Python Language

Enes Kemal Ergin

Questions of the Week

- How to handle list of objects?
- Which features does lists have?
- What is reusable code?
- How to use functions?
- How cool is List Comprehension?

Lists

- Lists contain zero or more objects and are used to keep track of collections of data.
- Lists can be modified, unlike strings…
- Creating a list:

```
list_objects = [1,2,3,4,5,6,7,8,9,0]
>>> list_objects
[1,2,3,4,5,6,7,8,9,0]
```

- This is how you store objects inside of the lists.
- [] is expressed version of empty list.

Lists(Cont'd)

- We can choose specific elements inside the lists using indexes of the list

whales = [5, 4, 7, 3, 2, 3, 2, 6, 4, 2, 1, 7, 1, 3]

whales[0]

5

whales[12]

1

- Indexation starts from 0 to n-1, when n is the length of the list
- How to get the length?
  - len(name_of_the_list)

whales[-1]

3

- Indexation with negative integers means starting from at the end, but they from end they starts with -1, to n

Lists(Cont'd)

- Lists can have different kinds of objects in them

  krypton = ['Krypton', 'Kr', -157.2, -153.4]

  krypton[1]

  'Kr'

  krypton[2]

  -157.2

- We can change the lists, they are mutable.

  nobles = ['helium', 'none', 'argon', 'krypton', 'xenon', 'radon']

  nobles[1] = 'neon'

  nobles

  ['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']

- Let's code them and see clearly…

List's Operations

- **len(L):** Returns the number of items in list L
- **max(L):** Returns the maximum value in list L
- **min(L):** Returns the minimum value in list L
- **sum(L):** Returns the sum of the values in list L
- **sorted(L):** Returns a copy of list L where the items are in order from smallest to largest (This does not mutate L.)

```
half_lives = [887.7, 24100.0, 6563.0, 14, 373300.0]
len(half_lives)
5
max(half_lives)
373300.0
min(half_lives)
14
sum(half_lives)
404864.7
sorted(half_lives)
[14, 887.7, 6563.0, 24100.0, 373300.0]
```

List's Operations

- + operators work here as well as a concatenation operator…

```
original = ['H', 'He', 'Li']
final = original + ['Be']
final
['H', 'He', 'Li', 'Be']
```

- Let's See how operations work…

List Slicing

- This is the same principle as what we learned in strings.

```
celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
useful_markers = celegans_phenotypes[0:4]
useful_markers
['Emb', 'Him', 'Unc', 'Lon']
```

Slicing has a lot of usage in dynamic programming, let's see some examples…

List Methods

- **L.append(v):** Appends value v to list L
- **L.clear():** Removes all items from list L
- **L.count(v):** Returns the number of occurrences of v in list L
- **L.extend(v):** Appends the items in v to L
- **L.index(v):** Returns the index of the first occurrence of v in L—an error is raised if v doesn't occur in L.
- **L.index(v, beg):** Returns the index of the first occurrence of v at or after index beg in L—an error is raised if v doesn't occur in that part of L.
- **L.index(v, beg, end):** Returns the index of the first occurrence of v between indices beg (inclusive) and end (exclusive) in L; an error is raised if v doesn't occur in that part of L.

List Methods(Cont'd)

- **L.insert(i, v):** Inserts value v at index i in list L, shifting subsequent items to make room
- **L.pop():** Removes and returns the last item of L (which must be nonempty)
- **L.remove(v):** Removes the first occurrence of value v from list L
- **L.reverse():** Reverses the order of the values in list L
- **L.sort():** Sorts the values in list L in ascending order (for strings with the same letter case, it sorts in alphabetical order)
- **L.sort(reverse=True):** Sorts the values in list L in descending order (for strings with the same letter case, it sorts in reverse alphabeticalorder)
- Hands on Time Again…

Reusable Code: Function

- A **routine** is a named group of instructions performing some task.
- A routine can be **invoked** (*called*) as many times as needed in a given program
- A **function** is Python's version of a program routine.
- We use functions to reuse the code again and again…

```
def function_name(parameters):
        # All bunch of stuff
        return #something or nothing
```

- First line of the function called the the Function Header
- Indented part called function body…

Function Basics

- Value-Returning Functions:
  - These types of functions have their return value at the very end of the function body.

```
def avgerage(a, b, c): # It takes 3 number
        return (a+b+c) / 3.0 # returns the average of those 3 numbers


avg(3,4,5) # Function call
4.0
```

- Without function call you won't be able to use your function.

Function Basics(Cont'd)

- Non-Value returning Functions:
  - It basically does not have a return at the end, but it has some side effects for the overall code…
  - A side effect is an action other than returning a function value, such as displaying output on the screen.

```
def displayWelcome():
        print('This program will convert between….')
        print('Enter (F)…..')
        print('Enter (C)…..')
displayWelcome()
 ## Those messages…
```

To understand more of Functions, Let's practice on the interpreter.

List Comprehensions

- List Comprehension is very elegant way of using for loop and list idea together. It shortcut most of the syntax for you.
- **List comprehensions** in Python can be used to generate more varied sequences.

```
[x ** 2 for x in [1, 2, 3]]          ->  [1, 4, 9]
[x ** 2 for x in range(5)]           ->  [0, 1, 4, 9, 16]
nums = [-1, 1, -2, 2, -3, 3, -4, 4]
[x for x in nums if x >= 0]          ->  [1, 2, 3, 4]
[ord(char) for ch in “Hello”]        -> [72, 101, 108, 108, 111]
```

To Do List

- Solve Exercise #3 ~ lists
- Exercise #3 ~ Functions
- Exercise #3 ~ List Iteration and Comprehension
- Programming Concepts #3
- Please Remember to study Programming Concepts because more and detailed information always there…