Cheatography

nus cs1010s python Cheat Sheet

by otkl via cheatography.com/179565/cs/37358/

Types of Errors

IndexError	Raised when the index of a sequence is out of range
NameError	Raised when a variable is not found in the local and global scope
SyntaxError	Raised by the parser when a syntax error is encountered
TypeError	Raised when a function/operation is applied to an object of an incorrect type
UnboundLocalError	Raised when a reference is made to a local variable in a function/method, but no value has been bound to that variable
ZeroDivisionError	Raised when the second operand of a division/module operation is zero
ValueError	Raised when a function gets an argument of a correct type but improper value
MemoryError (RecursionError)	Raised when an operation runs out of memory
RuntimeError	Raised when an error does not fall under any other category

Alphabetical Order (ASCII Table, ord & chr)

48: 0 49: 1 50: 2 51: 3 52: 4 53: 5 54: 6 55: 7 56: 8 57: 9 58: : 59: ; 60: < 61: = 62: > 63: ? 64: @

65: A 66: B 67: C 68: D 69: E 70: F 71: G 72: H 73: I 74: J 75: K 76: L 77: M 78: N 79: O 80: P 81: Q 82: R 83: S 84: T 85: U 86: V 87: W 88: X 89: Y 90: Z

97: a 98: b 99: c 100: d 101: e 102: f 103: g 104: h 105:i 106: j 107: k 108: l 109: m 110: n 111: o 112: p 113: q 114: r 115: s 116: t 117: u 118: v 119: w 120: x 121: y 122:z

ord('A') = 65, chr(66) = 'B'

0 < 9 < 'A' < 'Z' < 'a' < 'z'

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Loop Statements

break	Terminates the whole loop
continue	Stops the current iteration of the loop, and goes on to the next iteration of the loop
pass	Does nothing and continues the rest of the code inside the current iteration of the loop

Boolean Values

False evaluates to 0; int(False) == 0, while True evaluates to 1; int(True) = 1

On the other hand, any empty str, tuple, list ('', (), []), the value 0 and None evaluates to False; bool(0/None/""/()) = False, and any other expression will evaluate to True; bool(1/-95/"CS1010S is fun"/("C", "S", "S", "U", "C", "K", "S")) = True

String Slicing Mechanism

s = 'abcdef '

0 1 2 3 4 5

-6-5-4-3-2-1

s[start(inclusive):stop(exclusive):step]

e.g.

s[1:] = 'bcdef'

s[3::-1] = 'dcba'

s[6:] = ''

s[2:-6:-1] = 'cb'

Tuple and string functions

len()	Returns the length of the string/number of items in the tuple
max()	Returns the largest item in the tuple
min()	Returns the smallest item in the tuple
sum()	Returns the sum of all elements in the tuple
tuple()	Converts an iterable into a tuple
tuple.count(ele)	Counts the number of occurrences of an element in a tuple
str.index(ele)	Searches the string for a specified ele from the left and returns the position of where it was found

Checking data type

type(value) == Type	isinstance(value, Type)

Orders of Growth (OOG)

O(1) < O(log n) < O(n) < O(n log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)

O(1): Indexing, replacing variable name

O(log n): Constantly halving/doubling a number (depending on direction)

O(n): Going through the whole tuple/string (for loop/recursion)

O(n^2): Going through the whole tuple once for each element (Usually nested for loop)

O(2^n): The tree splits into 2/x number of branches for each level (Usually for recursion tree)

Sample Answer:

Time: O(n), there is a total of n recursive calls.

Space: O(n), there is a total of n recursive calls, and each call will take up space on the stack.

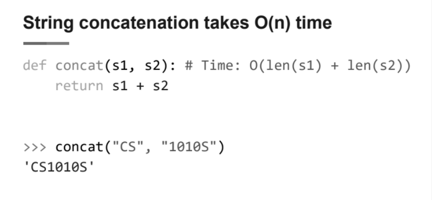
Time: O(n), the loop will iterate n times.

Space: O(1), no extra memory is needed because the variables are overwritten with the new values.

Big O Notation

Time Complexity: Sum of time taken at each level of the recursion tree (number of recursive calls, intensive operations)	Time Complexity: Count the loops, and the intensive operations (eg string concatenation)
Space Complexity: Height of the recursion tree (Also check for strings, tuples, etc)	Space Complexity: Count the variables stored (need to store individual chars for strings)

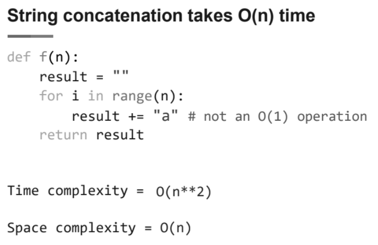
String Concatenation



String concatenation takes O(n) time

```
def concat(s1, s2): # Time: O(len(s1) + len(s2))
    return s1 + s2


>>> concat("CS", "1010S")
'CS1010S'
```

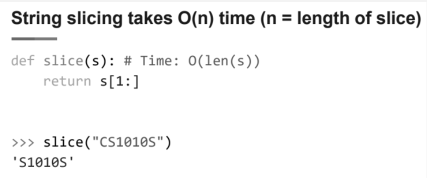
String Concatenation (2)



String concatenation takes O(n) time

```
def f(n):
    result = ""
    for i in range(n):
        result += "a" # not an O(1) operation
    return result


Time complexity = O(n**2)

Space complexity = O(n)
```

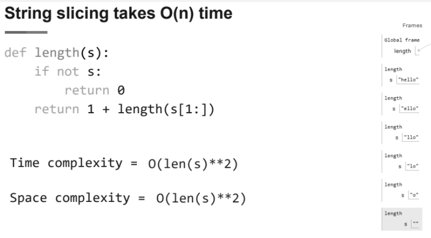
String Slicing



String slicing takes O(n) time (n = length of slice)

```
def slice(s): # Time: O(len(s))
    return s[1:]


>>> slice("CS1010S")
'S1010S'
```

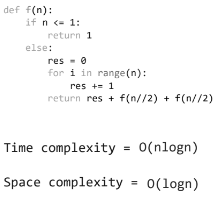
String Slicing (2)



String slicing takes O(n) time

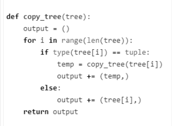
```
def length(s):
    if not s:
        return 0
    return 1 + length(s[1:])


Time complexity = O(len(s)**2)

Space complexity = O(len(s)**2)
```

Extra OOG



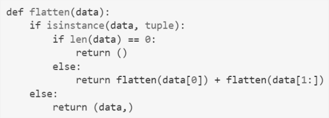
```
def f(n):
    if n <= 1:
        return 1
    else:
        res = 0
        for i in range(n):
            res += 1
        return res + f(n//2) + f(n//2)


Time complexity = O(nlogn)

Space complexity = O(logn)
```

Copy of Tree



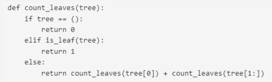
```
def copy_tree(tree):
    output = ()
    for i in range(len(tree)):
        if type(tree[i]) == tuple:
            temp = copy_tree(tree[i])
            output += (temp,)
        else:
            output += (tree[i],)
    return output
```

Flatten Tuples



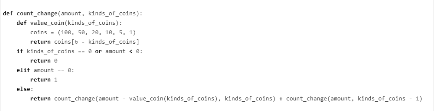
```
def flatten(data):
    if isinstance(data, tuple):
        if len(data) == 0:
            return ()
        else:
            return flatten(data[0]) + flatten(data[1:])
    else:
        return (data,)
```

Counting Leaves



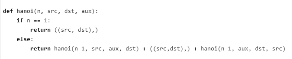
```
def count_leaves(tree):
    if tree == ():
        return 0
    elif is_leaf(tree):
        return 1
    else:
        return count_leaves(tree[0]) + count_leaves(tree[1:])
```

Counting Change Problem



```
def count_change(amount, kinds_of_coins):
    def value_coin(kinds_of_coins):
        coins = (100, 50, 20, 10, 5, 1)
        return coins[6 - kinds_of_coins]
    if kinds_of_coins == 0 or amount < 0:
        return 0
    elif amount == 0:
        return 1
    else:
        return count_change(amount - value_coin(kinds_of_coins), kinds_of_coins) + count_change(amount, kinds_of_coins - 1)
```

Towers of Hanoi



```
def hanoi(n, src, dst, aux):
    if n == 1:
        return ((src, dst),)
    else:
        return hanoi(n-1, src, aux, dst) + ((src,dst),) + hanoi(n-1, aux, dst, src)
```