Table of Contents

Python lecture series

> **Note :** This tutorial book can be downloaded from:
>
> https://www.gitbook.com/book/nitish6174/python-lecture-series/details

Which version of Python :

We will be using python 3 in this series.

It is preferred to use Python 3.6 which is the latest version although any version higher than 3.4 is fine.

Running Python :

- **Ubuntu** and **Mac** already have python3 in terminal.
  Write your code in a file and run the file in terminal with `python3` command
- **Windows** user can install Python 3 in Command Prompt and run files with python3 command (similar to Ubuntu)

  Alternately, install `IDLE (Python GUI)` application.

- **Running online** : https://try.jupyter.org/

  - Choose New -> Python3
  - Type code in textbox area and then use `run cell` button.

Text editors :

- Sublime Text
- Atom
- PyCharm
- Spyder

> See Sublime Text tweaks for a list of useful plugins in Sublime Text

Online resources :

- TheNewBoston video tutorials
- Udacity video tutorials

- Python documentation
- Tutorials point reference

Speaker :

Nitish Garg

Python lecture series - plan

Lecture 1

> **Topics :**
>
> - Input and output
> - Variables and basic data types
> - Typecasting (between int, float, str)
> - Arithmetic and logical operators
> - Decision making (if-elif-else)
> - Looping (for, while)
>
> **Application programs :**
>
> - Simple Calculator
> - Greatest of 3 numbers
> - Table of a number
> - Factorial
> - Pattern printing
> - e^x approximation

Lecture 2

> **Topics :**
>
> - Loops (pending portion from Lecture 1)
> - Comments
> - Arrays (lists)
> - Lists - insert, delete, merge, extend
> - Strings
> - Operations - split, join, stats, find, sort
>
> **Application programs :**
>
> - Printing multiples of a number
> - Decimal to binary
> - Finding square of numbers in a list
> - Mean and median in list
> - Find primes using sieve method

Lecture 3

> **Topics :**
>
> - Functions
> - Modules and import
> - Mapping
> - Recursion
> - Array/string splicing
> - List comprehensions
>
> **Application programs :**
>
> - nCr and nPr
> - Date format conversion
> - Tower of Hanoi
> - Binary search
> - Palindrome
> - Finding pending assignments

Lecture 4

> **Topics :**
>
> - Tuples
> - zip function
> - Sets
> - Dictionary
> - Array of dictionary
> - Comprehensions with dictionary
> - Determining datatype of variable
>
> **Application programs :**
>
> - Adding 2 matrices
> - Frequency analysis in array
> - Update course registration

Lecture 5

> **Topics :**
>
> - File input/output
> - JSON data file
> - pip
> - CSV files
>
> **Application programs :**
>
> - Sort lines in file
> - Article title
> - Contest Leaderboard

Lecture 6

> **Topics :**
>
> - sys library
> - os library
> - Error handling

Lecture 7

> **Topics :**
>
> - urllib library
> - using curl

Lecture 8

> **Topics :**
>
> - regex

Lecture 9

> **Topics :**
>
> - Functional programming
> - map, filter, reduce
> - Iterators

Other topics

> - Object-oriented programming
> - Web development using Django, Flask
> - Data analysis with pandas, numpy, scipy, matplotlib
> - Machine learning using scikit
> - Image processing using OpenCV
> - Machine learning using scikit
> - GUI development using GTK

Python lecture series - Lecture 1

> **Topics :**
>
> - Input and output
> - Variables and basic data types
> - Typecasting (between int, float, str)
> - Arithmetic and logical operators
> - Decision making (if-elif-else)
> - Looping (for, while)
>
> **Application programs :**
>
> - Simple Calculator
> - Greatest of 3 numbers
> - Table of a number
> - Factorial
> - Pattern printing
> - e^x approximation

Running our first program

- Open a new file, say `main.py`
- Type the following in file :

  ```
  print("My first program")
  ```

- Save file and in terminal/command prompt, `cd` to the directory where the file is saved
- Run `python3 main.py`

Input and Output

- **input()** - take input from console
- **print()** - output to console

Example :

We will take in a name from user and greet him

```
print("Enter your name")
name = input()
print("Hello, "+name)
```

Variables and basic data types

Basic types of data that we use :

- Text data :
  - Characters
  - Strings
- Numerical data :
  - Integers
  - Decimals (floating-point numbers)

Rules on variable naming :

- Start with letter or underscore
- Rest name can have letters, numbers, underscore
- Do not use reserved words (for,sum,in etc)

Assigning or changing value of a variable :

On LHS of `=` symbol, keep the variable in which we want to store the value or result of calculation which will be on RHS

- `myVariable = value or expression`

Following code adds the value of variable `n` with 5 and stores the result back in `n` itself.

- `n = n+5`

Below is the short notation form for `n = n + 2`

- `n += 2`

Typecasting (converting a variable to another type) :

`input()` always gives us string type of data. But if the input was intended to be numerical data, we must convert the string type variable to numerical form.

If a variable is of string type but has number-type value, we can get the integer form by passing the variable to `int()` function

```
a = '5'
b = int(a)
```

Thus, to take integer type input, we can directly pass `input()` to `int()` function :

```
n = int( input() )
```

Similarly, to get decimal value from string, use `float()` function

```
a = '5.82'
b = float(a)
```

To convert numerical variable to string, use `str()` function

```
a = 3
b = str(a)
```

Arithmetic operators

- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Division
- `%` : Modulus (remainder)
- `**` : Exponent (power)

Logical operators

To check equality between LHS and RHS. Answer is `True` or `False`

- `==` : is equal to
- `<` : is less than
- `>` : is greater than
- `<=` : is less than or equal to
- `<=` : is greater than or equal to

Joining strings and more on printing

Strings can simply be joined with `+`

```
a = "one"
b = "two"
c = "three"
d = a+b+c
e = a+" "+b
print(d)
print(e)
print(e+","+c)
```

Output of above code:

```
onetwothree
one two
one two,three
```

Both the variables on either side of `+` should be of same type i.e. numerical value cannot be added with string value.

So, the following code will give error as `count` is integer type variable and we are adding it with string :

```
count = 5
print("There are "+count+" students")
```

To fix this, you must typecast `count` variable to string form with `str()` :

```
count = 5
print("There are "+str(count)+" students")
```

An easier way to print numerical and string values together in a `print()` while avoiding typecasting is to use commas (`,`) :

```
count = 5
print("There are",count,"students")
```

> **Note :** `,` puts extra space in print while `+` does not.

How to avoid print() coming on next line automatically

Add `end=""` inside the `print()` function.

```
print("line1",end="")
print("line2",end="@")
print("line3",end="\n")
print("line4",end=" ")
print("line5",end=":")
```

Output :

```
line1line2@line3
line4 line5:
```

Decision making

Syntax of if-elif-else :

```
if var==0:
    print("answer is 0")
elif var>0:
    print("answer is positive")
else:
    print("answer is negative")
```

It starts with one `if` statement, followed by any number of `elif` statements (there might even be no `elif` at all) and `else` is used if we want to do something if none of the above conditions match. Hence, `else` can be omitted when not needed.

Application : Simple calculator

This program will take 2 numbers and a symbol as input, decide which arithmetic operation is to be performed based on symbol and display the answer

```
print("Enter first number")
a = int(input())
print("Enter second number")
b = int(input())
print("Enter symbol")
c = input()

if symbol=='+':
    print("Sum is",a+b)
elif symbol=='-':
    print("Difference is",a-b)
elif symbol=='*':
    print("Product is",a*b)
elif symbol=='/':
    print("Division is",a/b)
elif symbol=='%':
    print("Remainder is",a%b)
else:
    print("Not a valid symbol")
```

Application : Greatest of 3 numbers

This program will take 3 numbers and find the largest of them.

The logic is that we first compare first 2 numbers. If first is greater, we compare first with third and greater of them will be the answer. Similarly, if second number was greater or equal to first number, we find the answer by comparing second and third number.

```
print("Enter first number")
a = int(input())
print("Enter second number")
b = int(input())
print("Enter third number")
c = int(input())

if a>b:
    if a>c:
        print(a,"is the largest")
    else:
        print(c,"is the largest")
else:
    if b>c:
        print(b,"is the largest")
    else:
        print(c,"is the largest")
```

Looping

- ### for loop

Syntax :

```
for counter_variable in range( start_from , stop_before , increment_by ) :
    statement1
    statement2
```

Example :

```
for i in range(4,13,1):
    print(i)
```

We use variable `i` as the loop counter i.e. `i` will start from value `4` , then becomes `5` , then `6` , ... and finally `12` . It will never reach or exceed 13.

If we want our counter to increase by 2 every time instead of 1-1, we write:

```
for i in range(4,13,2):
```

This way, `i` will take values 4, 6, 8, 10, 12.

Since most of the time we want to increase our loop counter by 1-1, we can just give the `start_from` and `stop_before` values and python will automatically set `increment_by` to 1 :

```
for j in range(2,6):
```

In this code, `j` will take values 2, 3, 4, 5

Also, many times, we want to start from value 0 and increment by 1-1. In that case, we can just give on value i.e. `stop_before` and python will take care of other 2 values :

```
for k in range(5):
```

In this code, `k` will take values 0, 1, 2, 3, 4

If we want to do reverse counting, say 10 to 1, we will start from 10, stop before 0 and increment loop counter by -1 :

```
for i in range(10,0,-1):
```

This will make `i` take values 10, 9, 8, . . . , 2, 1

- **while loop**

  Syntax :

  ```
  while condition:
      statement1
      statement2
  ```

  Example :

  ```
  count = 1
  while count < 5 :
      print(count)
      count += 1
  ```

  We set the initial value of `count` to be 1. Then, as long as `count` is less than 5, the loop body will run. In the loop body, we are displaying the value of `count` and then increasing `count` by 1.

  Hence, `count` will take on values 1, 2, 3, 4. As soon as it becomes 5, the loop will end.

Application : Table of a number

This program will take a number, say `n` and display its table.

We need to loop from 1 to 10 and multiply `n` with the loop counter.

```
print("Enter a number to find its table")
n = int(input())

for i in range(1,11):
    print(n*i)
```

Awesome! Just 4 lines of code.

Further, replace `print(n*i)` with `print(n,"x",i,"=",n*i)` to get the exact format which we have in tables.

Application : Factorial

Factorial of a number *n* = 1 x 2 x 3 .... x *n*

This program will take a number, say `n` as input and display its factorial.

We begin with 1 as our product. Then, we need to loop from 1 to `n` and update the current product by multiplying it with loop counter.

```
print("Enter a number to find its factorial")
n = int(input())

answer = 1
for i in range(1,n+1):
    answer = answer * i

print("Factorial of",n,"is",answer)
```

Application : Pattern printing

Let us write a program to make a rectangle of 5 lines each containing 10 `o` .

For this, we need to keep a loop for 5 rows and inside this loop, another loop to print `o` 10 times.

Note that, after printing every `o` , we don't want to come on next line. So, we add `end=""` in `print()` to handle that. Once we have printed 10 `o` in a line i.e. inner loop is complete, we write `print("")` to come on the next line.

```
for i in range(1,6):
    for j in range(1,11):
        print("o",end="")
    print("")
```

Output :

```
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
```

Let's modify the above program to print triangle instead of rectangle.

st nd ***th***

So now, we want 1 symbol in 1st row, 2 in 2nd row, ... , ***i*** in ***i**th* row. Thus, we want the inner loop to run ***i*** times (row number is i)

```
for i in range(1,6):
    for j in range(1,i+1):
        print("*",end="")
    print("")
```

Output :

```
*
**
***
****
*****
```

Application : e^x approximation

We will use the formula :

$e^x = 1 + x^1/1! + x^2/2! + x^3/3! + x^4/4! + \ldots\ldots\ldots$

But this is an infinite series. We can't keep on adding infinite number of terms in our program.

So let's add just the first few terms, say first 10.

We notice that numerator of i^{th} term is x^{i-1} and denominator is $(i-1)!$

Thus, numerator of first term is *1* and gets multiplied by *x* in every next term. Similarly, denominator of first term is *1* and gets multiplied by *1*, then *2*, then *3* and so on in each next term.

So, in the below program, we start with variables `answer` as 0, `numerator` and `denominator` as 1. Then we loop from 1 to 10 using loop variable `i` . In the loop, we add the value of `numerator` / `denominator` to `answer` and then multiply `numerator` by `x` and `denominator` by the loop variable i.e. `i`

```
print("Enter x :")
x = int(input())

answer = 0
numerator = 1
denominator = 1

for i in range(1,11):
    term = numerator / denominator
    answer += term
    numerator *= x
    denominator *= i

print("e^x =",answer)
```

Sample run of this program :

```
Enter x :
1
e^x = 2.7182815255731922
```

Quite close to exact value of *e*. Looks good !

But this program works well only for small value of `x` (gave good enough answer upto x = 6). As we enter higher value of `x` in our program, the answer will become less accurate. This was bound to happen as we are considering only 10 terms of an infinite series. For small value of *x*, x^i is not that big as compared to *i !* and so the terms after 10th term could be neglected without losing much accuracy. For higher value of *x*, we need to take the sum of more terms to get close to correct sum.

If we simply run the loop more times, say 1000000 times, the program will consider many more terms thus calculating accurate answer for quite large values of *x*. But we know that the terms in this infinite series keep on getting smaller eventually and so the sum of all the terms which are very small won't affect the answer much. Thus, simply running the loop a huge number of times will unnecessarily add terms having negligible value.

To fix this, we will only consider the terms which are more than `0.1`. Once the terms are smaller than `0.1`, we will stop.

In the below program, we have increased the loop range and used `break` to stop the loop if `term` is less than `0.1` :

```
print("Enter x :")
x = int(input())

answer = 0
numerator = 1
denominator = 1

for i in range(1,1000000):
    term = numerator / denominator
    answer += term
    numerator *= x
    denominator *= i
    if term < 0.1 :
        break

print("e^x =",answer)
```

We can change `0.1` to smaller value like `0.001` to improve the accuracy as then, the program would consider more terms.

Here is the same program written with `while` loop :

```
print("Enter x :")
x = int(input())

answer = 0
numerator = 1
denominator = 1
i = 1
term = numerator / denominator

while term > 0.001 :
    term = numerator / denominator
    answer += term
    numerator *= x
    denominator *= i
    i += 1

print("e^x =",answer)
```

Python lecture series - Lecture 2

> **Topics :**
>
> - Loops (pending portion from Lecture 1)
> - Comments
> - Arrays (lists)
> - Lists - insert, delete, merge, extend
> - Strings
> - Operations - split, join, stats, find, sort
>
> **Application programs :**
>
> - Printing multiples of a number
> - Decimal to binary
> - Finding square of numbers in a list
> - Mean and median in list
> - Find primes using sieve method

Application : Printing multiples of a number

Let's say we want to print all multiples of a number `n` which are between 1 and 100.

Multiple approaches are possible :

```
n = 7
for i in range(1,101):
    if i%n==0:
        print i
```

```
n = 7
for i in range(n,101,n):
    print i
```

```
n = 7
multiple = n
while multiple<=100:
    print multiple
    multiple += n
```

Application : Decimal to binary

To convert a number to binary, we repetitively take its remainder by 2 and reduce the number to half.

```
print("Enter the number to convert")
n = int(input())

while n>0:
    digit = n%2
    print(digit)
    n = int(n/2)
```

But this program is printing digits in reverse order. Lets fix that by storing answer in a string.

```
print("Enter the number to convert")
n = int(input())

answer = ""

while n>0:
    digit = n%2
    answer = str(digit) + answer
    n = int(n/2)

print(answer)
```

Comments

Anything written in a line after `#` is a comment in Python and is ignored while running program.

To use multi-line comments or in a block, use triple quotes.

```
# This is a comment
n = int(input())   # Take input from user
"""
This is a multi-line comment.
We display the square of entered number below.
"""
print(n*n)
```

Arrays (lists)

Making a list

```
# List of integers
runs = [ 78 , 93 , 69 , 84 , 48 ]

# List of strings
names = [ "Sachin" , "Kohli" , "Dhoni" , "Yuvraj" , "Dhawan" ]

# Lists can be printed directly
print(runs)
```

Getting value of i^{th} item of list

For computers, counting starts from *0*.

So, first item in a list is represented with index 0. Similarly, index of 2nd element is 1 and so on.

To get the value of item having index `i` , we put `[i]` after the list name.

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
answer = runs[2]   # answer will become 69
print(runs[4])     # 48 will be printed
```

Also, to get the i^{th} item from the last, we use index `-i`

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
# runs[-1] refers to last element of list
print(runs[-1])     # 48
print(runs[-4])     # 93
```

Inserting items in list

Use the `append()` function on list to add item in the end.

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
runs.append(21)   # give the item to be inserted in append()
print(runs)

# Output will be : [78, 93, 69, 84, 48, 21]
```

To insert in middle of the list say at index `pos` , use `insert()` function.

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
runs.insert(2,21)  # give the position and item to be inserted
print(runs)

# Output will be : [78, 93, 21, 69, 84, 48]
```

Deleting items from list

Use the `del()` function to delete the element at given index.

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
del(runs[1])      # Delete item at index 1 in runs
print(runs)

# Output will be : [78, 69, 84, 48]
```

To remove a particular value from the list, use the `remove()` function on list.

```
runs = [ 78 , 93 , 69 , 84 , 48 ]
runs.remove(84)    # Remove the first 84 in the list
print(runs)

# Output will be : [78, 93, 69, 48]
```

Merging lists

Merging or concatenating lists in Python is as easy as adding numbers

```
# Make 3 lists
a = [ 4 , 8 , 2 , 1 ]
b = [ 7 , 2 ]
c = [ 9 , 4 , 3 ]
# Add the 3 lists
d = a + b + c

print(d)
# Output will be : [4, 8, 2, 1, 7, 2, 9, 4, 3]
```

Extending lists

To add all the numbers of a list `b` to list `a` , we can either simply add the lists back to `a` (same as merging) or, we can use the `extend()` function

```
# Method 1
a = [ 4 , 8 , 2 , 1 ]
b = [ 7 , 2 ]
# Add elements of b to a
a.extend(b)

print(a)
# Output will be : [4, 8, 2, 1, 7, 2]
```

```
# Method 2 : Merging
a = [ 4 , 8 , 2 , 1 ]
b = [ 7 , 2 ]
# Add elements of b to a
a = a + b

print(a)
# Same output as method 1
```

Application : Finding square of numbers in a list

Write a program to convert each number in a list to its square.

First, the program should ask the number of elements we want to insert in the list. Then we run a loop to enter that many numbers and append them to the list. Finally, we loop through the indices of array and print square of the number at that index.

```
print("Enter the number of elements :")
n = int(input())

a = []                # Start with an empty list

# Loop n times to enter numbers
for i in range(n):
    print("Enter number",i)
    num = int(input())
    a.append(num)     # Insert the number in list

# Go through each index and convert element to its square
for i in range(n):
    num = a[i]        # Get the element at index i
    sq = num*num
    a[i] = sq         # Store the square back at that index

print(a)
```

nd

The 2nd loop in the above program can simply be written as :

```
for i in range(n):
    a[i] = a[i] * a[i]
```

Strings

Strings can be seen as a list of characters and so, there are many operation that can be performed on lists as well as strings.

- Just like arrays, the ith character in a string `s` can be accessed using `s[i-1]` as ith character has index `i-1` .
- Strings can be concatenated/added as seen earlier using `+`
- But unlike arrays where we can assign value at an index using statements like `a[i] = 5` , characters in a string cannot be changed. Thus, statements like `s[2] = 'a'` are not allowed.
- We can split a string into list on the basis of some string.

  ```
  s = "this is a sentence"
  a = s.split(" ")          # Break s at every space found
  print(a)
  # output : ['this', 'is', 'a', 'sentence']
  ```

  ```
  s = "49-+-3438-+-22-+-91"
  a = s.split("-+-")
  print(a)
  # output : ['49', '3438', '22', '91']
  ```

  This helps in taking a list as input in a single line separated by space as we can quickly get the array by splitting the input.

  ```
  a = input().split(" ")
  print(a)
  # Sample input : 56 2 9 45 27 332
  # Sample output : ['56', '2', '9', '45', '27', '332']
  ```

Operations on list and string

There are a huge number of operations available on string and list. We discuss some of them. See Python documentation / reference guides for use of others.

- **join** : Join is just the opposite of `split()`. It takes a list of strings and converts it into a string while joining the elements with a given string.

```
a = [ "one" , "two" , "three" , "four" , "five" ]
s = "/".join(a)
print(s)
# Output : one/two/three/four/five
```

- **len** : `len` function can be applied on strings as well as lists and gives the length of given variable.

```
a = [ 3 , 6 , 1 , 0 ]
b = [ "phy" , "chem" , "maths" ]
s = "python"

print( len(a) )       # 4
print( len(b) )       # 3
print( len(s) )       # 6
```

- **min** : For a list of numbers, `min` function gives the smallest number in the list; for a list of strings, it gives the alphabetically smallest string; and for a string, it gives the smallest character in that string.

```
a = [ 3 , 6 , 1 , 0 ]
b = [ "phy" , "chem" , "maths" ]
s = "python"

print( min(a) )       # 0
print( min(b) )       # chem
print( min(s) )       # h
```

- **max** : Similar usage as `min` function

```
a = [ 3 , 6 , 1 , 0 ]
b = [ "phy" , "chem" , "maths" ]
s = "python"

print( max(a) )       # 6
print( max(b) )       # phy
print( max(s) )       # y
```

- **sum** : `sum` function is used on list of numbers.

```
a = [ 3 , 6 , 1 , 0 ]

print( sum(a) )       # 10
```

- **sort** : List of numbers as well as list of strings can be sorted

```
a = [ 3 , 6 , 1 , 0 ]
b = [ "one" , "two" , "three" , "four" , "five" ]

a.sort()
b.sort()

print(a)    # Output : [0, 1, 3, 6]
print(b)    # Output : ['five', 'four', 'one', 'three', 'two']
```

- **find** : To check if a value is present in a list or string, we use the expression `if value in listname`

```
a = [ 3 , 6 , 1 , 0 ]

if 1 in a :
    print("present")
else:
    print("not present")

# Output : present
```

```
b = [ "one" , "two" , "three" , "four" , "five" ]

if "six" in b :
    print("present")
else:
    print("not present")

# Output : not present
```

```
s = "python"

if "o" in s :
    print("present")
else:
    print("not present")

# Output : present
```

Application : Mean and median in list

Mean of a list can be easily found by dividing sum of list by its length

```
a = [ 8 , 2 , 3 , 12 , 5 , 3 , 7 , 9 ]

mean = sum(a) / len(a)

print(mean)
```

For median, we need to sort the list and look at the middle index. If there are even numbers in list, we have 2 middle positions hence take average of the numbers at those index.

```
a = [ 8 , 2 , 3 , 12 , 5 , 3 , 7 , 9 ]

a.sort()
l = len(a)

if l%2 == 1 :
    medianPos = (l-1) / 2
    median = a[medianPos]
else:
    upperMedianPos = l / 2
    lowerMedianPos = upperMedianPos - 1
    median = ( a[lowerMedianPos] + a[upperMedianPos] ) / 2

print(median)
```

Application : Find primes using sieve method

Find the prime numbers till 100.

We make an array `a` to store the status of 100 numbers. If `a[i]` is 1, then it means that `i` is a prime number and if `a[i]` is 0, then it is composite.

Initially, we assume that all numbers are prime and so our array is filled with 1s. Since all multiples of 2 are definitely not prime, we make `a[4]` , `a[6]` , `a[8]` , . . . , `a[100]` to be 0. Then look for next prime number i.e. next value of `i` such that `a[i]` is still 1. Repeat the process of striking off multiples of current prime number and looking for next prime number.

```
a = [1]*101         # Since we need a[100], list should have 101 elements

for i in range(2,101):                # Loop from 2 to 100
    if a[i]==1:                       # If the element has not been striked-off already
        for j in range(2*i,101,i):    # Loop for multiples of current element
            a[j] = 0                  # Strike-off the multiple

for i in range(2,101):                # Run loop to see final status of numbers
    if a[i]==1:                       # If a[i] is not striked-off, it is prime
        print(i,"is prime")
```

Python lecture series - Lecture 3

> **Topics :**
>
> - Functions
> - Modules and import
> - Mapping
> - Recursion
> - Array/string splicing
> - List comprehensions
>
> **Application programs :**
>
> - nCr and nPr
> - Date format conversion
> - Tower of Hanoi
> - Binary search
> - Palindrome
> - Finding pending assignments

Functions

Declaring and using a function :

The variables which we pass to functions are called parameters.

```
def cube(n):
    ans = n*n*n
    return ans

number = int(input("Enter number : "))
print( cube(number) )
```

```
def average(a,b):
    return (a+b)/2

a = int(input("Enter number 1 : "))
b = int(input("Enter number 2 : "))

avg = average(a,b)

print("Average is",avg)
```

Application : nCr and nPr

We will make functions for factorial, nCr and nPr

```
def factorial(n):
    p = 1
    for i in range(2,n+1):
        p *= i

def nPr(n,r):
    answer = factorial(n) / factorial(r)

def nCr(n,r):
    answer = factorial(n) / ( factorial(r) * factorial(n-r) )


print("Enter n : ",end="")
n = int(input())
print("Enter r : ",end="")
r = int(input())

if n<r or r<0 :
    print("Invalid n/r")
else:
    result1 = nCr(n,r)
    result2 = nPr(n,r)
    print(n,"C",r,"=",result1)
    print(n,"P",r,"=",result2)
```

Modules and import

Module is a python file containing functions which we want to use in another python file/program.

Python comes with many useful modules which we can use by importing that module in our program. We can also make our own module and import it.

Using Python in-built module :

```python
# Import math module containing many mathematical functions
import math

a = 16
b = math.sqrt(a)   # Use the square root function from math module
print(b)

print(math.pi)
print(math.e)
```

We can directly import particular functions from a module.

```python
# Import sin, cos functions from math module
from math import sin,ceil
# Now, instead of math.sin(), we can directly use sin()
a = 1.123
b = sin(a)
c = ceil(a)
print(b)
print(c)
```

Or, import all functions of that module, use:

```python
from math import *
```

Making our own module :

Make a file, say `combination.py` . Write the factorial, nCr, nPr functions in it.

Now, in another file, say `main.py` , import the above made file and use those functions :

```python
from combination import *

n,r = 4,2
print( nCr(n,r) )
```

Mapping

To apply a function on each item in a list, we will have to go through the list and call that function on the element.

The shortcut way to do this is use mapping. `map()` takes two parameters: the function name which is to be applied on each item of list and the list variable. Also, we need to convert the result back into list form, so we use `list()` function on the result.

```
def cube(n):
    return n*n*n

a = [ 3 , 1 , 6 , 8 , 2 ]
b = map( cube , a )
b = list(b)
print(b)
```

We often use `map()` and `split()` to input a list of numbers in a single line from console.

```
a = input().split()
b = list( map(int,a) )
# Sample input:  4 3 -2 9 0 10
# a will become : ['4', '3', '-2', '9', '0', '10']
# b will become : [4, 3, -2, 9, 0, 10]
```

or combining the 2 lines of code :

```
a = list( map( int , input().split() ) )
```

Application : Date format conversion

We have a list of dates in mm-dd-yyyy format. The dates are to converted to dd-mm-yyyy format.

We make a function which takes a date in mm-dd-yyyy format and gives back the converted date. Then, to apply this function on each item of the list, use mapping.

```
# Function to convert date from mm-dd-yyyy to dd-mm-yyyy format
def convert(old_date):
    l = list(old_date)          # Get the string in list form
    l[0],l[3] = l[3],l[0]       # Swap first digits of date and month
    l[1],l[4] = l[4],l[1]       # Swap second digits of date and month
    new_date = ''.join(l)       # Convert list back into string
    return new_date

# Make a list of dates in mm-dd-yyyy format
date_list = [ "08-13-2017" , "12-31-2016" , "01-26-2017" , "05-01-2016" ]
# Map the convert function on this list
converted_dates = list(map(convert,date_list))

print(converted_dates)
```

Recursion

We use recursive technique to solve a problem by breaking it into smaller problem.

Ex: To find *n!*, we find *(n-1)!* instead and keep on reducing our number till the base case which is *0!*

A function which calls itself is called a recursive function.

Recursively finding sum of array :

```
def recursive_sum(a,pos):
    if pos==len(a):
        return 0
    else:
        return a[pos]+recursive_sum(a,pos+1)

a = [ 5 , -2 , 3 , 4 , -1 ]
s = recursive_sum(a,0)
print(s)
```

Application : Tower of Hanoi

This famous puzzle has a simple recursive solution.

The problem of moving `n` disks from `Pole A` to `Pole C` can be broken down into 3 steps:

- Move `n-1` disks from `A` to `B`
- Move the n^{th} disk from `A` to `C`

- Move `n-1` disks from `B` to `C`

This way, the problem is reduced recursively until we have only 1 disk to be moved.

```
def hanoi(size, start, middle, end):
    if size > 0 :
        hanoi( size-1 , start , end , middle )
        print( "Move disk" , size , "from" , start , "to" , end )
        hanoi( size-1 , middle , end , start )

n = int(input())
hanoi(n,"A","B","C")
```

Application : Binary search

```
def bin_search(a,value,left,right):
    if left>right:
        return False
    else:
        mid = int( (left+right)/2 )
        if value == a[mid] :
            return True
        elif value > a[mid] :
            return bin_search(a,value,mid+1,right)
        else :
            return bin_search(a,value,left,mid-1)

def search(a,value):
    r = len(a)-1
    if bin_search(a,value,0,r) == True :
        print("Found")
    else :
        print("Not found")


a = [ -34 , -20 , -3 , 2 , 9 , 94 , 482 , 3693 ]

search(a,-3)
search(a,67)
```

Array/string splicing

Just like the element at index `i` in an array or string can be accessed with `a[i]`, we can select a sub-array or substring using `a[start:end]` which will take the elements from index `start` upto index `end` (start index is inclusive, end index is exclusive)

```
a = [ -34 , -20 , -3 , 2 , 9 , 94 , 482 , 3693 ]
b = "this is a sentence"

c = a[2:6]
d = b[3:15]

print(c)        # Output : [-3, 2, 9, 94]
print(d)        # Output : s is a sente
```

To take alternate elements, we can add `:2` in the square brackets. Similarly, that number can be used to control the steps to jump while splicing the list/string. By default, this jump value is 1.

```
a = [ -34 , -20 , -3 , 2 , 9 , 94 , 482 , 3693 ]
b = "this is a sentence"

c = a[2:7:2]
d = b[3:15:3]

print(c)        # Output : [-3, 9, 482]
print(d)        # Output : ss n
```

We can omit start index to take the list/string from beginning. Similarly, end index index can be omitted to take upto the last item.

```
a = [ -34 , -20 , -3 , 2 , 9 , 94 , 482 , 3693 ]
b = "this is a sentence"

c = a[:6]
d = b[3:]

print(c)        # Output : [-34, -20, -3, 2, 9, 94]
print(d)        # Output : s is a sentence
```

To get the list/string in reverse order, the jump value should be kept negative.

```
a = [ -34 , -20 , -3 , 2 , 9 , 94 , 482 , 3693 ]
b = "this is a sentence"

c = a[::-1]
d = b[:3:-3]

print(c)        # Output : [3693, 482, 94, 9, 2, -3, -20, -34]
print(d)        # Output : eeeai
```

Application : Palindrome

Check whether a string is a palindrome or not.

```
def palindrome(s):
    if s == s[::-1]:
        print("Yes")
    else:
        print("No")

a = "abcdcba"
b = "nfiebfie"
c = "aa"
d = "r"
palindrome(a)       # Yes
palindrome(b)       # No
palindrome(c)       # Yes
palindrome(d)       # Yes
```

List comprehensions

Till now, we have seen that to go through a list, we loop through the indices and access the element at given index `i` using `a[i]`

```
for i in range( len(a) ):
    val = a[i]
    print(val)
```

Alternate way to access the elements of the list is to use this syntax :

```
for item in a:
    print(item)
```

In the first method, variable `i` stores indices 0, 1, 2, . . . and we use look for the element at that index.

Whereas, in the second way, the variable `item` is directly holding the elements of list one by one. This syntax makes it easier to access the elements. The only drawback is that we cannot update the elements of loop.

Now, to extract the even numbers from a list, we can do :

```
a = [ 5 , 49 , 20 , 28 , 95 , 68 , 394 ]

b = []
for val in a:
    if val%2 == 0 :
        b.append(val)

print(b)
```

List comprehensions are a short way to achieve this.

Syntax :

```
new_list = [ expression(item) for item in old_list if condition(item) ]
```

Using list comprehensions, the above program can be written as :

```
a = [ 5 , 49 , 20 , 28 , 95 , 68 , 394 ]

b = [ val for val in a if val%2==0 ]

print(b)
```

The second line in code means : `b` is composed of every such `val` such that `val` is present in `a` and `val%2` is 0.

> **DIY** : Extract out the palindromic items from a list of strings

Another usage of list comprehensions is with `range()`

```
a = [ p for p in range(5,20) ]
print(a)
```

The variable `a` will be populated with every such `p` which lies in the range of 5 to 20 (20 exclusive)

> **DIY** : Find the first 10 powers of 2

Application : Finding pending assignments

Two lists are given : one containing all the due assignments and another containing those which you have completed. Find the ones which are still pending.

```
assignments = [ "CS101" , "MA102" , "PH102" , "BT101" , "ME101" , "EE110" ]
completed = [ "PH102" , "CS101" ]

pending = [ item for item in assignments if item not in completed ]

print(pending)
```

Python lecture series - Lecture 4

> **Topics :**
>
> - Tuples
> - zip function
> - Sets
> - Dictionary
> - Array of dictionary
> - Comprehensions with dictionary
> - Determining datatype of variable
>
> **Application programs :**
>
> - Adding 2 matrices
> - Frequency analysis in array
> - Update course registration

Tuples

Used to represent objects with 2 or more attributes. Eg : Points in 2-D plane are tuples of form (x,y)

Just like arrays, we use `p[i]` to access ith index of a tuple `p` . However, a tuple element cannot be re-assigned.

The following program takes as input a list of 3-D points each of which is stored as tuple :

```
n = int(input("Enter the number of points : "))

points = []
print("Enter 3 decimals separated by space for each point")

for i in range(n):
    print("Coordinates of point",i+1,":",end=" ")
    x,y,z = map(float,input().split())
    p = (x,y,z)
    points.append(p)

print(points)
```

---

zip function

`zip()` function iterates through elements of passed array one by one and returns them as a tuple.

```
a = [ 5, 6, 8, 9, 1, 0 ]
b = [ 8, 2, 10, 0, 5, 7 ]

ans = list( zip(a,b) )
print(ans)              # [(5, 8), (6, 2), (8, 10), (9, 0), (1, 5), (0, 7)]
```

If the lists given to zip function are of different sizes, the shortest list will determine how many elements are to be considered.

```
a = [ 5, 6, 8, 9, 1, 0 ]
b = [ 9, 2, 4, 6 ]
c = [ 8, 2, 10, 0, 5 ]

ans = list( zip(a,b,c) )
print(ans)             # [(5, 9, 8), (6, 2, 2), (8, 4, 10), (9, 6, 0)]
```

Application : Adding 2 matrices

We will deal with 2 dimensional array here i.e list of lists.

To add the matrices, loop through the row numbers and in each row, loop through the column numbers and add the elements from both matrices at that position.

```
a = [ [ 3, -5, -7, 9 ],
      [ 13, 0, -2, 1 ],
      [ -9, 8, 3, -1 ] ]

b = [ [ 0, -5, 10, 6 ],
      [ 6, 8, 13, -4 ],
      [ 8, 9, -7, -1 ] ]

# First check if dimensions of a and b are same
# For this, see len() of both to check no of rows
# Then, loop through rows and compare len() of each row

r = len(a)                              # No of rows
c = len(a[0])                           # No of cols

ans = []                                # Start with empty list as result

for i in range(r):
    ans.append([])                      # Append new row in c
    for j in range(c):
        ans[i].append(a[i][j]+b[i][j])  # Append element in current row of c

print(ans)
```

Further, using list comprehensions, we can write the above code as :

```
# Declare a and b here and check their dimensions

r = len(a)                              # No of rows
c = len(a[0])                           # No of cols

ans = []                                # Start with empty list as result

for i in range(r):
    ans.append( [ x+y for (x,y) in zip(a[i],b[i]) ] )  # Add entire ith row of a and b

print(ans)
```

Sets

List, tuple and set can be converted from each other using `list()` , `tuple()` and `set()` function.

- A new set can be made as :

```
rolls = set({2,4,7,2})    # Duplicate values will be removed
print(rolls)              # {2, 4, 7}
rolls2 = set()            # Make empty set
```

- Sets do not have index to access element. Instead, to go through its elements, we can use the following syntax :

```
for x in rolls:
    print(x)
```

- `add()` , `remove()` , `len()` functions have their usual meaning.
- `clear()` function removes all elements from set. This function is also available for lists.
- Searching in list, tuple and set can be done in the same way :

```
if val in a :
    print("yes")
else:
    print("no")
```

- Various set operations can performed using intuitive symbols or by using functions :

```
rolls1 = {2,4,7,6,5,8}
rolls2 = {1,6,8,3,9,7}

rolls_uni = rolls1 | rolls2
rolls_int = rolls1 & rolls2
rolls_xor = rolls1 ^ rolls2
rolls_1d2 = rolls1 - rolls2
rolls_2d1 = rolls2 - rolls1

print(rolls_uni)       # {1, 2, 3, 4, 5, 6, 7, 8, 9}
print(rolls_int)       # {8, 6, 7}
print(rolls_xor)       # {1, 2, 3, 4, 5, 9}
print(rolls_1d2)       # {2, 4, 5}
print(rolls_2d1)       # {1, 3, 9}
```

- Sets can be compared similarly :

```
rolls1 = {1,3,5,6}
rolls2 = {1,5}
rolls3 = {2,4}

if rolls2 <= rolls1 :
    print("rolls2 is subset of rolls1")

if rolls1.isdisjoint(rolls3) :
    print("disjoint sets")
```

Dictionary

In arrays, numbers are used for indexing. Dictionary can be thought as an array where anything can be used for indexing.

```
student = {}
student["name"] = "ABC"
student["year"] = 1
student["cpi"] = 9.0
student["dept"] = "CSE"
student["courses"] = [ "CS101" , "MA102" , "CS110" ]
student["gratuated"] = False

print(student)
print(student["courses"])
```

The second way of creating a dictionary is :

```
glossary = {
    "word1" : "meaning1",
    "word2" : "meaning2",
    "word3" : "meaning3",
    "word4" : "meaning4",
    "word5" : "meaning5"
}

print(glossary)
print(glossary["word3"])
```

Iterating through dictionary

Let's see what the following code gives when run on a dictionary:

```
for k in student:
    print(k)
```

We see that this gives all the keys (index names) of the dictionary.

Also, we have already seen that the value corresponding to key `k` in a dictionary `student` can be accessed by `student[k]`

Combining these, we use the following code :

```
for k in student:
    print(k,"=",student[k])
```

Also, if you want to avoid using `student[k]` , there is another option:

```
for k,v in student.items():
    print(k,"=",v)
```

Adding key to dictionary

We already saw this above. When writing `student[k] = something` , if the the key `k` is not there in the dictionary, it will be created otherwise the value for that key will get updated.

Checking if a key exists in dictionary

```
if "cpi" in student:
    print("Student dictionary contains cpi key")
else:
    print("Student dictionary does not contain cpi key")
```

Deleting key from dictionary

```
if "cpi" in student:
    del student["cpi"]
```

OR

```
student.pop("cpi, None)
```

Application : Element frequency in array

This can be done very easily using a dictionary. Start with an empty dictionary and add elements of list as a key in the dictionary with value 1 which denotes its frequency. In case the element was alreayd present in the dictionary, just increment the value of corresponding key.

```
a = list(map(int,input.split()))     # Input the list of numbers

d = {}                               # Start with empty dictionary
for x in a:
    if x not in d:                   # If this element occures first time
        d[x] = 1                     # Add new key to dictionary with value 1
    else:                            # If this element has already occured
        d[x] += 1                    # Increase count of this element

print(d)
```

Array of dictionary

A python list can contain any type of element: Number, String, Boolean, list . . .

Similarly, we can have a dictionary as an element of an array.

```
contacts = [
    {
        "name1" : 9999999999,
        "name2" : 9999999999,
        "name3" : 9999999999
    },
    {
        "name4" : 9999999999,
        "name2" : 9999999999
    },
    {
        "name3" : 9999999999,
        "name5" : 9999999999,
        "name6" : 9999999999
    }
]
```

Below is an example showing dictionary of dictionary :

```
contacts = {
    "school" : {
        "name1" : 9999999999,
        "name2" : 9999999999,
        "name3" : 9999999999
    },
    "college" : {
        "name4" : 9999999999,
        "name2" : 9999999999
    },
    "work" : {
        "name3" : 9999999999,
        "name5" : 9999999999,
        "name6" : 9999999999
    }
}
```

Comprehensions with dictionary

Given a list of dictionaries where each dictionary stores details of a student, get all the names.

```
data = [
    {
        "name" : "name1",
        "year" : 1,
        "cpi"  : 8.5,
        "dept" : "ECE"
    },
    {
        "name" : "name2",
        "year" : 2,
        "cpi"  : 8.2,
        "dept" : "MNC"
    },
    {
        "name" : "name2",
        "year" : 3,
        "cpi"  : 8.1,
        "dept" : "CSE"
    }
]

names = [ x["name"] for x in data ]
print(names)
```

Application : Update course registration

Consider the following task :

There is a dictionary of students where roll no is used as key and we store the list of courses which the student has completed as the value. The result of recent semester is given in the form of dictionary which has course no as key and the list of students who have passed that course as the value. Update the list of completed course list for each student by looking the new courses in which he has passed recently.

```
students = {
    "140101001" : [ "CS101" , "EE101" , "CS201" ],
    "140102001" : [ "MA101" , "EE101" , "EE201" ],
    "140103001" : [ "CS101" , "ME101" ],
    "140123001" : [ "MA101" , "CS101" , "MA201" ]
}

result = {
    "MA321" : [ "140101001" , "140123001" ],
    "EE301" : [ "140102001" , "140103001" ],
    "ME301" : [ "140103001" ]
}

for course in result:
    for roll in result[course]:
        students[roll].append(course)

print(students)
```

Determining datatype of variable

Pass the variable in `type()` function to get its type. To check if the variable is of certain type, use `isinstance()` function

```python
a = 9
b = [1,2,3]
c = '78'
d = 'qwerty'

print( type(a) )
print( type(b) )

if type(c)==type(d):
    print("same type")
else:
    print("different type")

if isinstance(b,list):
    print("b is list type")
```

Python lecture series - Lecture 5

> **Topics :**
>
> - File input/output
> - JSON data file
> - pip
> - CSV files
>
> **Application programs :**
>
> - Sort lines in file
> - Find article title
> - Contest Leaderboard

> **Note :** Download the Lecture-5 folder from this link for the data files used in this lecture.

File input/output

```
f = open("sample.txt","r")
```

This enables reading `sample.txt` file with the variable `f` . The string `"r"` stands for reading mode.

To write to file, use `"w"` instead of `"r"` . Note that this overwrites the entire content of file.

Similarly, using `"a"` option will allow appending text to end of file.

After you are done reading/modifying the file, it must be closed :

```
f.close()
```

Reading from file

This line will copy entire content of file opened with `f` as a string to variable `s` :

```
s = f.read()
```

Or, to read line by line, use this syntax :

```
for s in f:
    print(s)
```

To read first 5 lines, this syntax is better :

```
for i in range(5):
    s = f.readline()
    print(s)
```

To directly get all the lines in an array, just use `readlines()`

```
a = f.readlines()
print(a)
```

> **Note :** Use `strip()` or `rstrip()` function to avoid newline coming in each line of file text

Writing to file

Use the following functions:

- `write(s)` : Writes string `s` to file.
- `writelines(a)` : Write each string in array `a` to file.

Appending to file

Appending is same as writing and uses same functions. Only, the previous contents of that file are not erased.

Application : Sort lines in files

You have some files say `list1.txt` , `list2.txt` , `list3.txt` containing some numbers. Merge the contents of these files into a single file with the data sorted.

```
f1 = open("rolls/list1.txt","r")
f2 = open("rolls/list2.txt","r")
f3 = open("rolls/list3.txt","r")
fw = open("rolls/result.txt","w")
# Read content from files
l1 = f1.readlines()
l2 = f2.readlines()
l3 = f3.readlines()
# Merge lines
l = l1 + l2 + l3
# Convert to integer array
l = [ int(x.strip()) for x in l ]
# Sort
l.sort()
# Convert back to string form
l = [ str(x)+"\n" for x in l ]
# Write to result file
fw.writelines(l)
# Close files
f1.close()
f2.close()
f3.close()
fw.close()
```

Note : To avoid closing file explicitly, use the following syntax :

```
with open("sample.txt","r") as f:
    s = f.read()

# Rest program
```

Application : Find article title

You have some text files containing article on some topic. The files are not labelled and you have to determine the topic on which each file is.

We can solve this by finding the most important word in each file i.e., the most frequently occuring word. However, words like `the` , `is` etc will occur more than the required word. This will give wrong result. But, such common words will occur in each file whereas, the required title word will not be occuring frequently in other files.

So, one way to assign importance of a word is : `frequency in particular file / total frequency in each file`

Then, the word with highest importance will give us the title for article.

This basic formula can improved further to give correct result in even more situations.

```
def main():
    # Read content from files
    with open("articles/wiki1.txt","r") as f1:
        text1 = f1.read()
    with open("articles/wiki2.txt","r") as f2:
        text2 = f2.read()
    with open("articles/wiki3.txt","r") as f3:
        text3 = f3.read()
    # Split text to get words
    words1 = text1.split()
    words2 = text2.split()
    words3 = text3.split()
    # Clean the words and convert to lower case
    words1 = [ (x.lower()).strip(",-\"\'().") for x in words1 ]
    words2 = [ (x.lower()).strip(",-\"\'().") for x in words2 ]
    words3 = [ (x.lower()).strip(",-\"\'().") for x in words3 ]
    # Get frequency dictionary of above lists
    d1 = frequency(words1)
    d2 = frequency(words2)
    d3 = frequency(words3)
    # List of these dictionaries
    dlist = [d1, d2, d3]

    # See words in sorted by frequency
    # print(sorted(d1,key=d1.__getitem__))
    # print(sorted(d2,key=d2.__getitem__))
    # print(sorted(d3,key=d3.__getitem__))

    # Use the function to get answer
    print(mostImp(d1,dlist))
    print(mostImp(d2,dlist))
    print(mostImp(d3,dlist))

# Function to get frequency dictionary for an array of words
def frequency(wordlist):
    d = {}
    for x in wordlist:
        if x in d:
            d[x] += 1
        else:
            d[x] = 1
    return d

# Function to calculate importance of a word in a particular document
def importance(word, d, dlist):
    freq_doc = d[word]
    freq_total = 0
    for x in dlist:
        if word in x:
            freq_total += x[word]
    return freq_doc / freq_total

# Function to find most important word in a document
def mostImp(d, dlist):
    max_score = 0
    max_doc_freq = 0
    for word in d:
        doc_freq = d[word]
        score = importance(word, d, dlist)
        if (score>max_score) or (score==max_score and doc_freq>max_doc_freq):
            ans = word
            max_score = score
            max_doc_freq = doc_freq
    return ans

main()
```

JSON data file

We can store data in various ways. There are various database softwares to choose from.

But for simple and small data, we can simply use files. Even a text file with a fixed pattern might do.

Also, when making web applications, we need to transfer data in a proper format from the server to the user's device. JSON is the most popular format for this currently. JSON files (`.json`) are also used for various purpose.

JSON data looks pretty much like a nested combination of Python dictionary and lists. See the JSON files in the folder whose link is given above.

Here's how to read data from a JSON file in Python:

```
import json

f = open("sample.json",'r')
data = json.load(f)
f.close()

print(data)
# Or to print in a clean format, use the below line
# print( json.dumps(data,indent=2) )
```

or

```python
import json

with open("sample.json",'r') as f:
    data = json.load(f)

print(data)
# Or to print in a clean format, use the below line
# print( json.dumps(data,indent=2) )
```

Application : Contest Leaderboard

Let's store records of users participating in a contest using JSON file.

We need to provide functionality to add more users, remove users and display the leaderboard.

```python
import json

filename = "json/sample.json"

def main():
    choice = 1
    while choice>0:
        print("0) Exit")
        print("1) Add record")
        print("2) Delete record")
        print("3) Show leaderbord")
        choice = int(input("Enter choice : "))
        print("")
        if choice==1:
            addRecord()
        elif choice==2:
            deleteRecord()
        elif choice==3:
            showLeaderboard()
        print("")

def addRecord():
    with open(filename,"r") as f:
        data = json.load(f)
    username = input("Enter username : ")
    name = input("Enter name : ")
    age = int(input("Enter age : "))
    score = int(input("Enter score : "))
    d = {
        "username" : username,
        "name" : name,
        "age" : age,
            "score" : score
        }
        data[username] = d
        with open(filename,"w") as f:
            f.write(json.dumps(data,indent=4))


def deleteRecord():
    with open(filename,"r") as f:
        data = json.load(f)
    username = input("Enter username to delete : ")
    if username in data:
        del data[username]
        with open(filename,"w") as f:
            f.write(json.dumps(data,indent=4))
    else:
        print("Username not found")

def showLeaderboard():
    with open(filename,"r") as f:
        data = json.load(f)
    profiles = [v for k,v in data.items()]
    profiles.sort(key=lambda x: x["score"],reverse=True)
    for x in profiles:
        print(x["name"],x["score"])

main()
```

pip

pip is the package manager for Python.

We have seen some modules like `math` and `json` which are built-in with Python.

There are many such modules which do not come with Python and need to be installed separately.

pip enables us to install these modules/libraries easily.

Install pip in your system using this command in terminal:

```
sudo apt-get install python3-pip
```

Check the version of pip using `pip --version`

> **Note :** If `pip` corresponds to python2 in your system or does not exist, use `pip3` . e.g. - `pip3 --version`

To install the python module `csv`, use the following command in terminal:

```
pip install csv
```

To use pip behind proxy, use :

```
pip install --proxy=user:pass@host:port csv
```

> **Note :** Learn to use virtual environments once comfortable with pip stuff

CSV files

To store data in tabular form, CSV is the simplest way.

CSV file is like simplifed excel sheet. It can be opened in any software which is used for excel sheets.

Each line is a row and we use commas(,) to separate columns in a row.

See the CSV files in the folder whose link is given above.

Read data from CSV file

```
import csv

filename = "csv/credits.csv"

f = open(filename,'rt')
reader = csv.reader(f)

for row in reader:
    print(row)

f.close()
```

Write data to CSV file

```
import csv

filename = "csv/new.csv"

data = [
    [1,2,3],
    [4,5,6]
]

f = open(filename,'w')
writer = csv.writer(f)

for row in data:
    writer.writerow(row)

f.close()
```

Python lecture series - Practice problems

1) Fibonacci numbers

Make a program which takes an integer `n` as input and prints the first `n` fibonacci numbers.

Sample input :

```
10
```

Sample output :

```
0
1
1
2
3
5
8
13
21
34
```

2) Print the abbreviation

Input a list of names/terms and print the initials/abbreviations of each.

Sample input :

```
New York
Shah Rukh Khan
Counter Strike : Global Offensive
Blue Screen of Death
```

Sample output :

```
NY
SRK
CS:GO
BSoD
```

Note : Try to shorten code using list comprehensions and mapping

3) GCD

Make a function which takes 2 integers and returns their GCD.

You must have done Euclid's method to find GCD in mathematics. Use that method.

The function can be written in 2 different ways :

- Recursion
- Using while loop

4) Commas in number

Input a list of numbers and display them in international format (with a comma after every 3 digits).

You can either use loop or to reduce code, use recursion and splicing in your function.

Sample input :

```
12345
165616
656
1589
89494942942
```

Sample output :

```
12,345
165,616
656
1,589
89,494,942,942
```

5) Frequency of each element of list

Make a function which takes a list and displays the count of each distinct element in that list.

There are multiple approaches to this problem. One of the simple approaches is to sort the list and loop through the elements. Even simpler way is to use dictionary to keep count of elements.

Sample input :

```
7 1 9 3 7 2 8 9 3
```

Sample output :

```
1 : 1
2 : 1
3 : 2
7 : 2
8 : 1
9 : 2
```