

  ```python
  def quotient_and_remainder(x, y):
      q = x // y  # integer division
      r = x % y
      return (q, r)
  
  (quot, rem) = quotient_and_remainder(4, 5)
  ```

Tuple

Manipulating tuples: iterating over a tuple pf tuples

def get_data(aTuple):
    nums = ()
    words = ()
    for t in aTuple:
        nums = nums + (t[0],)
        if t[1] not in words:
            words = words + (t[1],)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
Lists

- **ordered sequence** of information, accessible by index
- a list is denoted by **square brackets**, 
  - [ ]
- a list contains **elements**
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common)
- list elements can be changed so a list is **mutable**

Lists

\[
a_{\text{list}} = [] \\
L = [2, 'a', 4, [1, 2]]
\]

\[
\text{len}(L) \rightarrow \text{evaluates to 4} \\
L[0] \rightarrow \text{evaluates to 2} \\
L[2]+1 \rightarrow \text{evaluates to 5} \\
L[3] \rightarrow \text{evaluates to } [1, 2], \text{ another list!} \\
L[4] \rightarrow \text{gives an error} \\
i = 2 \\
L[i-1] \rightarrow \text{evaluates to } 'a' \text{ since } L[1] = 'a'
\]
Lists

- lists are **mutable**!
- assigning to an element at an index changes the value
  
  \[
  L = [2, 1, 3]
  \]
  
  \[
  L[1] = 5
  \]
- \(L\) is now \([2, 5, 3]\), note this is the **same object** \(L\)
Lists

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

```python
total = 0
for i in range(len(L)):
    total += L[i]
print total
```

```python
total = 0
for i in L:
    total += i
print total
```

- notice
  - list elements are indexed 0 to len(L) - 1
  - `range(n)` goes from 0 to n-1

Lists

- **add** elements to end of list with `L.append(element)`
- **mutates** the list!
  
  ```python
  L = [2, 1, 3]
  L.append(5)  # L is now [2, 1, 3, 5]
  ```

- what is the dot?
  - lists are Python objects, everything in Python is an object
  - objects have data
  - objects have methods and functions
  - access this information by `object_name.do_something()`
  - will learn more about these later

Lists

- to combine lists together use **concatenation**, + operator, to give you a new list
- **mutate** list with `L.extend(some_list)`

L1 = [2, 1, 3]
L2 = [4, 5, 6]
L3 = L1 + L2  →  L3 is [2, 1, 3, 4, 5, 6]
L1, L2 unchanged
L1.extend([0, 6])  →  mutated L1 to [2, 1, 3, 0, 6]

Lists

- delete element at a **specific index** with `del(L[index])`
- remove element at **end of list** with `L.pop()`, returns the removed element
- remove a **specific element** with `L.remove(element)`
  - looks for the element and removes it
  - if element occurs multiple times, removes first occurrence
  - if element not in list, gives an error

```
L = [2,1,3,6,3,7,0]  # do below in order
L.remove(2)         # mutates L = [1,3,6,3,7,0]
L.remove(3)         # mutates L = [1,6,3,7,0]
L.pop()             # mutates L = [1,3,7,0]
L.pop()             # returns 0 and mutates L = [1,3,7]
```

Lists to String

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

```python
s = "I<3 cs"
list(s) → s is a string
s.split('<') → returns ['I','<','3',' ','c','s']
L = ['a','b','c']
'' .join(L) → L is a list
'_' .join(L) → returns "abc"
'_' .join(L) → returns "a_b_c"
```

Lists: Other Operations

- `sort()` and `sorted()`
- `reverse()`

```python
L=[9, 6, 0, 3]

sorted(L) # returns sorted list, does not mutate L
L.sort() # mutates L=[0, 3, 6, 9]
L.reverse() # mutates L=[9, 6, 3, 0]
```

Lists

- lists are mutable
- behave differently than immutable types
- is an object in memory
- variable name points to object
- any variable pointing to that object is affected
- key phrase to keep in mind when working with lists is side effects
Lists: Aliasing

- `hot` is an alias for `warm` – changing one changes the other!

- `append()` has a side effect

```
1 a = 1
2 b = a
3 print(a)
4 print(b)
5
6 warm = ['red', 'yellow', 'orange']
7 hot = warm
8 hot.append('pink')
9 print(hot)
10 print(warm)
```
Lists: Cloning

- create a new list and *copy every element* using

```python
chill = cool[:]
```

```python
cool = ['blue', 'green', 'grey']
chill = cool[:]
chill.append('black')
print(chill)
print(cool)
```

**Frames**
- Global frame
  - `cool`
  - `chill`

**Objects**
- `list`
  - 0: "blue"
  - 1: "green"
  - 2: "grey"
  - 3: "black"
Lists: Sorting

- calling `sort()` **mutates** the list, returns nothing
- calling `sorted()` **does not mutate** list, must assign result to a variable

```python
1  warm = ['red', 'yellow', 'orange']
2  sortedwarm = warm.sort()
3  print(warm)
4  print(sortedwarm)
5  cool = ['grey', 'green', 'blue']
6  sortedcool = sorted(cool)
7  print(cool)
8  print(sortedcool)
```