**Python**

Python is a general purpose interpreted, Object-oriented and high-level programming language.

Advantages :-

* Open Source and community development
* Easy to learn
* Python allow you to split your program into modules that can be reused in another Python program.
* High level data types allow you to express complex operations in a single statement
* Statement grouping is done by indentation instead of beginning and ending brackets
* No variable or argument declarations are necessary
* The biggest strength of Python is huge collection of standard library which can be used for the following:
* Machine Learning
* GUI Applications (like Kivy, Tkinter, PyQt etc. )
* Web frameworks like Django (used by YouTube, Instagram, Dropbox)
* Image processing (like OpenCV, Pillow)
* Web scraping (like Scrapy, BeautifulSoup, Selenium)
* Test frameworks
* Multimedia
* Scientific computing
* Text processing and many more..

Keys points :-

**Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP

**Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

**Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Comments :-

Single Line Comment

Python single line comment starts with hashtag symbol with no white spaces (#) and lasts till the end of the line.

Multi Line Comment

Python multi-line comment is a piece of text enclosed in a delimiter (""") on each end of the comment

Singleline Comment – Syntax -> #

Example :-

# Single Line Comment in Python

Multiline Comment – Syntax -> ‘’’ ‘’’ OR “”” “””

Example :-

'''  
Multi  
line  
comme  
'''

Escape Sequence :-

Example :-

print('Hello World', end=' and ')  
print('Welcome') *# Output -> Hello World and Welcome*

Typecasting :-

We can change data type with method

Example :-

i = "10"print(int(i))

F-Strings :-

a = 'our'  
b = 'World'  
print(f'Welcome to {a} {b}') *# Output -> Welcome to our World*

Namespace :-

A namespace is a system to have a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary.

Namespace is an area where you create and store object/variable

**Types of Namespaces**

Built-in Namespace, Global Namespace, Local Namespace

Example :

# var1 is in the global namespace

var1 = 5

def some\_func():

# var2 is in the local namespace

var2 = 6

def some\_inner\_func():

# var3 is in the nested local

# namespace

var3 = 7

Statement :-

Instructions written in the source code for execution are called statements. There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These all help the user to get the required output.

Multi-line Statement

Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\). When the programmer needs to do long calculations and cannot fit his statements into one line, one can make use of these characters.

Example :

Declared using Continuation Character (\):

s = 1 + 2 + 3 + \

4 + 5 + 6 + \

7 + 8 + 9

Declared using parentheses () :

n = (1 \* 2 \* 3 + 7 + 8 + 9)

Declared using square brackets [] :

footballer = ['MESSI',

'NEYMAR',

'SUAREZ']

Declared using braces {} :

x = {1 + 2 + 3 + 4 + 5 + 6 +

7 + 8 + 9}

Declared using semicolons(;) :

flag = 2; ropes = 3; pole = 4

Input :-

Python provides us with inbuilt functions to read the input from the keyboard.

Example : input(“Enter a no”)

Taking multiple inputs from users

Python user can get multiple inputs or values in one line by two methods

Using split() method

This function helps in getting a multiple inputs from user . It breaks the given input by the specified separator. If separator is not provided then any white space is a separator. Generally, user use a split() method to split a Python string but one can used it in taking multiple input.

Syntax : input().split(separator, maxsplit)

Example :

# taking two inputs at a time

x, y = input("Enter a two value: ").split() #split(‘,’) then value separated by ( , )

print("Number of boys: ", x)

print("Number of girls: ", y)

Using List comprehension

Example :

# taking two input at a time

x, y = [int(x) for x in input("Enter two value: ").split()]

print("First Number is: ", x)

print("Second Number is: ", y)

Output :-

The simplest way to produce output is using the print() function where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string before writing to the screen.

Syntax : print()

Python end parameter in print()

Example :

# ends the output with a <space>

print("Welcome to" , end = ' ')

print("GeeksforGeeks", end = ' ') # Output -> Welcome to GeeksforGeeks

Python sep parameter in print()

Example :

#code for disabling the softspace feature

print('G','F','G', sep='') # Output -> GFG

#for formatting a date

print('09','12','2016', sep='-') # Output -> 09-12-2016

#another example

print('pratik','geeksforgeeks', sep='@') # Output -> pratik@'geeksforgeeks

Data Types :-**1 - Number Data Type**x = 10 *# Int*x = 10.2 *# Float*x = 10 > 3 *# Bool*x = 10j *# Complex***2 - String Data Type** *# String define in " " OR '*

*# Strings in Python can be created using single quotes or double quotes or even triple quotes.  
# Individual characters of a String can be accessed by using the method of Indexing.  
# In Python, Updation or deletion of characters from a String is not allowed. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.'*x = 'Hello' *# Str*x = "Hello" *# Str*print(x) *# Print string. Output -> Hello*print(len(x)) *# len(variable\_name) for length of string. Output -> 5*print(type(x)) *# type(variable\_name) for type of variable. Output -> <class 'str'>*print(x[2]) *# show string letter using index. Output -> l*print(x[2:4]) *# x[index\_start\_point : index\_End\_point].  
# x[-1] for last index in a string x[-2] for second last index of string.... so on  
# Output -> ll  
# x[2] = 'p' String cannot change. Output -> 'str' object does not support item assignment*print(x.upper()) *# string convert into Uppercase. Output -> HELLO*print(x.lower()) *# String convert into Lowercase. Output -> hello*print(x.find('wor')) *# Find the index position of letter or string in the variable. Output -> 6*

*# Formatting of String  
# Default order*String1 = "{} {} {}".format('Geeks', 'For', 'Life')  
print(String1) *# Output -> Geeks For Life  
# Positional Formatting*String1 = "{1} {0} {2}".format('Geeks', 'For', 'Life')  
print(String1) *# Output -> For Geeks Life  
# Keyword Formatting*String1 = "{l} {f} {g}".format(g='Geeks', f='For', l='Life')  
print(String1) *# Output -> Life For Geeks*

*# String alignment*String1 = "|{:<10}|{:^10}|{:>10}|".format('Geeks', 'for', 'Geeks')  
print(String1) *# Output -> |Geeks | for | Geeks|  
# To demonstrate aligning of spaces*String1 = "\n{0:^16} was founded in {1:<4}! {2:>10}".format("GeeksforGeeks", 2009, 22222)  
print(String1) *# Output -> GeeksforGeeks was founded in 2009! 22222*

*# show alternate character*i = 'Hello world'  
print(i[0:16:2]) *# Output -> Hlowrd*

*# Reverse string*xx = "Hello World"  
*# After second colon number is negative then it return reverse string*print(xx[::-1]) *# Output -> dlrow olleH*print(xx[::-2]) *# Output -> drWolH   
# In above eg. first reverse string then return alternate character*

print(xx.endswith('ld')) *# Output -> True*print(xx.replace('World', 'Sandip')) *# Output -> Hello Sandip*

**3 - List Data Type** *# Ordered, can be changed. Duplicate entries are present*

*# A single list may contain DataTypes like Integers, Strings, as well as Objects.*

*# Lists are mutable, and hence, they can be altered even after their creation.*

*# append() method - only works for addition of elements at the end of the List.  
# insert() method - for addition of element at the desired position.  
# extend() method - add multiple elements at the same time at the end of the list.*

*# remove() method - Elements can be removed from the List by using built-in remove() function but an Error arises if element doesn’t exist in the set. Remove() method only removes one element at a time, to remove range of elements, iterator is used. The remove() method removes the specified item.*

*# pop method - Pop() function can also be used to remove and return an element from the set, but by default it removes only the last element of the set, to remove element from a specific position of the List, index of the element is passed as an argument to the pop() method.*

*# We can merge all Data Types in a list*x = ['John', 'Sam', 'Andrew', 'John']  
x = [1, 3, 4, 6, 77, 3, 5, 6, 'John', "Smith", {3, 6, 7}]  
x = [1, 3, 5, 6, 43j, {3, 56, 7, 7}]  
x.append('Sam') *# append for add the value in the last index*print(x) *# Output -> [1, 3, 5, 6, 43j, {56, 3, 7}, 'Sam']*x.insert(2, "Sss") *# variable\_name.insert(Index, add\_value). Add the value on mention index in the list*print(x) *# Output -> [1, 3, 'Sss', 5, 6, 43j, {56, 3, 7}, 'Sam']*

x.extend([8, 'Geeks', 'Always'])  
print(x) *# Output -> [1, 3, 'Sss', 5, 6, 43j, {56, 3, 7}, 'Sam', 8, 'Geeks', 'Always']*

x.remove('Sam')  
print(x) *# Output -> [1, 3, 'Sss', 5, 6, 43j, {56, 3, 7}, 8, 'Geeks', 'Always']*

x.pop(5)  
print(x) *# Output -> [1, 3, 'Sss', 5, 6, {56, 3, 7}, 'Sam', 8, 'Geeks', 'Always']*x.reverse() *# reverse the list*print(x) *# Output -> ['Sam', {56, 3, 7}, 43j, 6, 5, 'Sss', 3, 1]*

**4 - Dictionary Data Type** *# Unordered, can be changed. No duplicate entries are present.  
# In that Key Value pair. Key can not be duplicate and Value can be duplicate. Key and Value can be any Data Type*x = {  
 1: 'John',  
 2: 'Smith',  
 'third': 232,  
 'John': [53, 43]  
}  
print(x) *# Output -> {1: 'John', 2: 'Smith', 'third': 232, 'John': [53, 43]}*print(x['John']) *# We can access the value using Key. Output -> [53, 43]*x['third'] = [233, 724] *# Update the value using Key*print(x) *# Output -> {1: 'John', 2: 'Smith', 'third': [233, 724], 'John': [53, 43]}***5 - Tuple Data Type** *# Ordered , can not be changed. Duplicate entries are present.*x = (3, 6, 7, 3, 8, 3, 8, 356, 8, 'Smith')  
print(x.count(3)) *# count function show how many time value present in Tuple***6 - Set Data Type** *# Unordered. No Duplicate entries are present.  
# Collection of List and Dictionary.*x = {4, 153, 83, 7, 2, 7, 'Sam', 'John'}  
*# x[2] Set does not support indexing***---- All Data Types ----**x = 10 *# Int*x = 10.2 *# Float*x = 10 > 3 *# Bool*x = 10j *# Complex*x = **'Hello'** *# Str*x = **"Hello"** *# Str*x = [1, 3, 4, 6, 77, 3, 5, 6, **'John'**, **"Smith"**, {3, 6, 7}] *# List*x = {  
 1: **'John'**,  
 2: **'Smith'**,  
 **'third'**: 232,  
 **'John'**: [53, 43]  
} *# Dictionary*x = (3, 6, 7, 3, 8, 3, 8, 356, 8, **'Smith'**) *# Tuple*x = {4, 153, 83, 7, 2, 7, **'Sam'**, **'John'**} *# Set*

Array :-

*# Python Arrays and lists have the same way of storing data.  
# Arrays take only a single data type elements but lists can have any type of data.  
# Therefore, other than a few operations, the kind of operations performed on them are different.  
# Array are muttable  
  
# Create an array in 3 ways  
# 1st method to create an Array*import array  
i = array.array(**'i'**,[2,4,6,7,8,3])  
print(i) *# Output -> array('i', [2, 4, 6, 7, 8, 3])*print(type(i)) *# Output -> <class 'array.array'>  
  
# 2nd method to create an Array*import array as arr  
i = arr.array(**'d'**,[2.0,4.3,3.2])  
print(i) *# Output -> array('d', [2.0, 4.3, 3.2])*print(type(i)) *# Output -> <class 'array.array'>  
  
# 3rd method to create an Array*from array import \*  
i = array(**'i'**, [2,6,3,7,4])  
print(i) *# Output -> array('i', [2, 6, 3, 7, 4])*print(type(i)) *# Output -> <class 'array.array'>  
  
# Accessing array elements*from array import \*  
i = array(**'i'**, [2,6,3,7,4])  
print(i[2]) *# Output -> 3  
  
# Array operation  
# 1 - Finding length of Array*import array as arr  
i = arr.array(**'d'**, [2.3,3.5,5.8,2.0])  
print(len(i)) *# Output -> 4  
  
# 2 - Adding element to an Array  
# append() -> Used when you want to add a single element at the end of an array  
# extend() -> Used when you want to add more than one element at the end of an array  
# insert() -> Used when you want to add an element at a specific position in an array*import array as arr  
i = arr.array(**'i'**, [2,4,5,7,6])  
i.append(33333)  
print(i) *# Output -> array('i', [2, 4, 5, 7, 6, 33333])*i.extend([9,5,333,6,42])  
print(i) *# Output -> array('i', [2, 4, 5, 7, 6, 33333, 9, 5, 333, 6, 42])*i.insert(3,431)  
print(i) *# Output -> array('i', [2, 4, 5, 431, 7, 6, 33333, 9, 5, 333, 6, 42])  
  
# 3 - Removing elements of an Array  
# pop() -> Used when you want to remove an element and return it  
# remove() -> Used when you want to remove an element with a specific value without returning it*import array as arr  
i = arr.array(**'i'**, [1,2,4,5,7,6])  
print(i.pop()) *# Output -> 6*print(i) *# Output -> array('i', [1, 2, 4, 5, 7])*print(i.pop(2)) *# Output -> 4*print(i) *# Output -> array('i', [1, 2, 5, 7])*i.remove(2) *# put the value which is you want to remove*print(i) *# Output -> array('i', [1, 5, 7])  
  
# 4 - Array Concatenation*import array as arr  
i = arr.array(**'i'**, [1,2,4,5,7,6])  
p = arr.array(**'i'**, [99,98,97,96])  
q = i + p  
print(q) *# Output -> array('i', [1, 2, 4, 5, 7, 6, 99, 98, 97, 96])  
  
  
# 5 - SLicing an Array*import array as arr  
i = arr.array(**'i'**, [1,2,4,5,7,6])  
print(i[0:4]) *# Output -> array('i', [1, 2, 4, 5])  
  
# 6 - For loop*import array as arr  
i = arr.array(**'i'**, [1,2,55,4,5,7,6])  
for x in i[1:3]:  
 print(x) *# Output -> 2 55. Or show all value then use i[1:3] replace with i  
  
# 7 - While loop*import array as arr  
i = arr.array(**'i'**, [1,2,55,4,5,7,6])  
b = 0 *# initialize the iterator*while b < i[1]: *# all value show using len(i)* print(i[b]) *# Output -> 1 2* b = b + 1 *# b += 1*

Hash Table and Hashmap :-

*# Hash table or a Hashmap is a type of data structure that maps keys to its value pairs  
# create dictionary in 2 method  
# Method 1*my\_dect = {**'John'**: **'001'**, **'Smith'**: **'002'**, **'Andrew'**: **'003'**}  
print(my\_dect) *# Output -> {'John': '001', 'Smith': '002', 'Andrew': '003'}*print(type(my\_dect)) *# Output -> <class 'dict'>  
  
# Method 2*new\_dects = dict(Smith=**'001'**,John=**'002'**)  
print(new\_dects) *# Output -> {'Smith': '001', 'John': '002'}*print(type(new\_dects)) *# Output -> <class 'dict'>  
  
# Nested Dictionary*emp\_details= {**'Employee'**: {**'John'**:{ **'ID'**: **'001'**, **'Salary'**: **'1000'**, **'Designation'**: **'TL'**},  
 **'Smith'**:{ **'ID'**: **'002'**, **'Salary'**: **'2000'**, **'Designation'**: **'Analyst'**},  
 }  
 }  
print(emp\_details)  
print(type(emp\_details)) *# Output -> <class 'dict'>  
  
# Operation on Hash table  
# Accessing values*print(my\_dect[**'John'**]) *# Output -> 001*print(my\_dect.keys()) *# Output -> dict\_keys(['John', 'Smith', 'Andrew'])*print(my\_dect.values()) *# Output -> dict\_values(['001', '002', '003'])*print(my\_dect.get(**'Smith'**)) *# Output -> 002*for x in my\_dect:  
 print(x) *# Output -> John Smith Andrew*for x in my\_dect.values():  
 print(x) *# Output -> 001 002 003*for x,y in my\_dect.items():  
 print(x,y) *# Output -> John 001 Smith 002 Andrew 003  
  
# Updating values*my\_dect[**'John'**] = **'004'**my\_dect[**'Kris'**] = **'005'**print(my\_dect) *# Output -> {'John': '004', 'Smith': '002', 'Andrew': '003', 'Kris': '005'}  
  
# Deleting*print(my\_dect.pop(**'Smith'**)) *# Output -> 002*print(my\_dect) *# Output -> {'John': '004', 'Andrew': '003', 'Kris': '005'}*print(my\_dect.popitem()) *# Output -> ('Kris', '005'). popitem() delete last key value pair*print(my\_dect) *# Output -> {'John': '004', 'Andrew': '003'}*del my\_dect[**'Andrew'**]  
print(my\_dect) *# Output -> {'John': '004'}  
  
# Converting Dictionary into a Dataframe  
# DataFrame is a 2-D data structure that consist of columns of various types.  
# It is very similar to a Python Dictionary and you can even convert into a pandas dataframe*import pandas as pd  
df = pd.DataFrame(emp\_details[**'Employee'**])  
print(df)  
*# Output ->  
# John Smith  
# ID 001 002  
# Salary 1000 2000  
# Designation TL Analyst*

Python Operator :-

Arithmetic Operator (+, -, \*, /, //, \*\*, %)

Relational Operator (>, <, ==, !=, >=, <=)

Logical Operator (and, or, not)

Bitwise Operator (&, |, ~, …)

Assignment Operator (+, +=, -=, ….)

Special Operator

**1 – Identity Operator (is and is not)**

is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

is - True if the operands are identical

is not - True if the operands are not identical

# Examples of Identity operators

a1 = 3

b1 = 3

a2 = 'GeeksforGeeks'

b2 = 'GeeksforGeeks'

a3 = [1,2,3]

b3 = [1,2,3]

print(a1 is not b1) # Output -> False

print(a2 is b2) # Output -> True

**2 – Membership Operator**

in and not in are the membership operators; used to test whether a value or variable is in a sequence.

in - True if value is found in the sequence

not in - True if value is not found in the sequence

# Examples of Membership operator

x = 'Geeks for Geeks'

y = {3:'a',4:'b'}

print('G' in x) # Output -> True

print('geeks' not in x) # Output -> True

print('Geeks' not in x) # Output -> False

print(3 in y) # Output -> True

print('b' in y) # Output -> False

**3 – Arithmetic Operator**

# Arithmetic operators are used to perform arithmetic between variables  
# Addition(+), Subtraction(-), Multiplication(\*), Division(/), Modules(%), Exponentiation(\*\*), Floor Division(//)x = 10  
y = 20  
print(x + y) *# Output -> 30*print(x - y) *# Output -> -10*print(x \* y) *# Output -> 200*print(x / y) *# Output -> 0.5*print(x % y) *# Output -> 10*print(x \*\* y) *# Output -> 100000000000000000000*print(x // y) *# Output -> # 0*

**4 – Assignment Operator**

# Assignment operator are used to assign valuesx = 100  
x += 10 *# x = x + 10*print(x)  
x -= 10 *# x = x - 10*print(x)  
x \*= 10 *# x = x \* 10*print(x)  
x /= 10 *# x = x / 10*print(x)

**5 – Comparison Operator**

# Comparison operators are used to compare to values  
# Equal(==), Not Equal(!=), Greater Than(>), Less Than(<), Greater than OR Equal(>=), Less Than OR Equal(<=)

**6 – Logical Operator**

# Logical Operator - and, or, not  
# Logical operator are used to combine condition statements  
# Conditional statementval = 10  
num = 10  
if val == num:  
 print(**'Equal'**)  
elif val > num:  
 print(**'Greater'**)  
else:  
 print(**'Smaller'**)  
  
x = 10  
print(x > 10 and x > 5) *# Output -> False*print(x > 10 or x > 5) *# Output -> True*print(not(x > 10 or x > 5)) *# Output -> False*

**7- Bitwise Operator**

# Bitwise operator are used to compare binary operator  
# Bitwise AND(&) - Sets each bit to 1 if both bits are 1.  
# Bitwise OR(|) - Sets each bit to 1 if one of the bits is 1.  
# Bitwise XOR(^) - Sets each bit to 1 if only one of the bits is 1.  
# Bitwise NOT(~) - Inverts all bits.  
# Left Shift(<<) - Shift left by pushing in zeroes from the right and left the leftmost bits fell off  
# Right Shift(>>) - Shift left by pushing copies of the leftmost bit in from the left, and let the rightmost bit fall off  
print(10 & 12) *# Output -> 8. => 10 - 1010, 12 - 1100 => 1010 1100 => 1000 => 8*

**Operator overloading in Python**

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

Examples :

# + operator for different purposes.

print(1 + 2) # Output -> 3

# concatenate two strings

print("Geeks"+"For") # Output -> GeeksFor

# Product two numbers

print(3 \* 4) # Output -> 12

# Repeat the String

print("Geeks"\*4) # Output -> GeeksxGeeksGeeksGeeks

**Difference between == and is operator in Python**

The **==** operator compares the values of both the operands and checks for value equality. Whereas **is** operator checks whether both the operands refer to the same object or not.

Examples :

list1 = []

list2 = []

list3=list1

if (list1 == list2):

print("True")

else:

print("False")

if (list1 is list2):

print("True")

else:

print("False")

# Output -> True False

Output of the first if condition is “True” as both list1 and list2 are empty lists.

Second if condition shows “False” because two empty lists are at different memory locations.

Function :-

# Functions is used for code optimization

# Built-in and user define function can used

# Doc String is used for function description.  
  
a = 8  
b = 7  
x = sum((a, b)) #Built-in function  
print(x)  
  
def avg(a, b): # User define function  
 '''This is the avg function'''  
 c = (a + b)/2  
 return c  
result = avg(5,6)  
print(result) # Output -> 5.5  
print(avg.\_\_doc\_\_) # This is the Doc string. We can access for function discription.  
# Output -> This is the avg function

Range() function

# range(start,stop, stepsize)  
# range() - Default start value started from 0 and Stop value end with (stop\_value - 1)  
# if range(10) - one value then this is the Stop value  
# if range(2,10) - two value then this is the Start and Stop value  
# if range(0,10,2) - third value is stop value - 1  
for i in range(10):  
 print(i, end=**' '**) # Output -> 0 1 2 3 4 5 6 7 8 9  
print()  
for i in range(2,10):  
 print(i, end=**' '**) # Output -> 2 3 4 5 6 7 8 9  
print()  
for i in range(1,10,2): # Output -> 1 3 5 7 9  
 print(i, end=**' '**)

Try Except exception handling :-

print('Enter first number')  
i1 = input()  
print('Enter second number')  
i2 = input()  
try:  
 print('Sum', int(i1) + int(i2))  
except Exception as e:  
 print(e)  
  
print('This is very important line')  
'''Output  
Enter first number  
3  
Enter second number  
d  
invalid literal for int() with base 10: 'd'  
This is very important line **'''**

File IO Basics :-Two type of memory Volatile(RAM) and Nonvolatile(HardDisk)  
Mode

**"r"** - Open file for reading – Default

**"w"** - Open file for writing

**"x"** - Create file if not exists

**"a"** - Add more content to a file

**"t"** - Text mode – Default

**"b"** - Binary mode

**"+"** - Read and Write

# read() - All content in that file  
f = open(**"Notes.txt"**)  
content = f.read() # OR f.read(30) - display starting 30 character in that file  
#print(content)  
f.close()  
  
# readable() - True/False  
f = open(**"Notes.txt"**)  
content = f.readable()  
#print(content)  
f.close()  
  
# readline() - print first line in that file  
f = open(**"Notes.txt"**)  
content = f.readline()  
#print(content)  
f.close()  
  
# Create new file - write  
f = open(**"Notes1.txt"**, **'w'**) # 'w' - write mode for create file if not exists OR replace content with new content  
f.write(**'Hello'**)  
f.close()  
  
# Add content in a file  
f = open(**"Notes1.txt"**, **'a'**) # 'a' - append content in file  
content = f.write(**'**\n**Sir'**)  
print(content) # Return number of character in that file  
f.close()  
  
# Handle read and write both  
f = open(**'Notes1.txt'**,**'r+'**) # 'r+' for read and write mode  
print(f.read()) # read the content in a file  
c = f.write(**'**\n**Thank you'**) # add content in a file  
f.close()

# tell() and seek()  
f = open(**'Notes.txt'**)  
print(f.tell()) # print the starting index value  
f.seek(3) # print where we can start printing character on that line  
print(f.readline())  
f.close()  
  
# Using with block to open file  
# By using syntax we could not define close file syntax  
with open(**'Notes1.txt'**) as f:  
 a = f.read()  
 print(a)  
f = open(**'Notes1.txt'**)  
f.read()

Scope, Global Variables and Global Keyword :-  
# Local Variable - can not access out of the local scope  
# Global Variable - Global variable can not edit in local scope  
# Global Keyword - Global keyword used for edit global variable in local scope using 'global' keyword  
l = 10  
def function1(n):  
 global l # edit global variable in local scope using 'global' keyword  
 l = l + 5 # after global keyword define we can edit that global variable  
 print(n, **'I have printed'**)  
 print(l)  
print(l)  
function1(**'this is me'**)  
  
x = 20  
def print1():  
 x = 10  
 def print2():  
 global x  
 x = 5  
 print(**'local : '**,x) # Output -> 5  
 print(**'before calling print2 : '**,x) # Output -> 10  
 print2()  
 print(**'after calling print2 : '**,x) # Output -> 10  
print1()  
print(x) # Output -> 5

Recursion :-

# Recursive Vs Iterative  
# Factorial using Iterative method  
# n! = n \* n-1 \* n-2 \* n-3 \*......1  
# n! = n \* (n-1)!  
def factorial\_interative(n):  
 fact = 1  
 for i in range(n):  
 fact = fact \* (i+1)  
 return fact  
  
# Factorial using Recursive method  
def factorial\_recursive(n):  
 if n == 1:  
 return 1  
 else:  
 return n \* factorial\_recursive(n-1)  
  
  
print(**"Enter a number"**)  
get\_no = int(input())  
print(**"Result in factorial interative method : "**, factorial\_interative(get\_no))  
print(**"Result in factorial recursive method : "**, factorial\_recursive(get\_no))

Lambda function or anonymous function :-

# lambda function used for minimum code for define a function

minus = lambda x, y : x – y  
print(minus(50,8))  
*# OR normal function*def minus(x,y):  
 print(x - y)  
minus(50,8)  
  
*# sort function using def function*def a\_first(a):  
 return a[0] *# a[0] for first position sorting*a = [[2,55],[502,100],[56,21]]  
a.sort(key=a\_first)  
print(**'Sorting using def function :'**,a) *# Output -> Sorting using def function : [[2, 55], [56, 21], [502, 100]]  
  
# sort function using lambda function*a = [[2,55],[502,100],[56,21]]  
a.sort(key=lambda x: x[1]) *# x[1] for second position sorting*print(**'Sorting using lambda function :'**,a) *# Output -> Sorting using lambda function : [[56, 21], [2, 55], [502, 100]]*

\_args and \_kwargs :-

# In list or tuple value are added then retrieve by using \*args and \*\*kwargs  
# Sequence of argument are (normal\_var, \*args, \*\*kwargs)  
# \*args and \*\*kwargs are optional  
# We can define any keyword of \*args or \*\*kwargs  
# \*args and \*\*kwargs used for retrieve list or dictionary or tuple  
def fuargs(normal, \*args, \*\*kwargs):  
 """any list retrieve in \*args or \*\*kwargs it will convert to Tuple format"""  
 print(normal)  
 for item in args:  
 print(item)  
 print(**'**\n**Now I would like to introduce some of our skills'**)  
 for key, value in kwargs.items():  
 print(key, value)  
  
har = [**'Harry'**, **'Rohan'**, **'Sandip'**,**'Tom'**]  
normal = **'I am a normal argument and the students are : '**kws = {**'1'**: **'Python'**, **'2'**: **'Javascript'**, **'3'**: **'Flask'**}  
fuargs(normal, \*har, \*\*kws)

Time Module :-  
import time  
  
*# Calculate while loop execution time Vs for loop execution time*initial = time.time()  
print(initial)  
for i in range(45):  
 print(**f'**{i}**'**, end=**' '**)  
  
print(**'**\n**For loop time in '**, time.time() - initial, **'second'**)  
initial2 = time.time()  
i = 0  
while i < 45:  
 print(i, end=**' '**)  
 i += 1  
print(**'**\n**while loop time in '**, time.time() - initial2, **'second'**)  
  
*# Local time*locatime = time.asctime(time.localtime(time.time()))  
print(locatime) *# Output -> Tue Jul 21 14:11:55 2020*time.sleep(2) *# after 2second next code execute*print(**'Sleep function'**)

Enumerate Function :-

# In Enumerate function we can access index with value  
# A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers’ task by providing a built-in function enumerate() for this task.  
# Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.  
# Syntax - enumerate(iterable, start=0)  
  
i = [**'Sandip'**, **'Swapnil'**,**'John'**, **'Pratik'**, **'Ravi'**, **'Dipak'**, **'Amol'**]  
  
# We can access only odd number value using for loop  
x = 0  
for item in i:  
 if x%2 == 0:  
 print(**f'**{item} **is our team'**)  
 x += 1  
  
# We can access only odd number using enumerate function  
for index, item in enumerate(i):  
 if index%2 == 0:  
 print(**f'**{item} **is our team'**)  
  
  
l1 = [**"eat"**, **"sleep"**, **"repeat"**]  
s1 = **"geek"**# creating enumerate objects  
obj1 = enumerate(l1)  
obj2 = enumerate(s1)  
print(**"Return type:"**, type(obj1)) # Output -> Return type: <class 'enumerate'>  
print(list(enumerate(l1))) # Output -> [(0, 'eat'), (1, 'sleep'), (2, 'repeat')]  
# changing start index to 2 from 0  
print(list(enumerate(s1, 2))) # Output -> [(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]

\_\_name == ‘\_\_main\_\_’ :-  
# When we used any function from other files then used  
# tutorial24 and tutorial25 is a example of that accessing other file content  
# file name - tutorial25.py  
def printhar(strg):  
 return **f'String is** {strg}**'**def add(a, b):  
 return **f'Sum of** {a + b + 5}**'**if \_\_name\_\_ == **'\_\_main\_\_'**: # this code not execute in other import file but this below code execute in current file  
 x = printhar(**'World'**)  
 print(x)  
 y = add(5, 6)  
 print(y)

# file name - tutorial25.py

# import file(tutorial24.py) for access there function  
import tutorial24  
print(tutorial24.add(7,8))

Map, Filter and Reduce :-  
# Map function  
# map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)  
# Applies a given function to all the iterables and return a new list  
# Syntax - map(fun, iter)  
# fun : It is a function to which map passes each element of given iterable.  
# iter : It is a iterable which is to be mapped.  
  
# List in string convert to integer and addition of that list  
numbers = [**'2'**,**'8'**,**'12'**,**'100'**]  
numbers = list(map(int, numbers))  
print(**f'List value :** {numbers} **'**)  
res = 0  
for i in range(len(numbers)):  
 res = res + numbers[i]  
print(**f'Total :** {res}**'**)  
  
# List value Square using def function  
def sq(a):  
 return a \* a  
num = [2, 4, 6, 3, 7, 44, 7, 88]  
square = list(map(sq, num))  
print(square) # Output -> [4, 16, 36, 9, 49, 1936, 49, 7744]  
  
# List value Square using lambda function  
num = [2, 4, 6, 3, 7, 44, 7, 88]  
square = list(map(lambda x: x \* x, num))  
print(square) # Output -> [4, 16, 36, 9, 49, 1936, 49, 7744]  
  
# Square and Cube function  
def square(a):  
 return a\*a  
def cube(a):  
 return a\*a\*a  
func = [square,cube]  
x = [3,5,7,8,9,10]  
for i in range(len(x)):  
 val = list(map(lambda x: x(i), func))  
 print(**f'Value is :** {val}**'**)  
  
# Two list multiplication using map  
list1 = [1,2,3,4,6,7]  
list2 = [8,7,5,4,7,8]  
res = list(map(lambda x,y: x\*y , list1, list2))  
print(**f"Two list multiplication :** {res}**"**) # Output -> [8, 14, 15, 16, 42, 56]  
  
  
# Filter function  
# The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.  
# filter greater value than 5 in a list  
list1 = [2,3,5,7,8,8,23,566]  
def is\_greater(num):  
 return num > 5  
result = list(filter(is\_greater, list1))  
print(**f'def :** {result}**'**) # Output -> def : [7, 8, 8, 23, 566]  
# OR using lambda function  
res = list(filter(lambda x: (x>5), [2,3,5,7,8,8,23,566]))  
print(**f'Lambda :** {res}**'**) # Output -> Lambda : [7, 8, 8, 23, 566]  
# Reduce function  
# Addition of list value using reduce function  
from \_functools import reduce  
list2 = [1,2,4,5,7]  
num = reduce(lambda x,y: x + y, list2)  
print(num)  
  
  
# Map within Filter  
res = list(map(lambda x: x+x, filter(lambda x: (x>=3),[2,3,4,5,6,7])))  
print(**f'Map within Filter =** {res}**'**) # Output -> Map within Filter = [6, 8, 10, 12, 14]  
  
# Map and Filter within Reduce  
res = reduce(lambda x,y: x+y, map(lambda x: x+x, filter(lambda x: (x >= 4), [1,2,3,4,5,76,8,9])))  
print(**f'Map and Filter within Reduce =** {res}**'**) # Output -> Map and Filter within Reduce = 204

Decorator :-

# In Python, functions are the first class objects, which means that –  
# Functions are objects; they can be referenced to, passed to a variable and returned from other functions as well.  
# Functions can be defined inside another function and can also be passed as argument to another function.  
# Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.  
# In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.  
  
def funct1():  
 print(**'Subscribe now'**)  
func2 = funct1  
func2()  
  
  
def functret(num):  
 if num == 0:  
 return sum  
 if num == 1:  
 return print  
  
a = functret(0)  
print(a) # Output -> <built-in function sum>  
  
  
def exercutor(funct):  
 funct(**'Hello World'**)  
exercutor(print) # Output -> Hello World  
  
  
def dec1(func1):  
 def nowexec():  
 print(**"Executing now"**)  
 func1()  
 print(**'Executed'**)  
  
 return nowexec  
  
  
@dec1  
def funct2():  
 print(**"Hello World"**)  
  
# funct2 = dec1(funct2)  
funct2()

OOPs Concept :-

***Class***

* *The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods.*

***Object***

* *The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.*
* *Everything in Python is an object, and almost everything has attributes and methods.*
* *Object - Is an instance of a Class  
  Object consists of - State, Behavior, Identity*
* *When we define a class, it needs to create an object to allocate the memory.*
* *We have created a object to access the class attribute.*

***Method***

* *The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods.*

*# Create Class*class Employee:  
 no\_of\_leave = 9  
 pass  
  
harry = Employee()  
sandip = Employee()  
  
sandip.name = **"Sandip"** *# sandip is a instance*sandip.salary = 16  
harry.name = **'Harry'**harry.salary = 17  
print(sandip.name)  
print(sandip.no\_of\_leave)  
print(sandip.\_\_dict\_\_) *# Output -> {'name': 'Sandip', 'salary': 16}*sandip.no\_of\_leave = 19 *# Create new instance not replace in Employee class*print(sandip.no\_of\_leave)  
print(sandip.\_\_dict\_\_) *# Output -> {'name': 'Sandip', 'salary': 16, 'no\_of\_leave': 19}*Employee.no\_of\_leave = 10 *# Update in a class*print(harry.no\_of\_leave) *# Output -> 10*

The Self :-

* Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it.
* If we have a method which takes no arguments, then we still have to have one argument.

class Employee:  
 no\_of\_leave = 9  
 # Method  
 def empdetails(self): # a method for printing data members  
 return **f'My name is** {self.name} **and salary is** {self.salary}**'**# creating object of the class  
sandip = Employee()  
harry = Employee()  
  
sandip.name = **"Sandip"**sandip.salary = 16  
harry.name = **'Harry'**harry.salary = 17  
  
print(sandip.empdetails()) # Output -> My name is Sandip and salary is 16  
print(harry.empdetails()) # Output -> My name is Harry and salary is 17

\_\_init\_\_() or Constructor :-

# The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created.In Python the \_\_init\_\_() method is called the constructor and is always called when an object is created.  
# Syntax -  
# def \_\_init\_\_(self):  
# body of the constructor  
  
class Employee:  
 no\_of\_leave = 9  
  
 def \_\_init\_\_(self, aname, asalary, arole): # aname, asalary, arole is argument - \_\_init\_\_ is a Constructor  
 self.name = aname # name is instance variable name  
 self.salary = asalary # salary is instance variable name  
 self.role = arole # role is instance variable name  
  
 def empdetails(self): # method  
 return f'My name is {self.name} salary is {self.salary} and role is {self.role}'  
  
  
sandip = Employee('Sandip', 235, 'Software')  
harry = Employee('Harry',352, 'Designer')  
  
  
print(sandip.empdetails()) # Output -> My name is Sandip and salary is 16  
print(harry.empdetails()) # Output -> My name is Harry and salary is 17

Class Method :-

* *Class method used for access to instance or access to class*
* *By using Class method we can change value of class attribute using instance or class name*

class Employee:  
 no\_of\_leave = 9  
  
 def \_\_init\_\_(self, aname, asalary, arole):  
 self.name = aname  
 self.salary = asalary  
 self.role = arole  
  
 def empdetails(self):  
 return f'My name is {self.name} salary is {self.salary} and role is {self.role} Leaves is {self.no\_of\_leave}'  
  
 @classmethod *# Decorator* def change\_leave(cls, newleaves): *# this is class method used for getting class* cls.no\_of\_leave = newleaves *#*sandip = Employee('Sandip', 235, 'Software')  
harry = Employee('Harry',352, 'Designer')  
  
Employee.change\_leave(34) *# By using Class method we can change value of class attribute*sandip.change\_leave(56) *# By using Class method we can change value of class attribute*print(sandip.empdetails()) *# Output -> My name is Sandip salary is 235 and role is Software Leaves is 56*print(harry.empdetails()) *# Output -> My name is Harry salary is 352 and role is Designer Leaves is 56*

Inheritance :-

* *It specifies that the child object acquires all the properties and behaviors of the parent object.*
* *By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.*

Public, Private and Protected

A Class in Python has three types of access modifiers

Public Access Modifier

Protected Access Modifier

Private Access Modifier

Instance Variable and Class/Static Variable

* In \_\_init\_\_/Constructor define a variable this is the Instance variable
* Outside of \_\_init\_\_ but in the class we define variable this is Class Variable

class Employee:  
 var1 = 5 *# this is Class variable* def \_\_init\_\_(self):  
 self.var2 = 5 *# this is Instance Variable*

Program

Access list using for loop  
list1 = ['Sandip', 'John', 'Marry']  
list2 = [['Sandip',1],['John',2],['Swap',3]]  
for item, item3 in list2:  
 print(item, item3)  
'''Output  
Sandip 1  
John 2  
Swap 3  
'''  
  
Access Dictionary using for looplist3 = {'sandip':'Salunhkhe',  
 'Temp':'woz',  
 'Queen':'pop'}  
for pips, pips2 in list3.items():  
 print(pips,'value is', pips2)  
'''Output  
sandip value is Salunhkhe  
Temp value is woz  
Queen value is pop  
'''

Print the all values greater than 6  
list4 = [3,4,6,2,4,'sandip',22,'John',4,53]  
for items in list4:  
 if str(items).isnumeric() and items > 6:  
 print(items)  
'''Output  
22  
53'''

While Loop

*# While Loop*i = 0  
while i < 45:  
 print(i)  
 i += 1

Show value which is greater than 4 upto less than 45i = 0  
while(True):  
 if i+1<5:  
 i = i + 1  
 continue  
  
 print(i+1, end=' ')  
 if(i==44):  
 break  
 i = i + 1  
*# Output -> 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45*

User enter a number upto if number is greater than 100 then stop that looplists = []  
i =0  
while i < 100:  
 i = int(input("Enter a number"))  
 lists.append(i)  
print(lists)

*# Other method OR*

*# User enter a number upto if number is greater than 100 then stop that loop*lists = []  
while(True):  
 input1 = int(input('Enter a number'))  
 lists.append(input1)  
 if input1 > 100:  
 print('Value is greater than 100')  
 break  
 else:  
 continue  
print(lists)

Guess the number with guess chancesguess = 18  
times = 9  
while times > 0:  
 user\_val = int(input('Enter a number\n'))  
  
 if user\_val > 18:  
 print('Value is greater than guess value')  
 elif user\_val < 18:  
 print('Value is less than guess value')  
 else:  
 print('Guess number is correct')  
 print('Number of the guesses he took to finish ', 10 - times)  
 break  
 times = times - 1  
 print('No. of guesses left ',times)  
else:  
 print('Game over')

Star Program

# Pattern Printing  
# Input = Integer n  
# Boolean = if True then increment pattern or False then decrement pattern  
# \*  
# \*\*  
# \*\*\*  
# \*\*\*\*  
def star():  
 print(**'Star Pattern'**)  
 print(**'Enter How many rows you want :'**)  
 n = int(input())  
 print(**'Enter 1 for True |OR| 0 for False'**)  
 bool\_val = input()  
 if bool\_val == **'1'**:  
 for i in range(0,n+1):  
 print(**'\*'** \* i)  
 if bool\_val == **'0'**:  
 for i in range(n,0,-1):  
 print(**'\*'** \* i)  
 star\_cond()  
  
  
def star\_cond():  
 print(**'Enter Yes for Play again |OR| No for Quit'**)  
 s = input()  
 if s == **'Yes'**:  
 star()  
 else:  
 print(**"----Game Over-----"**)  
  
star()

Fibonacci Sequence   
# Fibonacci sequence  
# 0 1 1 2 3 5 8 13 21 ....  
# In that sequence last two number addition and start with 0 and 1  
# start with 0 and 1 and next number is addition of previous two number  
  
def fibonacci(number):  
 if number == 0:  
 return 0  
 elif number == 1:  
 return 1  
 else:  
 return fibonacci(number -1) + fibonacci(number - 2)  
  
result = fibonacci(6)  
print(result) # Output -> 8

Health Management System   
# Total 6 Files, 3 For exercise and 3 For for diet  
# 3 clients - Harry, Rohan and Hammad  
# write a function that when executed takes as input client name  
# one more function to retrieve exercise or food for any client  
  
client\_list = {1: **"Harry"**, 2: **"Rohan"**, 3: **"Hammad"**}  
lock\_list = {1: **"Exercise"**, 2: **"Diet"**}  
  
  
def getdate():  
 import datetime  
 return datetime.datetime.now()  
  
  
try:  
 print(**"Select Client Name:"**)  
 for key, value in client\_list.items():  
 print(**"Press"**, key, **"for"**, value, **"**\n**"**, end=**""**)  
 client\_name = int(input())  
  
 print(**"Selected Client : "**, client\_list[client\_name], **"**\n**"**, end=**""**)  
  
 print(**"Press 1 for Log"**)  
 print(**"Press 2 for Retrieve"**)  
 op = int(input())  
  
 if op == 1:  
 for key, value in lock\_list.items():  
 print(**"Press"**, key, **"to log"**, value, **"**\n**"**, end=**""**)  
 lock\_name = int(input())  
 print(**"Selected Job : "**, lock\_list[lock\_name])  
 f = open(client\_list[client\_name] + **"\_"** + lock\_list[lock\_name] + **".txt"**, **"a"**)  
 k = **'y'** while (k != **"n"**):  
 print(**"Enter"**, lock\_list[lock\_name], **"**\n**"**, end=**""**)  
 mytext = input()  
 f.write(**"[ "** + str(getdate()) + **" ] : "** + mytext + **"**\n**"**)  
 k = input(**"ADD MORE ? y/n:"**)  
 continue  
 f.close()  
 elif op == 2:  
 for key, value in lock\_list.items():  
 print(**"Press"**, key, **"to retrieve"**, value, **"**\n**"**, end=**""**)  
 lock\_name = int(input())  
 print(client\_list[client\_name], **"-"**, lock\_list[lock\_name], **"Report :"**, **"**\n**"**, end=**""**)  
 f = open(client\_list[client\_name] + **"\_"** + lock\_list[lock\_name] + **".txt"**, **"rt"**)  
 contents = f.readlines()  
 for line in contents:  
 print(line, end=**""**)  
 f.close()  
 else:  
 print(**"Invalid Input !!!"**)  
except Exception as e:  
 print(**"Wrong Input !!!"**)

Snake Water Gun Game

*# Snake - Water = Snake win  
# Gun - Snake = Gun win  
# other combination are draw*import random  
i = 1  
user\_counts = 0  
comp\_counts = 0  
while i < 11:  
 a = [**'s'**, **'w'**, **'g'**]  
 print(**"----Enter your choice----** \n **'s' for Snake** \n **'w' for Water** \n **'g' for Gun"**)  
 user\_input = input().lower()  
 comp\_input = random.choice(a)  
 while user\_input not in (**'s'**,**'w'**,**'g'**):  
 print(**'**\n**----Invalid input. Please enter a right input----**\n**'**)  
 print(**"----Enter your choice----** \n **'s' for Snake** \n **'w' for Water** \n **'g' for Gun"**)  
 user\_input = input().lower()  
 print(**f'User Input =** {user\_input} **|| Computer Input =** {comp\_input}**'**)  
 if user\_input == **'s'**:  
 if comp\_input == **'w'**:  
 user\_counts += 1  
 elif comp\_input == **'g'**:  
 comp\_counts += 1  
 elif user\_input == **'g'**:  
 if comp\_input == **'s'**:  
 user\_counts += 1  
 i += 1  
 print(**'User points ='**, user\_counts, end=**' || '**)  
 print(**'Computer points = '**, comp\_counts)  
if user\_counts > comp\_counts:  
 print(**"**\n**-----User WINNER-----"**)  
 print(**'User points ='**, user\_counts, end=**' || '**)  
 print(**'Computer points = '**, comp\_counts)  
elif comp\_counts > user\_counts:  
 print(**'Computer WINNER'**)  
 print(**'User points ='**, user\_counts, end=**' || '**)  
 print(**'Computer points = '**, comp\_counts)  
else:  
 print(**"Game Draw"**)  
 print(**'User points ='**, user\_counts, end=**' || '**)  
 print(**'Computer points = '**, comp\_counts)

Healthy Programmer

# 9am - 5pm  
# Water - water.mp3 (3.5 litres) - Drank - log - Every 40 min  
# Eyes - eyes.mp3 - every 30 min - EyDone - log - Every 30 min  
# Physical activity - physical.mp3 every - 45 min - ExDone - log  
# Rules  
# Pygame module to play audio  
  
from pygame import mixer  
from datetime import datetime  
from time import time  
  
def musiconloop(file, stopper):  
 mixer.init()  
 mixer.music.load(file)  
 mixer.music.play()  
 while True:  
 input\_of\_user = input()  
 if input\_of\_user == stopper:  
 mixer.music.stop()  
 break  
  
def log\_now(msg):  
 with open("mylogs.txt", "a") as f:  
 f.write(f"{msg} {datetime.now()}\n")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 init\_water = time()  
 init\_eyes = time()  
 init\_exercise = time()  
 watersecs = 40\*60  
 exsecs = 30\*60  
 eyessecs = 45\*60  
  
 while True:  
 if time() - init\_water > watersecs:  
 print("Water Drinking time. Enter 'drank' to stop the alarm.")  
 musiconloop('water.mp3', 'drank')  
 init\_water = time()  
 log\_now("Drank Water at")  
  
 if time() - init\_eyes >eyessecs:  
 print("Eye exercise time. Enter 'doneeyes' to stop the alarm.")  
 musiconloop('best.mp3', 'doneeyes')  
 init\_eyes = time()  
 log\_now("Eyes Relaxed at")  
  
 if time() - init\_exercise > exsecs:  
 print("Physical Activity Time. Enter 'donephy' to stop the alarm.")  
 musiconloop('faded.mp3', 'donephy')  
 init\_exercise = time()  
 log\_now("Physical Activity done at")