# Google Guide

October 6, 2018

## 1 Mark Abramov's super awesome Google interview guide

### 1.1 Made by your 1 tru

## 2 Table of Contents

### 2.1 Data Structures

#### 2.1.1 Lists

```
In [1]: greeting = list('HiMark') #string to list
        greeting

Out[1]: ['H', 'i', 'M', 'a', 'r', 'k']

In [2]: greeting.append('!') #adds 1 element to the end

In [3]: greeting.extend(['<', '3']) #extend appends all the elements of the given list

In [4]: greeting

Out[4]: ['H', 'i', 'M', 'a', 'r', 'k', '!', '<', '3']

In [5]: greet_string = ''.join(greeting) #joins with '' between every element

In [6]: greeting * 2 # the list twice
```

```
Out[6]: ['H',
         'i',
         'M',
         'a',
         'r',
         'k',
         '!',
         '<',
         '3',
         'H',
         'i',
         'M',
         'a',
         'r',
         'k',
         '!',
         '<',
         '3']
```

```
In [7]: greet_string * 2 #the string twice
```

```
Out[7]: 'HiMark!<3HiMark!<3'
```

Converting to Ascii: - ord(c) converts a character to its ascii value - doing this in a list comprehenstion (thats what the [ for in ] thing is called) does it for every char in the string

```
In [8]: greet_ascii = [ord(c) for c in greet_string]
        greet_ascii
```

```
Out[8]: [72, 105, 77, 97, 114, 107, 33, 60, 51]
```

```
In [9]: ''.join(chr(i) for i in greet_ascii) # you can do a little list comprehension inside a
```

```
Out[9]: 'HiMark!<3'
```

```
In [10]: sorted_ascii = sorted(greet_ascii)
         sorted_ascii
```

```
Out[10]: [33, 51, 60, 72, 77, 97, 105, 107, 114]
```

- you can use the optional key argument and pass it a lambda function
- `lambda VARIABLE_NAME_YOU_DEFINE: whatever you want to do to the VAR`

```
In [11]: greeting.sort(key = lambda char: char.upper())
         greeting = greeting[::-1] # REVERSE the list
         greeting
```

```
Out[11]: ['r', 'M', 'k', 'i', 'H', 'a', '<', '3', '!']
```

| Operation | Big O |
|---|---|
| append | O(1) |
| Pop Last | O(1) |
| Insert | O(n) |
| Get/Set | O(1) |
| Length | O(1) |
| Sort | O(n log n) |

**List Complexities**

---

### 2.1.2 Dictionaries

- `len(d)` returns the number of key,value pairs
- `del d[k]` deletes the key k and its value
- `d.pop(k)` deletes key and value and returns the value
  - can put a default value for if k isnt in the dict: `d.pop(k, 'nope')`
  - otherwise there will be a *KeyError*
- `if k in d` if there is a key k in dictionary d
- `d1.update(d2)` merges the dictionaries
- `L = list(D)` converts dictionary into list of 2-item tuples
- `D = dict(zip(L1, L2))` zips and converts 2 lists into a dictionary

```
In [12]: rappers = {} # creating empty dictionary
```

```
In [13]: rappers['Kanye'] = 1
         rappers['J Cole'] = 2
         rappers['Nicki Minaj'] = 15
         rappers
```

```
Out[13]: {'Kanye': 1, 'J Cole': 2, 'Nicki Minaj': 15}
```

```
In [14]: len(rappers)
```

```
Out[14]: 3
```

```
In [15]: for ranking in rappers.values(): #also: rappers.keys()
             print(ranking)
```

```
1
2
15
```

```
In [16]: for rapper, ranking in rappers.items():
             print('the number ', ranking, ' rapper is ', rapper)
```

```
the number  1  rapper is  Kanye
the number  2  rapper is  J Cole
the number  15  rapper is  Nicki Minaj
```

| Operation | Big O |
|---|---|
| Delete | O(1) |
| Insert | O(n) |
| Get/Set | O(1) |
| Length | O(1) |
| Sort | O(n log n) |

**Dictionary Complexities**

---

### 2.1.3 Sets

- like a 1 dimensional dictionary
- unordered, no duplicates

```
In [17]: your_friends = set()
         your_friends.add("Maxwell")
         your_friends.add("Stelios")
         your_friends.add("Jazmyn")
         your_friends
```

```
Out[17]: {'Jazmyn', 'Maxwell', 'Stelios'}
```

```
In [18]: your_friends.add("Maxwell")
         your_friends
```

```
Out[18]: {'Jazmyn', 'Maxwell', 'Stelios'}
```

```
- oh well, guess you cant have 2 Maxwells
```

| Operation | Big O |
|---|---|
| Delete | O(1) |
| x in s | O(1) |
| Get/Set | O(1) |
| Length | O(1) |

**Set Complexities**

### 2.1.4 Binary Trees

- not native to python but heres a nice implementation (and a little intro to how classes work)

```
In [19]: class Node:

             def __init__(self, data):
```

```
        # init is the constructor
        self.left = None
        self.right = None
        self.data = data

    def insert(self, data): # every method takes self
        # Compare the new value with the parent node
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data

# Print the tree - not a great representation but ya know
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data),
        if self.right:
            self.right.PrintTree()

# Use the insert method to add nodes
root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)

root.PrintTree()# notice you dont have to write self
```

```
3
6
12
14
```

| Operation | Big O |
|---|---|
| Search | O(n) |
| Insertion | O(n) |