```
2
3
4
5
6
8
9
```

```
# function in python

#print the 'HAVE A NICE DAY' using function
def greet():
    print('HAVE A NICE DAY')
greet()
```

```
HAVE A NICE DAY
```

```
#python program to find the sum of two numbers
#required argument
#argument=a,b
def add(a,b):
    print('the value is:',a+b)
add(50,70)
add(5,7)
```

```
the value is: 120
the value is: 12
```

```
#PEMDAS


#return function
def div(x,y):
    return x/y
print('the value is:',div(8,4))
print('the value is:',div(40,5))
```

```
the value is: 2.0
the value is: 8.0
```

```
#default argument

def bod(a,b,c=7):
    print('the value is:',a/5*a+b-c)
bod(50,7)
bod(50,7,9)
```

```
the value is: 500.0
the value is: 498.0
```

```
#convert a number to its equivalent word using python
num=int(input('enter a number:'))
```

```
def number_to_word(num):

units=['zero','one','two','three','four','five','six','seven','eight',
'nine','ten','eleven','twelve','thirteen','fourteen','fifteen','sixtee
n','seventeen','eighteen','nineteen']

tens=['','','twenty','thirty','fourty','fifty','sixty','seventy','eigh
ty','ninety']
    if num<20:
        return units[num]
    elif num <100:
        return tens[num//10] + (''if num % 10==0 else' '+ units[num
%10])
    elif num<1000:
        return units[num//100] + 'hundred' + ('' if num%100==0 else' '
+ number_to_word(num%100))
    elif num<10000:
        return units[num//1000] + 'thousand' + ('' if num%1000==0
else' ' + number_to_word(num%1000))
    elif num<1000000:
        return number_to_word(num//1000) + 'thousand' + ('' if num
%1000==0 else ' ' +number_to_word(num%1000))
    else:
        return'number out of range'
print(number_to_word(num))


enter a number:8323
eightthousand threehundred twenty three
```

DAY - 14 JULY-9

```
#count vowels in a string using python string of vowels
vowels='aeiou'
#enter an input string
user_input=input('enter the string:')
#convert string in case insensitive
string=user_input.casefold()
#create a dictionary with each vowel a key and value 0
count_vowels={}.fromkeys(vowels, 0)
#count the number of each vowels
for x in string:
    if x in count_vowels:
        count_vowels[x]+= 1
print('total number of vowels -',count_vowels)

enter the string:LAVANYA SREE P
total number of vowels - {'a': 3, 'e': 2, 'i': 0, 'o': 0, 'u': 0}
```

DAY-15 JULY-15

```
#RETURN FUNCTION

#return Expression

def arthmetic(a,b):
    return a+b/5
x=int(input('enter the number:'))
y=int(input('enter the number:'))
print('value of a and b is :',arthmetic(x,y))


enter the number:50
enter the number:70
value of a and b is : 64.0

#USING DEFAULT ARGUMENT
def div (x,y,z=70):
    return(x/y)+z
print('value is:',div(80,40))
print('value is:',div(30,10,50))

value is: 72.0
value is: 53.0

#KEYWORD ARGUMENT

def stud(name,degree,native):
    print(f'my name is {name},have completed {degree} and coming from
{native}')
stud('Lavanya Sree','BSc AZB','Chennai')
stud('Radha','BSc Economics','Madhura')


my name is Lavanya Sree,have completed BSc AZB and coming from Chennai
my name is Radha,have completed BSc Economics and coming from Madhura

# using default argument

def stud(name,degree,native,salary=50000):
    print(f'my name is {name},have completed {degree} and coming from
{native}')
    print('Im gettng salary of :',salary)
stud('Lavanya Sree','BSc AZB','Chennai',800000)
stud('Radha','BSc Economics','Madhura')


my name is Lavanya Sree,have completed BSc AZB and coming from Chennai
Im gettng salary of : 800000
```

```
my name is Radha,have completed BSc Economics and coming from Madhura
Im gettng salary of : 50000
```

```
#ARBITARY OR VARIABLE LENGTH ARGUMENT-*,**

def lang(*arg):
    print('I like to learn',arg)
lang('Bharatanatyam','carnativ music','Medical')
```

```
I like to learn ('Bharatanatyam', 'carnativ music', 'Medical')
```

```
# to make it in seperate statement
def lang(*arg):
    for x in arg:
        print('I like to learn',x)
lang('Bharatanatyam','carnativ music','Medical')
```

```
I like to learn Bharatanatyam
I like to learn carnativ music
I like to learn Medical
```

```
#Dic  IN FUNCTION AND LOOP


def student(**kargs):
    for key,value in kargs.items():
        print(key,value)
student(Name='Lavanya Sree',Degree='BSc AZB',Role='GOAT')
```

```
Name Lavanya Sree
Degree BSc AZB
Role GOAT
```

```
# to put symbols in between (use %s[s- denotes string] '%s == %s',%)


def student(**kargs):
    for key,value in kargs.items():
        print('%s == %s'%(key,value))
student(Name='Lavanya Sree',Degree='BSc AZB',Role='GOAT')
```

```
Name == Lavanya Sree
Degree == BSc AZB
Role == GOAT
```

```
#A function test whether a number is even or odd
def even_odd(num):
    if num%2==0:
```

```
        print(num,'is a even number')

    else:
        print(num,'is a odd number')
even_odd(10)
even_odd(7)
```

```
10 is a even number
7 is a odd number
```

DAY-16 JULY-16

```
def add (a,b):
    return a+b
print ('value is:',add (12,34))
```

```
value is: 46
```

```
#LAMBDA FUNCTION
#syntax:
variable name=lambda arguments:expression
x=variable name only used

x=lambda a,b:a+b
print('value of a and b is:',x(55,77))
print('value of a and b is:',x(44,99))
print('value of a and b is:',x(88,66))
```

```
value of a and b is: 132
value of a and b is: 143
value of a and b is: 154
```

```
#FILTER
# find the odd or even number using lambda
list=[1,2,3,4,6,5,7,8,10,33,22,66,99]
even=tuple(filter(lambda x:x%2==0,list))
print('even number in the list',even)
```

```
even number in the list (2, 4, 6, 8, 10, 22, 66)
```

```
odd=set(filter(lambda x:x%2==1,list))
print('odd number in the list',odd)
```

```
odd number in the list {1, 33, 3, 99, 5, 7}
```

```
#MAP - to find square root
l=set(range(15))
print(l)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```

```
#finding square of a number using map functiom

l=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
square=set(map(lambda a:a**2,l))
print('square os the list is',square)
```

```
square os the list is {0, 1, 64, 121, 4, 36, 100, 196, 9, 169, 16, 49,
81, 144, 25}
```

```
print(help(str))
print(help('keywords'))
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(self, format_spec, /)
 |      Return a formatted version of the string as described by
format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
```

```
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(...)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
|      Return self%value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __sizeof__(self, /)
|      Return the size of the string in memory, in bytes.
|
|  __str__(self, /)
|      Return str(self).
```

```
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case
and the rest lower
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless
comparisons.
 |
 |  center(self, width, fillchar=' ', /)
 |      Return a centered string of length width.
 |
 |      Padding is done using the specified fill character (default is
a space).
 |
 |  count(...)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring
sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |  encode(self, /, encoding='utf-8', errors='strict')
 |      Encode the string using the codec registered for encoding.
 |
 |      encoding
 |        The encoding in which to encode the string.
 |      errors
 |        The error handling scheme to use for encoding errors.
 |        The default is 'strict' meaning that encoding errors raise a
 |        UnicodeEncodeError.  Other possible values are 'ignore',
'replace' and
 |        'xmlcharrefreplace' as well as any other name registered
with
 |        codecs.register_error that can handle UnicodeEncodeErrors.
 |
 |  endswith(...)
 |      S.endswith(suffix[, start[, end]]) -> bool
 |
 |      Return True if S ends with the specified suffix, False
otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      suffix can also be a tuple of strings to try.
 |
```

```
 |  expandtabs(self, /, tabsize=8)
 |      Return a copy where all tab characters are expanded using
spaces.
 |
 |      If tabsize is not given, a tab size of 8 characters is
assumed.
 |
 |  find(...)
 |      S.find(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  format(...)
 |      S.format(*args, **kwargs) -> str
 |
 |      Return a formatted version of S, using substitutions from args
and kwargs.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  format_map(...)
 |      S.format_map(mapping) -> str
 |
 |      Return a formatted version of S, using substitutions from
mapping.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  index(...)
 |      S.index(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  isalnum(self, /)
 |      Return True if the string is an alpha-numeric string, False
otherwise.
 |
 |      A string is alpha-numeric if all characters in the string are
alpha-numeric and
 |      there is at least one character in the string.
 |
 |  isalpha(self, /)
 |      Return True if the string is an alphabetic string, False
```

```
otherwise.
 |
 |      A string is alphabetic if all characters in the string are
alphabetic and there
 |      is at least one character in the string.
 |
 |  isascii(self, /)
 |      Return True if all characters in the string are ASCII, False
otherwise.
 |
 |      ASCII characters have code points in the range U+0000-U+007F.
 |      Empty string is ASCII too.
 |
 |  isdecimal(self, /)
 |      Return True if the string is a decimal string, False
otherwise.
 |
 |      A string is a decimal string if all characters in the string
are decimal and
 |      there is at least one character in the string.
 |
 |  isdigit(self, /)
 |      Return True if the string is a digit string, False otherwise.
 |
 |      A string is a digit string if all characters in the string are
digits and there
 |      is at least one character in the string.
 |
 |  isidentifier(self, /)
 |      Return True if the string is a valid Python identifier, False
otherwise.
 |
 |      Call keyword.iskeyword(s) to test whether string s is a
reserved identifier,
 |      such as "def" or "class".
 |
 |  islower(self, /)
 |      Return True if the string is a lowercase string, False
otherwise.
 |
 |      A string is lowercase if all cased characters in the string
are lowercase and
 |      there is at least one cased character in the string.
 |
 |  isnumeric(self, /)
 |      Return True if the string is a numeric string, False
otherwise.
 |
 |      A string is numeric if all characters in the string are
```

```
numeric and there is at
 |      least one character in the string.
 |
 |  isprintable(self, /)
 |      Return True if the string is printable, False otherwise.
 |
 |      A string is printable if all of its characters are considered
printable in
 |      repr() or if it is empty.
 |
 |  isspace(self, /)
 |      Return True if the string is a whitespace string, False
otherwise.
 |
 |      A string is whitespace if all characters in the string are
whitespace and there
 |      is at least one character in the string.
 |
 |  istitle(self, /)
 |      Return True if the string is a title-cased string, False
otherwise.
 |
 |      In a title-cased string, upper- and title-case characters may
only
 |      follow uncased characters and lowercase characters only cased
ones.
 |
 |  isupper(self, /)
 |      Return True if the string is an uppercase string, False
otherwise.
 |
 |      A string is uppercase if all cased characters in the string
are uppercase and
 |      there is at least one cased character in the string.
 |
 |  join(self, iterable, /)
 |      Concatenate any number of strings.
 |
 |      The string whose method is called is inserted in between each
given string.
 |      The result is returned as a new string.
 |
 |      Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
 |
 |  ljust(self, width, fillchar=' ', /)
 |      Return a left-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is
a space).
```

```
 |
 |  lower(self, /)
 |      Return a copy of the string converted to lowercase.
 |
 |  lstrip(self, chars=None, /)
 |      Return a copy of the string with leading whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars
instead.
 |
 |  partition(self, sep, /)
 |      Partition the string into three parts using the given
separator.
 |
 |      This will search for the separator in the string.  If the
separator is found,
 |      returns a 3-tuple containing the part before the separator,
the separator
 |      itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing
the original string
 |      and two empty strings.
 |
 |  removeprefix(self, prefix, /)
 |      Return a str with the given prefix string removed if present.
 |
 |      If the string starts with the prefix string, return
string[len(prefix):].
 |      Otherwise, return a copy of the original string.
 |
 |  removesuffix(self, suffix, /)
 |      Return a str with the given suffix string removed if present.
 |
 |      If the string ends with the suffix string and that suffix is
not empty,
 |      return string[:-len(suffix)]. Otherwise, return a copy of the
original
 |      string.
 |
 |  replace(self, old, new, count=-1, /)
 |      Return a copy with all occurrences of substring old replaced
by new.
 |
 |        count
 |          Maximum number of occurrences to replace.
 |          -1 (the default value) means replace all occurrences.
 |
 |      If the optional argument count is given, only the first count
```

```
occurrences are
 |      replaced.
 |
 |  rfind(...)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  rindex(...)
 |      S.rindex(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  rjust(self, width, fillchar=' ', /)
 |      Return a right-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is
a space).
 |
 |  rpartition(self, sep, /)
 |      Partition the string into three parts using the given
separator.
 |
 |      This will search for the separator in the string, starting at
the end. If
 |      the separator is found, returns a 3-tuple containing the part
before the
 |      separator, the separator itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing
two empty strings
 |      and the original string.
 |
 |  rsplit(self, /, sep=None, maxsplit=-1)
 |      Return a list of the substrings in the string, using sep as
the separator string.
 |
 |        sep
 |          The separator used to split the string.
 |
 |          When set to None (the default value), will split on any
```

```
whitespace
 |          character (including \\n \\r \\t \\f and spaces) and will
discard
 |          empty strings from the result.
 |        maxsplit
 |          Maximum number of splits (starting from the left).
 |          -1 (the default value) means no limit.
 |
 |      Splitting starts at the end of the string and works to the
front.
 |
 |  rstrip(self, chars=None, /)
 |      Return a copy of the string with trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars
instead.
 |
 |  split(self, /, sep=None, maxsplit=-1)
 |      Return a list of the substrings in the string, using sep as
the separator string.
 |
 |        sep
 |          The separator used to split the string.
 |
 |          When set to None (the default value), will split on any
whitespace
 |          character (including \\n \\r \\t \\f and spaces) and will
discard
 |          empty strings from the result.
 |        maxsplit
 |          Maximum number of splits (starting from the left).
 |          -1 (the default value) means no limit.
 |
 |      Note, str.split() is mainly useful for data that has been
intentionally
 |      delimited.  With natural text that includes punctuation,
consider using
 |      the regular expression module.
 |
 |  splitlines(self, /, keepends=False)
 |      Return a list of the lines in the string, breaking at line
boundaries.
 |
 |      Line breaks are not included in the resulting list unless
keepends is given and
 |      true.
 |
 |  startswith(...)
 |      S.startswith(prefix[, start[, end]]) -> bool
```

```
 |
 |      Return True if S starts with the specified prefix, False
otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(self, chars=None, /)
 |      Return a copy of the string with leading and trailing
whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars
instead.
 |
 |  swapcase(self, /)
 |      Convert uppercase characters to lowercase and lowercase
characters to uppercase.
 |
 |  title(self, /)
 |      Return a version of the string where each word is titlecased.
 |
 |      More specifically, words start with uppercased characters and
all remaining
 |      cased characters have lower case.
 |
 |  translate(self, table, /)
 |      Replace each character in the string using the given
translation table.
 |
 |        table
 |          Translation table, which must be a mapping of Unicode
ordinals to
 |          Unicode ordinals, strings, or None.
 |
 |      The table must implement lookup/indexing via __getitem__, for
instance a
 |      dictionary or list.  If this operation raises LookupError, the
character is
 |      left untouched.  Characters mapped to None are deleted.
 |
 |  upper(self, /)
 |      Return a copy of the string converted to uppercase.
 |
 |  zfill(self, width, /)
 |      Pad a numeric string with zeros on the left, to fill a field
of the given width.
 |
 |      The string is never truncated.
 |
```

```
 |
----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  maketrans(...)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping
Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings
or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal
length, and
 |      in the resulting dictionary, each character in x will be
mapped to the
 |      character at the same position in y. If there is a third
argument, it
 |      must be a string, whose characters will be mapped to None in
the result.

None

Here is a list of the Python keywords.  Enter any keyword to get more
help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not

None

# math module
print(help('math'))


Help on built-in module math:

NAME
    math
```

```
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)


None
```

```
import math
print(math.sqrt(49))
```

```
7.0
```

```
from math import sqrt
print(sqrt (16))
```

```
4.0
```

```
#functools-module : only 2 values added but just like sum totally
every number is added

l=[1,2,3,4,5,6,7,8,9,10]
from functools import reduce
result=reduce(lambda x,y:x+y,l)
print('value is',result)
```

```
value is 55
```

DAY-17 JULY-22

```
#RECURSIVE FUNCTION-call by itselffunction

#factorial- n*fact(n-1)
n=5
5*4*3*2*1==120
```

```
True
```

```
#find the factorial of a number using recursive function
def fact(n):
    if n==0 or n==1:
        return 1
    else:
        return n*fact(n-1)
```

```
x=int(input('enter the factor number:'))
print('factor of x is:',fact(x))
```

```
enter the factor number:7
factor of x is: 5040
```

```
import math as m
print('factorial of a number:',m.factorial(7))
```

```
factorial of a number: 5040
```

```
#CEIL AND FLOOR FUNCTION
print('ceil value is:',m.ceil(70.77))
print('floor valueis:',m.floor(70.77))
```

```
ceil value is: 71
floor valueis: 70
```

```
#REMAINDER
print('Remainder value is:',m.remainder(6,3))
print('Remainder value is:',m.remainder(7,3))
```

```
Remainder value is: 0.0
Remainder value is: 1.0
```

```
from math import*
#GCD-greatest commom divisor
print('GCD of a number:',gcd(24,12))
```

```
GCD of a number: 12
```

```
print('sum of value is:',fsum([1,2,3,4,5,6,7]) )
```

```
sum of value is: 28.0
```

```
from math import*

#TRIGNOMETRIC FFUNCTION
print('sine value is:',sin(0))
print('cos value is:',cos(0))
print('tan value is:',tan(0))
print('asine value is:',asin(0))
```

```
sine value is: 0.0
cos value is: 1.0
tan value is: 0.0
asine value is: 0.0
```

```
# CONSTANT FUNCTIONS
print('Not a Number:',nan)
print('infinity is:',inf)
print('Tau value is:',tau)
print('pi value is:',pi)
print('Euler value is:',e)
```

```
Not a Number: nan
infinity is: inf
Tau value is: 6.283185307179586
pi value is: 3.141592653589793
Euler value is: 2.718281828459045
```

```
import cmath
print(cmath.sqrt(85))
```

```
(9.219544457292887+0j)
```

```
#area of circle
r=int(input('enter the radius os the circle:'))
import math
print('area of circle is:',math.pi*r**2)
```

```
enter the radius os the circle:7
area of circle is: 153.93804002589985
```

```
#circumference of the circle
r=int(input('enter the radius os the circle:'))
import math
print('circumference of the circle:',2*math.pi*r)
```

```
enter the radius os the circle:7
circumference of the circle: 43.982297150257104
```

DAY-18 JULY 23

```
#RANDOM MODULE
import random
ran=random.random()#ans in float
print('Random value is:',ran)
```

```
Random value is: 0.3346839237892345
```

```
#same random
random.seed(7)#random value will not get changed
ran=random.random()#ans in float
print('Random value is:',ran)
```

```
Random value is: 0.32383276483316237
```

```
import random
#random range
print(random.randrange(77))

print(random.randrange(5,77))
```

```
19
55
```

```
import random
#random in int

ran7=random.randint(5,50)
print(ran7)

ran5=random.randint(-70,-5)
print(ran5)
```

```
46
-64
```

```
# import random
#random choice
Colours=['sky blue','pink','red','black','green']
print(random.choice(Colours))


import random
#shuffle set
lis=set(range(17))
print(lis)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

```
lis=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
random.shuffle(lis)
print(lis)
```

```
[10, 5, 2, 9, 7, 4, 0, 12, 8, 11, 3, 14, 13, 16, 15, 1, 6]
```

DAY-19 JULY-29

```
#syntax:
open('File path name','Modes of operation')

f=open('D:\\python programming\\text.txt','r')
print(f.read())
```