https://hackernoon.com/technical-data-science-interview-questions-sql-and-coding-jv1k32bf (https://hackernoon.com/technical-data-science-interview-questions-sql-and-coding-jv1k32bf)
https://www.interviewquery.com/blog-machine-learning-interview-questions/ (https://www.interviewquery.com/blog-machine-learning-interview-questions/)
https://medium.com/swlh/how-to-answer-data-science-interview-coding-questions-b5e6b2335c7e (https://medium.com/swlh/how-to-answer-data-science-interview-coding-questions-b5e6b2335c7e)
https://github.com/kojino/120-Data-Science-Interview-Questions (https://github.com/kojino/120-Data-Science-Interview-Questions)

1) FizzBuzz. Print numbers from 1 to 100
If it's a multiplier of 3, print "fizz"
If it's a multiplier of 5, print "buzz"
If both 3 and 5 — "fizzbuzz"
Otherwise, print the number itself

Example of output: 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, 31, 32, Fizz, 34, Buzz, Fizz, ...

2) Factorial. Calculate a factorial of a number
factorial(5) = 5! = 1 *2* 3 *4* 5 = 120
factorial(10) = 10! = 1 *2* 3 *4* 5 *6* 7 *8* 9 * 10 = 3628800<

In [1]:
```
#Recursively
def factorial(n):
    if n<0:
        return 0
    elif n==1 or n==0:
        return 1
    else:
        return n*factorial(n-1)
print('Recursive Method Factorial of 7 is : ',factorial(7))

# Using iteration
def factorialIterative(n):
    # handle 0,1 and -ves else compute factorial
    fact=1
    while(n>1):
        fact*=n
        n-=1
    return fact
print('Iterative Method Factorial of 7 is : ',factorialIterative(7))


# Using in-built math function
import math
def factorialMath(n):
    return math.factorial(n)
print('Math Function Factorial of 7 is : ',factorialMath(7))
```

```
Recursive Method Factorial of 7 is :  5040
Iterative Method Factorial of 7 is :  5040
Math Function Factorial of 7 is :  5040
```

In [3]:
```

```

Out[3]:
```
5040
```

In [5]:
```
def addNumbers(x,y):
    if y==0: # if y==0 then return x since x+0 =x
        return x
    summation=x^y # add two numbers using ^ operator without carrying
    carry=(x & y)<<1 # carry without adding
    return addNumbers(summation,carry)
addNumbers(7,3)
```

Out[5]: 10

Add two numbers without using arithmetic operators

In [2]:
```
# Half adder logic
def add(a,b):
    if b==0: # 1. If b==0 then a
        return a
    summation=a^b # 2. Add two numbers without carrying
    carry=(a & b)<<1 # 3. Carry without adding
    return add(summation,carry) # 4. Recursively compute the sumation
print(add(7,3))
```

```
10
```

Recursive function to check if a string is palindrome

In [3]:
```
def palindrome(s):
    if len(s)<2: # 1. If string is empty then it's a palindrome.
        return "Palindrome"
    if s[0]==s[-1]:  # 2. If the first and last elements are same.
        return palindrome(s[1:-1]) # 3. Check the elements from second index to last index, excluding the last index v
alue.
    else:
        return "Not palindrome"

print(palindrome('MoM'))
```

```
Palindrome
```

Remove Duplicates from a string

In [4]:
```
word='mama'
# Without considering order
print(''.join(set(word)))
# Considering order

print(''.join(sorted(set(word), key=word.index)))

# Remove duplicate words from a sentence
sentence='Web Design, Web Development, Web Technology'
' '.join(set(sentence.split()))
```

```
am
ma
```

Out[4]:
```
'Design, Web Technology Development,'
```

3) Mean. Compute the mean of number in a list

In [5]:
```
lst=[12,9,4,6,8]
# 1. Using the sum() and len() functions. sum(list)/len(list)
def listAveragewithSumFunc(lst):
    return sum(lst)/len(lst)
print(listAveragewithSumFunc(lst))

from statistics import mean
def listAveragewithMeanStatFunc(lst):
    return mean(lst)
print(listAveragewithMeanStatFunc(lst))

# importing reduce()
from functools import reduce
def listAveragewithReduceFunc(lst):
    return reduce(lambda a, b: a + b, lst) / len(lst)
print(listAveragewithReduceFunc(lst))


def listAveragewithLoopFunc(lst):
    sumOfNumbers = 0
    for t in lst:
        sumOfNumbers = sumOfNumbers + t

    avg = sumOfNumbers / len(lst)
    return avg
print(listAveragewithLoopFunc(lst))
```

```
7.8
7.8
7.8
7.8
```

4) Variance. Calculate the variance of elements in a list.

In [6]:
```
lst=[12,9,4,6,8]
def calVariance(lst):
    mean=sum(lst)/len(lst)
    variance=sum((xi - mean) **2 for xi in lst) /len(lst)
    return variance

print(calVariance(lst))

# Sample Variance
from statistics import variance
variance(lst)

# Population Variance
from statistics import pvariance
pvariance(lst)
```

```
7.360000000000001
```

Out[6]:
```
7.36
```

5) STD. Calculate the standard deviation of elements in a list.

5) RMSE. Calculate the RMSE (root mean squared error) of a model. The function takes in two lists: one with actual values, one with predictions.

rmse([1, 2], [1, 2]) = 0 rmse([1, 2, 3], [3, 2, 1]) = 1.63

6) Remove duplicates. Remove duplicates in list. The list is not sorted and the order of elements from the original list should be preserved.

1

[1, 2, 3, 1] ⇒ [1, 2, 3] [1, 3, 2, 1, 5, 3, 5, 1, 4] ⇒ [1, 3, 2, 5, 4]

In [36]:
```
lst=[1, 3, 2, 1, 5, 3, 5, 1, 4]
set(lst)
```

Out[36]:
```
{1, 2, 3, 4, 5}
```

7) Count. Count how many times each element in a list occurs.

[1, 3, 2, 1, 5, 3, 5, 1, 4] ⇒

1: 3 times 2: 1 time 3: 2 times 4: 1 time 5: 2 times

8) Palindrome. Is string a palindrome? A palindrome is a word which reads the same backward as forwards.

“ololo” ⇒ Yes “cafe” ⇒ No

In [41]:
```
def pallindrome(s):
    if len(s)<2:
        return "Palindrome"
    elif s[0]==s[-1]:
        return pallindrome(s[1:-1])
    else:
        "Not Palindrome"
s="ololo"
print(pallindrome(s))
```

```
Palindrome
```

11) RLE. Implement RLE (run-length encoding): encode each character by the number of times it appears consecutively.

'aaaabbbcca' ⇒ [('a', 4), ('b', 3), ('c', 2), ('a', 1)] (note that there are two groups of 'a')

12) Jaccard. Calculate the Jaccard similarity between two sets: the size of intersection divided by the size of union.

jaccard({'a', 'b', 'c'}, {'a', 'd'}) = 1 / 4

13) IDF. Given a collection of already tokenized texts, calculate the IDF (inverse document frequency) of each token.

input example: [['interview', 'questions'], ['interview', 'answers']]

Where:

t is the token, n(t) is the number of documents that t occurs in, N is the total number of documents

14) PMI. Given a collection of already tokenized texts, find the PMI (pointwise mutual information) of each pair of tokens. Return top 10 pairs according to PMI.

input example: [['interview', 'questions'], ['interview', 'answers']] PMI is used for finding collocations in text — things like “New York” or “Puerto Rico”. For two consecutive words, the PMI between them is:

1

The higher the PMI, the more likely these two tokens form a collection. We can estimate PMI by counting:

Where:

N is the total number of tokens in the text, c(t1, t2) is the number of times t1 and t2 appear together, c(t1) and c(t2) — the number of times they appear separately.

These questions can also be used to check the knowledge of NumPy — some of them may be solved in NumPy with just one or two lines.

Next, we’ll look at a slightly different type of coding tasks — algorithmic questions.

1) Two sum. Given an array and a number N, return True if there are numbers A, B in the array such that A + B = N. Otherwise, return False.

[1, 2, 3, 4], 5 ⇒ True [3, 4, 6], 6 ⇒ False

2) Fibonacci. Return the n-th Fibonacci number, which is computed using this formula:
F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)
The sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

3) Most frequent outcome. We have two dice of different sizes (D1 and D2). We roll them and sum their face values. What are the most probable outcomes?

6, 6 ⇒ [7] 2, 4 ⇒ [3, 4, 5]

4) Reverse a linked list. Write a function for reversing a linked list.

The definition of a list node: Node(value, next) Example: a -> b -> c ⇒ c -> b -> a

5) Flip a binary tree. Write a function for rotating a binary tree.

The definition of a tree node: Node(value, left, right)

6) Binary search. Return the index of a given number in a sorted array or -1 if it’s not there.

[1, 4, 6, 10], 4 ⇒ 1 [1, 4, 6, 10], 3 ⇒ -1

7) Deduplication. Remove duplicates from a sorted array.

[1, 1, 1, 2, 3, 4, 4, 4, 5, 6, 6] ⇒ [1, 2, 3, 4, 5, 6]

8) Intersection. Return the intersection of two sorted arrays.

[1, 2, 4, 6, 10], [2, 4, 5, 7, 10] ⇒ [2, 4, 10]

9) Union. Return the union of two sorted arrays.

[1, 2, 4, 6, 10], [2, 4, 5, 7, 10] ⇒ [1, 2, 4, 5, 6, 7, 10]

10) Addition. Implement the addition algorithm from school. Suppose we represent numbers by a list of integers from 0 to 9:

12 is [1, 2] 1000 is [1, 0, 0, 0]

Implement the “+” operation for this representation

[1, 1] + [1] ⇒ [1, 2]
[9, 9] + [2] ⇒ [1, 0, 1]

11) Sort by custom alphabet. You’re given a list of words and an alphabet (e.g. a permutation of Latin alphabet). You need to use this alphabet to order words in the list.

Example (taken from here):

Words: ['home', 'oval', 'cat', 'egg', 'network', 'green'] Dictionary: 'bcdfghijklmnpqrstvwxzaeiouy' Output:

['cat', 'green', 'home', 'network', 'egg', 'oval']

12) Check if a tree is a binary search tree. In BST, the element in the root is:

Greater than or equal to the numbers on the left Less than or equal to the number on the right The definition of a tree node: Node(value, left, right)

Linear Search Algorithm

In [7]:
```
def linear_search(list_items, query):
    #start seraching from position 0
    position=0
    #Loop through the list
    while position<len(list_items):
        #check if the item is at current position then return the position and exit
        if list_items[position]==query:
            print('Item found')
            return position
        #increment to the next position in the list
        position+=1
    return -1
```

In [8]:
```
list_items=[67,90,50,38,70,55]
query=70
linear_search(list_items, query)
```

```
Item found
```

Out[8]: 4

Arrays

In [13]:
```
months = ["Jam", "Feb", "Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
print(months)
```

```
['Jam', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

List

In [12]:
```
months = ["Jam", "Feb", "Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec",1,2,3,4,5,6,7,8,9,10,11,12]
print(months)
```

```
['Jam', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1, 12]
```

Using the list() constructer

In [43]:
```
months=list(("Jam", "Feb", "Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec",1,2,3,4,5,6,7,8,9,10,11,12))
print(months)
```

```
['Jam', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1, 12]
```

Dictionary

In [25]:
```
gdp= {'Country': 'USA', 'Pop. (M)': 325.08, 'GDP (T)': 19.485,
      'GDP Growth %':2.27, 'GDP Per capita':59939,
      'Share of World GDP %':24.08,
     'States':["New York","Washington","New Jersey"]}
print(gdp)
```

```
{'Country': 'USA', 'Pop. (M)': 325.08, 'GDP (T)': 19.485, 'GDP Growth %': 2.27, 'GDP Per capita': 59939, 'Share of Wo
rld GDP %': 24.08, 'States': ['New York', 'Washington', 'New Jersey']}
```

tuple

In [46]:
```
months=("Jam", "Feb", "Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec",1,2,3,4,5,6,7,8,9,10,11,12)
print(months)
```

```
('Jam', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1, 12)
```

Using the tuple() constructer

In [30]:
```
months=tuple(("Jam", "Feb", "Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec",1,2,3,4,5,6,7,8,9,10,11,12))
print(months)
```

```
('Jam', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1, 12)
```

Sets

In [33]:
```
set1={"Jar","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec","Dec"}
set2={1,1,1,2,2,3,4,5,6,7,8,9,10,11,12}

print(set1,set2)
```

```
{'Dec', 'Jul', 'Sep', 'May', 'Jun', 'Aug', 'Oct', 'Nov', 'Jar'} {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Series from a List

In [3]:
```
import pandas as pd
import numpy as np
continents = np.array(['Africa','Asia','Europe','America','Arctic'])
s = pd.Series(continents)
print(s)
```

```
0     Africa
1       Asia
2     Europe
3    America
4     Arctic
dtype: object
```

Series from dictionary

In [6]:
```
import pandas as pd
data = {'1' : 'Africa', '2' : 'Asia', '3' : 'Europe','4': 'America','5':'Arctic'}
s = pd.Series(data)
print(s)
```

```
1     Africa
2       Asia
3     Europe
4    America
5     Arctic
dtype: object
```

Creating Data Frame from Python List

In [15]:
```
import pandas as pd
gdp_share = [['Asia',47.4],['North America',21.7],['Europe',19.9],
             ['Africa',5.0],['South America',4.8],['Oceania',1.2]]
df = pd.DataFrame(gdp_share,columns=['Country','GDP Sahre %'])
print(df)
```

```
         Country  GDP Sahre %
0           Asia         47.4
1  North America         21.7
2         Europe         19.9
3         Africa          5.0
4  South America          4.8
5        Oceania          1.2
```

Creating Data Frame from Python Dictionary

In [16]:
```
import pandas as pd
gdp_share = {'Country':['Asia','North America', 'Europe', 'Africa', 'South America','Oceania'],
             'GDP Sahre %':[47.4,21.7,19.9,5.0,4.8,1.2]}
df = pd.DataFrame(gdp_share)
print(df)
```

```
         Country  GDP Sahre %
0           Asia         47.4
1  North America         21.7
2         Europe         19.9
3         Africa          5.0
4  South America          4.8
5        Oceania          1.2
```

Creating Empty Panel

In [41]:
```
import pandas as pd
p = pd.Panel()
print(p)
```

```
<pandas.Panel object at 0x000002085234C7B8>
```

Creating Panel from Dictionary of Data Frames

In [44]:
```
import pandas as pd
import numpy as np

df1 = pd.DataFrame({'Country':['Asia','North America', 'Europe', 'Africa', 'South America','Oceania'],
                    'GDP Sahre %':['4,641,054,775','592,072,212','747,636,026','1,340,598,147','430,759,766','43,111,7
04']})

df2 = pd.DataFrame({'Country':['Asia','North America', 'Europe', 'Africa', 'South America','Oceania'],
                    'GDP Sahre %':[47.4,21.7,19.9,5.0,4.8,1.2]})

data = {'population':df1, 'gdp':df2}

p = pd.Panel(data)
# print("panel['b'] is - \n\n", panel['b'])

# print("\nShape of panel['b'] is - ",  panel['b'].shape)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-ab89dbc5ad11> in <module>()
     10 data = {'population':df1, 'gdp':df2}
     11
---> 12 p = pd.Panel(data)
     13 # print("panel['b'] is - \n\n", panel['b'])
     14

TypeError: object() takes no parameters
```