using eval() mathematical expression may be input and then evaluated using input values

```
expr = input("Enter the function(in terms of x):")
```

```
Enter the function(in terms of x):x+x*5
```

```
x = (input("Enter the value of x:"))
```

```
Enter the value of x:5
```

```
y = eval(expr)
y
```

```
'555555'
```

```
L=list()
L=[1,2,3,4,5,6,89]
```

Double-click (or enter) to edit

replacing values using slice

```
L
```

```
[1, 2, 3, 4, 5, 6, 89]
```

```
L[2]=44
L
```

```
[1, 2, 44, 4, 5, 6, 89]
```

```
L[2:]='asd'
```

```
L
```

```
[1, 2, 'a', 's', 'd']
```

```
L[-4:]
```

```
[2, 'a', 's', 'd']
```

```
L[::-2]
```

```
['d', 'a', 1]
```

dictionary

```
Weekdays={1:'mon',2: 'tues',3:'wed',4:'thurs',5:'fri'
print(Weekdays)
print(Weekdays.keys())
print(Weekdays.values())
print(Weekdays.items())
```

```
{1: 'mon', 2: 'tues', 3: 'wed', 4: 'thurs', 5: 'fri'}
dict_keys([1, 2, 3, 4, 5])
dict_values(['mon', 'tues', 'wed', 'thurs', 'fri'])
dict_items([(1, 'mon'), (2, 'tues'), (3, 'wed'), (4, 'thurs'), (5, 'fr
```

```
fruits={'apple':120,'mango':100}
print(sum(fruits.values()))
print(sum(Weekdays.keys()), max(Weekdays.values()))
del Weekdays[2]  #deleting key 2
print(Weekdays)
```

```
220
15 wed
{1: 'mon', 3: 'wed', 4: 'thurs', 5: 'fri'}
```

```
W=Weekdays
W
```

```
{1: 'mon', 3: 'wed', 4: 'thurs', 5: 'fri'}
```

```
W.pop(15,'notdefined')
```

```
'notdefined'
```

```
W.pop(10)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-66-f77699183ffd> in <module>()
----> 1 W.pop(10)

KeyError: 10
```

```
Weekdays
```

```
{1: 'mon', 3: 'wed', 4: 'thurs', 5: 'fri'}
```

```
W=Weekdays
del Weekdays
print(W)
print(W.pop(2,-3))
```

```
{1: 'mon', 3: 'wed', 4: 'thurs', 5: 'fri'}
-3
```

```
W[0]='Hello'
W
```

```
{0: 'Hello', 1: 'mon', 3: 'wed', 4: 'thurs', 5: 'fri'}
```

```
W[10]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-79-0ab714654504> in <module>()
----> 1 W[10]

KeyError: 10
```

To retrieve a dictionary element

```
value = W.get(5,'ends')
value
```

```
'not'
```

```
W
```

```
{0: 'Hi', 1: 'mon', 3: 'wed', 5: 'fri'}
```

To remove a dictionary element

```
value = W.pop(4, 'not')
value
```

```
'thurs'
```

```
hash((1, 2, (2, 3)))
```

```
1097636502276347782
```

```
hash((1, 2, [2, 3]))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
<ipython-input-82-8ffc25aff872> in <module>()
----> 1 hash((1, 2, [2, 3]))

TypeError: unhashable type: 'list'
```

```
L=['Delhi','Haryana','UP','punjab']
```

```
L=['Delhi','Haryana','UP','punjab']
print(L)
dictstates={}
for i,val in enumerate(L):
  dictstates[i]=val
dictstates
```

```
['Delhi', 'Haryana', 'UP', 'punjab']
{0: 'Delhi', 1: 'Haryana', 2: 'UP', 3: 'punjab'}
```

ZIP()

```
seq1 = ['Delhi','Haryana','UP','punjab']
seq2 = ['one', 'two', 'three','four']
zipped = zip(seq1, seq2)
print(zipped,type(zipped))
Z=dict(zipped)
```

```
<zip object at 0x7ff7a669cb40> <class 'zip'>
```

```
Z
```

```
{'Delhi': 'one', 'Haryana': 'two', 'UP': 'three', 'punjab': 'four'}
```

```
L1=W.values()
L1=list(L)
L1
```

```
['Delhi', 'Haryana', 'UP', 'punjab']
```

```
seq1
```

```
['Delhi', 'Haryana', 'UP', 'punjab']
```

```
seq2
```

```
['one', 'two', 'three', 'four']
```

```
Z1=(zip(seq1,seq2))
Z2=zip(L,Z1)
dict(Z2)
```

```
{'Delhi': ('Delhi', 'one'),
 'Haryana': ('Haryana', 'two'),
 'UP': ('UP', 'three'),
 'punjab': ('punjab', 'four')}
```

```
L=list(zip(seq1,seq2))
```

```
del L1
```

```
L1=[1,2,3]
```

```
dict(zip(L1,L))
```

Creating dictionary from sequences

```
mapping = dict(zip(range(5), reversed(range(5))))
mapping
```

```
{0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

```
L=list(range(5))
L
```

```
[0, 1, 2, 3, 4]
```

iterating over multiple sequences using zip()

```
print(seq1,seq2)
```

```
['Delhi', 'Haryana', 'UP', 'punjab'] ['one', 'two', 'three', 'four']
```

```
list(zip(seq1,seq2))
```

```
{'Delhi': 'one', 'Haryana': 'two', 'UP': 'three', 'punjab': 'four'}
```

```
for i, (x, y) in enumerate(zip(seq1, seq2)):
    print('{0}: {1}, {2}'.format(i, x, y))
```

```
0: Delhi, one
1: Haryana, two
2: UP, three
3: punjab, four
```

dict.get(): to get specific values at a key

```
mapping.get(10,'999')
```

```
'999'
```

What is the output?

```
dict1 = { 40 : 'Hello', 11 : 'from', (30,10) : 'All'
```

```
print(set(dict1))
```

```
{40, 11, (30, 10)}
```

Double-click (or enter) to edit

```
dict1.setdefault(40,'absent')
```

```
'Hello'
```

```
dict1.setdefault(50,'absent')
```

```
'absent'
```

```
dict1
```

```
{(30, 10): 'All', 40: 'Hello', 50: 'absent'}
```

```
dict1.pop(50,'invalid')
```

```
'absent'
```

method to delete the specified key' pair without flash the deleted item

```
del dict1[11]
```

```
l1=list(dict1.values())
l1
```

```
['Hello', 'All', 'absent']
```

Generator: A special type of function **that returns an iterator object** with a sequence of values instead of a single value.

- need to typecast the iterator object before accessing it.
- A yield statement is used rather than a return statement.
- A concise way to construct a new iterable object

```
def f1():
  yield(10)
  yield(20)

print(list(f1()))
```

```
[10, 20]
```

reversed() generator

```
print(l1)
L2=reversed(l1)
```

```
['Hello', 'All', 'absent']
```

```
print(list(l1),list(L2))
```

```
['Hello', 'All', 'absent'] ['absent', 'All', 'Hello']
```

```
def squares(n=10):
    print('Generating squares from 1 to {0}'.format(n
    for i in range(1, n + 1):
        yield i ** 2
```

```
tuple(squares())
```

```
Generating squares from 1 to 100
(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

example generator expression

```
gen = (x ** 3 for x in range(10))
gen
```

```
<generator object <genexpr> at 0x7f70922e7850>
```

```
list(gen)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

```
sum(gen)
```

```
2025
```

difference in iterator and iterable

- An iterator: an object that contains a finite number of values
- iterable:Tuple, lists, tuples, dictionaries, and sets that returns iterator object.

> have iter() method to get an iterator object

```
l1
```

```
['Hello', 'All', 'absent']
```

```
O=iter(l1*2)
print(O,l1)
```

```
<list_iterator object at 0x7f70921d8290> ['Hello', 'All', 'absent']
```

```python
for i in O:
  print(i)
```

```
Hello
All
absent
Hello
All
absent
```

```python
l1
```

```
['Hello', 'All', 'absent']
```

```python
O
next(O,'ends')
```

```
'ends'
```

more example on generator

```python
def squares(n=10):
    print('Generating squares from 1 to {0}'.format(n
    for i in range(1, n + 1):
        yield i ** 2
```

```python
gen=squares(5)
```

```python
next(gen,'ends')
```

```
Generating squares from 1 to 25
1
```

```
next(gen, 'ends')
```

```
4
```

the iterated portion of iterator object is not traversed again until redefined

```
list(gen)
```

```
[9, 16, 25]
```

when using iterator, be careful about its usage

needs to reassign values to iterator object to get all values, if already accessed. Not needed in iterable objects

```
for i in O:
  print(i)
```

```
for i in l1:
  print(i)
```

```
Hello
All
absent
```

range() is agenerator that returns an iterator **object**

```
L=list(range(5))
L
```

```
[0, 1, 2, 3, 4]
```

Unzipping

```
Z
```

```
{'Delhi': 'one', 'Haryana': 'two', 'UP': 'three', 'punjab': 'four'}
```

```
new1,new2=zip(*Z)
new1, new2
```

```
(('D', 'H', 'U', 'p'), ('e', 'a', 'P', 'u'))
```

```
seq1
```

```
['Delhi', 'Haryana', 'UP', 'punjab']
```

reversed generator

```
reversed(seq1)
```

```
<list_reverseiterator at 0x7fad325dfed0>
```

```
list(reversed(seq1))
```

```
['punjab', 'UP', 'Haryana', 'Delhi']
```

```
dict1
```

```
{(30, 10): 'All', 40: 'Hello'}
```

Find the output?

Combine mutiple words starting with same letter together with same key output should be a dictionary where key is the letter and value is the word

```
words = ['apple', 'bat', 'bar', 'atom', 'book']
by_letter = {}
for word in words:
    letter = word[0]
    if letter not in by_letter:
        by_letter[letter] = [word]
    else:
        by_letter[letter].append(word)
by_letter
```

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

simple way to do above thing using setdefault()

returns the value of the passed key, else add the key (if not part of dictionary) with default mentioned value of either scalar/sequence

```
d1={1:'in',2:'out'}
letter=2
print(d1.setdefault(letter, []))
d1
```

```
out
{1: 'in', 2: 'out'}
```

```
letter=3
print(d1.setdefault(letter, []))
d1
```

```
[]
{1: 'in', 2: 'out', 3: []}
```

```
by_letter={}
for word in words:
  letter = word[0]
  by_letter.setdefault(letter, []).append(word)
by_letter
```

defaultdict module: to produce a new value for the key not in dictionary and returns an iterator object, Needs to pass the value as scalar or sequence

```
from collections import defaultdict
```

```
words
```

```
['apple', 'bat', 'bar', 'atom', 'book']
```

```
D1=defaultdict(str)
for word in words:
    D1[word[0]]=word
```

```
dict(D1)
```

```
{'a': 'atom', 'b': 'book'}
```

```
by_letter = defaultdict(list)
print(by_letter)
for word in words:
    by_letter[word[0]].append(word)

dict(by_letter)
```

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

comrehession

```
[x.upper() for x in words if len(x) > 3]
```

```
['APPLE', 'ATOM', 'BOOK']
```

```
for x in words:
  count=0
  if len(x)>3:
    if count==0:

      y=[x.upper()]
      count+=1
    else:
      y=y.append(x.upper())
```

```
y
```

```
['BOOK']
```

```
dict_lengths = {x:len(x) for x in words}
```

```
dict_lengths
```

```
{'apple': 5, 'atom': 4, 'bar': 3, 'bat': 3, 'book': 4}
```

```
words
```

```
['apple', 'bat', 'bar', 'atom', 'book']
```

what is the output?

```
dict_lengths = {x:y for x,y in enumerate(words)}

dict_lengths
```

```
{0: 'apple', 1: 'bat', 2: 'bar', 3: 'atom', 4: 'book'}
```

Nested comprehenssion

to get all words having letter 'o' more than once in a list

```
words2D = [['apple', 'mango'],['bat', 'ball'],['pen',

result = [x for names in words2D for x in names
          if x.count('o') > 1]
result
```

```
['book', 'goose', 'look']
```

```
[x for names in words2D for x in names if x.count('o'
```

```
['book', 'goose', 'look']
```

what is the output?

```
list_tuple = [(11, 12, 13), (14, 5, 16), (7, 8, 9)]

[[x for x in tup if x>10] for tup in list_tuple]
```

```
[[11, 12, 13], [14, 16], []]
```

output is to get all the elements with only values >10, using := operator

```
tot=0

[[tot:=tot+x for x in tup if x>10] for tup in list_tu
```

```
  File "<ipython-input-186-93c54b5f6454>", line 1
    [[tot:=tot+x for x in tup if x>10] for tup in list_tuple]
         ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

```
import platform
print(platform.python_version())
```

```
3.7.11
```

functions

```
gx=4
def fn(b,c,a=5):
    global gx
    gx=2
    print ('inside fn:gx=', gx)
    x1=10+5
    gx=20
    print ('inside fn:x=',x1)
    print ('inside fn:gx=', gx)
    return x1+2,gx+20
```

```
print ('outside fn:before fn() called  gx= ',gx)
print(fn(c=6,b=10))
print ('outside fn:after fn() called  gx= ',gx)
print ('outside fn:x= ',x1)
```

```
outside fn:before fn() called  gx=  4
inside fn:gx= 2
inside fn:x= 15
inside fn:gx= 20
(17, 40)
outside fn:after fn() called  gx=  20
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-203-30e69986354d> in <module>()
      2 print(fn(c=6,b=10))
      3 print ('outside fn:after fn() called  gx= ',gx)
----> 4 print ('outside fn:x= ',x1)

NameError: name 'x1' is not defined
```

map() function returns a map object, an iterator, of the results after applying the given function to each item of passed iterable (list, tuple etc.)

```
states = ['   Alabama$ ', 'Georgia!', 'Georgia', 'geo
          'south   carolina##', 'West virginia?','abc
```

```
def fnlen(n):
  if 'G' in n:
    return len(n)
  else:
    return 0

print(fnlen(states))
```

```
8
```

```
for s in states:
```

```
for e in states:
  print(fnlen(e))

L=list(map(fnlen,states))


L
```

```
    [0, 8, 7, 0, 0, 0, 0, 0]
```

Function to remove special characters '$#%' from strings

```
import re   #regular expression module


def remove_punctuation(value):
  value=value.strip()
  return re.sub('[!#?$/]', '', value)
```

passing function as an argument to another function

```
for x in map(remove_punctuation, states):
    print(x,x.isalpha())
```

```
    Alabama True
    Georgia True
    Georgia True
    georgia True
    FlOrIda True
    south   carolina False
    West virginia False
    abc True
```

lambda function:A lambda function is a small anonymous function that

- take any number of arguments
- has only one expression
- function object is never given a name using **name**attribute

```
fn=lambda a,b: ~(a**b)

print(fn(3,2))
```

```
-10
```

Bitwise operators

&: AND Sets each bit to 1 if both bits are 1 |: OR Sets each bit to 1 if one of two bits is 1 ^: XOR Sets each bit to 1 if only one of two bits is 1 ~: NOT Inverts all the bits (result displayed is 2's complement)

```
print('{0:5} and {1:7}'.format(9&5,9|5))
```

```
    1 and      13
```

right (>) and left (<) aligned output

```
print('{0:>5} and {1:<06}'.format(str(9&5),str(9|5)))
```

```
    1 and 130000
```

usage of lambda function

- hassle free for doing simple repetitve task
- useful when a function is to be passed as argument of another function

example: compute expression x^3+2*x for x in L=[2,3,56,67]

```
def fn1(L):
  for x in L:
    print(x**3+2**x)


L=[2,3,56,67]
fn1(L)
```

```
12
35
72057594038103552
147573952589676713691
```

```
def fnexpression(L1,f1):
  #for e in L1:
    #print(f1(e))
  return [f1(e) for e in L1]


L=[2,3,56,67]
fnexpression(L,lambda x:x**3+x*2)
```

```
[12, 33, 175728, 300897]
```

Currying: Partial Argument Application

Double-click (or enter) to edit

```
def exp_numbers(x, y,z):
    return x +2* y//z
```

redefining already defined function using partial argument. keeping two values constant and passing third arugment

```
newz = lambda z: exp_numbers(5,7,z)
```

```
newz(2)
```

```
    12
```

```
from functools import partial
newpartial = partial(exp_numbers,8,3)
newpartial(6)
```

```
    9
```

key arguments can be used as per the rule of default arguments

```
newpartial = partial(exp_numbers,y=3,z=3)
newpartial(2)
```

```
    4
```

itertools module: collection of generators

```
import itertools
first_letter = lambda x: x[0]
names = ['Alan', 'Adam', 'Albert', 'Wes','Will', 'Ste
```

```
names.sort()
print(names)
```

```
['Adam', 'Alan', 'Albert', 'Steven', 'Wes', 'Will']
A ['Adam', 'Alan', 'Albert']
S ['Steven']
W ['Wes', 'Will']
```

```
for letter,words  in  itertools.groupby(names, first_
    print(letter, len(list(words))) # words is a gene
```

```
A 3
S 1
W 2
```

```
list(itertools.groupby(names, first_letter))
```

```
[('A', <itertools._grouper at 0x7ff7a66f80d0>),
 ('W', <itertools._grouper at 0x7ff7a66f8310>),
 ('A', <itertools._grouper at 0x7ff7a66f8250>),
 ('S', <itertools._grouper at 0x7ff7a66f8350>)]
```

```
list()
```

other useful functions

Combination wiithout replacement: C(n,r)=(n)!/r!(n−1)! retuen a tuple of r-sized combination

```
list(itertools.combinations('xyzw', 3))
```

```
[('x', 'y', 'z'), ('x', 'y', 'w'), ('x', 'z', 'w'), ('y', 'z', 'w')]
```

C(n,r)=(n+r−1)!/r!(n−1)!

```
list(itertools.combinations_with_replacement('xyzw', 
```

to find the cartesian product from the given iterator, output is lexicographic ordered. itertools.product()

```
s1='asd'; s2='xyz'
list(itertools.product(s1,s2))
```

```
    [('a', 'x'),
     ('a', 'y'),
     ('a', 'z'),
     ('s', 'x'),
     ('s', 'y'),
     ('s', 'z'),
     ('d', 'x'),
     ('d', 'y'),
     ('d', 'z')]
```

```
list(itertools.product(s1,s1))
```

```
    [('a', 'a'),
     ('a', 's'),
     ('a', 'd'),
     ('s', 'a'),
     ('s', 's'),
     ('s', 'd'),
     ('d', 'a'),
     ('d', 's'),
     ('d', 'd')]
```

try-except block:

Exceptions: Errors detected during execution are called exceptions and are not unconditionally fatal

for undeclared variable

```
try:
```

```
try:
  print(value)
except:
  print("An exception occurred")
```

```
An exception occurred
```

Different exceptions:

- NameError: Variable not defined
- TypeError: an operation or function is applied to an object of inappropriate type.
- ValueError: when a built-in operation or function receives an argument that has the right type but an inappropriate value
- ZeroDivisionError: denominator is zero

```
try:
  value='a'
  print(value)
  value='2'+5
  print(float(value))
  x=list(value)

except NameError:
  print("Variable value is not defined")
except ValueError:
  print("Variable value is  not compatible")
except TypeError:
  print("Conversion type is  not compatible")
except:
  print("Something else went wrong")

del value
```

```
a
Conversion type is  not compatible
```

#

- Else: executed if try clause does not raise an exception.
- finally: executed always irrespective of the try block raises an error or not.

```
try:
  #print(xx)
  print("Hello everyone")
except:
  print("wrong action")
else:
  print("All is fine")
finally:
  print('explored')
```

```
Hello everyone
All is fine
explored
```

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
print(x+5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-41261ede40e9> in <module>()
      2 
      3 if not type(x) is int:
----> 4   raise TypeError("Only integers are allowed")
      5 print(x+5)

TypeError: Only integers are allowed
```

raise: raise an exception if a given condition is meet. used for debugging

```
xx=int(input('enter value: '))
try:
  assert xx == 10
  print(xx+5)
except:
  print('missing data')
```

```
enter value: 11
missing data
```

given are two lists L1 L2 of names find unique names in both list as well as all common names in both lists

```
names1 = ['Alan', 'Adam', 'Albert', 'Wes','Will', 'St
names2 = ['Alan', 'Adam', 'Alberto', 'Wesely','Will',

s1=set(names1).union(set(names2))
s1
```

```
{'Adam', 'Alan', 'Albert', 'Alberto', 'Steven', 'Wes', 'Wesely', 'Will
```

```
s1=set(names1).intersection(set(names2))
s1
```

```
{'Adam', 'Alan', 'Steven', 'Will'}
```

✓ 0s completed at 1:00 PM