Post Bootcamp Expectations

Module 1 : Basic Coding

Introduction to Module:-

Welcome to the module "Programming in Python".

Programming is one of the most fundamental skills you will need as a data scientist/analyst. Among the numerous tools available for analysing data, we choose Python as it is the most robust, has an intuitive syntax, and is a modern and widely used language.

In this module:

The module will be focused on developing problem-solving ability using Python and familiarising you with programming. This will mostly be done by solving various problems and a lot of practice.

This module has been broken down in 4 major portions:

- Basic Programming
  - Here you will learn about how to think and approach coding questions. This portion will be mostly based on building a strong sense of logic using only the basics.
- Lists
  - In this session, you will solve questions based on lists. You already know most of the basic syntax from the previous module. Here you will be challenged to use those learnings in an effective way to tackle different problems.
- Strings
  - Here you will solve questions based on strings. You already know the basic syntax from the previous module. In this session, you will see how to leverage those to try and solve different problems and build a strong logical thought process.
- Other Data Structures
  - In this session, you will solve questions based on the dictionary, which is one of the most important data structure for a data scientist. You will be able to leverage your learnings from the previous sessions where the keys and values of a dictionary can be analogous to lists and strings.

We have introduced a new feature called 'Programming Bootcamp' that helps learners from non-coding background to strengthen their programming fundamentals. Refer to the link below

that contains the detailed documentation of this feature.

Session Overview:-

In this session:

In this session, you will first revise the basics of conditional statements, looping syntax and operators in Python. This session focuses on building a strong sense of logic and the importance of making a flow diagram or writing pseudocode before solving any problem. You will also see how to break down an intimidating problem and approach it with sound logic.

We will do this by:

- First, breaking down a few famous industry application problems and demonstrate how that can be done using simple concepts that you have learned in the previous module - Introduction to Python.
- Next, we will take up a few problems starting from an easy one and slowly increasing the difficulty. It is recommended you solve all of them and also try them on your own employing different approaches than the ones that are illustrated.
  We will consider the following problems in this session:
  - Swapping
  - Even or Odd
  - Alarm Clock
  - Factorial
  - Reverse The Digits
  - How Many Chocolates
  - Print the Pattern

Basic Refresher:-

In this session, we will be learning how to approach different problems and develop a strong foundation in logic building. Here are a few questions for your revision, kindly revise the basic conditionals, loops and operators from the previous module - Introduction to Python before starting.

Real-Life Scenarios of Programming - I:-

The first step to solving any problem is to think about what you are going to do. Only after deciding the right process and flow of the solution, you explain the same process to the computer using a programming language. So no matter what language you write your code in, to explain the computer what you want it to do, the first step is always deciding the flow. This is called writing pseudocode.

Let’s listen to Sajan as he shows how to break down some complex real-life

problems into smaller problems and write pseudocode for those using flow diagrams.

Our first problem for this discussion is the implementation of the back button. The back button is not only used in browsers but also in smartphones and different applications. Ever wondered how that works? Of course, we will not be designing the real back button for different applications or browsers, but we can still try and break down the basic implementation. This basic thought or flow of the problem is then worked on and improved with different things like giving different illustrations and applying a similar concept for different data forms and so on. Let's see how you will solve the implementation of the back button.

Here you saw a very rudimentary application of the back button using stacks. Another concept similar to stack is a queue. You can read more about stacks and queues **here**. We saw how the back button can be implemented as a stack using lists. This is also how the 'undo' function works in your text editors. The programs are built upon many times to improve performance and speed but the core idea never changes.

The next problem is YouTube views and upGrad views. You must have realised by now that YouTube keeps a count of the unique views each video gets. Even upGrad platform keeps track of the video being seen by its student to later analyse the content and student performance. How do you think this is done? Let us see Sajan explain this algorithm in the video below, which we are tracking if you viewed or not as well :P

Now you know why different websites store cookies or collect IP addresses or even ask you to signup or link accounts before viewing the content.

With this much data being saved, the next big problem is to search the entry of interest from this accumulated ton of data. Our next problem is the same. While shopping online on a platform you enter the product name and the platform finds all the relevant results for displaying, or while using any other filter as well. How would you implement this functionality on a platform?

Although a 'search bar' in real life has been further developed to perform more complex searches than just basic string matching, this should serve as a very good starting point to further develop a more sophisticated search bar. This solution might seem very trivial now, but that is only because we selected the right data structure. Selecting the right data structure to store a particular data depending on the problem at hand can make the problem extremely easy or hard.

Try these following very easy problem statements and see if you can come up with a flow diagram to solve the problem, remember to put some thought into what data you will be saving and in what format.

Real-Life Scenarios of Programming - II:-

The next problem on our plate is the food delivery application platforms. If you have ever placed a food order online, you might have noticed that you only see the restaurants that will deliver an order to you, at the top of the display list, from the thousands of restaurants registered with the application. How do you think this happens? How would you do it if you were asked to solve this problem?

Like you realised, you don't only have to design the flow or think of storing the data, you also have to think about what more information is needed from the users or some other source of data. In this case, we had to take GPS locations and map data from users. Storing this, we can later analyse the behaviour of different users to see if they tend to order from restaurants closer to them or some other analysis to help make better business decisions.

You should have a pretty good idea by now about how to draw flowcharts depicting the solutions of different problems. Let us conclude this discussion with this slightly complex problem about any dating website or application. Can you try and think of some way to design an application similar to Tinder?

Important Note: In the video below, we have demonstrated a simple use-case of Tinder which shows the interaction between 2 straight users - one male user (pronouns: him/his) and one female user (pronouns: she/her). Please note that the actual Tinder algorithm is very complex and considers all genders and preferences as mentioned by the person using it.

That was an interesting video. Here are a few problem statements for you to try and design a flow diagram to solve them. You can make few assumptions but don't forget to state those so that your peers can also comment on the same along with the TAs.

Swapping:-

Let’s start writing codes now. Sajan will help you break down the problem and then show you how to code it on the console. Let’s start with a very simple problem first. You will be given two integers x and y. You have to swap values stored in them. Have a look at the detailed problem statement in the coding quiz question at the end of this page.

Having understood the logic behind the problem, let's now take a look at how to code it using the coding console.

#Take input using input()

```
#input() takes input in form of the string
in_string=input()

#here extract the two numbers from the string
inp_lst = [int(x) for x in in_string.split(',')]
x = inp_lst[0]
y = inp_lst[1]

#print x and y before swapping
print('x before swapping: {0}'.format(x))
print('y before swapping: {0}'.format(y))
print()

#Writing your swapping code here
x = x + y
y = x - y
x = x - y

#print x and y after swapping
print('x after swapping: {0}'.format(x))
print('y after swapping: {0}'.format(y))
```

Even Or Odd:-

Let’s take up another problem of finding whether a given integer is even or odd. You may have a look at the problem statement from the coding quiz question at the end of the page before proceeding with the video.

The aim of this question is to revise the if-else conditions and also to learn to draw a flow chart for every problem and to convert it to code. Let's see Sajan make a simple flow diagram to show how to solve the question.

Making the flow diagram is the same as representing the solution in a diagrammatic form. You may choose to write it and explain it in simple words too, but it is easier with a diagram.

Let us now see Sajan write a code for the same diagram that he explained above.

Alarm Clock:-

Do you wake up at the same time every morning? No, right? If it is a normal working day where you're swamped with work and meetings starting early morning, you try to wake up as soon as possible. If it is a weekend or a vacation day, you mostly want to give yourself some extra rest, right? Now, you might set alarms for this manually periodically depending on what day it is. Instead

of manually setting up an alarm, what if you try to write a code that automates the time the alarm goes off. If there was some way for the alarm to know the kind of day it is, it would automatically ring according to the preferences set by you. Let's attempt to do this.

So the problem statement is, you're trying to automate your alarm clock by writing a function for it. You're given a day of the week encoded as **1 = Mon**, **2 = Tue**, ..., **6 = Sat**, **7 = Sun**, and whether you are on vacation as a boolean value (a boolean object is either True or False. Google "booleans Python" to get a better understanding).

Based on the day and whether you're on vacation, write a function that returns a time in form of a string indicating when the alarm clock should ring.

When **not** on a vacation, on weekdays, the alarm should ring at "7:00" and on the weekends (Saturday and Sunday) it should ring at "10:00".

While **on** a vacation, it should ring at "10:00" on weekdays. On vacation, it should **not** ring on weekends, i.e., it should return "off". You may have a look at the problem statement from the coding quiz question before proceeding with the video.

You must have a pretty good idea of how to draw simple flow diagrams now. Let's try and convert it to a code now.

You can do it using if... elif... elif... elif... else statement as well. Can you try writing a code with that approach? (Hint: You will find the 'and' operator quite useful here).

So you have now automated your alarm clock to ring according to your needs so that you can always catch that extra hour of sleep depending on the day. Feeling lazy already? Well, right now you have to keep your minds open and brains charged as there are many such questions to go in the upcoming segments.

```
#Take input here
#we will take input using ast sys
import ast
input_str = input()

#ast.literal_eval() will evaluate the string and make a data structure for the same
#here the input is a list since input is in '[...]', so ast.literal_eval() will
#make a list with the same data as passed
input_list = ast.literal_eval(input_str)
```

```
#the data or the two values in list is now changed to separate variables
day_of_the_week = input_list[0] #first element is an integer denoting the day of the week
is_on_vacation = input_list[1] #this is a boolean denoting if its vacation or not

# write your code here
weekends = [6,7]
def alarm_time(day_of_the_week,is_on_vacation):
    if is_on_vacation:
        if day_of_the_week not in weekends:
            return '10:00'
        else:
            return 'off'
    else:
        if day_of_the_week not in weekends:
            return '7:00'
        else:
            return '10:00'

print(alarm_time(day_of_the_week,is_on_vacation))
```

Factorial:-

The next problem is to find the factorial of a number. A factorial is defined for integers greater than or equal to zero and is defined as: n! = nx(n-1)! Till we reach 1

Now, if you need to find the factorial of a number 'n', you just need to multiply 1 with 2, then the result with 3, succeeded with a multiplication with 4 and so on until 'n'. Sounds like this question needs to be done using a loop. So given a number ***n,*** can you find **n!**? Let's hear from Sajan on how to approach it.

```
#take the input here
number= int(input())



#the function definition starts here
def factorial(n):
    #write the funtion here that finds and RETURNS factorial of next
    if n <= -1:
        return -1
    elif n == 0:
```

```
    return 1
  else:
    f = 1
    for i in range(1,n+1):
      f = f * i
    return f


#function definition ends here

#do not alter the code typed below
k=factorial(number)
print(k)
```

Reverse The Digits:-

Until now, we saw how to use loops and conditionals. Let's now move a step further and solve a tougher problem. Here you will reverse a number.

Given a number, say **24312**, you have to reverse it and print it, **21342**. (hint: you will find % and // operators useful here). Can you try making a flow diagram and then converting it to code? Try it below!

```
#take input of the number here
n = int(input())


#write code to reverse the number here
s = 0
while n > 0 :
  s = s * 10 + n % 10
  n = n // 10
print(s)
```

How Many Chocolates?:-

This question is actually a puzzle that some of you must have solved as a kid. Let's say you have ***m*** Rupees and one chocolate costs rupees ***c***. The shopkeeper will give you a bar of chocolate for free if you give him 3 wrappers. Can you determine how many chocolates can you get with the **m** Rupees you have?

This might seem like a tough problem to code in the first glance, but let's see Sajan break it down and make it easier

```
#take input here
import ast
```

```
input_str = input()

input_list = [int(i) for i in input_str.split(',')]
m = input_list[0]
c = input_list[1]

#start writing your code here
choco = m // c
wrap = m // c
while wrap // 3 != 0:
    choco  = choco + wrap // 3
    wrap = wrap // 3 + wrap % 3
print(choco)
#dont forget to print the number of chocolates Sanjay can eat
```

Print The Pattern:-

Pattern printing is an exercise done frequently to learn and master loop iterations. This type of exercise is recommended for all new coders and professionals. Please go to the coding quiz question below and check the detailed problem statement.

Given a positive integer, you have to print the pattern as illustrated. We recommend solving a lot more of these types of problems on your own. You will also be solving these types of problems in the upcoming lessons and modules.

Don't worry if you are still confused, watch Sajan explain the problem in detail and then you will feel confident to solve it on your own.
Let's see Sajan convert the above logic to code

These types of problems may seem a little overwhelming at first, but you have already learnt how to break them into pieces and solve them. Pattern printing problems make the best questions for practice. Try and make an account on different coding platforms and attempt more of these for practice.

```
#please take input here
n = int(input())

#start writing your code here
for i in range(1,n+1):
    for j in range(n-i):
        print(' ',end='')
    for k in range(i-1):
        print('*_',end='')
    print('*')
```

```
#please take input here
n = int(input())

#start writing your code here

for i in range(1,n+1):
   for j in range(n,i,-1):
      print(' ',end='')
   for k in range(1,i+1):
      if k == i:
         print('*',end='')
      else:
         print('*_',end='')
   print()
```

Summary:-

In this session, you learnt about how to think and approach coding questions. The following topics were covered in this session:

- Swapping
- Even or Odd
- Alarm Clock
- Factorial
- Reverse the digits
- How many chocolates?
- Print the pattern

Module 2 : List

Session Overview:-

In this session, we will build upon the basics of lists learned in the previous module - Introduction to Python. You will learn to iterate on lists and use loops and conditionals with lists, apply list comprehensions to make code more comprehensible, short, and clean. You will also learn about different inbuilt Python functionalities to make use of while coding. We will also demonstrate how shallow copy can backfire and collapse your code and logic.

We will do this by covering the following problems:

- Smallest Element
- Above Average
- Recruit New Members
- Calendar
- Fenced Matrix

Refresher:-

Smallest Element:-

Let's start with a very basic simple question of finding the smallest or minimum element from the given list of integers. You will be using these type of codes like finding the minimum, average, maximum, median, and a lot more when dealing with new data. Let's listen to Sajan explain the approach that you can take.

We can also use the Python inbuilt functionality to perform a lot of actions. Let's see Sajan code the problem explained above and show us the Python inbuilt functionality to do the same.

You can read more about min() functionality of Python **here**. Other important Python inbuilt functions can be found **here**. It is always better to use these than to write code that performs the same action. However, the aim of this question is to develop a coding intuition and help you understand the working of these inbuilt functions.

Above Average:-

Just like the previous problem, finding the average is another action you will perform quite frequently while analysing the data. Given a list of integers, can you find the average and determine if the given number is above the list's average or not?

Recruit New Members:-

This can be seen as an extension of the previous segment. Suppose you are the manager of a big firm and are looking to hire new members for your team. You published an advertisement and have received a few applications.

You rate people on a scale of 0 to 100 and have given scores to all the members of your team and the new applicants. The selection process is very straightforward - if the applicant improves the average of the team then you hire the applicant or else reject the applicant. Remember the order of processing applications is going to be important here.

Maintaining modularity in your code is a very important aspect of writing code in the industry. You will be building on programs that are developed by your colleagues and vice versa. To make that easy for them and you, it's always better to have a modular code. Let us see Sajan use the code from the previous question and solve this one.

Calendar:-

Let’s raise the bar a little now by taking level-2 lists (2-D list) into consideration. Here is a simple problem to start.

You are planning to go to your friend's wedding and you have long events all month, going on for at least a few days. You have the start and end dates of events and your task is to find out events that are overlapping with the wedding date.

Let’s understand how Sajan approaches the problem.

```
#taking input
import ast
input_str1 = input()
input_list1 = ast.literal_eval(input_str1)
events = input_list1

wedding = int(input())


#start writing from here
clash = 0
for s,e in events:
   if wedding>=s and wedding<=e:
      clash +=1

print(clash)
```

Fenced Matrix:-

You will be given two positive integers ***m*** and ***n***. You have to make a list of lists (which can be visualised as a matrix) of size **m*n**, that is m sublists (rows), with each sublist having n integers (columns).

The matrix should be such that it should have ***1*** on the border
and ***0*** everywhere else. Check the coding quiz question below for more details.

Deep and Shallow copy is one of the most significant error encountered. To illustrate the difference let us consider the following piece of code.

```
original_list=[0, 0, 0]
copy1=original_list
copy2=list(original_list)
copy3=list.copy(original_list)
original_list[0] = 1
print(copy1)
print(copy2)
print(copy3)
```

The output we were expecting is
[**0**, **0**, **0**]
[**0**, **0**, **0**]
[**0**, **0**, **0**]

Since all 3, copy1, copy2, copy3, are copies of original_list, any change in the original list should not be reflected in copy1, copy2 or copy3. However, the result we get is
[**1**, **0**, **0**]
[**0**, **0**, **0**]
[**0**, **0**, **0**]

Why is that?
Let's dig in a little deeper. When you make ANY data structure, you ask the computer to give you some space in its memory. Now when you asked the computer for original_list's space in the first line, it returned you the asked space.

Think of this as a box. You can store anything in this box: list, dict, anything. When you next asked ***copy1=original_list***, you wanted a new box with its contents same as the box named ***original_list***, but what the computer did was to give you the box named original_list instead of making a new box.

Now if you make changes in the content of the box
using **original_list** or **copy1**, because it is the same box it will be reflected in other as well. This is called a **shallow copy**. It is called a **deep copy** when you make the computer give you a new box as in the case of **copy2** and **copy3**.
Now try coding the same thing below.

Summary:-

In this session, you solved questions based on lists. You already know most of the basic syntax from the previous module. Here you were challenged to use those learnings in an effective way to tackle different problems.

Following topics were dicussed in this module:

- Smallest Element
- Above Average
- Recruit New Memebrs
- Calender
- Fenced Matrix

Module 3 : Strings

Session Overview:-

In this session

After covering lists, you can now start exploring the interaction of different data structures, namely list of strings. You will first revise basic syntax like the previous two sessions and then move on to coding questions. You will also learn to make use of basic functionalities like appending, slicing, typecasting etc.

One of the first steps in data analysing as a data scientist is acquiring the data. The acquired data is mostly in string format. Cleaning this data is the first step. Learning functionalities of the string is thus, one of the most important objectives of this session. You will be doing similar activities in the upcoming questions.

You will be understanding the following questions in this session:

- Palindrome
- Reverse Words
- No Spaces
- Move Vowels
- Common Prefix
- Anagrams

Palindrome String:-

Let’s start with a simple and common problem: Palindrome String.

A string is considered palindrome if it stays the same upon reversing it. For example ‘racecar’. Can you write a code that takes the input of a string and checks if it's palindrome or not?

Before we start off with the coding section, let’s understand the logic.

```
# Read the input
s = input()

#check for palindrome here

# Convert all the characters of the string to lowercase so that your program is
# not case-sensitive. You can also use 'lower()', alternatively.
s = s.lower()

# Initialise your reversed string as an empty string and store it in a variable
# called "r"
r =""
```

```
# Run a loop throughout the length of the string
for i in s:
# Keep adding the current character to rev_str, i.e. the string reversed so far.
# Make sure 'i' is kept before 'r' in the addition process or you will get
# the r exactly same as the original string.
   r = i+r

# Compare the reversed and the original string. If they are the same, the string
# is a palindrome; otherwise it is not a palindrome.
if r==s:
   print(1)
else:
   print(0)
```

Aliter: My Way

```
# Read the input
s = input()
s = s.lower()
strlen = len(s)
halfway = strlen // 2
f = 0
#check for palindrome here
for i in range(0,halfway):
   if s[i] != s[strlen -1 -i]:
      f = 1
      break

if f == 0:
   print(1)
else:
   print(0)
```

Reverse Words:-

In this segment, you will learn about a useful functionality of strings and lists called reverse().

Let’s take an example to illustrate it. You will be given a sentence in the form of a string. You have to reverse the order of the words in the sentence. Remember not to reverse the individual words, character by character, but the order of words. Before we see Sajan explain how we can do this with a very small code, can you try it out?

Note that Sajan uses split(' '), whereas using only split() would have worked too as default argument for split function is space. It is always a good practice to not depend on default arguments and pass them the same like space in split function.

```
#take input here
#no need to change input as its in string form
sentence=input()

#reverse the words of the sentence here

#first separate the words by spliting with space
words = sentence.split(' ')

#reversed_list = words[::-1]
#reverse this list
words.reverse()

#join using space
final_string = ' '.join(words)

print(final_string)
```

Aliter:

```
#take input here

sentence=input()
lst_sen = sentence.split(' ')
#reverse the words of the sentence here
lst_sen.reverse()
for i in lst_sen:
    print(i,end=' ')
```

No Spaces:-

The two fundamental skills for any data analyst is getting data, and then cleaning it. Only then can you work your magic around the data to extract or draw actionable insights from it. When you get your data, it is never in a useable form; one of the most common problems is having to deal with spaces. Spaces are always to be avoided in naming the variables and the column names and a lot more similar places. Another problem is values like cost or any other numbers have commas in their representation, e.g. one lakh is 1,00,000 and due to this, the system treats it as a string and not an integer or float.

In this segment, you will address the first problem with the help of Sajan, and

then address the second problem on your own in practice problems. You may look at the detailed problem statement in the coding quiz question below.

```
#take input here
s=input()
s = s.lower()
s = s.title()
new_s = s.replace(' ','_')
print(new_s)

#words = [wrds.capitalize() for wrds in s.split(' ')]

#write code to format the string s as asked
#sen = '_'.join(words)
```

Move Vowels:-

Next, we will write a code to shift all the vowels to the front and consonants to the back in the given string, without changing their order. This might seem quite tough at first, doesn't it? Give it a thought and try to think of an approach before we see Sajan break down this complex looking problem.

This goes to illustrate how a complex looking problem can be so easy beneath once we calmly break it down to smaller chunks and think about it in the right way.

Common Prefix:-

We often have to find the common part of two sets of data. It can be when we are merging two data sets, or you sometimes see if a certain data or a certain part of data appears more times implying it is either more important or highly reliable and a lot more.

In this question, we will take up a simpler problem of the same type. You will be given two strings. Can you think of a way to find only the common prefix part in the two strings? Let's see as Sajan explains it.

Anagrams:-

Let's try a fun problem now. Two words are called anagrams if they have the exact same letters but in a different order. For example, 'night' and 'thing' are anagrams.

Can you write a code to find out if two strings are anagrams or not? We strongly

recommend that you try this question on your own by thinking of the logic, building the flow chart, and translating that logic into code in the console below before you watch Sajan's explanation of the code.

Summary:-

Here you solved questions based on strings. You already know the basic syntax from the previous module.
In this session, you saw how to leverage those to try and solve different problems and build a strong logical thought process.

Following topics were discussed in this module:

- Palindrome String
- Reverse Words
- No Spaces
- Move Vowels
- Common Prefix
- Anagrams

Module 4: Others Data Structure

Session Overview:-

In this session

In this session, we will learn about all other data structures like tuples, sets and dictionary. From the data science perspective, the dictionary is one of the most important data structure and you will learn so in upcoming modules. You will focus on iterating through the dictionary keys and values. The key learnings from the previous sessions will also fit in this as more often than not, the keys and values are either strings or lists or lists of strings. You will also see different scenarios where a dictionary will make the problem much easier.

The learning objectives will be achieved with the help of the following problems:

- Remove Duplicates
- Dictionary and List
- upGrad String
- Balanced Brackets

Remove Duplicates:-

You often encounter duplicates in your data when you get it from different

sources. Although there are inbuilt functions in the Pandas library to remove duplicates which you will learn in upcoming modules, this raises the question of how do you really remove duplicates.

Let's take a smaller version of the same problem. You will be given a list of integers, you have to remove all duplicate values from the list. How would you do it?

Dictionary And List:-

One of the most common ways to store data is in a dictionary or in a data frame (you will learn about the data frame in later modules). So it is important to have a good grasp of iterating through dictionary keys and values. Please have a look at the coding quiz question below before you proceed to see Sajan solve it. Feel free to attempt it before you watch the video.

In this segment, you learned how to iterate on the keys and the values of the dictionary. You can iterate on values in two ways, by using **dict.values()** and also by iterating on **dict.keys()** and using **dict[k]** to access the value of key ***k***. Choosing what way to traverse through the dictionary will be crucial and will change with your objective for the same.

upGrad String:-

The next problem will help you in developing a strong sense of dictionary key and value. For the purpose of this question, we will define something called an upGrad string. Note that this definition is not valid outside this question.

A string is an upGrad string if the frequency of its characters is something like 1, 2, 3, 4, ... i.e., a character appears only once, another appears twice, another appears thrice and so on.

For example string '**yrrssrsr**' is an upGrad string since the frequency of y:1, $:2, s:3, r:4, however string '**yrrssrsr%**' will **not** be an upGrad string since it has two characters (y and %) with frequency 1. The frequency of characters should be of the form 1, 2, 3, 4, 5... only. Given a string, can you determine if the string is upGrad string or not?

Balanced Brackets:-

By now you must have realised that matching brackets get highlighted when you are coding on the console. Your next problem is based on the same thing. Given a string on brackets, can you determine if the string of brackets is balanced or not?

```
# take input
inp=input()

stack = []

for i in inp:
    if len(stack)==0:
        stack.append(i)
    else:
        if i==')' and stack[-1]=='(':
            stack.pop()
        elif i=='}' and stack[-1]=='{':
            stack.pop()
        elif i==']' and stack[-1]=='[':
            stack.pop()
        else:
            stack.append(i)

if stack:
    print('No')
else:
    print('Yes')
```

Aliter:-

```
inp=input()
d = {
    '(':0,
    ')':0,
    '[':0,
    ']':0,
    '{':0,
    '}':0
}
for i in inp:
    if i in d:
        d[i] = d[i]+1
if d['('] == d[')'] and d['{'] == d['}'] and d['['] == d[']']:
    print('Yes')
else:
    print('No')
```

Summary:-

In this session, you solved questions based on the dictionary, which is one of

the most important data structure for a data scientist.

Following topics were discussed in this module:

- Remove Duplicates
- Dictionary and List
- upGrad String
- Balanced Brackets

Module 5 : Python Additional Links