

HackerRank: 30 Days of Code Challenge

Nicholas P. Ross

August 16, 2016

Abstract

This is my (NPR's) set of notes from the HackerRank "30 Days of Code Challenge". The general idea here is to help myself learn/brush up on my Python skills. The .tex and .pdf of these notes can be found at: https://github.com/d80b2t/Research_Notes.

Contents

0	Day 0: Hello, World.	5
1	Day 1: Data Types	6
2	Day 2: Operators	7
3	Day 3: Intro to Conditional Statements	8
4	Day 4: Class vs. Instance	9
5	Day 5: Loops	11
6	Day 6: Let's Review	12
7	Day 7: Arrays	13
8	Day 8: Dictionaries and Maps	14
9	Day 9: Recursion	15
10	Day 10: Binary Numbers	16
11	Day 11: 2D Arrays	17
12	Day 12: Inheritance	19
13	Day 13: Abstract Classes	21
14	Day 14: Scope	23
15	Day 15: Linked List	25
16	Day 16: Exceptions - String to Integer	27
17	Day 17: More Exceptions	28
18	Day 18: Queues and Stacks	30
19	Day 19: Interfaces	33
20	Day 20: Sorting	34
21	Day 21: Generics	35
22	Day 22: Binary Search Trees	36

23 Day 23: BST Level-Order Traversal	37
24 Day 24: More Linked Lists	38
25 Day 25: Running Time and Complexity	39
26 Day 26: Nested Logic	40
27 Day 27: Testing	41
28 Day 28: RegEx, Patterns, and Intro to Databases	42
29 Day 29: Bitwise AND	43

General Notes

All in Python3.

0 Day 0: Hello, World.

```
# Read a full line of input from stdin and save it to our
    dynamically typed variable, input_string.
inputString = input()
print (inputString)
```

1 Day 1: Data Types

```
i = 4
d = 4.0
s = 'HackerRank '

# Declare second integer, double, and String variables.
i2 = int(input())    # read int
d2 = float(input())  # read double
s2 = input()         # read string

# print summed and concatenated values
print(i + i2)
print(d + d2)
print(s + s2)
```

2 Day 2: Operators

```
mealCost = float(input())
tipPercent = int(input())
taxPercent = int(input())

tip = mealCost * (tipPercent/100.)
tax = mealCost * (taxPercent/100.)

totalCost = mealCost + tip + tax
total = round(totalCost)

print ('The total meal cost is', int(total), 'dollars.')
```

3 Day 3: Intro to Conditional Statements

```
import sys

N = int(input().strip())
condition = 'Not Weird'

if N % 2 != 0:
    condition = 'Weird'
elif N % 2 == 0 and (N >= 6 and N <= 20):
    condition = 'Weird'
else:
    condition = 'Not Weird'

print(condition)
```

4 Day 4: Class vs. Instance

'''

Objective:

In this challenge, we are going to learn about the difference between a class and an instance; because this is an Object Oriented concept, it is only enabled in certain languages. Check out the Tutorial tab for learning materials and an instructional video!

Task:

Write a Person class with an instance variable, age, and a constructor that takes an integer, initialAge, as a parameter. The constructor must assign initialAge to age after confirming the argument passed as initialAge is not negative; if a negative argument is passed as initialAge, the constructor should set age to 0 and print Age is not valid, setting age to 0. In addition, you must write the following instance methods:

yearPasses() should increase the age instance variable by 1.

amIOld() should perform the following conditional actions:

If age < 13, print You are young.

If >= 13 and age < 18, print You are a teenager.

Otherwise, print You are old.

To help you learn by example and complete this challenge, much of the code is provided for you, but you'll be writing everything in the future. The code that creates each instance of your Person class is in the main method. Dont worry if you dont understand it all quite yet!

'''

```
class Person:
```

```
    def __init__(self, initialAge):
```

```
        # Add some more code to run some checks on initialAge
```

```
        self.age = 0
```

```
        if initialAge < 0:
```

```
            print ("Age is not valid, setting age to 0.")
```

```
        else:
```

```
            self.age = initialAge
```

```
    def amIOld(self):
```

```
        # Do some computations in here and print out the correct  
        # statement to the console
```

```
        if age < 13:
```

```
            print("You are young.")
```

```
        elif 13 <= age < 18:
```

```

        print("You are a teenager.")
    elif age >= 18:
        print("You are old.")

    def yearPasses(self):
        # Increment the age of the person in here
        global age #NPR: don't quite understand what global does
        here...
        age += 1

t = int(input())
for i in range(0, t):
    age = int(input())
    p = Person(age)
    p.amIOld()
    for j in range(0, 3):
        p.yearPasses()
    p.amIOld()
    print("")

```

5 Day 5: Loops

```
'''
```

Objective:

In this challenge, we are going to use loops to help us do some simple math. Check out the Tutorial tab to learn more.

Task

Given an integer, `n`, print its first multiples. Each multiple (where `i` is from 1 to 10) should be printed on a new line in the form: `N x i = result`.

```
'''
```

```
import sys
```

```
N = int(input().strip())
```

```
for ii in range(1, 11):  
    print (N, 'x', ii, '=', N*ii)
```

6 Day 6: Let's Review

Task:

Given a string, S, of length N that is indexed from 0 to N-1, print its even-indexed and odd-indexed characters as space-separated strings on a single line (see the Sample below for more detail).

Note: 0 is considered to be an even index.

Sample Input:

```
2
Hacker
Rank
```

Sample Output:

```
Hce akr
Rn ak
'''
```

```
for i in range(int(eval(input()))):
    s=eval(input())
    print((*["".join(s[::2]),"".join(s[1::2])]))
```

7 Day 7: Arrays

```
'''
Task: Given an array, A, of N integers, print A's elements in
      reverse order as a single line of space-separated numbers.

http://docs.scipy.org/doc/numpy/reference/routines.array-manipulation.html
http://www.scipy-lectures.org/intro/numpy/numpy.html
'''

import sys

n = int(input().strip())
arr = [int(arr_temp) for arr_temp in input().strip().split(' ')]

# print(arr[::-1])
print(" ".join(map(str, arr[::-1])))
```

8 Day 8: Dictionaries and Maps

```
'''
Objective:: Today, we are learning about Key-Value pair mappings
            using a Map or Dictionary data structure. Check out the
            Tutorial tab for learning materials and an instructional video!

Task:: Given N names and phone numbers, assemble a phone book that
        maps friends names to their respective phone numbers. You will
        then be given an unknown number of names to query your phone
        book for; for each name queried, print the associated entry
        from your phone book (in the form ) or if there is no entry for
        .
'''
```

Note: Your phone book should be a Dictionary/Map/HashMap data structure.

Sample Input:

```
3
sam 99912222
tom 11122222
harry 12299933
sam
edward
harry
'''
```

```
import sys

# Read input and assemble phoneBook
n = int(input())
phoneBook = {}
for i in range(n):
    contact = input().split(' ')
    phoneBook[contact[0]] = contact[1]

# Process Queries
lines = sys.stdin.readlines()
for i in lines:
    name = i.strip()
    if name in phoneBook:
        print(name + '=' + str( phoneBook[name] ))
    else:
        print('Not found')
```

9 Day 9: Recursion

'''

Objective:: Today, we are learning and practicing an algorithmic concept called Recursion. Check out the Tutorial tab for learning materials and an instructional video!

Task:: Write a factorial function that takes a positive integer, N, as a parameter and prints the result of N!

Note: If you fail to use recursion or fail to name your recursive function factorial or Factorial, you will get a score of 0.

Input Format: A single integer, N (the argument to pass to factorial).

'''

```
def factorial(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)  
  
print((factorial(int(eval(input())))))
```

10 Day 10: Binary Numbers

```
'''
Task:: Given a base-10 integer, n, convert it to binary (base-2).
       Then find and print the base-1- integer denoting the maximum
       number of consecutive 1's in n's binary representation.

'''

bin(11111113)
# '0b101010011000101011001001'

bin(11111113)[2:]
# '101010011000101011001001'

bin(11111113)[2:].split()
# ['101010011000101011001001']

bin(11111113)[2:].split('0')
# ['1', '1', '1', '', '11', '', '', '1', '1', '11', '', '1', '',
  '1']

max(bin(11111113)[2:].split('0'))
# '11'

#len(max(bin(11111113)[2:].split('0')))

print(len(max(bin(int(input().strip()))[2:].split('0'))))
```

11 Day 11: 2D Arrays

```
'''
```

Context: Given a 6x6 2D Array, A:

```
1 1 1 0 0 0
0 1 0 0 0 0
1 1 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

We define an hourglass in A to be a subset of values with indices falling in this pattern in A's graphical representation:

```
a b c
  d
e f g
```

There are 16 hourglasses in A, and an hourglass sum is the sum of an hourglass's values.

Task:: Calculate the hourglass sum for every hourglass in A, then print the maximum hourglass sum.

Sample Input::

```
1 1 1 0 0 0
0 1 0 0 0 0
1 1 1 0 0 0
0 0 2 4 4 0
0 0 0 2 0 0
0 0 1 2 4 0
```

Sample Output::

```
19
'''
```

```
import sys
```

```
arr = []
for arr_i in range(6):
    arr_t = [int(arr_temp) for arr_temp in input().strip().split(' ')]
    arr.append(arr_t)
```

```
res = []
for x in range(0, 4):
    for y in range(0, 4):
        print(x,y)
```

```
print('arr[x] [y:y+3]', arr[x] [y:y+3])
print('arr[x+1] [y+1]', arr[x+1] [y+1])
print('arr[x+2] [y:y+3]', arr[x+2] [y:y+3])
s = sum(arr[x] [y:y+3]) + arr[x+1] [y+1] + sum(arr[x+2] [y:y+3])
res.append(s)

print(max(res))
```

12 Day 12: Inheritance

'''

Objective:: Today, we are delving into Inheritance. Check out the Tutorial tab for learning materials and an instructional video!

Task:: You are given two classes, Person and Student, where Person is the base class and Student is the derived class. Completed code for Person and a declaration for Student are provided for you in the editor. Observe that Student inherits all the properties of Person.

Complete the Student class by writing the following:

A Student class constructor, which has parameters:

A string, firstName.

A string, lastName.

An integer, id.

An integer array (or vector) of test scores, scores.

A char calculate() method that calculates a Student object's average and returns the grade character representative of their calculated average.

Sample Input::

Heraldo Memelli 8135627

2

100 80

Sample Output::

Name: Memelli, Heraldo

ID: 8135627

Grade: O

'''

```
class Person:
```

```
    def __init__(self, firstName, lastName, idNumber):
```

```
        self.firstName = firstName
```

```
    self.lastName = lastName
```

```
    self.idNumber = idNumber
```

```
    def printPerson(self):
```

```
        print("Name:", self.lastName + ", ", self.firstName)
```

```
        print("ID:", self.idNumber)
```

```
class Student(Person):
```

```
    def __init__(self, firstName, lastName, idNumber, scores):
```

```
        Person.__init__(self, firstName, lastName, idNumber)
```

```

        self.testScores = scores

    def calculate(self):
        average = 0
        for i in self.testScores:
            average += i

        average = average / len(self.testScores)

        if(average >= 90):
            return 'O' # Outstanding
        elif(average >= 80):
            return 'E' # Exceeds Expectations
        elif(average >= 70):
            return 'A' # Acceptable
        elif(average >= 55):
            return 'P' # Poor
        elif(average >= 40):
            return 'D' # Dreadful
        else:
            return 'T' # Troll

line = input().split()
firstName = line[0]
lastName = line[1]
idNum = line[2]
numScores = int(input()) # not needed for Python
scores = list( map(int, input().split()) )
s = Student(firstName, lastName, idNum, scores)
s.printPerson()
print("Grade:", s.calculate())

```

13 Day 13: Abstract Classes

'''

Objective:: Today, we are taking what we learned yesterday about Inheritance and extending it to Abstract Classes. Because this is a very specific Object-Oriented concept, submissions are limited to the few languages that use this construct. Check out the Tutorial tab for learning materials and an instructional video!

Task:: Given a Book class and a Solution class, write a MyBook class that does the following:

-- Inherits from Book

-- Has a parameterized constructor taking these parameters:

-- string title

-- string author

-- int price

Implements the Book class abstract display() method so it prints these 3 lines:

1. Title, a space, and then the current instances title.
2. Author, a space, and then the current instances author.
3. Price, a space, and then the current instances price.

Note: Because these classes are being written in the same file, you must not use an access modifier (e.g.:) when declaring MyBook or your code will not execute.

Input Format:: You are not responsible for reading any input from stdin. The Solution class creates a Book object and calls the MyBook class constructor (passing it the necessary arguments). It then calls the display method on the Book object.

Output Format:: The void display() method should print and label the respective, title, author and price of the MyBook objects instance (with each value on its own line) like so:

Title: \$title

Author: \$author

Price: \$price

Note: The \$ is prepended to variable names to indicate they are placeholders for variables.

Sample Input::

The following input from stdin is handled by the locked stub code in your editor:

The Alchemist

Paulo Coelho

```

248
'''

from abc import ABCMeta, abstractmethod

class Book(object, metaclass=ABCMeta):
    def __init__(self, title, author):
        self.title=title
        self.author=author
    @abstractmethod
    def display(): pass

class MyBook(Book):
    price = 0
    def __init__(self, title, author, price):
        super(Book, self).__init__()
        self.price = price

    def display(self):
        print("Title: " + title)
        print("Author: " + author)
        print("Price: " + str(price))

title =input()
author =input()
price =int(input())
new_novel=MyBook(title,author,price)
new_novel.display()

```

14 Day 14: Scope

"""

Objective:: Today we're discussing scope. Check out the Tutorial tab for learning materials and an instructional video!

The absolute difference between two integers, a and b , is written as $|a-b|$. The maximum absolute difference between two integers in a set of positive integers, elements, is the largest absolute difference between any two integers in elements.

The Difference class is started for you in the editor. It has a private integer array (elements) for storing N non-negative integers, and a public integer (maximumDifference) for storing the maximum absolute difference.

Task:: Complete the Difference class by writing the following:

- A class constructor that takes an array of integers as a parameter and saves it to the instance variable.
- A computeDifference method that finds the maximum absolute difference between any numbers in and stores it in the instance variable.

Input Format:: You are not responsible for reading any input from stdin. The locked Solution class in your editor reads in 2 lines of input; the first line contains N , and the second line describes the elements array.

Sample Input::

```
3
1 2 5
```

Sample Output::

```
4
"""
```

```
class Difference:
    def __init__(self, a):
        self.__elements = a
# Add your code here
    def computeDifference(self):
        self.maximumDifference = abs(max(self.__elements) -
                                     min(self.__elements))
# End of Difference class

##
## NOTE TO NPR: The indentation really had to be correct here!!
## aka, you kinda really need to understand these Classes a bit
## better!!
```

```

##

"""
Your constructor must save the argument passed as its integer array
parameter to the integer array instance variable ().

The computeDifference method must then access the the integer array
instance variable () and find its maximum and minimum elements.
Once these are found, the method must save their absolute
difference to the instance variable.

Note: The use of Math.abs is not really necessary. Because the
problem constraints stipulate that we are only dealing with
positive numbers, will always be positive.
"""

_ = input()
a = [int(e) for e in input().split(' ')]

d = Difference(a)
d.computeDifference()

print(d.maximumDifference)

```

15 Day 15: Linked List

"""

Objective:: Today we're working with Linked Lists. Check out the Tutorial tab for learning materials and an instructional video!

A Node class is provided for you in the editor. A Node object has an integer data field, data, and a Node instance pointer, next, pointing to another node (i.e.: the next node in a list).

A Node insert function is also declared in your editor. It has two parameters: a pointer, head, pointing to the first node of a linked list, and an integer data value that must be added to the end of the list as a new Node object.

Task:: Complete the insert function in your editor so that it creates a new Node (pass data as the Node constructor argument) and inserts it at the tail of the linked list referenced by the head parameter. Once the new node is added, return the reference to the head node.

Note: If the head argument passed to the insert function is null, then the initial list is empty.

Input Format:: The insert function has 2 parameters: a pointer to a Node named head, and an integer value, data. The constructor for Node has 1 parameter: an integer value for the data field.

You do not need to read anything from stdin.

Output Format::

Your insert function should return a reference to the head node of the linked list.

Sample Input::

The following input is handled for you by the locked code in the editor:

The first line contains T, the number of test cases.

The T subsequent lines of test cases each contain an integer to be inserted at the list's tail.

4
2
3
4
1

Sample Output::

The locked code in your editor prints the ordered data values for each element in your list as a single line of space-separated

```

    integers:

2 3 4 1
"""

class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
class Solution:
    def display(self,head):
        current = head
        while current:
            print(current.data,end=' ')
            current = current.next

    def insert(self,head,data):
        #Complete this method
        current = head
        if(current):
            while(current.next):
                current = current.next
            current.next = Node(data)
            return head
        else:
            return Node(data)

mylist= Solution()
T=int(input())
head=None
for i in range(T):
    data=int(input())
    head=mylist.insert(head,data)
mylist.display(head);

```

16 Day 16: Exceptions - String to Integer

```
#!/bin/python3
"""
```

Objective: Today, we're getting started with Exceptions by learning how to parse an integer from a string and print a custom error message. Check out the Tutorial tab for learning materials and an instructional video!

Task: Read a string, *S*, and print its integer value; if *S* cannot be converted to an integer, print Bad String.

Note: You must use the String-to-Integer and exception handling constructs built into your submission language. If you attempt to use loops/conditional statements, you will get a 0 score.

Input Format: A single string, *S*.

Output Format: Print the parsed integer value of *S*, or Bad String if *S* cannot be converted to an integer.

Sample Input 0

3

Sample Output 0

3

Sample Input 1

za

Sample Output 1

Bad String

```
"""
```

```
import sys
```

```
S = input().strip()
```

```
try:
```

```
    print(int(S))
```

```
except:
```

```
    print("Bad String")
```

17 Day 17: More Exceptions

"""

Objective: Yesterday's challenge taught you to manage exceptional situations by using try and catch blocks. In today's challenge, you're going to practice throwing and propagating an exception. Check out the Tutorial tab for learning materials and an instructional video!

Task: Write a Calculator class with a single method: `int power(int,int)`. The power method takes two integers, `n` and `p`, as parameters and returns the integer result of n^p . If either `n` or `p` is negative, then the method must throw an exception with the message: `n and p should be non-negative`.

Note: Do not use an access modifier (e.g.: `public`) in the declaration for your Calculator class.

Input Format: Input from stdin is handled for you by the locked stub code in your editor. The first line contains an integer, `t`, the number of test cases. Each of the subsequent lines describes a test case in space-separated integers denoting `n` and `p`, respectively.

Constraints: No Test Case will result in overflow for correctly written code.

Output Format: Output to stdout is handled for you by the locked stub code in your editor. There are `t` lines of output, where each line contains the result of `n^p` as calculated by your Calculator class' power method.

Sample Input:

```
4
3 5
2 4
-1 -2
-1 3
```

Sample Output

```
243
16
n and p should be non-negative
n and p should be non-negative
"""
```

```
class Calculator:
    def power(self,n,p):
```

```

        self.n = n
        self.p = p
        if n < 0 or p < 0:
            raise Exception("n and p should be non-negative")
        else:
            return n**p

myCalculator=Calculator()
T=int(input())
for i in range(T):
    n,p = map(int, input().split())
    try:
        ans=myCalculator.power(n,p)
        print(ans)
    except Exception as e:
        print(e)

```

18 Day 18: Queues and Stacks

"""

Welcome to Day 18! Today we're learning about Stacks and Queues.
Check out the Tutorial tab for learning materials and an instructional video!

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backwards and forwards. Can you determine if a given string, `s`, is a palindrome?

To solve this challenge, we must first take each character in `s`, enqueue it in a queue, and also push that same character onto a stack. Once that's done, we must dequeue the first character from the queue and pop the top character off the stack, then compare the two characters to see if they are the same; as long as the characters match, we continue dequeuing, popping, and comparing each character until our containers are empty (a non-match means `s` isn't a palindrome).

Write the following declarations and implementations:

1. Two instance variables: one for your stack, and one for your queue.
2. A void `pushCharacter(char ch)` method that pushes a character onto a stack.
3. A void `enqueueCharacter(char ch)` method that enqueues a character in the instance variable.
4. A `char popCharacter()` method that pops and returns the character at the top of the instance variable.
5. A `char dequeueCharacter()` method that dequeues and returns the first character in the instance variable.

Input Format:: You do not need to read anything from `stdin`. The locked stub code in your editor reads a single line containing string `s`. It then calls the methods specified above to pass each character to your instance variables.

Constraints: `s` is composed of lowercase English letters.

Output Format: You are not responsible for printing any output to `stdout`.

If your code is correctly written and `s` is a palindrome, the locked stub code will print `Yes`; otherwise, it will print `No`.

Sample Input:
`racecar`

Sample Output:
`Yes`

```

The word, racecar, is a palindrome
"""

import sys

class Solution:
    # Write your code here
    def __init__(self):
        self.mystack = list()
        self.myqueue = list()
        return(None)

    def pushCharacter(self, char):
        self.mystack.append(char)

    def popCharacter(self):
        return(self.mystack.pop(-1))
        ## pop(0) removes and returns the first entry from the list,
        ## pop(-1) removes and returns the last entry in the list.

    def enqueueCharacter(self, char):
        self.myqueue.append(char)

    def dequeueCharacter(self):
        return(self.myqueue.pop(0))

# read the string s
s=input()

#Create the Solution class object
obj=Solution()

l=len(s)
# push/enqueue all the characters of string s to stack
for i in range(l):
    obj.pushCharacter(s[i])
    obj.enqueueCharacter(s[i])

isPalindrome=True
'''
pop the top character from stack
dequeue the first character from queue
compare both the characters
'''
for i in range(l // 2):
    if obj.popCharacter()!=obj.dequeueCharacter():
        isPalindrome=False
        break

```

```
#finally print whether string s is palindrome or not.  
if isPalindrome:  
    print("The word, "+s+", is a palindrome.")  
else:  
    print("The word, "+s+", is not a palindrome.")
```

19 Day 19: Interfaces

"""

Nota Bene: Day 19 was about Interfaces.

This was acutally not a Python solveable challenge (only C++, C#, Java 7, Java 8 and PHP) were offered, so no code was written here or progress made.

"""

20 Day 20: Sorting

21 Day 21: Generics

22 Day 22: Binary Search Trees

23 Day 23: BST Level-Order Traversal

24 Day 24: More Linked Lists

25 Day 25: Running Time and Complexity

26 Day 26: Nested Logic

27 Day 27: Testing

28 Day 28: RegEx, Patterns, and Intro to Databases

29 Day 29: Bitwise AND
