

(Design philosophy)

(Batteries included)

(Awesome Community)

DO NOT REINVENT THE WHEEL

WHEEL BAN CHUKA HE TOH GADI BANAV

End goal is to serve better with best applications and services not to build the code from scratch.

Compilation is the process which includes converting code into binary language.

Interpreters compile code line by line therefore are slow in comparison to Compilers which does it as a whole.

1) Python source code using its extension (Hello.py)	
2 <u>Compiler</u> compiles the code and converts into byte code)	Extension for bytecode file is (hello.pyc)
3 Byte code need to be converted into machine code since) machines don't understand byte code	
4 Python virtual Machine (PVM) converts the byte code) into machine code using <u>Interpreter</u>	Extension machinecode (Hello.exe)

Python is a general purpose (used to create a range of applications)

High level programming language (completely understandable by humans)

Python was developed by guido van rossam in 1989 in netherlands

Official date is feb 20th 1991

APPLICATIONS OF PYTHON

for developing desktop application

for developing web application

for developing database application

for network programming

for developing games

for data analysis application

for machine learning
for ai

FEATURES OF PYTHON

Simple and easy to learn

Freeware and open source

Interpreted : all code is executed line by line we are not required to compile the python programs explicitly internally python interpreters will take care of that compilation.

Python has rich inbuilt library

Python is platform independent language

Python is dynamically typed language we are not required to declare the type variables

Python has rich inbuilt source of library

LIBRARY FUNCTION

there are three types of library functions

1) tokens - essential elements for writing python program.

keywords

variables

identifiers

datatypes

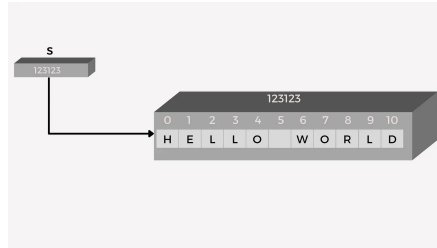
2) operators -

3) built in functions -

VARIABLES

Memory allocation :

As soon as the variable is created python controller allocates address of



value to the variable space and value is stored in memory block

Variable space - is the memory where address of the value is stored

value space - is the memory where variable value is stored

named memory block where we are going to store some values

```
meet = 4
print(meet)
print(id(meet))
```

Output :

4

4354940072

```
# multiple variable creation
a,b,c = "meet","krupal","heet"
print(a,b,c)
```

Output :

meet krupal heet

- In python datatypes follow dynamic typing it means variables are not needed to declare its datatype explicitly. Compiler understands the datatype on its own. This is called **Dynamic Typing**.
- In C language u mention type int name = 'meet' this is **Static Typing**. Also Python variables are not binded to any particular datatype, those can store values of different datatypes as well after declaring. This is called **Dynamic Binding**. For example,

```
a = True    print(a) >>> True
a = 4       print(a) >>> 4
```

IDENTIFIERS

user defined names given to variable, class, module, function, etc collectively identifiers

KEYWORD - these words have special meaning.

Compiler or Interpreter has few sets of keywords (special words) which are used to convert high level language to low level language

```
a = "hello world"
print(a)
```

```
import keyword
print(keyword.kwlist)
```

Output:

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

first 3 keywords start with capital letter rest all are starting with small False True None and are the only three which can be used as value for variables

Dynamic softwares are the softwares which asks for the inputs and works accordingly for example, mostly application are dynamic eg. youtube #, those softwares where u only see and infer something from its working is called static software for example, clock, blog, etc

We use input() function to receive input from the user.

DATATYPES

Single value datatypes

- 1) Integer
- 2) Float
- 3) Complex
- 4) Boolean

Multi value datatypes

- 1) String **immutable**
- 2) List **mutable**
- 3) Tuple **immutable**
- 4) Set **mutable**
- 5) Dictionary **mutable**

```
a = 20  
print(type(a))
```

```
# Integer  
# every datatype has default and non-default value  
# default value of integer is "False" "0"  
# and non default value is every other value other than zero "True"
```

```
b = 20.20  
print(type(b))
```

```
# Float  
# default value of integer is "0.0"  
# non default value is other then "0.0"
```

```
c = 0.3 + 4j  
print(type(c))
```

```
# Complex
```

```
# combination of real and imaginary numbers
# a + bj where a and b are integer or float and j equals to  $\sqrt{-1}$  (root
sign)
# default value of complex datatype is 0j
```

```
d = True
print(type(d))
```

```
# Boolean
# default value of boolean datatype is False
# and non default value is True
```

```
Output :
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'bool'>
```

#String - Multi value datatype

Strings are set or collection of characters. They are immutable.

```
my_string = "Hello, world!"  
# Attempt to modify the string  
my_string[0] = 'h'  
# Raises TypeError: 'str' object does not support item assignment
```

You can construct a string object using "str" constructor eg. word = str('Meet sudra')

Constructor is a special method which is called when an object is created

```
a1 = 'Meet Sudra'  
a2 = 'Heet Sudra'  
a3 = 'Krupal Sudra'
```

```
print(a1)  
print(a2)  
print(a3)
```

Output :
Meet Sudra
Heet Sudra
Krupal Sudra

Indexing

we can index the collection using positive and negative indexing
indexing from right to left is negative indexing
while indexing from left to right it is positive indexing

Positive indexing

0	1	2	3	4	5
	y	t	h	o	n
p					

Negative indexing

-6	-5	-4	-3	-2	-1
p	y	t	h	o	n

```
aa = a1[-1]  
bb = a1[0]  
print(bb)  
print(aa)
```

Output:

```
M  
a
```

```
print(len(a1)) #len function  
print(len('meetsudra'))
```

Output :

```
10  
9
```

Regular String vs Raw String

In Python strings, the backslash "\" is a special character, also called the "escape" character. It is used in representing certain whitespace characters: "\t" is a tab, "\n" is a newline, and "\r" is a carriage return.

Regular string considers backslash as a special sequence

```
p = "Hello \task from pytho\n world"  
print(p)
```

Raw string considers entire string as a character

```
p = r"Hello \task from pytho\n world"  
print(p)
```

```
print("\n") # it will return a blank line
```


Output:

Hello ask from pytho

world

Hello \task from pytho\n world

```
m = "lorem gipsum mysun heelus gelusm \t heer aje gandsum \n this way  
backslash (escape character) and regular strings work"
```

```
print(m)
```

```
m = r"lorem gipsum mysun heelus gelusm \t heer aje gandsum \n this way  
backslash (escape character) and raw strings work"
```

```
print(m)
```

```
lorem gipsum mysun heelus gelusm      heer aje gandsum
```

```
this way backslash (escape character) and raw strings work
```

```
lorem gipsum mysun heelus gelusm \t heer aje gandsum \n this way  
backslash (escape character) and raw strings work
```

Memory Allocation and Deallocation

Memory management is automatic

Two scripts :

- 1) Python memory manager - allocation
- 2) Python Garbage Collector - deallocation

Memory Allocation

There are two types of memory in ram.

- 1) Stack memory or static memory (simply stores the reference of the object)

slow as compared to heap and Direct access allowed

- 1) global references
- 2) Function calls and names

- 2) Heap memory (Private heap space) (stores the object)

Can access using references or variables

- 1) All values and objects

The stack keeps track of variable names and their corresponding references/ids to objects in the heap memory. This mechanism allows Python to manage memory efficiently and supports dynamic typing. This kind of memory allocation is also known as Temporary memory allocation because as soon as the method finishes its execution all the data belonging to that method flushes out from the stack automatically.

Lets take an example

```
x = [10,20,22]
```

```
y = x
```

```
x.append(40)
```

```
x = [35,45]
```

```
y.append(25)
```

what will be the value of x and y after step 5?

```
X = [35,45]
```

```
Y = [10,20,22,40,25]      o/p
```

Del keyword is used for **deallocation**

Slicing

Phenomenon of extracting group of data items from a given collection.

[starting index:ending index:step]

We can apply slicing only on string list, tuple, dictionary (if values of keys are string, list, tuple)

```
string = "Hi welcome to Python!"
```

```
#print the string's alternate characters
```

```
q1 = string[::2] # step = 2 for alternate charcters
```

```
print(q1)
```

```
#print the string's alternate character in reversed order
```

```
q2 = string[::-1] # ::-1 is used to reverse the string
```

```
print(q2)
```

```
print(q2[::2])
```

```
#print the string in reversed order
```

```
q3 = string[::-1]
```

```
print(q3)
```

```
#print the extension of the file name "Youtube.com"
q4 = "Youtube.com"
print(q4[8:])
```

Output :

```
H ecm oPto!
!nohtyP ot emoclew iH
!otPo mce H
!nohtyP ot emoclew iH
com
```

Formated strings

String formatting is also known as String interpolation. It is the process of inserting a custom string or variable in predefined text.

There are three ways of formatted strings

- 1) Formatting string using placeholders
- 2) Formatting with .format() string method
- 3) Formatting with string literals, called f-string

.format method and placeholder

```
print("My name is {firstname} {lastname}".format(firstname='Meet', lastname
= 'Sudra'))
```

```
print('We all are {}'.format('equal'))
```

```
print('we live in {}.{} is in {}'.format('Dahisar','Dahisar','Mumbai'))
```

Output :

```
We all are equal
we live in Dahisar.Dahisar is in Mumbai
```

f-literals

```
aaa = 24
bbb = 'Day'
ans = f"There are {aaa} Hours in a {bbb}"
```

```
print(ans)
```

```
ccc = "One"
```

```
ddd = 1224
```

```
ans2 = f"{ccc} has {ddd} minutes in a day"
```

```
print(ans2)
```

```
eee = "Sudra"
```

```
fff = "B-402"
```

```
ans3 = f"{eee} family stays in {fff}, Minal Park"
```

```
print(ans3)
```

Output :

There are 24 Hours in a Day

One has 1224 minutes in a day

Sudra family stays in B-402, Minal Park

methods

dir() - It is a list of inbuilt functions that returns list of attributes that are attached to the object or class

```
message = "Hello world!"
```

```
print(dir(message))
```

#String methods

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',  
'__eq__', '__format__', '__ge__', '__getattr__', '__getitem__',  
'__getnewargs__', '__getstate__', '__gt__', '__hash__', '__init__',  
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__',  
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',  
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',  
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',  
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',  
'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',  
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',  
'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind',
```

'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

These are the methods/ attributes/ functions which can be implemented only on string class or object

```
print('\n','#lower method' )  
print(message.lower())
```

```
print('\n','#upper method' )  
print(message.upper())
```

```
print('\n','#swap case' )  
print(message.swapcase()) # Converts a lower case character into upper  
case and vicerversa
```

```
print('\n','# count() method')  
print(message.count('H')) # counts the occourences of H in hello world!  
print(message.count('h')) # counts the occourences of h in hello world!  
print(message.count('Hello',0,13)) # counts the no. of word 'Hello' in the  
mentioned range  
# Hello occurs once from si 0 to ei 23  
print('\n','#index() method')  
print(message.index('o',3,13)) # how many times the letter o occurs in the  
given range from 3 to 13
```

```
print('\n','#rindex')  
print(message.rindex('l')) # will return the last position where it was found  
print(message.rindex('w'))  
print(message.rindex('Meet')) # keyvalue error
```

```
print('\n','# find() method')  
print(message.find('H'))  
print(message.find('d'))
```

```
print('\n','# rfind() method')  
print(message.rfind('l'))  
print(message.rfind('d'))
```

Both index() and find() method are identical as they both return the index position of the first occurrence. The main difference is that index method returns keyvalue error if it is unable to find the substring whereas find() method returns -1

```
print('\n','# replace() method')
print(message.replace('World','Universe'))
print('banana'.replace('a','A'))
print('banana'.replace('a','A',1)) # banana replace a at index position 1 with A
replace('old_val', 'new_val', index_number)
```

```
print('\n','# startswith() method')
print(message.startswith('universe')) # returns True if the string starts
with specified value
```

```
print('\n','# endswith() method')
print('Hello Universe'.endswith('universe')) # returns True if the string
starts with specified value
```

```
print('\n','# split method') # split method splits a string to a list
print(message.split())
# string.split(separator,maxsplit)
# You can specify the separator, default value is any whitespace.
# maxsplit specifies how many split to do. default value is -1, which is all
occurrences
print('Hello, my name is Peter, I am 26 years old'.split(',',1))
print('apple#banana#cherry#orange'.split('#',2)) # split max 2 items
```

```
print('\n','# rsplit() method')
# split splits the string into list starting from the right
# if no max is provided it is same as the split method
print('Hello, my name is Peter, I am 26 years old'.rsplit(',',1))
```

```
print('\n', '# join() method')
# joins the element of the sequence using the string specified
print(message)
print('hi'.join(message)) # hi after every element
print(' '.join(message)) # space after every element
```

```
print('\n','# strip() method')
```

strip function is used to remove extra whitespaces and specified characters from the start and from the end irrespective of how the parameter is passed.

```
print('      Hey,myself Meet Sudra'.strip())
xyz='$$$Hey dude, wassup?$$$$$$$$$'
print(xyz.strip("$"))
txt = ",,,,,rrttgg.....banana....rrr"
print(txt.strip("rtg."))
print(txt.rstrip("rtg.")) # rstrip() method
print(xyz.rstrip("$")) # Removes $ from the end
print(xyz.lstrip("$")) # Removes $ from the start    lstrip() method
print(txt.lstrip("rtg.")) # from start till banana
```

```
print(xyz.isalnum()) # returns true if all the characters in the string are
alphanumeric
print(xyz.isalpha()) # returns true if all the characters in the string are
alphabets
print(xyz.isdigit()) # returns true if all the characters in the string are
digits
print(xyz.isupper()) # returns true if all the characters in the string are in
upper case
print(xyz.islower()) # returns true if all the characters in the string are in
lower case
print(xyz.isspace()) # returns true if all the characters in the string are
spaces
print(xyz.isidentifier()) # whether it is identifier or not
```

All these string methods return new value. These do not change the original string

Concatenation

Combining two or more strings using (+)

```
print('\n','# Concatenation')
print('Good'+ ' '+Morning!)
h = 'Rome'
j = 'was'
```

```
k = 'not'  
o = 'built'  
l = 'in'  
p = 'a'  
q = 'day'  
print(h+ ' '+j+ ' '+k+ ' '+o+ ' '+l+ ' '+p+ ' '+q)
```

Replication

Creating copies of a string using asterick

```
print('\n','# Replication')  
print('Sale! '*5)
```


List - Multi value Datatype

- 1) List is a collection of homogenous (of the same or a similar kind or nature) and heterogenous (consisting of different kinds of people or things) elements separated by comma.
- 2) Elements in the list are ordered and Lists are mutable (we can modify original list)
- 3) `len(list)` function to find the length of the list.
- 4) Default value of list is empty `[]` length=0.
- 5) List can have a list within itself it is called nested list

```
a = [1,2,3,4,5]
print(type(a))
print(a)
```

```
print('\n')
b = [1,'Hello',9.78,4+4j,False]
print(type(b))
print(len(b))
print(b)
```

```
print('\n')
c = [1,4,3,['hello',3.2,True],'Bava']
print(c)
print(type(c))
print(len(c))
```

```
print('\n','Indexing in lists')
d = ['yahoo','chrome','firefox','safari','redhat','chrome','gmail']
print(d.index('chrome')) # returns index number using index() method
print(d.index('yahoo'))
print(d[1])
print(d[3])
print(d.index('chrome',2,6))
```

```
print(d[1]) # returns element at index number 1
print(d[-3])
print(d[2])
```

```
e = [1.2,3,'hello',['world',24+7j,6.6],'universe']
```

```
print(e[3][1])
print(e[3][2])
```

```
print(e[1:4]) # starting index : ending index
print(e[2:4:2])
print(e[1:4:2])
print(e)
print(e[3][1])# targeting nested list's element > (24+7j)
print(e[3][0]) # > world
print(e[3][0][0:4:2]) # > wr
print(e[3][0][-1:]) # > dlrow
print(e[3][0][-1:-4:-1]) # >dlr
```

```
names = ['apple','google','yahoo','amazon','facebook','instagram','microsoft']
print(names[2:5]) # all elements from index no.2 to index no.4
print(names[0:4]) # all elements from index no.0 to index no.3
print(names[-4:-2]) # print amazon facebook
print(names[-6:6]) # from google to instagram
print(names[0:-1])
print(names[-7:]) # print entire list
print(names[-7::2]) # print alternate items
print(names[::-1]) # print elements in reverse order
print(names[-1:2:-1]) # print elements microsoft to yahoo
```

Output :

```
['yahoo', 'amazon', 'facebook']
['apple', 'google', 'yahoo', 'amazon']
['amazon', 'facebook']
['google', 'yahoo', 'amazon', 'facebook', 'instagram']
['apple', 'google', 'yahoo', 'amazon', 'facebook', 'instagram']
['apple', 'google', 'yahoo', 'amazon', 'facebook', 'instagram', 'microsoft']
['apple', 'yahoo', 'facebook', 'microsoft']
['microsoft', 'instagram', 'facebook', 'amazon', 'yahoo', 'google', 'apple']
['microsoft', 'instagram', 'facebook', 'amazon']
```

Methods in List

1) append()

Adds an element at the end of the list

```
print('\n','# append()')
names.append('gmail')
print(names)
f = [0,1,2,3]
f.append([4,5,6])
```

2) `extend()`
 Extends the list using specified sequence

```
print('\n','# extend()')
f.extend([7,8,9])
print(f)
```

3) `insert()`
 Inserts an element at specified position

```
print('\n','# insert()')
f.insert(0,'Macbook')
print(f)
```




















4) `pop()`
 Removes the element at specified position

```
print('\n','# remove()')
f.remove(8) # need to mention the element name
print(f)
```

```
print('\n','# clear()')
print(f.clear()) # clears the list without deleting the list
print(f)
```

```
print('\n','# sort()')
# In order to sort the list it must be homogenous
g = [45,75,47,89,84,22,25,7]
g.sort() # print(g.sort) returns > none
print(g) # It modifies the original list itself
h = ['apple','google','yahoo','amazon','facebook','instagram','microsoft']
```

Python List Methods

Input	Method	Output
	<code>.append()</code>	
	<code>.insert(1, )</code>	
	<code>.pop(1)</code>	
	<code>.remove()</code>	
	<code>.reverse()</code>	
	<code>.sort()</code>	
	<code>.index()</code>	2
	<code>.count()</code>	2

```
h.sort(reverse=True) # By default reverse is False
print(h)
h.sort(key=len) # sorts the element wrt their length
print(h)
```

```
print('\n','# index() method')
print(h)
print(h.index('amazon'))
print(h[3]) # access list items
# slicing
print(h[0:6:2])
print(h[-4:-1])
print(h[4:-1])
print(h[1:4])
print(h[-3::-1]) # negative indexing
```

```
print('\n')
print(dir(list))
```

Tuple - Multi value Datatype

- 1) Collection of homogenous or heterogenous data values enclosed between ().
- 2) A = (23,34+6j,9.5,'hello').
- 3) Tuples are immutable elements of the tuple cannot be changed, replaced, appended, deleted after creation.
- 4) Tuples can hold duplicate values.

```
A = (23,34+6j,9.5,'hello')
print(type(A))
print(A)
B = ('Meet Sudra','B',402,'Minal Park')
print(B)
```

List of tuples

```
temperature = [('Mumbai',28),('Delhi',54),('Chennai',43),('Kolkata',24)]
print(type(temperature))
```

Memory allocation in tuple

Indexing

```
print('\n','# Indexing')
names = ('apple' , 'google' , 'yahoo' , 'gmail' , 'amazon' , 'flipkart')
print(names[1])
print(names.index('apple'))
print(names[-1][-1])

company = ('apple' , 'google' , 'yahoo' , ('gmail' , 'amazon') , 'flipkart' )
print(company[3][1][3])

print('\n','# slicing')
print(company[1:4])
print(company[4:1:-1]) # reverse
print(company[::-1])
C = ('apple', 'google', 'yahoo', 'amazon', 'facebook', 'instagram', 'microsoft')

print(C[::-2])
```

Methods

Count() method returns number of occurrences of element in the tuple

```
print('\n','# count() method')
print(C.count('yahoo'))
d=(2,3,2,5,5,6,7,6,5,8,2)
print(d.count(2))
```

index() method returns position number of the specified element.

Set - Multi value Datatype

To be remembered:

Sets do not allow mutable types in it.

S = {square} # lets say square is any function

```
{<function __main__.square(num)>}
```

- 1) Set is unordered non duplicate, homogenous or heterogenous data values enclosed between { }
- 2) Set do not have indexing hence they are unordered.

```
a = {1,2,3,4,6}
print(type(a))
```

```
print(dir(a))
```

```
print(a.add(7))
```

 built in function used to add the new value to the set

```
print(a)
```

```
print(a.remove(7))
```

 built in function used to remove a element from the set

```
print(a)
```

Dictionary - Multi value Datatype

- 1) Collection of key-value pairs.
- 2) Separated by comma operator.
- 3) Each element is associated with unique keys.

4) Length function returns the number of keys present in the dict.

Ways of creating dictionaries

```
A1 = {}          # empty dict
```

```
A2 = dict() # using dict constructor
```

```
A22 = dict(Bangalore=34, Chennai=67, mumbai=56)
```

```
print(A2)
```

```
print(A22)
```

```
print(type(A22))
```

```
A3 = dict([("Bangalore",34),("Chennai",67),("mumbai",56)])
```

```
print(A3)
```

```
A4 = dict({"Bangalore":34, "Chennai":67, "mumbai":56})
```

```
print(A4)
```

```
# tuple inside a dictionary
```

```
B = dict({(11,5):"Meet",(2,6):"Krupal",(14,7):"Nirali"})
```

```
print(B)
```

```
# Nested dictionary
```

```
C = dict({"Khimji Sudra":  
{"Ramji":"Bindu","Sanjay":"Jayshree","Nilesh":"Chandrika","Bhavesh":"Bhavi  
sha"}})
```

```
print(C)
```

```
# list inside dictionary
```

```
D = dict({"triji pedhi":["Swati","Meet","Krupal","Heet","Arav","Het"],"biji  
pedhi":["Ramjibhai","Sanjaybhai","Kailashben","Nileshbhai","Bhaveshbhai"]})
```

```
print(D)
```

Accessing and update values from a dictionary.

```
print('\n','# accessing values')
```

```
print(D.get('biji pedhi'))
```

```
print(D.get('triji pedhi'))
```

```
print('\n','# updating values')
```

```
A4['Bangalore']=35465475999999
```

```
print(A4)
```

```
# methods
```

```
print('\n','# methods on directory')
```

```
print(dir(A4))
```

```
['__class__', '__class_getitem__', '__contains__', '__delattr__',  
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',  
'__getattr__', '__getitem__', '__getstate__', '__gt__', '__hash__',  
'__init__', '__init_subclass__', '__ior__', '__iter__', '__le__', '__len__',  
'__lt__', '__ne__', '__new__', '__or__', '__reduce__', '__reduce_ex__',  
'__repr__', '__reversed__', '__ror__', '__setattr__', '__setitem__',  
'__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get',  
'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

get() method returns the value of the specified key.

```
print('\n','# get method')
```

```
print(A22.get('Banglore'))
```

```
print(A22.get('Rajkot')) # returns None if there is no such key in the dict
```

```
A22['Banglore']=92468414
```

```
print(A22)
```

```
print('\n,' # update method')
```

```
A4.update({'Rajkot':90248,'Bihar':278732})
```

```
print(A4)
```

```
B.update({'(31,10)': 'Binduben', '(3,1)': 'Swatiben'})
```

```
print(B)
```

```
D . u p d a t e ( { " p e l i p e d h i " :
```

```
["Khimji","Mansukhbhai","Amrutlal","Rambhaben","Liliben"]})
```

```
print(D)
```

```
D [ ' p e l i
```

```
pedhi]=["Khimji","Mansukhbhai","Amrutlal","Rambhaben","Liliben","xyz"]
```

```
print(D)
```

```
print('\n,'# setdefault method')
```

```
Data = {'Banglore':12,'Chennai':34,'Mumbai':56}
```

```
print(Data)
```

```
Data.setdefault('Mangrol') # setdefault method adds the specified key with  
None as a value if not specified
```



```
print(Data)
Data.setdefault('London',78)
print(Data)
```

```
print('\n','# fromkeys() method')
company = {'apple','microsoft','facebook'}
namethem = dict.fromkeys(company,'Mumbai')
print(namethem)
thiscomp = dict.fromkeys(company)
print(thiscomp)
```

```
print('\n','# items() method')
# item method returns key-value pairs in the form of tuple in the list
Dictionary1 = { 'A': 'Geeks', 'B': 4, 'C': 'Geeks' }
print(Dictionary1.items())
```

```
print('\n','# keys() method')
# returns all the keys in the dictionary
print(Dictionary1.keys())
```

```
print('\n','# values() method')
# returns all the values in the dictionary
print(Dictionary1.values())
```

```
print('\n','# pop() method')
print(Dictionary1.pop('C'))
print(Dictionary1)
```

```
print('\n','#popitem() method')
# popitem() method removes the last item added in the dictionary
Alphavsnum = {
    'A':1,
    'B':2,
    'C':3,
    'D':4
}
print(Alphavsnum.popitem())
```

```
# Merging dictionaries
print('\n')
```

```
M = {
    'Fname':'Meet Sudra',
    'Fid': 12345
}
N = {
    'Faddress':'Mumbai',
    'Fage':21
}
```

```
mn = {**M,**N}
print(mn)
xy = M|N
print(xy)
```

difference between the pop() and popitem() method

pop() can remove any item from a dictionary as long as you specify the key. On the other hand, popitem() can only remove and return the value of the last element.

```
print('\n','# activity')
# increment the value of b
print('\n')
points = {'a': 1, 'b': 2, 'c': 3}
points['b']=3
print(points)
```

create a dictionary with "No value" as value for all the keys.

```
print('\n')
I = ["a", "b", "c", "d"]
valuesofi = dict.fromkeys(I,'No value')
print(valuesofi)
```

Merge two dictionaries.

```
print('\n')
merge = points|valuesofi
print(merge)
```

points = {'a': 1, 'b': 2, 'c': 3}, add one more key "d" with value 4

```
print('\n')
alphanum = {'a': 1, 'b': 2, 'c': 3}
```

```
alphanum.setdefault('d',4)
print(alphanum)
```

```
# dictt = {"flowers": ["Rose", "lily", "lotus"], "animals": ["Cat", "dog"]}
# Add "sunflower" to the flowers list and "Lion" to animals
print('\n')
dictt = {"flowers": ["Rose", "lily", "lotus"], "animals": ["Cat", "dog"]}
dictt['flowers'].append('sunflower')
dictt['animals'].append('Lion')
print(dictt)
```

Properties	String	List	Tuple	Set	Dictionary
Boundary	str() or variable =" "'"	[]	()	{ }	dict({ key: value})
Empty datatype	str() or Empty quotes	list() or []	tuple() or ()	set()	dict() or { }
Mutable	No	Yes	No	Yes	Yes
Indexing and slicing	Yes	Yes	Yes	No	No
Duplicates	Allowed	Allowed	Allowed	Not allowed	Key duplication not possible
Datatypes allowed	All	All	All	Immutable / hashable	Keys - washable values - any

Sequence vs iterables

- 1) A sequence is a object which can be indexed. For eg. Strings, Lists, Tuples.
- 2) All sequences are iterables. But all iterables are not sequences. For eg. Sets and dictionaries.
- 3) Note: All sequences can be merged using concatenation operator(+).

Object Oriented Programming

- 1) Python is object oriented language because it allows us to write user defined class and datatypes
- 2) Major Principles of oop are :

Class

Object

Inheritance

Polymorphism

Abstraction

Encapsulation

Class

Class is a set/collections of functions/methods that carries out various operations on the data.

Class is a blueprint of object which consists of functionalities of real time entities.

Two types of classes

- Inbuilt class, It is a pre-defined class with predefined method in it.
- User defined class, It is user defined class based on user requirement.

Object

Object is a instance of a class. It is a variable created for a particular class

Class creation

```
class Person:
    # properties
    name = 'Meet Sudra'
    age = 21
    occupation = 'Aspiring Data Scientist'
```

accessing class members -- class_name.propertyname

```
print(Person.name)
print(Person.age)
```

Object Creation

```
a = Person()
```

```
# accessing object members -- object_name.propertyname
print(a.name)
print(a.occupation)
```

```
# Object Creation
b = Person()
# Modifying the values
b.name = 'Heet Sudra'
b.age = 14
b.occupation = 'Student'
print(b.name)
print(b.age)
print(b.occupation)
```

Output :

```
Meet Sudra
21
Meet Sudra
Aspiring Data Scientist
Heet Sudra
14
Student
```

Note:

If we do modification with respect to class then it will affect class along with the object

Changes with respect to object modify that particular object only

Memory allocation for class

When python interpreter encounters class keyword it creates a dictionary in the memory to store class methods and attributes.

Classes have two types of states/properties/members

1. Generic / static / Class members:

– Class variables

Definition : Variables that are stored across all the instances of the class

usage : Useful for the data that should be consistent across all the instances of the class

access : Can be accessed by class name or any instance.

- Class methods

Definition: Methods that are bound to the class and not the instance.

Usage: Defined using the @classmethod decorator. The first parameter is cls, which refers to the class itself.

Access: Can be called using the class name or an instance.

2. Instance level members

- Instance variables

Definition: Variables that are unique to each instance of a class

Usage: Useful for data that vary from instance to instance

Access: Can be accessed using the instance

- Instance methods

Definition: Methods that are bound to the instance of the class

Usage: Defined normally without any decorators. The first parameter is 'self', which refers to the instance itself.

Access: Can be called using the instance itself.

__init__ method

Every time u initiate using __init__() method, method gets called every time u create an instance for example:

```
class Vehicle:
    def __init__(self):
        print('Hey i am vehicle')
```

```
a = Vehicle()
b = Vehicle()
c = Vehicle()
```

Output:

```
Hey i am vehicle
Hey i am vehicle
Hey i am vehicle
```

```
print('\n', '# Example4')
class bank():
    bname = 'BOI'
```

```

    CEO = 'Meet sudra'
c1 = bank()
c2 = bank()
c3 = bank()
def c_details(org,cname,cid):
    org.cname = cname
    org.cid = cid
c_details(c1,'Heet Sudra',16510110008418)
c_details(c2,'Mila repa',16510110007635)

print(c1.cname)

print('\n','# Example6')
class bank2:
    bname = 'State bank of India'
    CEO = 'Bindu Sudra'

    def __init__(self,cname,cid): # ensure __init__() method is defined
inside the class
        self.cname = cname
        self.cid = cid

c1 = bank2('Krupal Sudra',876543)
c2 = bank2('Het Sudra',345678)

print(c1.cname)

```

Is it necessary to use self argument to create a method in a class?

Ans is NO we can use @static method without using self as an argument

@static method

Static method neither belongs to class neither belongs to instance it acts as supportive method for class and object

```

class maths:
    @staticmethod
    def add(a,b):
        return a+b

```

```
aa = maths()  
print(aa.add(2,4)) # instance.method()  
print(maths.add(3,7)) # class.method()
```

@classmethod

Class method is used to access and modify the members of the class.

```
def changecomp(cls,newcomp): # here by default it takes cls as instance  
    cls.company = newcomp
```

```
@classmethod
```

```
def changecomp(cls,newcomp): # here it takes cls as class  
    cls.company = newcomp
```

Object method

Object method is used to access and modify the members of the objects.

Inheritance

- 1) Inheritance is the property of deriving properties of one class to another class.
- 2) In this case the class from which we inherit the properties is called parent class/ base class/ super class.
- 3) The class to which we inherit the properties is called the child class or sub class or derived class.
- 4) Inheritance is a mechanism for creating a new class that specialises or modifies the behaviour of existing class
- 5) When a class is created via inheritance, It inherits all the attributes in the parent class. However a derived class may define any of these attributes and can add new attributes on its own !

single level inheritance

It is the phenomenon of deriving properties from a single parent class to single child class

Constructor Chaining


```
print('\n','Example2 constructor chaining')
class Vehicle:
    def __init__(self,engine):
        print('inside vehicle Constructor')
        self.engine = engine
```

```
class Car(Vehicle):
    def __init__(self,engine,max_speed):
        super().__init__(engine)
        print('Inside car constructor')
        self.max_speed = max_speed
```

```
class Electric_car(Car):
    def __init__(self,engine,max_speed,km_range):
        super().__init__(engine,max_speed)
        print('Inside electric_car')
        self.km_range = km_range
```

object of electric car

```
e1 = Electric_car('1000cc',190,900)
print(e1.engine,e1.max_speed,e1.km_range)
```

Constructor chaining refers to a process where child class constructor calls the constructor of its parent class. This ensures that the initialisation defined in the parent class is executed when object of child class is created

```
print('\n','Example4 ')
class Parent:
    def __init__(self,name):
        self.name = name
class Child(Parent):
    def __init__(self,name,age):
        super().__init__(name) # super(). decorator is used
        self.age = age
ch1 = Child('Meet',21)
print(ch1.name,ch1.age)
```


FUNCTIONS

GLOBAL AND LOCAL VARIABLE

Variables in the global frame are referred as global variables. Global variables are those who come in programs scope.

Whereas those created after calling the function are called local variables. Local variables come in functions scope.

y=5 ko u can use in g()
But program won't use y=5

global thodi use karega local ko
local uses global

To be remembered!!!!!!!

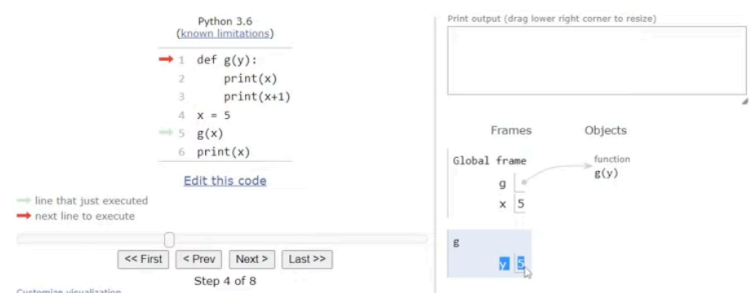
Function (local) can use global variable but can't change it.

```
def h(y):  
    x+=1      #idher fat jayega  
x=5  
h(x)  
print(x)
```

Option bhi deta he python, "global" keyword which tells python that dekh me global me modification kar raha hu

```
def h(y):  
    global x  
    x+=1  
x=5  
h(x)  
print(x)
```

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)



Functions are 1st class citizens

Jo kam datatypes karte he (in terms of operations) functions do them as well.

DATASTRUCTURES AND ALGORITHMS

Linear and Non-linear data structures

Data structures where data is stored linearly are called linear data structures.

Whereas where data is non-linear are called non-linear data structures.

Linear ds : arrays, linked lists, stacks, queues and hashing

Non-Linear ds : trees, graph

Arrays

Array is a linear ds used to store multiple item of same type in continuous memory location.

Marks of students in class

75 89 92 76 59 63

1000 1004 1008 . . .

if starting address of an array is 1000 and int takes 4bytes(32bits) of space the next elements would be addressed 1004 10008 and so on.

This is called **call by value**

Disadvantages of Arrays:

- Fixed storage # in c c++ we declare int A[50] lets say used gave 2 tasks only means again memory wastage!
- Homogenous (same type)

I. Referential arrays

Values are first stored in different memory locations

2	3	4	5	6	(VALUE)
0012	3829	2438	3286	3284	(ADDRESS)

Then these addresses are stored in an array where each element stores address to the values

0012 3829 2438 3286 3284

REFERENTIAL ARRAYS ARE REFFERED AS LISTS IN PYTHON

Lets say I want to add Hello

2	3	'Hello'	5	6	(VALUE)
0012	3829	3248	3286	3284	(ADDRESS)

0012 3829 3248 3286 3284

still Homogenous but stored heterogeneous values

This is called **call by reference**

II. The issue of fixed size in array is resolved using the concept of **dynamic array (LIST)**

Lets say I have an array of size 1 where in I stored value "1" as user said.

| 1 |

Now user wants to store 2. Thus I will double the size of previous array and copy "1" in the new array along with "2".

| 1 | 2 |

Now user wants to store 3. Again the same process **double > copy previous elements > store new element** (doubling the list depends)

| 1 | 2 | 3 | |

Now user want to store 4. Simply add 4 to the remaining space

| 1 | 2 | 3 | 4 |

Now user wants to store 5. Again the same process double > copy previous elements > store new element

| 1 | 2 | 3 | 4 | 5 | | |

Altho these are still static !

IN PYTHON LISTS ARE DYNAMIC ARRAYS

	Dynamic Arrays List in Python	Arrays	Numpy Array
	Heterogenous	Homogenous	Homogenous
Computation power	Slow	Fast then list	Fastest
Memory consumption	High	Low then lists	Lowest

NUMPY IN PYTHON

Library in python used for scientific operations. The core of numpy package is the **ndarray object**. This encapsulates n-dimensional arrays of homogenous datatype.

Numpy is built in C language but is wrapped in such a manner that it is used in python

Why Numpy was introduced when Lists were already there?

Lists in python become slow in term of performance, they are not optimised for numerical operations. Also Numpy arrays take less memory then lists.

Why Numpy over List?

Numpy is convenient fast in terms of time and since it is flexible with respect to space