Cheatography

Python Programming Competition Cheat Sheet

by 0llieC (0llieC) via cheatography.com/38321/cs/18889/

File I/O	
Open a File	`f = open("test.txt")`
Open a File as Read-Only	`f = open("test.txt", 'r')`
Open a File to Write	`f = open("test.txt", 'w')`
Open a File to Append	`f = open("test.txt", 'a')`
Close a File	`f.close()`
Write to a File	`f.write("write this line\n")`
Write a list of lines	`f.writelines(lines)`
Read a File	`f.read()` reads to the EOF `f.read(n)` reads n characters
Current File Position	`f.tell()`
Change the File Position	`f.seek(n)` where n is the new position.
Read a single line	`f.readline()`
Put all file lines into a list	`f.readlines()`

By **0llieC** (0llieC)
cheatography.com/0lliec/

Not published yet.
Last updated 20th February, 2019.

Tree Algorithms

Tree

```
class Node:

    def __init__(self, data):

        self.left = None
        self.right = None
        self.data = data
# Insert Node
    def insert(self, data):

        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
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()
```

Tree Algorithms (cont)

In-Order

```
# Inorder traversal
# Left -> Root -> Right
    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.data)
            res = res + self.inorderTraversal(root.right)
        return res
```

Pre-Order

```
# Preorder traversal
# Root -> Left ->Right
    def PreorderTraversal(self, root):
        res = []
        if root:
            res.append(root.data)
            res = res + self.PreorderTraversal(root.left)
            res = res + self.PreorderTraversal(root.right)
        return res
```

Tree Algorithms (cont)

Post-Order

```
# Postorder traversal
# Left ->Right -> Root
    def PostorderTraversal(self, root):
        res = []
        if root:
            res = self.PostorderTraversal(root.left)
            res = res + self.PostorderTraversal(root.right)
            res.append(root.data)
        return res
```

By **0llieC** (0llieC)
cheatography.com/0lliec/

Not published yet.
Last updated 20th February, 2019.

Tree Algorithms (cont)

Find a Value

```
def findval(self, lkpval):
        if lkpval < self.data:
            if self.left is None:
                return str(lkpval)+" Not Found"
            return self.left.findval(lkpval)
        elif lkpval > self.data:
            if self.right is None:
                return str(lkpval)+" Not Found"
            return self.right.findval(lkpval)
        else:
            print(str(self.data) + ' is found')
```

Queues and Stack

Queue

```
class Queue:


  def __init__(self):
      self.queue = list()


  def addtoq(self,dataval):
# Insert method to add element
      if dataval not in self.queue:
          self.queue.insert(0,dataval)
          return True
      return False
# Pop method to remove element
  def removefromq(self):
      if len(self.queue)>0:
          return self.queue.pop()
      return ("No elements in Queue!")
```

Queues and Stack (cont)

Stack

```
class Stack:

    def __init__(self):
        self.stack = []

    def add(self, dataval):
# Use list append method to add element
        if dataval not in self.stack:
            self.stack.append(dataval)
            return True
        else:
            return False

# Use list pop method to remove element
    def remove(self):
        if len(self.stack) <= 0:
            return ("No element in the Stack")
        else:
            return self.stack.pop()
```

Regex in Python

import module	`import re`
Return a list of matches	`x = re.findall("ai", str)`
Split a string	`x = re.split("\s", str)`
Replace	`x = re.sub("\s", "9", str)`
Finds occurrences. Returns a Match object	`x = re.search("\s", str)`
	`x.span()` returns a tuple of start and end of each match
	`x.string()` returns the string searched
	`x.group()` returns the part of the string matching

Regular Expressions

.	Any character except newline
a|b	a or b
a*	0 or more a's
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
\d	One digit
\D	One non-digit

Regular Expressions (cont)	
\s	One whitespace
\S	One non-whitespace
^	Start of string
$	End of string
(...)	Capturing group
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more
{,5}	Up to 5
\n	Newline
\t	Tab

Graph Algorithms

Breadth First Traversal

```
import collections
class graph:
    def __init__(self,gdict=None):
        if gdict is None:
            gdict = {}
        self.gdict = gdict

def bfs(graph, startnode):
# Track the visited and unvisited nodes using queue
        seen, queue = set([startnode]), collections.deque([startnode])
        while queue:
            vertex = queue.popleft()
            marked(vertex)
            for node in graph[vertex]:
                if node not in seen:
                    seen.add(node)
                    queue.append(node)

def marked(n):
    print(n)

# The graph dictionary
gdict = { "a" : set(["b","c"]),
                "b" : set(["a", "d"]),
                "c" : set(["a", "d"]),
                "d" : set(["e"]),
                "e" : set(["a"])
                }

bfs(gdict, "a")
```

Graph Algorithms (cont)

Depth First Traversal

```
class graph:


    def __init__(self,gdict=None):
        if gdict is None:
            gdict = {}
        self.gdict = gdict
# Check for the visisted and unvisited nodes
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

gdict = { "a" : set(["b","c"]),
                "b" : set(["a", "d"]),
                "c" : set(["a", "d"]),
                "d" : set(["e"]),
                "e" : set(["a"])
                }


dfs(gdict, 'a')
```

Searching Algorithms

Linear Search

```
def linear_search(values, search_for):
    search_at = 0
    search_res = False

# Match the value with each data element
    while search_at < len(values) and search_res is False:
        if values[search_at] == search_for:
            search_res = True
        else:
            search_at = search_at + 1

    return search_res

l = [64, 34, 25, 12, 22, 11, 90]
print(linear_search(l, 12))
print(linear_search(l, 91))
```

By **0llieC** (0llieC)
cheatography.com/0lliec/

Not published yet.
Last updated 20th February, 2019.

Searching Algorithms (cont)

Binary Search

```
def bsearch(list, val):

    list_size = len(list) - 1

    idx0 = 0
    idxn = list_size
# Find the middle most value

    while idx0 <= idxn:
        midval = (idx0 + idxn)// 2

        if list[midval] == val:
            return midval
# Compare the value the middle most value
        if val > list[midval]:
            idx0 = midval + 1
        else:
            idxn = midval - 1

    if idx0 > idxn:
        return None
# Initialize the sorted list
list = [2,7,19,34,53,72]

# Print the search result
print(bsearch(list,72))
print(bsearch(list,11))
```

Useful Functions

Counter

```
from collections import Counter
arr = [1, 3, 4, 1, 2, 1, 1, 3, 4, 3, 5, 1, 2, 5, 3, 4, 5]
counter = Counter(arr)
top_three = counter.most_common(3)
print(top_three)
```

Output

```
[(1, 5), (3, 4), (4, 3)]
```

Top/Bottom nth

```
import heapq
grades = [110, 25, 38, 49, 20, 95, 33, 87, 80, 90]
print(heapq.nlargest(3, grades))
print(heapq.nsmallest(4, grades))
```

Output

```
[110, 95, 90]
[20, 25, 33, 38]
```

Map Function

```
# Python code to apply a function on a list
income = [10, 30, 75]
def double_money(dollars):
  return dollars * 2

new_income = list(map(double_money, income))
print(new_income)
```

Useful Functions (cont)

Output

```
[20, 60, 150]
```

Sorting Algorithms

Bubble Sort

```
def bubblesort(list):

# Swap the elements to arrange in order
    for iter_num in range(len(list)-1,0,-1):
        for idx in range(iter_num):
            if list[idx]>list[idx+1]:
                temp = list[idx]
                list[idx] = list[idx+1]
                list[idx+1] = temp


list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

Sorting Algorithms (cont)

Merge Sort

```
def merge_sort(unsorted_list):
    if len(unsorted_list) <= 1:
        return unsorted_list
# Find the middle point and devide it
    middle = len(unsorted_list) // 2
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    left_list = merge_sort(left_list)
    right_list = merge_sort(right_list)
    return list(merge(left_list, right_list))

# Merge the sorted halves

def merge(left_half,right_half):

    res = []
    while len(left_half) != 0 and len(right_half) != 0:
        if left_half[0] < right_half[0]:
            res.append(left_half[0])
            left_half.remove(left_half[0])
        else:
            res.append(right_half[0])
            right_half.remove(right_half[0])
    if len(left_half) == 0:
        res = res + right_half
    else:
        res = res + left_half
    return res

unsorted_list = [64, 34, 25, 12, 22, 11, 90]

print(merge_sort(unsorted_list))
```