Lecture 5: Tuples, Lists, Aliasing, Mutability, and Cloning

Tuples

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

Ex:: te = () empty tuple

t = (2, "nit", 3)

- t[0] => 2
- (2, "nit", 3) + (5,6) => (2, "nit", 3, 5, 6)
- t[1:2] -> slicing => ("nit",)
  - extra comma means tuple with one element
  - if comma wasn't there we just have a string and parentheses make no sense to be there
- t[1:3] -> slicing => ("nit", 3)
- len(t) => 3
- t[1] = 4 -> Error: can't modify object

- tuples can be used to swap values;

```
temp = x
x = y
y = temp
```

(x, y) = (y, x)

- tuples used to return more than one value

```
def quotient_and_remainder(x, y):
    q = x // y   # integer division
    r = x % y
    return (q, r)
```

(quot, rem) = quotient_and_remainder(4, 5)

Manipulating Tuples

aTuple: (, ,) — strings, integer

elements of aTuple are tuples themselves

- can iterate over tuples
  - def getdata(aTuple):

empty Tuple 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)
```

Example on python file

Lists

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

Indices and ordering (lists)

```
a_list = []
L = [2, 'a', 4, [1,2]]
len(L) => evaluates to 4
L[0] = 2
L[2]+1 = 4+1 = 5
L[3] = [1,2]
L[4] = error
```

```
i=2
L[i-1] = L[2-1] = L[1]
=> 'a'
```

Changing Elements

- lists are mutable
- assigning to an element at an index changes the value
  - `L = [2, 1, 3]`
    `L[1] = 5`
    we get `L = [2, 5, 3]`, notice this is the same object L

Iterating Over a List

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

ex:

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

```
total = 0
for i in L:
    total += 1
print(total)
```

like strings, can iterate over lists elements directly

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

Operations On Lists - Add

- add elements to the end of list with `L.append(element)`
- mutates the list

```
L = [2,1,3]
L.append(5)  --> [2,1,3,5]
```

object_name.do_something

- to combine list together use concatenation, + operator
- mutate list with L.extend(some_list)

```
L1 = [2,1,3]
L2 = [4,5,6]
L3 = L1 + L2  → [2,1,3,4,5,6]

L1.extend([0,6]) → mutated to: [2,1,3,0,6]
```

Operations on Lists – Remove

- delete element at a specific index by; del(L[index])
- remove element at the end of list by; L.pop(), returns the removed element
- remove a specific element with; L.remove(element)
  - this looks for element and removes it
  - if element occurs multiple times, removes first occurence
  - if element is not listed, gives an error.

Convert Lists to Strings And Back

- convert string to list with list(s), returns a list with every character from s as 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 character into a string, can give a character in quotes to add char between every element

Examples →

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

~~Mutation~~ List in Memory

- 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