# Python Introduction:

## Overview:

* General purpose high level language -- designed to be used in a range of applications, including data science, software and web development, automation, and generally getting stuff done
* Object oriented, High level language -- **Object-oriented means this language is based around objects (such as data) rather than functions, and high-level means it's easy for humans to understand**

Basically, there are two main categories of computer languages, namely Low Level Language and High Level Language. Let us take a brief look at both these types of computer languages

## *1] Low Level Languages*

**Low level languages are the basic computer instructions or better known as machine codes. A computer cannot understand any instruction given to it by the user in English or any other high level language. These low level languages are very easily understandable by the machine.**

The main function of low level languages is to interact with the hardware of the computer. They help in operating, syncing and managing all the hardware and system components of the computer. They handle all the instructions which form the architecture of the hardware systems.

### Machine Language

This is one of the most basic low level languages. The language was first developed to interact with the first generation computers. It is written in binary code or machine code, which means it basically comprises of only two digits – 1 and 0.

### Assembly Language

This is the second generation programming language. It is a development on the machine language, where instead of using only numbers, we use English words, names, and symbols. It is the most basic computer language necessary for any processor.

## *2] High Level Language*

When we talk about high level languages, these are programming languages. Some prominent examples are PASCAL, FORTRAN, C++ etc.

The important feature about such high level languages is that they allow the programmer to write programs for all types of computers and systems. Every instruction in high level language is converted to machine language for the computer to comprehend.

**In a simple way high level language means human understandable language**

### Scripting Languages

Scripting languages or scripts are essentially programming languages. These languages employ a high level construct which allows it to interpret and execute one command at a time.

Scripting languages are easier to learn and execute than compiled languages. Some examples are **AppleScript, JavaScript, Pearl etc.**

### Object-Oriented Languages

These are high level languages that focus on the **‘objects’ rather than the ‘actions’**. To accomplish this, the focus will be on data than logic.

The reasoning behind is that the programmers really cares about the object they wish to manipulate rather than the logic needed to manipulate them. Some examples include **Java, C+, C++, Python, Swift** etc.

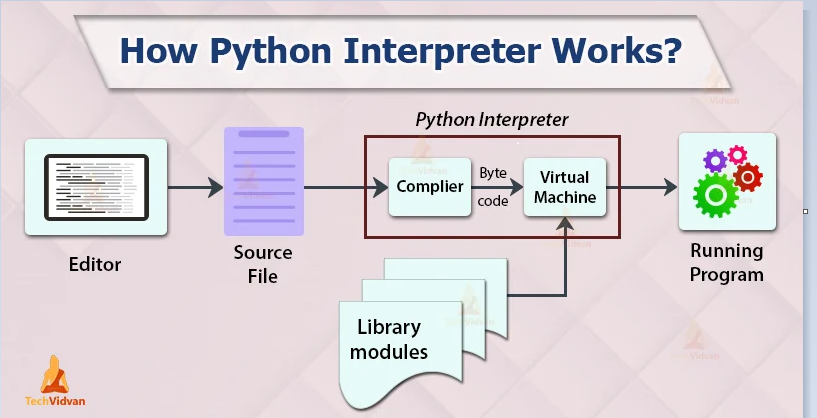
### Procedural Programming Language

This is a type of programming language that **has well structured steps and complex procedures** within its programming to compose a complete program.

It has a systematic order functions and commands to complete a task or a program**. FORTRAN, ALGOL, BASIC, COBOL are some examples.**

* It’s a interpreted language

A Compiler displays every error and warning while compiling. So, you can't run this program unless you fix the errors. An Interpreter reads every statement, then displays the errors, if any. A user must resolve these errors in order to interpret the next line.



The Python interpreter initializes its runtime engine called PVM which is the Python virtual machine. The interpreter loads the machine language with the library modules and inputs it into the PVM. This converts the byte code into executable code such as 0s and 1s (binary).

* It supports dynamic memory allocation
  + Memory is allocated to the objects at the run time. We use the Heap for implement dynamic memory management. We can use the memory throughout the program. As we know, everything in Python is an object means dynamic memory allocation inspires the Python memory management.

## Features of python:

* Readable syntax
* Cross platform working like ( windows, mac, linux)
* Large standard libraries
* Open source

# Identifiers:

User defined names to represent a vatriable, function, class

* If you assign a name to a programmable entity then in python it’s called as identifier
* To check whether the identifier is valid or not in python we have a method

‘string’.isidentifier()

**Rules to create Identifier:**

* An identifier should starts with either alphabet or underscore
* An identifier can be of with alphanumeric characters. But should starting letter either alphabet or \_ only
* Any of the special character should not included in identifier except ‘\_’
* An identifier should be written in smaller case only, it’s a python convention
* Keywords cannot be used as variables
* Python doc says we can have unlimited length of identifier but as per convention rule the total length would be 79 characters only.

## Keywords:

Keywords are the reserved words and having special meaning.

To get all the keywords: help(‘keywords’)

Import keyword

Keyword.kwlist()

To check whether string is keyword or not : **import keyword**

**keyword.iskeyword(‘string’)**

**In python 3.11 we have 35 keywords**

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import keyword

keyword.kwlist

['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']

print(len(keyword.kwlist))

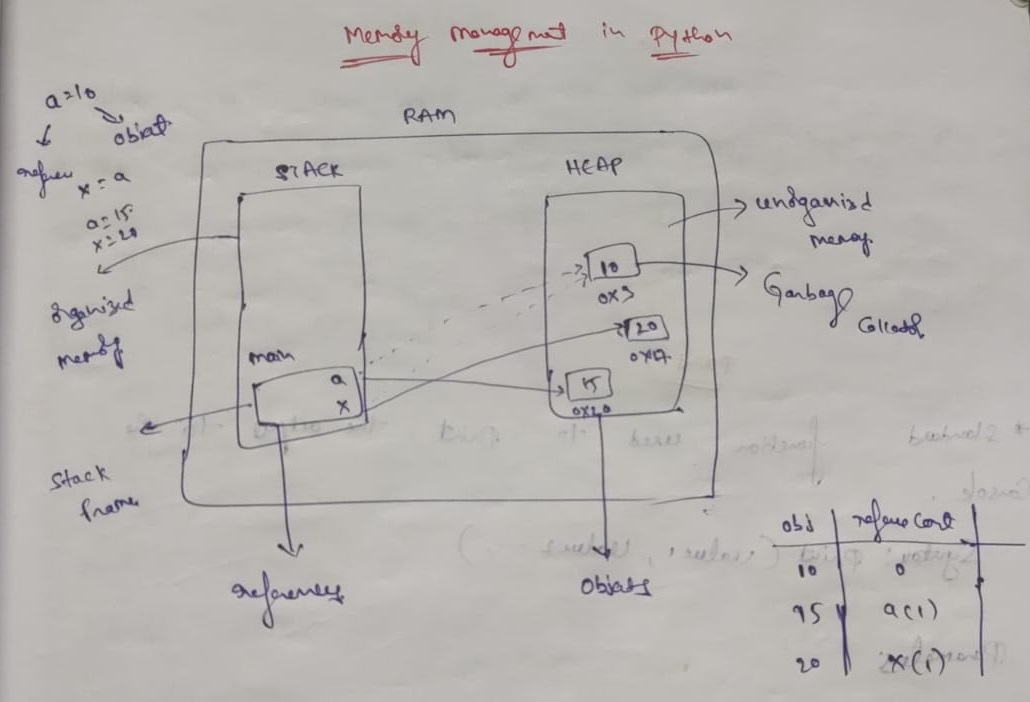
35

## Python variables:

* The name given to a memory location which hold the actual value is called as variable
* A variable can holds the object of different data types
* A variable should be in lower case and if it is in more than one word ot should be seperated by ‘\_’ it is a python convention

**Synatax : variable\_name = value**

## Memory management in python:



When we created a variable, internally RAM is created with stack & heap space. In stack the references are stored in main frame called as stack frame. In Heap space the objects are stored.

Stack space is called as organized memory & heap space will be called as Un-organized memory. The references in the stack space will be pointing to objects in the heap space through the address

## Data types:

* Data type represents what type of data being stored in a memory location.
* A variable can holds values. Every value will have the data type
* As python is a dynamic language no need of defining type of data while declaring a variable. The interpreter blinds the type of data with value implicitly.
* type( ) 🡪 returns the data type of the variable passed
* dir() 🡪 returns the all attributes of a given variable or function or method

Data types are two types

1. Individual data types
2. Collection data types

**Individual data types:**

1. Integer
2. Float
3. Complex
4. Boolean

**Integer:**

* It specifies the data stored in a memory location is integer
* It can be of positive or negative

**Ex:**

a = 1000

print(type(1000))

<class 'int'>

**Operations on integer:**

**Addition 🡪 3+2**

**Subtraction 🡪 3-2**

**Multiplication 🡪 3\*2**

**Division 🡪 3/2**

**Floor division 🡪 3//2 (gives the nearest integer which is not greater then result)**

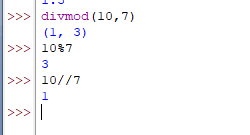
Ex: Normal division 🡪 3/2 🡪 1.5

Floor division 🡪 3//2 🡪 1

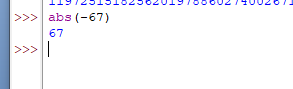
**Exponent: 3\*\*2**

Modules: 3%2

divmod(x,y): returns (x//y, x%y)

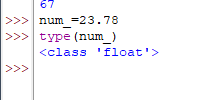


abs(): converts negative numbers into positive



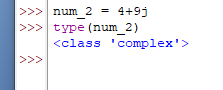
**Float:**

* it specifies the data stored in a memory location is a type decimal
* it is represented by float



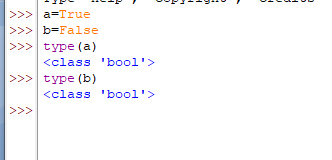
**Complex:**

* written In the form of a+bj, where a is the real part and b is the imaginary part
* It can be of positive or negative
* It is represented by complex

****

**Boolean:**

* Boolean values are represented by True and False keywords
* Useful in conditional expressions
* It is represented by bool

****

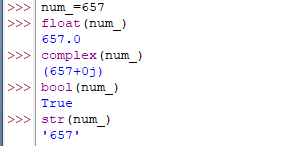
**Type casting:**

The process of converting one data type into another data type is called as type casting.

**Integer:**

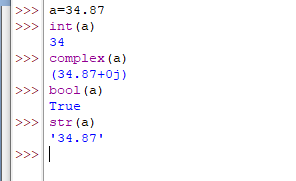
* Integer can be converted into all single data types

Ex:



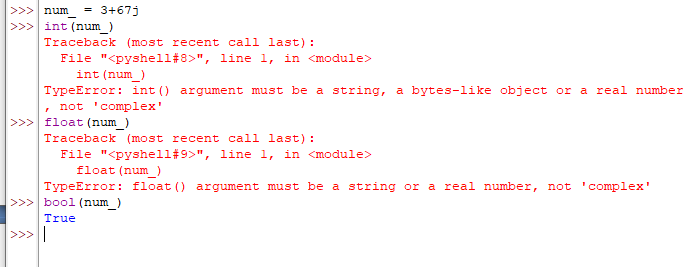
**Float:**

* Float also can be converted into all single data types

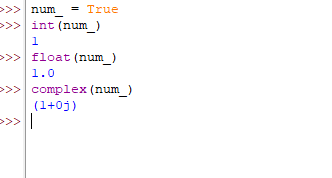


**Complex:**

* Complex type cannot converted into any of the single data type except Boolean



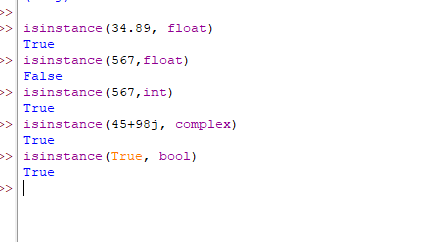
**Boolean:**



**Isinstance:**

* returns true if the specified object is of specified data type otherwise returns false

**Syntax: isinstance(value, datatype)**

****

## Mutable & Immutable data type difference:

**Mutable data type:**

* Allow modification on the original object
* **Eg: list, set, dictionary**

**Immutable data type:**

* Does not allow modification on the original object
* Even if the modification is done new object will be created by keeping original object

**Ex: string, tuple, integer, float, complex, boolean**

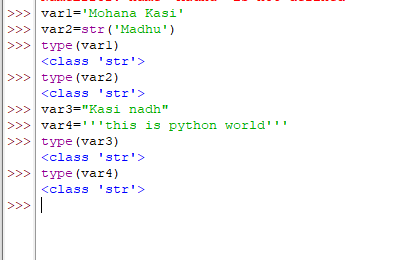
## Collection data types:

**String:**

* set or collection of characters
* Boundary: ‘-----‘ or “-----“ or ‘’’--------‘’’
* String is a immutable data type
* Length of the sting will be find out by inbuilt function called len

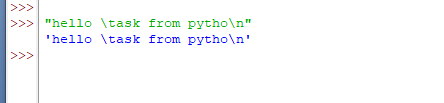
len(variable\_name) or len(string)

**Synatx: variable\_name = string**

****

**Raw strings:**

* Python raw strings are the string literals prefixed with r or R.
* Raw strings do not treat backslashes as an escape sequence. It will print normally as a string
* Usually we will use this raw string in editors like pycharm only. In python IDE back slashes will consider as normal string it self

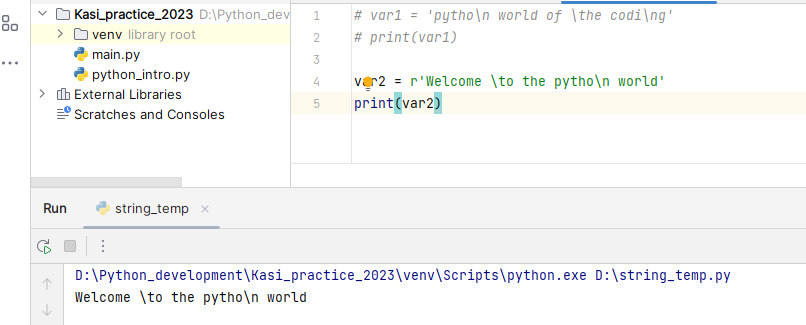
****

**Normal string:**

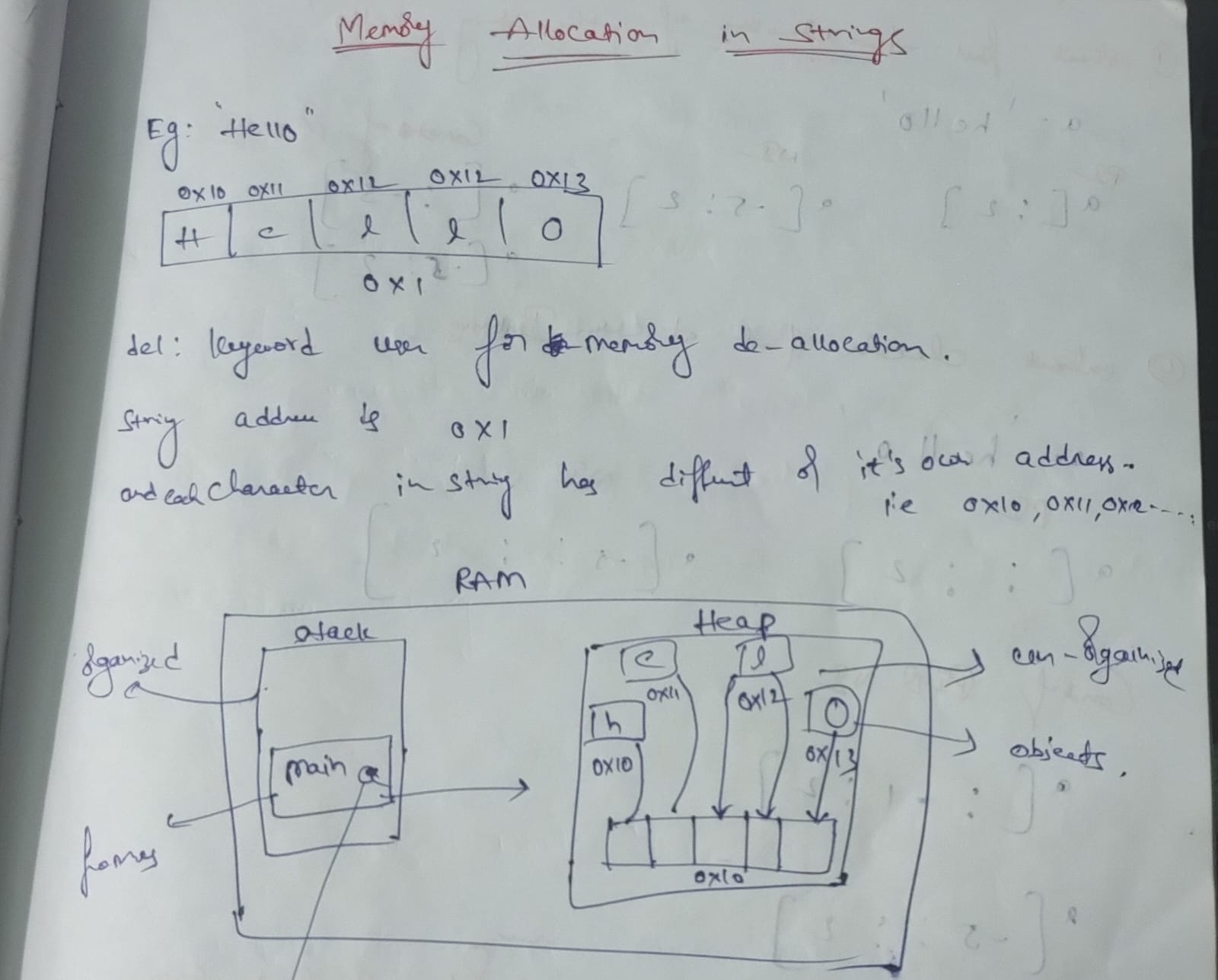
var1 = 'pytho\n world of \the codi\ng'  
print(var1)



**Raw string:**



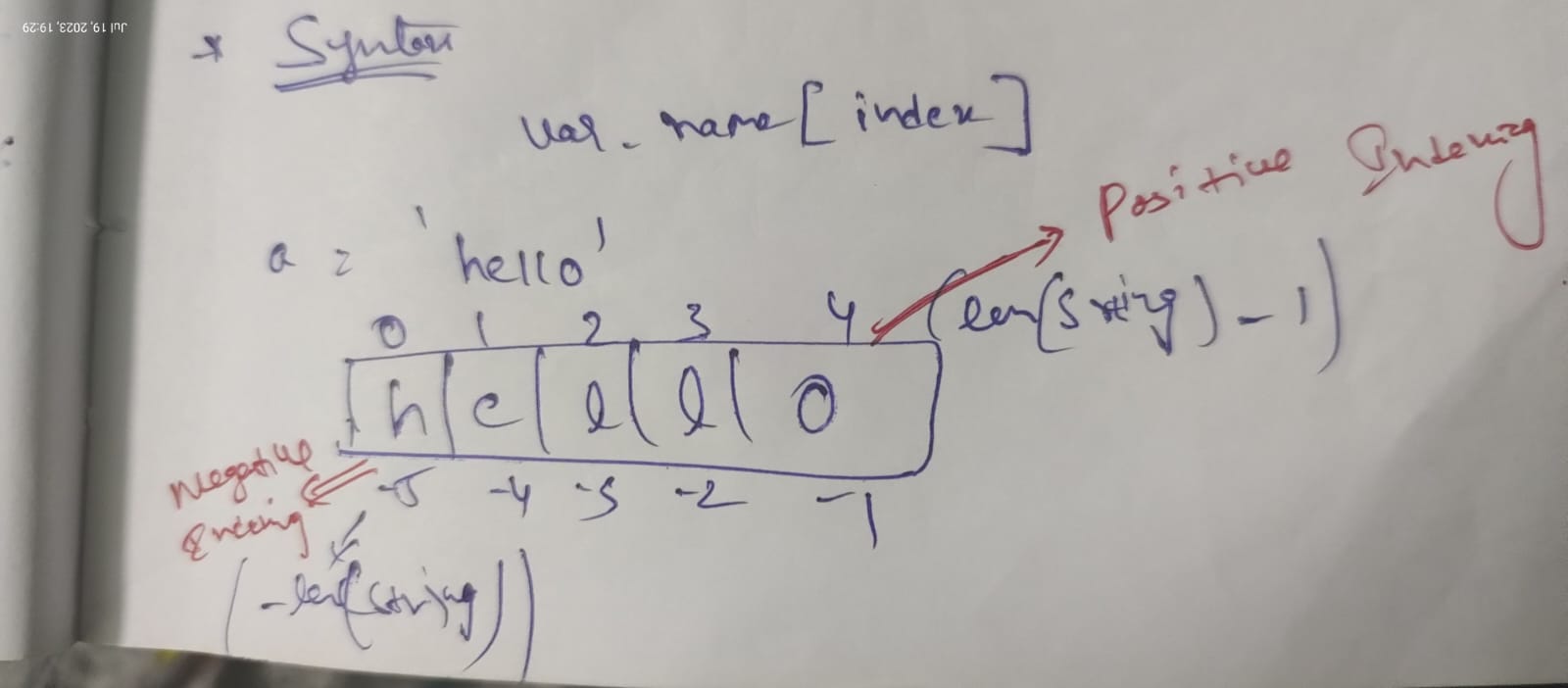
**Memory allocation in strings:**

****

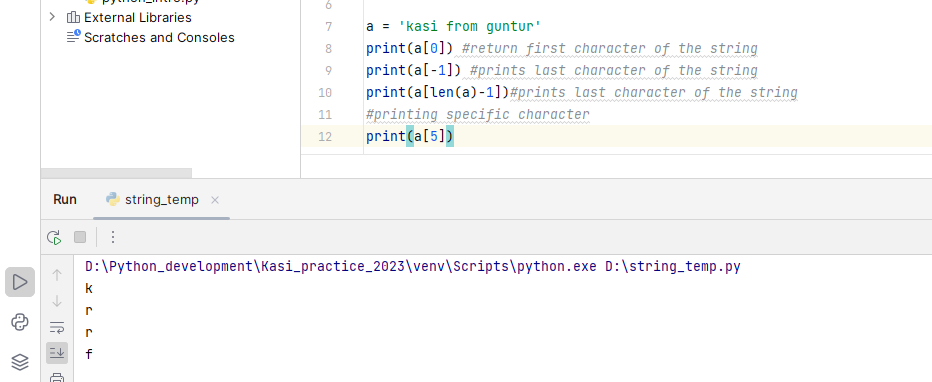
**Indexing:**

* The process of extracting single character at a time
* Indexing can be of positive or negative
* The index values starts from 0 and ending index will be len(string)-1

Syntax: var\_name[index]

****

**Examples:**

****

**Slicing:**

* The process of extracting multiple characters at a time

**Syntax: var\_name[starting index: Ending index: step value]**

In simple 🡪 var\_name[SI:EI:Step]

**Starting index 🡪 if you not mention starting index it will take default value as 0 that too you need to make empty before colon**

Ex: var\_name[ :8]

**Ending index🡪 🡪 if you not mention ending index it will take default value as len(string-1) that too you need to make empty afer colon**

Ex: var\_name[3: ]

**Step value 🡪 default value will be 1 when you make empty**

**Ex:** var\_name[1:2:]

a = 'kasi from guntur'

To print first two characters of the string

* print(a[:2])

To pring characters present at the even indexes

* print(a[::2])

To print characters present at the odd indexes

* print(a[1::2])

To print characters in the reverse order

* print(a[::-1])

To print characters in the reverse order of even indexes

* print(a[::-2])

## String formatting:

Infusing data dynamically into the strings at the time of execution is called as string formatting.

There are three types of string formatting

1. String formatting with place holders
2. String formatting with .format() method
3. String formatting with string literals called f strings

**String formatting with place holders:**

* It is the oldest method of string formatting
* String formatting is possible by using % (modules) operator
* Here % operator will be also called as string formatting operator

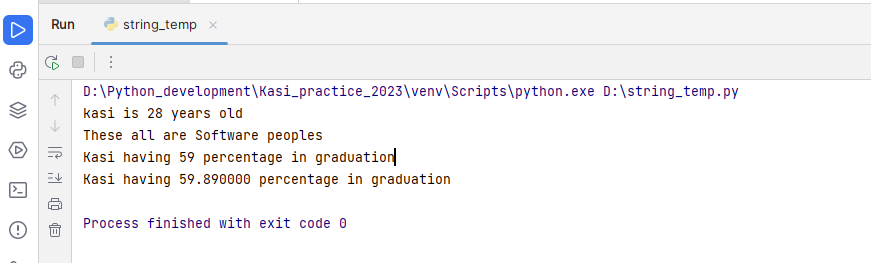
**‘%s’ 🡪 is used to inject strings**

**‘%d’ 🡪 is used to inject integers**

**‘%f’ 🡪 is used to inject float values**

**Examples:**

string\_ = %s is %d years old % (kasi, 28)  
print(string\_)  
  
string2\_ = 'These all are %s' % 'Software peoples'  
print(string2\_)  
  
string3\_ = Kasi having %d percentage in graduation % 59.89 *# when you not using %f you will not get decimal values*print(string3\_)  
  
string4\_ = Kasi having %f percentage in graduation % 59.89  
print(string4\_)

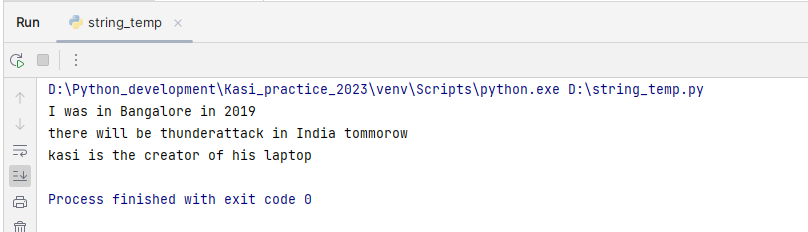
****

**String formatting using .format() method:**

* Here formatting is done by using .format() method
* Place holders {} are used to infuse the data into the string at the run time

**Examples:**

s = I was in {} in {}.format(Bangalore, 2019) #default takes indexes  
print(s)  
  
string1\_ = there will be {1} in {0} tommorow.format(India,thunderattack)   
print(string1\_)  
  
string2\_ = {name} is the creator of his {item\_name}.format(name='kasi',item\_name=laptop)  
print(string2\_)

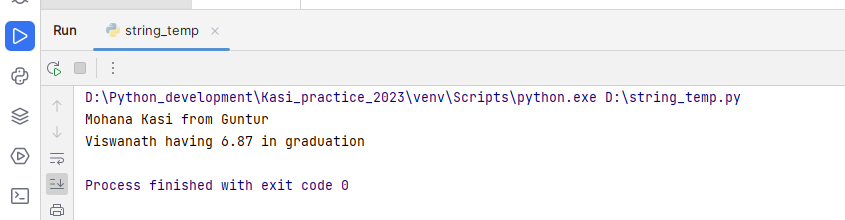
****

**String formatting with string literals f-strings:**

* Here outside variables are used to infuse into the string instead of passing them as arguments
* We will use {variable\_name} place holders to infuse strings at the time of execution

**Examples:**

name\_ = 'Mohana Kasi'  
place = 'Guntur'  
print(f{name\_} from {place})  
  
name2\_ = Viswanath  
percenrage = 6.87  
  
print(f{name2\_} having {percenrage} in graduation)

****

**Activity:**

**S = Hi welcome to python**

* **Print every alternative character in string**
* **Print alternative characters in revered order**
* **Print string in reverse order**

a = 'youtube.txt'

* **Print extension of file name**
* **Print only file name**

link\_ = 'https://web.whatsapp.com/'

* **Print only protocall in URL**
* **Print only domain name**

S = Hi welcome to python  
*#Print every alternative character in string*print(S[::2])  
  
*#Print alternative characters in revered order*print(S[::-2])  
*#Print string in reverse order*print(S[::-1])  
  
a = 'youtube.txt'  
*#Print extension of file name*print(a[8:])  
print(a.split('.')[1]) *#using inbuilt method  
  
#Print only file name*print(a[:7])  
print(a.split('.')[0]) *# inbuilt method*link\_ = 'https://web.whatsapp.com/'  
  
*#print only protocal in URL*print(link\_[:5])  
print(link\_.split(':')[0])  
*#print only domain*print(link\_[8:])  
print(link\_.split(':')[1])

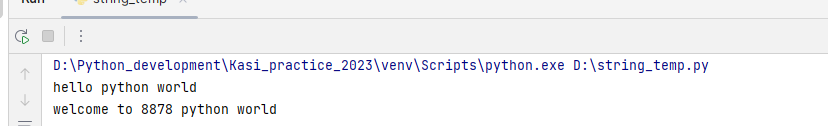
## String inbuilt methods:

**lower():**

* It will converts a string into lower case
* **Syntax: string.lower()**
* Returns string as output

**Examples:**

string\_ = Hello PYTHON World  
print(string\_.lower())  
  
string2\_ = welcome to 8878 PYTHon WORld  
print(string2\_.lower())

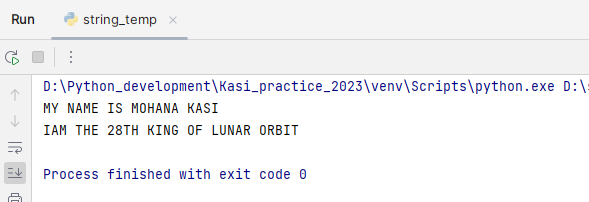
****

**Upper():**

* It will converts a string into Upper case
* **Syntax: string.upper()**
* Returns string as output

**Examples:**

string\_ = MY NAME is Mohana Kasi  
print(string\_.upper())  
  
string2\_ = Iam the 28th king of lunar orbit  
print(string2\_.upper())

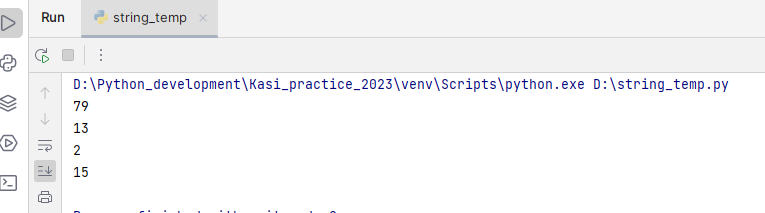
****

**Count():**

* Returns the number of occurrences of a specified substring in a string
* **Syntax: string.count(substring)**

**Examples:**

string\_ = we have only less time to complete this entire task and we have less people too  
print(len(string\_))  
print(string\_.count('e'))  
print(string\_.count(we))  
print(string\_.count( ))

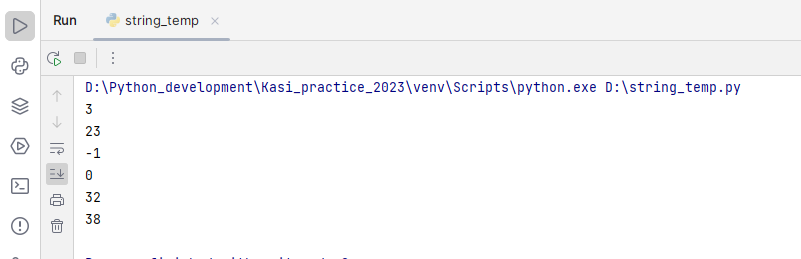
****

**Find()**

* It will search the string for a specified value & return the position where it found
* Find the first occurrence of specified value
* It will return -1 if the value not found
* **Synatx: string.find(value,[SI],[EI])**

Examples:

string1\_ = 'Where is the king of persia and Where is his queen'  
print(string1\_.find('r'))  
print(string1\_.find('r',4))  
print(string1\_.find('where')) *#proper case we need to give while searching*print(string1\_.find('Where'))  
print(string1\_.find('Where',4))  
print(string1\_.find('is',8))

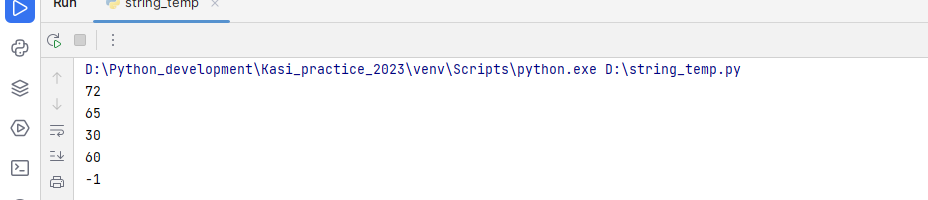


**rfind():**

* Searches the string for a specified value & returns the last position of where it is found

Example:

string2\_ = 'please process the desighn of the cylinder module & desighn of helix too'  
print(len(string2\_))  
print(string2\_.rfind('l'))  
print(string2\_.rfind('the'))  
print(string2\_.rfind('of'))  
print(string2\_.rfind('where'))

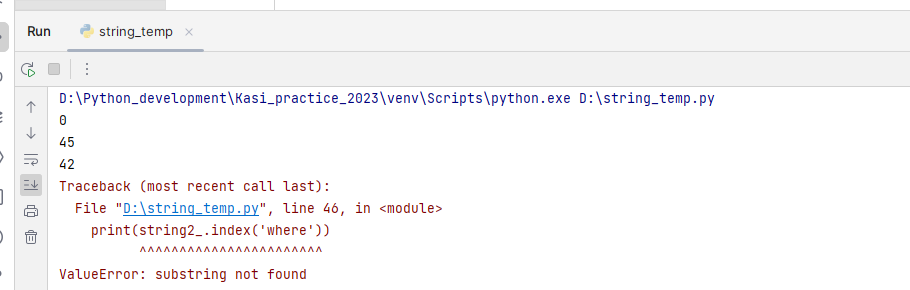


**Index():**

* It will search the string for a specified value & return the position where it found
* Find the first occurrence of specified value
* It will return error if the value not found
* **Synatx: string.index(value,[SI],[EI])**

Example:

string2\_ = the generation of human has been modified of the furure scope  
print(string2\_.index('the'))  
print(string2\_.index('the',5))  
print(string2\_.index('of',16))  
print(string2\_.index('where'))

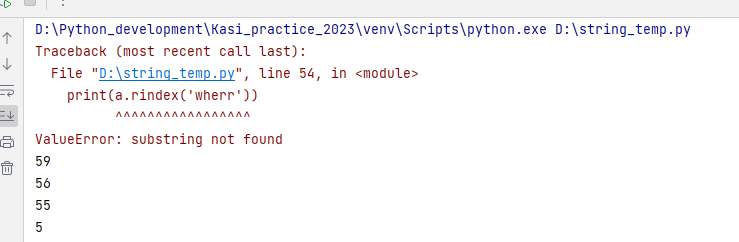


**rindex():**

* Searches the string for a specified value and returns the last position where it is found
* Returns error if substring not found

Examples:

a = 'this one has the fow of the river of krishna & brahma & this'  
print(a.rindex('s'))  
print(a.rindex('this'))  
print(a.rindex(' '))  
print(a.rindex('one'))  
print(a.rindex('wherr'))

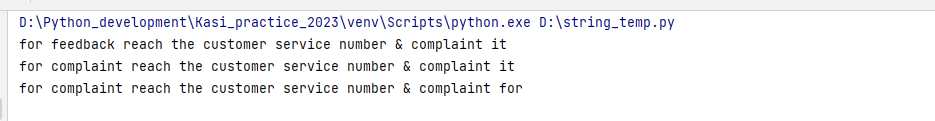


**replace():**

* Replaces a specified phrase with a new phrase given
* **Synatx: string.replace(old value, new value, [count])**

Examples:

string\_ = 'for complaint reach the customer service number & complaint it'  
print(string\_.replace('complaint','feedback',1))  
print(string\_.replace('complaint','feedback',0))  
print(string\_.replace('it','for'))



**Starts with():**

* Returns true if the string starts with the specified value or else returns false
* **Syantax: string.startswith(value,[si],[ei])**

Example:

string\_ = 'its very nice to meet you and its pleasure too'  
print(string\_.startswith('its'))  
print(string\_.startswith('is'))  
print(string\_.find('its',5))  
print(string\_.startswith('its',30))

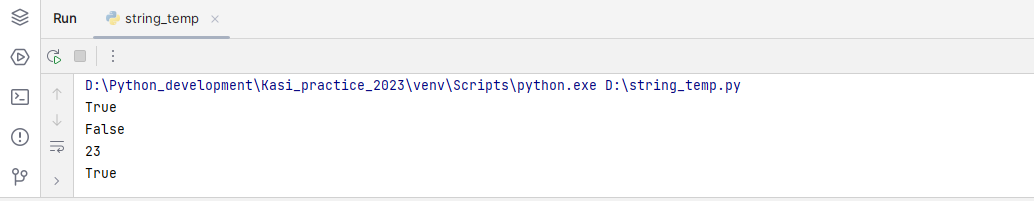


**Ends with():**

* Returns true if the string ends with the specified value or else returns false
* **Syantax: string.endswith(value,[si],[ei])**

Example:

s = 'its more far from here to reach early'  
print(s.endswith('early'))  
print(s.endswith('is'))  
print(s.find('to'))  
print(s.endswith('early',23))



**split():**

* Split the string at the separator and returns a list of strings
* If no separator is given it will separates based on the space as default separator
* If separator is given and if its matched in the string, it will separate string into two separate strings
* **Syntax: string.split([separator], [max split])**
* rsplit() method splits a string starting from the right. If no max not specified it will work as normal split() method

Example:

string\_ = 'there are,no more startagies, only, one way, need to folow'  
print(string\_.split())  
print(string\_.split(','))  
print(string\_.split(',',1))  
print(string\_.split(',',2))  
print(string\_.rsplit(',',2))  
print(string\_.rsplit(',')) *#seperates as like split() as max split not specified*string2\_ = 'take: one, alighn it to five:say to: cross all, and: do the margin fit'  
print(string2\_.split(':',2))

**o/p:**

['there', 'are,no', 'more', 'startagies,', 'only,', 'one', 'way,', 'need', 'to', 'folow']

['there are', 'no more startagies', ' only', ' one way', ' need to folow']

['there are', 'no more startagies, only, one way, need to folow']

['there are', 'no more startagies', ' only, one way, need to folow']

['there are,no more startagies, only', ' one way', ' need to folow']

['there are', 'no more startagies', ' only', ' one way', ' need to folow']

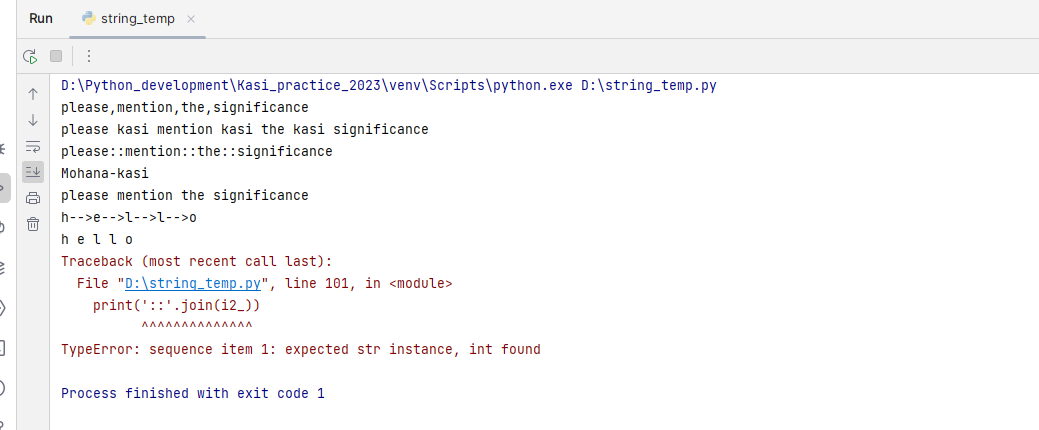
['take', ' one, alighn it to five', 'say to: cross all, and: do the margin fit']

**Join():**

* joins the elements in the iterable using the string specified
* In order to join elements of iterable using join, the elements must be of type string else type error will be raised
* After joins it will return the string
* Syntax: string.join(iterable)

Example:

elems = ['please', 'mention', 'the', 'significance']  
print(','.join(elems))  
print(' kasi '.join(elems))  
print('::'.join(elems))  
print('-'.join(['Mohana','kasi']))  
print(' '.join(elems))  
  
string\_ = 'hello'  
print('-->'.join(string\_))  
print(' '.join(string\_))  
  
i2\_ = ['only',3, 'chances', 'left']  
print('::'.join(i2\_)) *# returns error as iterable having integer as the element*

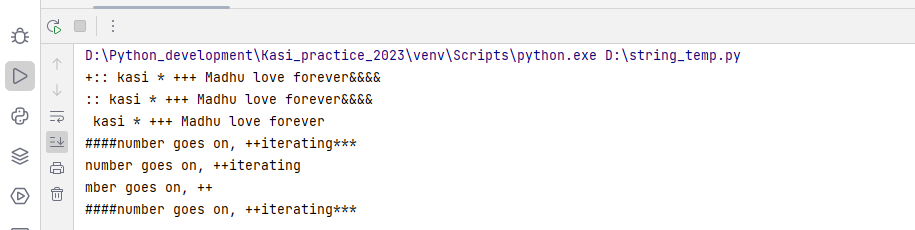


**Strip():**

* Removes leading and tailing characters of string with specified value
* If no value is specified it will checks with space as default
* If specified character not found it will return string as it is
* Strip() checks first character if it is match it will removes if not match it will go to end of the string and if its match it will removes or else it will return the string as it is. If you mentioned more than one character in the value it will starts checking the second character
* It will strips matched character if those are either at the leading or tailing only
* Synatax: string.strip(‘characters’)

Examples:

string\_ = '\*\*\*+:: kasi \* +++ Madhu love forever&&&&\*\*\*'  
print(string\_.strip('\*'))  
print(string\_.strip('\*+'))  
print(string\_.strip('\*+:&'))  
string2\_ = ' ####number goes on, ++iterating\*\*\* '  
print(string2\_.strip(' '))  
print(string2\_.strip(' #\*'))  
print(string2\_.strip(' #\*iteratingu'))  
print(string2\_.strip())

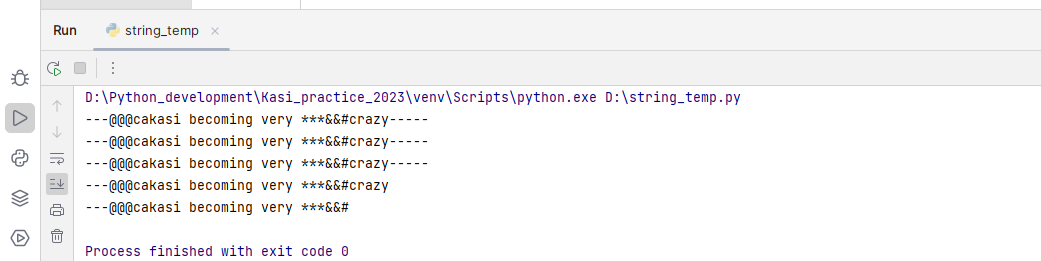


**Rstrip():**

* Removes tailing characters of the specified string
* If no value is specified it will checks with space as default
* If specified character not found it will return string as it is
* will strips matched character if those are either at the tailing only
* Syntax: string.rstrip(characters)

**Example:**

var1\_ = '---@@@cakasi becoming very \*\*\*&&#crazy----- '  
print(var1\_.rstrip('@'))  
print(var1\_.rstrip()) *# it will take space as default value*print(var1\_.rstrip(' '))  
print(var1\_.rstrip(' -'))  
print(var1\_.rstrip(' -crazy'))

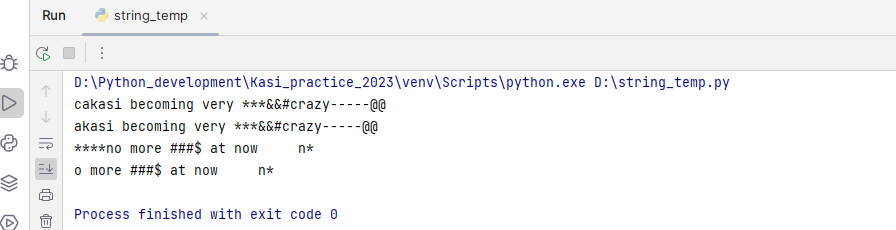


**lstrip():**

* Removes leading characters of the specified string
* If no value is specified it will checks with space as default
* If specified character not found it will return string as it is
* will strips matched character if it is at the leading edge only
* Syntax: string.lstrip(characters)

**Example:**

var1\_ = '@@@cakasi becoming very \*\*\*&&#crazy-----@@'  
print(var1\_.lstrip('-@'))  
print(var1\_.lstrip('@c'))  
var2\_ = ' \*\*\*\*no more ###$ at now n\* '  
print(var2\_.lstrip())  
print(var2\_.lstrip(' \*n'))



**isalnum()** 🡪 returns true if all the characters in the string are alpha numeric

**isalpha()** 🡪 returns true if all the characters in the string are alphabet only

**isdigit()** 🡪 returns true if all the characters in the string are digits

**islower()** 🡪returns true if all the characters in the string are lower case

**isupper()** 🡪 returns true if all the characters in the string are upper case

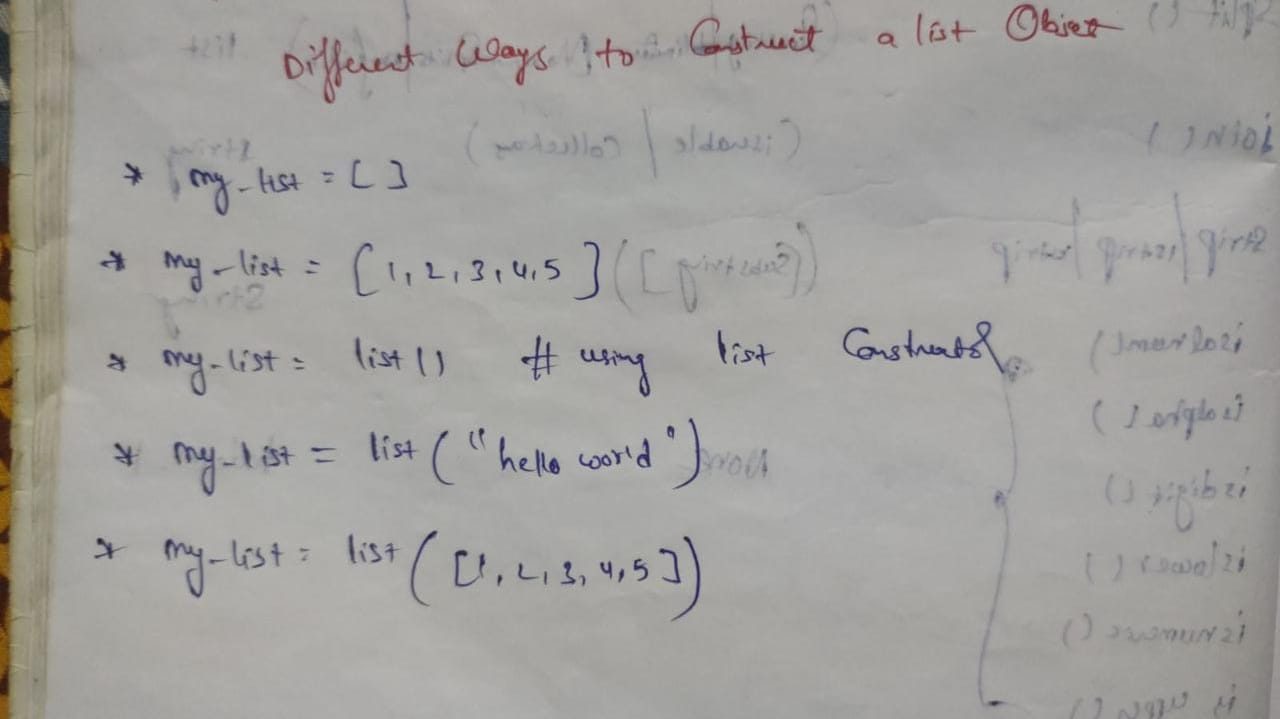
**isspace()** 🡪 returns true if all the characters in the string are spaces

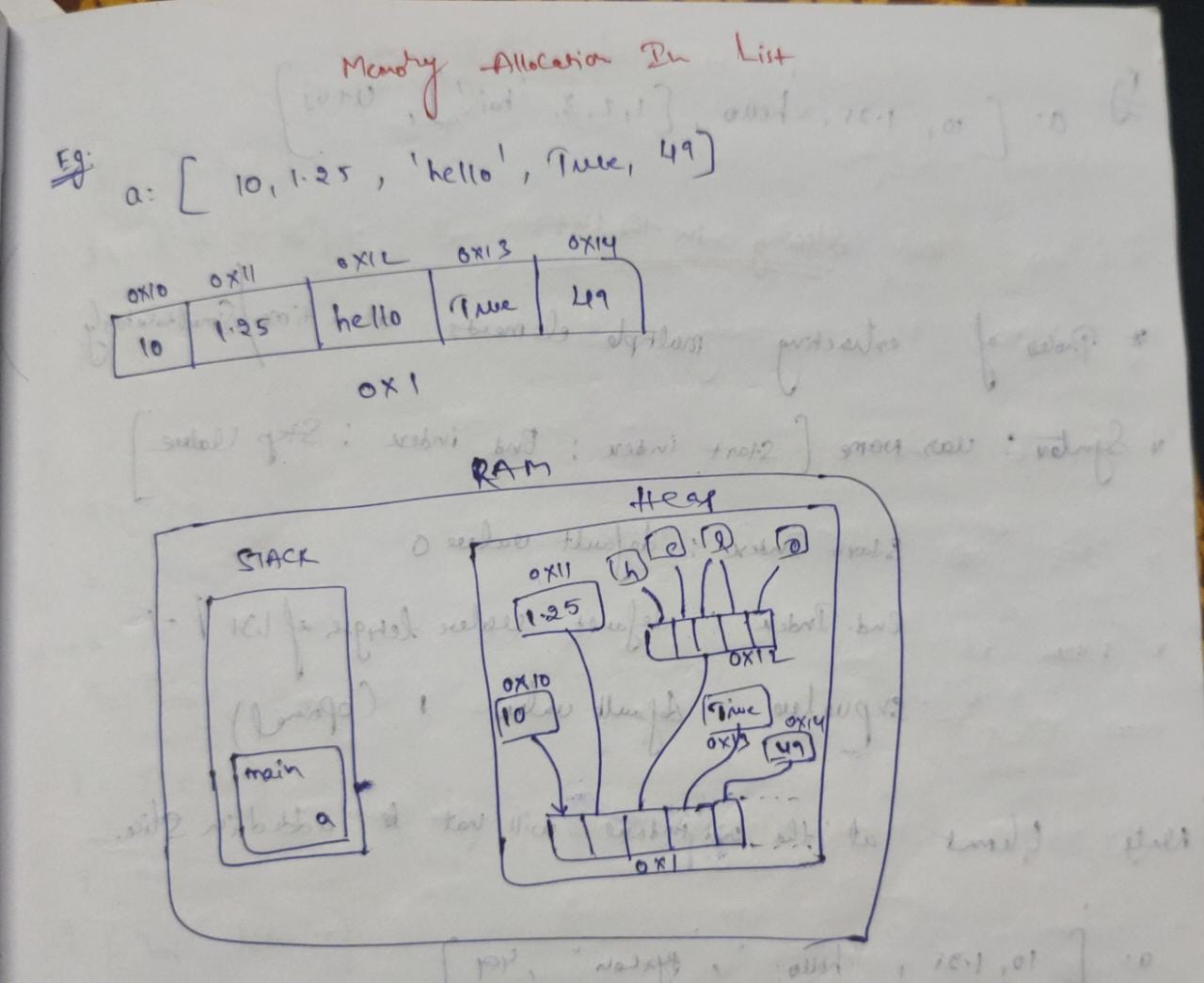
*#isalnum()*s = 'Mohana 45 &&&'  
print(s.isalnum())  
s1 = 'Kasi 143 Madhu'  
print(s1.isalnum())  
s1 = 'Kasi143Madhu'  
print(s1.isalnum())  
Var1 = 'Mohanakasi'  
print(Var1.isalnum())  
  
  
*#isalpha()*var1 = its a very starnge to hear  
print(var1.isalpha())  
var2 = 'thisoneisnotgood'  
print(var2.isalpha())  
var3 = 'kasi1madhu'  
print(var3.isalpha())  
  
  
*#isupper()*s1 = 'MOHANA KASI FROM BANGALORE'  
print(s1.isupper())  
s2 = 'MOHANA KASI fROM BANGALORE'  
print(s2.isupper())  
  
*#isspace()*string\_ = 'Mohana kasi is from guntur'  
print(string\_.isspace())

*#isdigit()*var1 = '45isfrom'  
print(var1.isdigit())  
var2 = '9848494'  
print(var2.isdigit())  
print(var2.isnumeric())

## Lists:

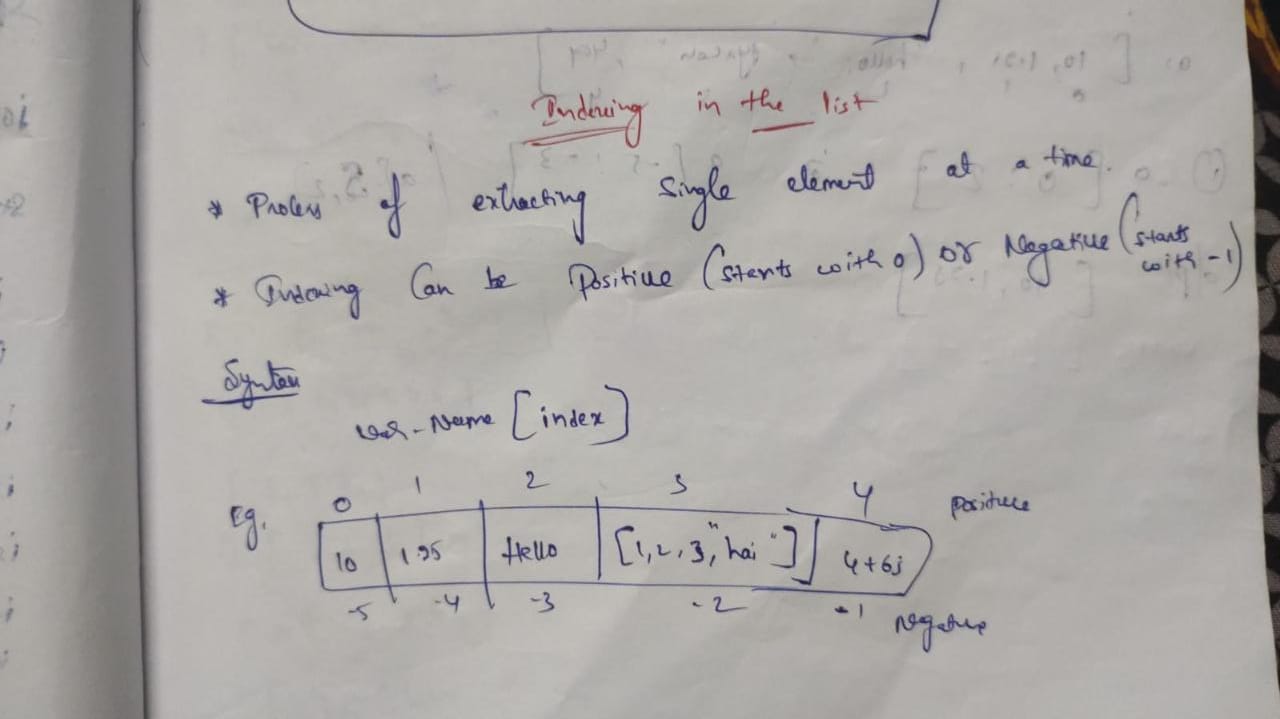
* Collection of homogeneous & heterogeneous elements
* Bounded by [] & Separated by coma
* List allows duplication of elements
* Items in the list are ordered
* Lists are mutable data type
* Syantax: variable\_name = [item1,item2……..]
* Length of the list is len(variable\_name)





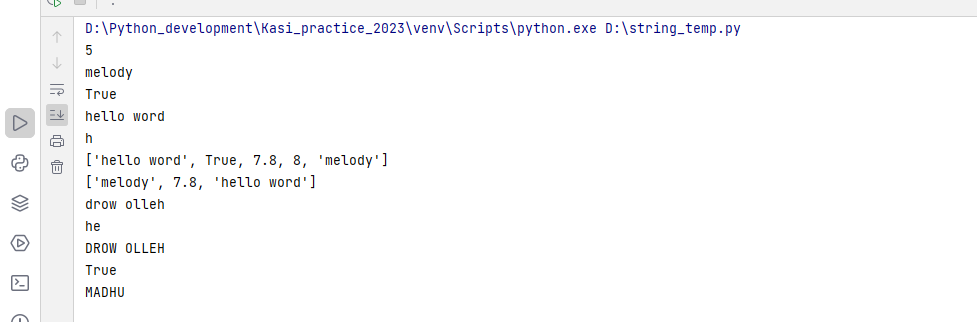
**Indexing & slicing in list:**

* Indexing & slicing in list rules are common like string. But the change is when the list having string or nested list the traversing will increase

****

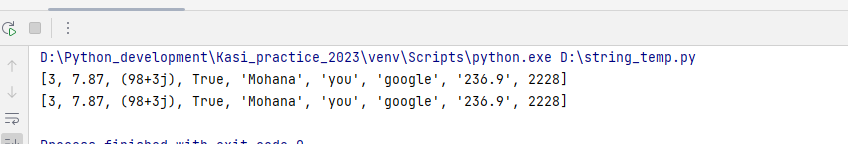
**Examples:**

list\_ = ['melody',8,7.8,True,'hello word']  
print(len(list\_))  
print(list\_[0])  
print(list\_[3])  
print(list\_[4])  
print(list\_[4][0])  
print(list\_[::-1])  
print(list\_[::2])  
print(list\_[4][::-1])  
print(list\_[4][:2])  
print(list\_[4][::-1].upper())  
print(list\_[4][0].startswith('h'))  
  
list2\_ = ['normal','\*\*@#',74848,bool,4+87j,False,12.87,[89,'Mohana','Madhu']]  
print(list2\_[7][2][::-1].upper()[::-1])



**Merging two lists:**

list1\_ = [3,7.87,98+3j, True, 'Mohana']  
list2\_ = ['you', 'google','236.9',2228]  
print(list1\_+list2\_) *#returns list by merging two lists*print([\*list1\_,\*list2\_])



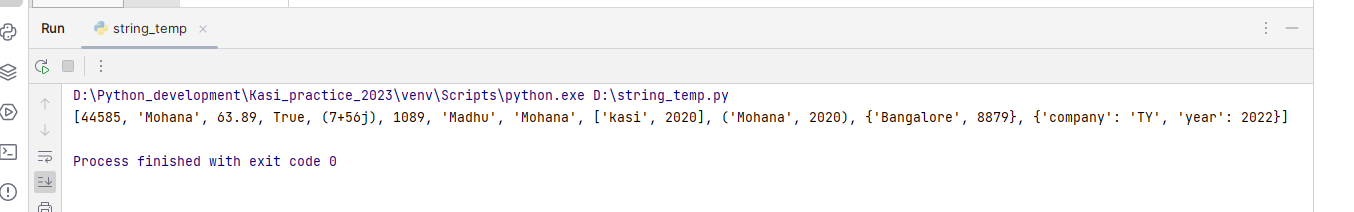
**List inbuilt methods:**

**Append():**

* It will add the element at the end of the list
* The element data type can be of any type
* It will take only one argument at a time
* It will return nothing, it will modify the original list
* **Synatax: list.append(element/item)**

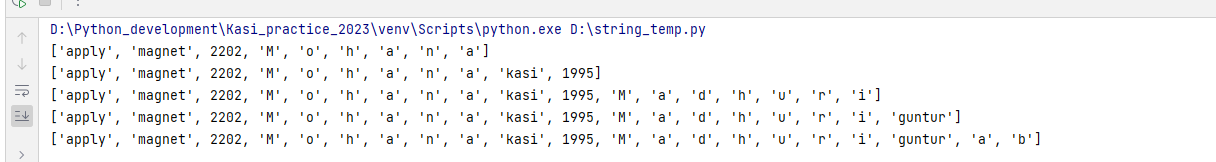
**Examples:**

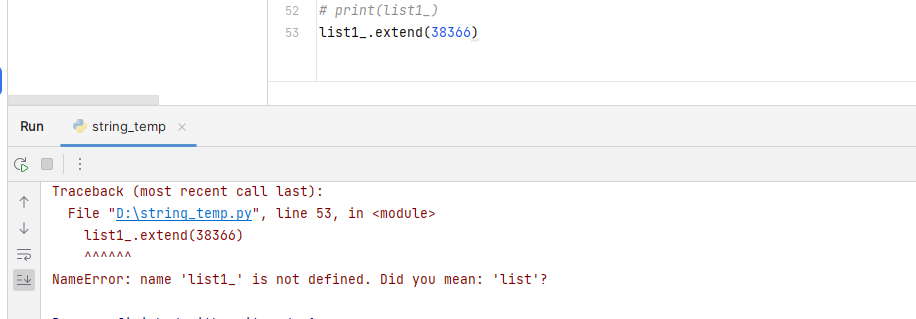
list\_ = [44585, 'Mohana']  
list\_.append(63.89)  
list\_.append(True)  
list\_.append(7+56j)  
list\_.append(1089)  
list\_.append('Madhu')  
list\_.append('Mohana')  
list\_.append(['kasi',2020])  
list\_.append(('Mohana',2020))  
list\_.append({8879,'Bangalore'})  
list\_.append({'company': 'TY', 'year': 2022})  
print(list\_)

****

**Extend():**

* It will extends the existing list elements with the items of given sequence
* It will supports only collection data types
* It will return nothing, it will modify the original list
* **Syntax: list.extend(iterable)**
* list1\_ = ['apply','magnet',2202]  
  list1\_.extend('Mohana')  
  print(list1\_)  
  list1\_.extend(['kasi',1995])  
  print(list1\_)  
  list1\_.extend(('Madhuri'))  
  print(list1\_)  
  list1\_.extend({'guntur'})  
  print(list1\_)  
  list1\_.extend({'a':10, 'b':1000})  
  print(list1\_)

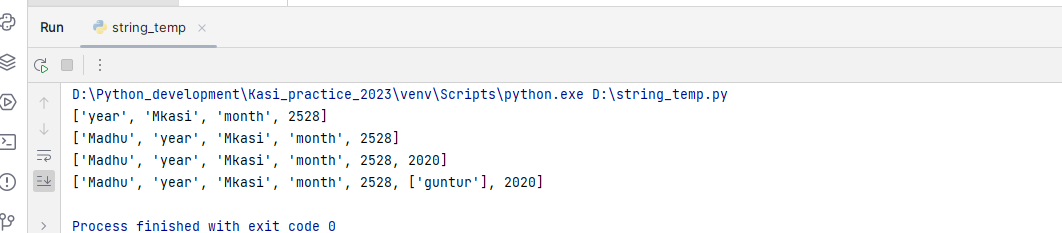
****

****

**Insert():**

* Insert elements at the specified position(both individual & collections)
* **Syntax: list.insert(pos, element)**

list1\_ = ['year','month',2528]  
list1\_.insert(1,'Mkasi')  
print(list1\_)  
list1\_.insert(0,'Madhu')  
print(list1\_)  
list1\_.insert(len(list1\_), 2020) *#adds the element at the last*print(list1\_)  
list1\_.insert(len(list1\_)-1, ['guntur']) *#adds the element before the last element*print(list1\_)

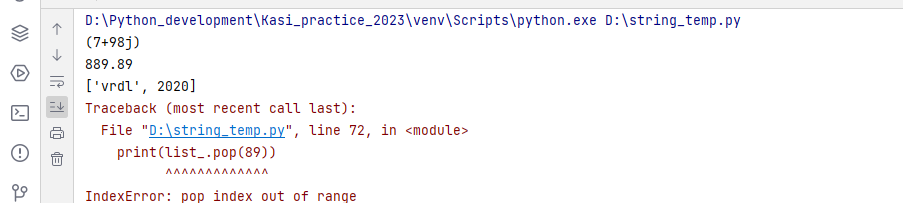


**pop():**

* Removes the element at the specified position
* If the position not defined by default removes the last element
* It will modify the original list & retruns the removed element
* If given position not found it will throws the index error
* **Syntax: list.pop([pos])**

Example:

list\_ = ['Kasi', 'Madhu', ['vrdl',2020], True, 'Bangalore','forever',889.89,7+98j]  
print(list\_.pop()) *#default it wil removes the last element as pos not specified*print(list\_.pop(len(list\_)-1)) *# as len(list)-1 given it will takes last element position & removes it*print(list\_.pop(2))  
print(list\_.pop(89))



**remove():**

* Removes the first occurrence of element specified
* If the element not found it will return value error
* It will modify the original list, it will not return anything
* Syntax: list.remove(element)

Examples:

list\_ = ['Kasi', 'Madhu', ['vrdl',2020], True, 'Bangalore','forever',889.89,7+98j]  
print(list\_.remove('forever'))  
print(list\_)  
print(list\_.remove('Kasi'))  
print(list\_)  
print(list\_.remove('forever'))

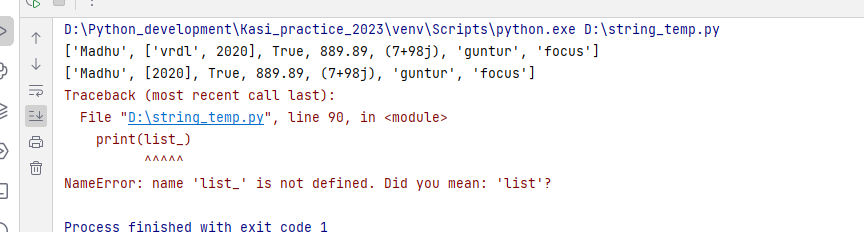


**del:** del is the keyword to deal locate the memory in the iterable

* It will removes the element or full list
* It will removes the elements based on the indexing & slicing
* It will not return anything, it will modifies the original list
* Synatax: del iterable

**Example:**

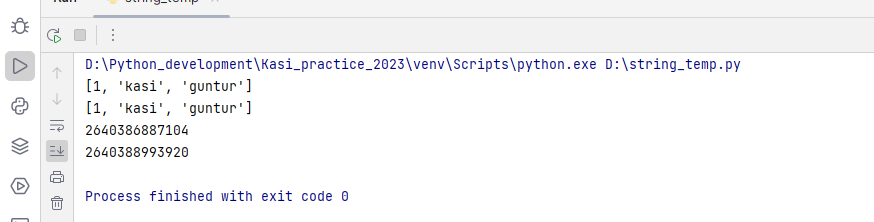
list\_ = ['Kasi', 'Madhu', ['vrdl',2020], True, 'Bangalore','forever',889.89,7+98j,'guntur','focus']  
del list\_[0]  
del list\_[3:5]  
print(list\_)  
del list\_[1][0] *# removing item in the nested list*print(list\_)  
del list\_  
print(list\_)



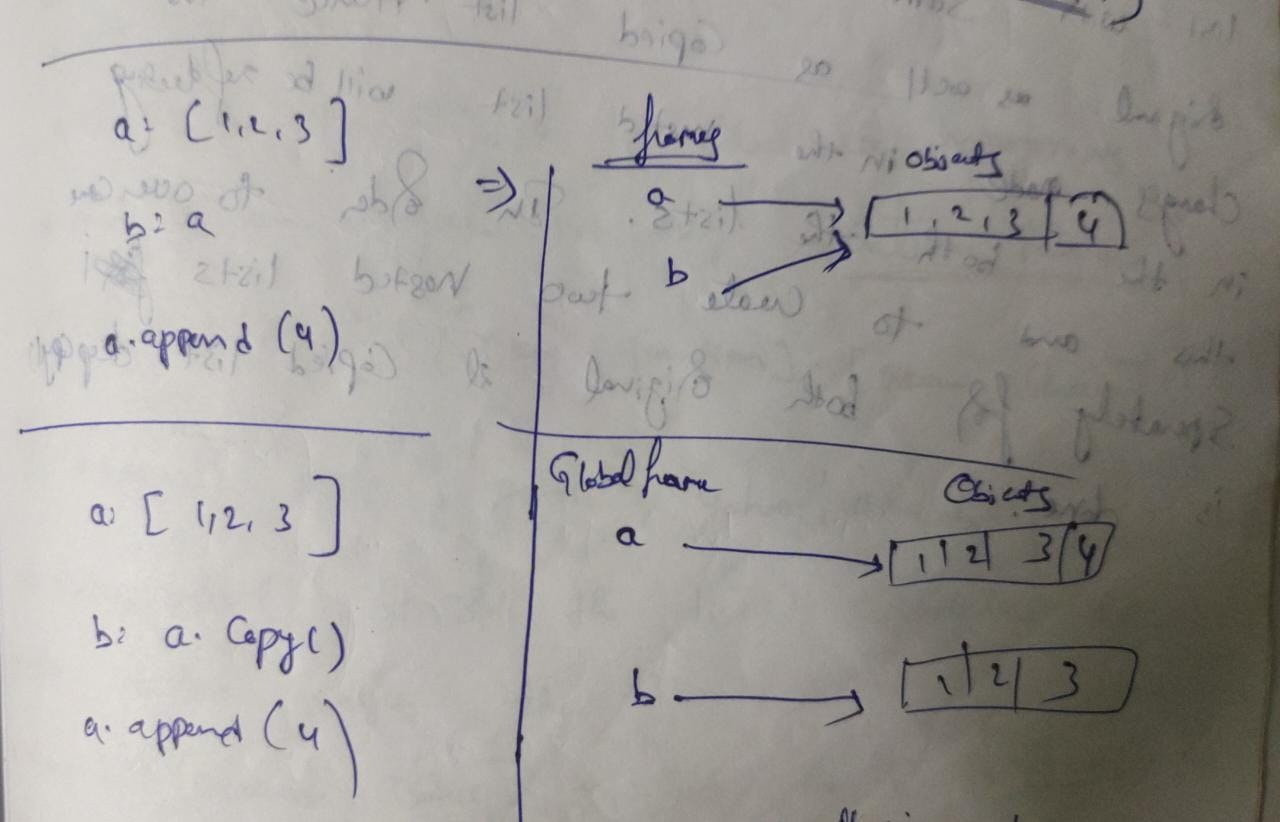
Copy():

* Returns a copy of the list

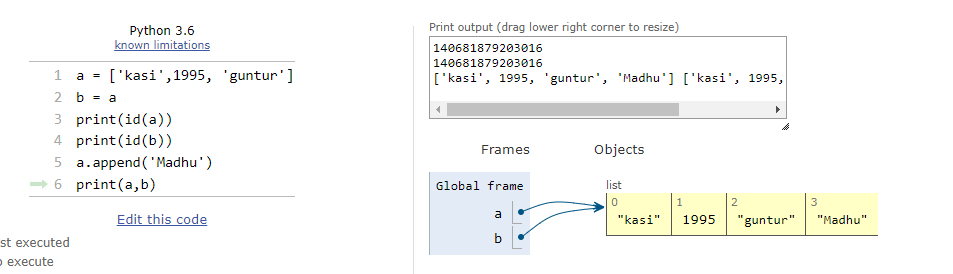
a = [1,'kasi','guntur']  
b = a.copy()  
print(a)  
print(b)  
print(id(a))  
print(id(b))



**Normal copy & copy() difference:**

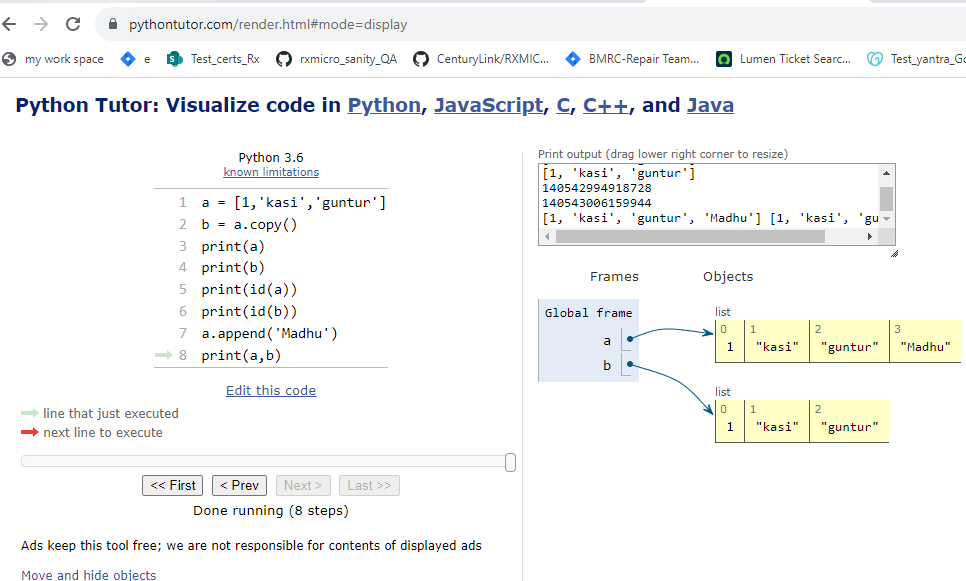
****

a = ['kasi',1995, 'guntur']  
b = a  
print(id(a))  
print(id(b))  
a.append('Madhu')  
print(a,b)

****

**Copy():**

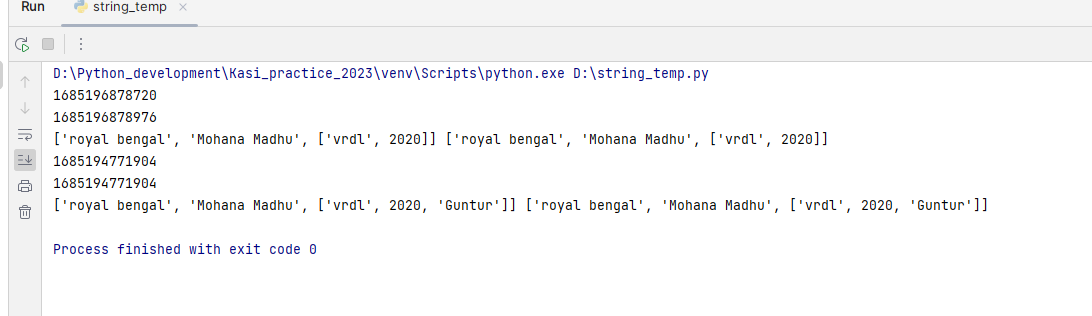
a = [1,'kasi','guntur']  
b = a.copy()  
print(a)  
print(b)  
print(id(a))  
print(id(b))  
a.append('Madhu')  
print(a,b)

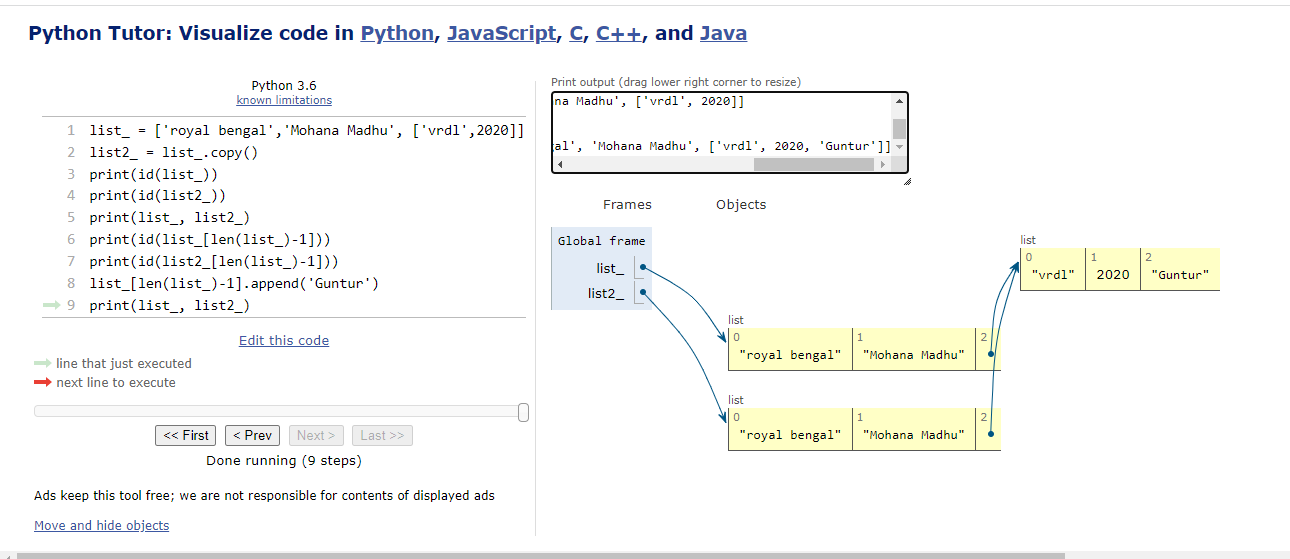
****

**#But the same copy() method will not works when the main list having nested list**

* This time as the list having nested list, the nested list address will common to both the lists, so if any modification done first list nested list. The changes will reflect in second list nested list also

list\_ = ['royal bengal','Mohana Madhu', ['vrdl',2020]]  
list2\_ = list\_.copy()  
print(id(list\_))  
print(id(list2\_))  
print(list\_, list2\_)  
print(id(list\_[len(list\_)-1])) *# eventhough both the lists address are differnet but nested list address common*print(id(list2\_[len(list\_)-1]))  
list\_[len(list\_)-1].append('Guntur')  
print(list\_, list2\_) *# due to nested list memory address common to both the list,  
# the item added in one list appeding to both the list*





Note: to avoide this problem we will use deep copy()

**Shallow copy():**

A shallow copy is a copy of an object that stores the reference of the original elements. It creates the new collection object and then occupying it with reference to the child objects found in the original.

It makes copies of the nested objects' reference and doesn't create a copy of the nested objects. So if we make any changes to the copy of the object will reflect in the original object. We will use the **copy()** function to implement it.

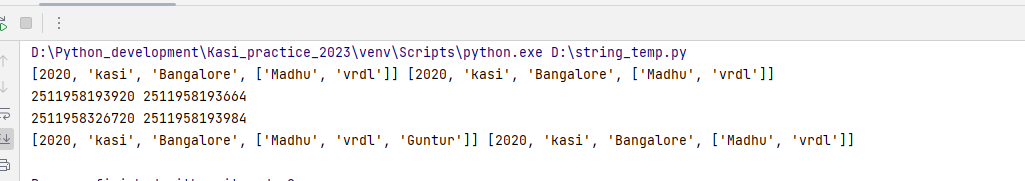
list\_ = [78, 'kasi',['Madhu']]  
import copy  
list2\_ = copy.copy(list\_)  
print(id(list\_))  
print(id(list2\_))  
print(list\_, list2\_)  
list\_[len(list2\_)-1].append(2020)  
print(list\_, list2\_)

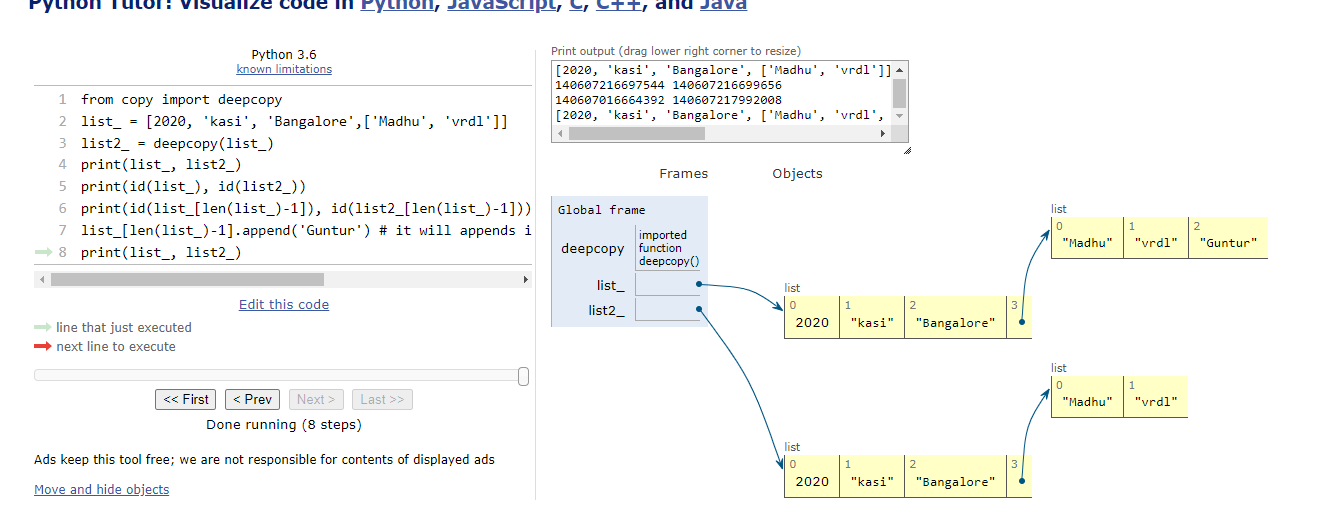
**deep copy():**

A deep copy is a process where we create a new object and add copy elements recursively. We will use the **deecopy()** method which present in copy module. The independent copy is created of original object and its entire object. Let's understand the following example.

**Example:**

from copy import deepcopy  
list\_ = [2020, 'kasi', 'Bangalore',['Madhu', 'vrdl']]  
list2\_ = deepcopy(list\_)  
print(list\_, list2\_)  
print(id(list\_), id(list2\_))  
print(id(list\_[len(list\_)-1]), id(list2\_[len(list\_)-1])) *#address of both nested lists are different*list\_[len(list\_)-1].append('Guntur') *# it will appends into first list only*print(list\_, list2\_)

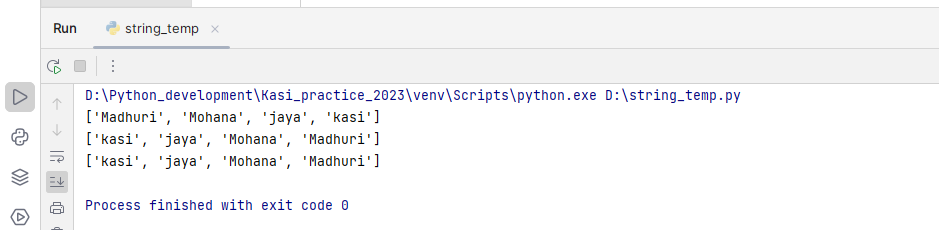




**sort():**

* Sort the list
* In order to sort the list, the list should contains the values of homogeneous
* The sort() modifies the original list, it will not return anything
* Syntax: list.sort([key=function], [reverse=true]) 🡪 # if something written in [] notes indicates optional

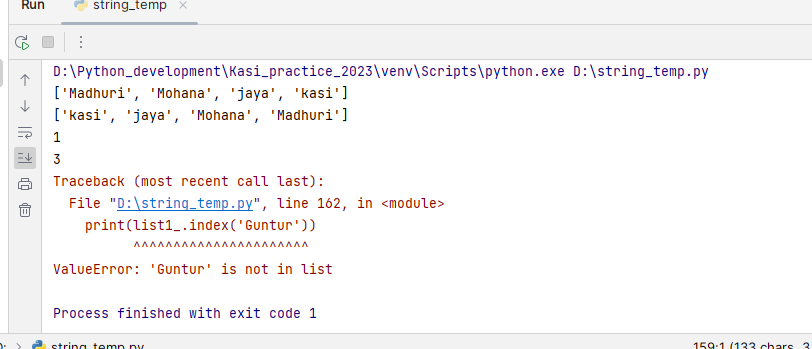
**Example:**  
list\_ = ['Mohana', 'kasi','jaya','Madhuri']  
list\_.sort()  
print(list\_)  
list\_.sort(reverse=True)  
print(list\_)  
  
*#using the function*def len\_base\_sort(ele):  
 return len(ele)  
  
list\_ = ['Mohana', 'kasi','jaya','Madhuri']  
list\_.sort(key=len\_base\_sort)  
print(list\_)



**index():**

* Returns the index of the specified element
* If the element not found gives the value error
* **Synatx: list.index(element)**

list1\_ = ['Mohana', 'kasi','jaya','Madhuri']  
print(list1\_.index('kasi'))  
print(list1\_.index('Madhuri'))  
print(list1\_.index('Guntur'))

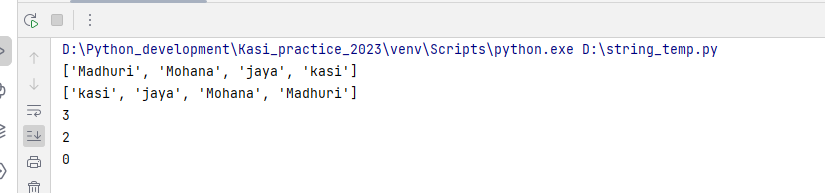


**count():**

* Returns the no of occurrences of the specified value
* It will return the count as integer
* If the value is not there in the list, it will return 0
* **Syntax: list.count(value)**

**Example:**

list\_ = [225, 125, 'kasi', 225, 'Madhuri', 2020, 'Mohana', kasi, 225, 'Madhuri']  
print(list\_.count(225))  
print(list\_.count(list\_[-1]))  
print(list\_.count('Guntur'))

****

**Tuples:**

* collection of homogeneous & heterogeneous items
* separated by coma & bounded by ()
* tuples are immutable data type
* duplication of the elements allowed
* Syntax: var\_name = (el1, el2,ele3……….)
* Indexing & slicing is similar to that of string & list
* Length of the tuple is len(var\_name)

**Note: Tuples are more fast & memory efficient compared to the list**

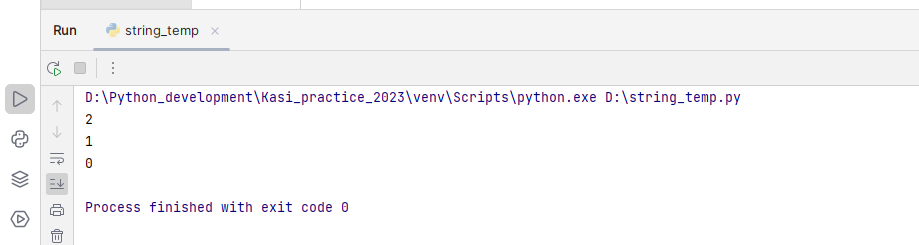
**Inbuilt methods:**

**Count():**

* Returns the no of occurrences of a specified element in the tuple
* If the specified value is not found it will show the count as 0
* Synatax: tuple.count(element)

Example:

var1 = ('kasi',1995,['Madhu'],1995)  
print(var1.count(1995))  
print(var1.count('kasi'))  
print(var1.count('guntur'))

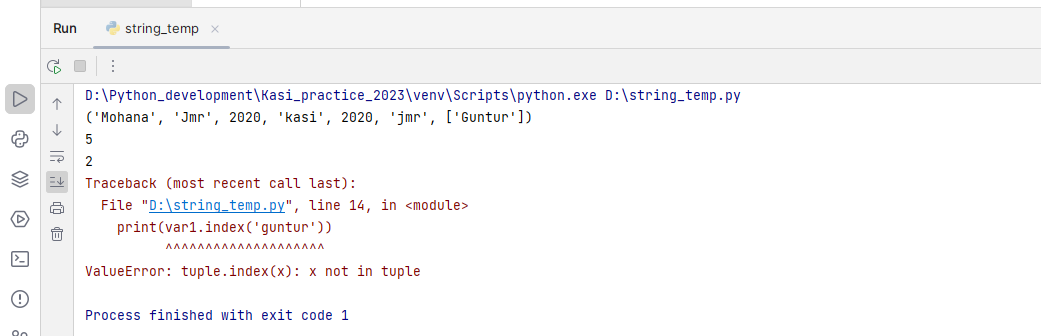


**index():**

* Returns the position of first occurrence of element in tuple
* If the specified value not found it will return value error
* Syntax: tuple.index(value)

Example:

var1 = ('Mohana','Jmr',2020,'kasi',2020,'jmr',['Guntur'])  
print(var1)  
print(var1.index('jmr'))  
print(var1.index(2020))  
print(var1.index('guntur'))



**Note: The collections or iterables, strings, lists, tuples are called as sequences because they are ordered ( their elements can be have indexes)**

**Sets:**

* Sets are underscored
* Elements inside the sets are unique
* Set will not allow to add the **mutable data** types
* Sets are mutable but elements inside the set must be hashable
* Sets cannot be **indexed or sliced**
* Not allows the duplication of elements
* These are separated by coma & bounded between {}
* Syntax: var\_name = {ele1,ele2,ele3…….}

**Hashable objects:**

* Hashable objects are the objects which implements the \_\_hash\_\_ magic method & hash() method can be called
* Only immutable objects will have hash values, hence hash() can be be used to check mutability & immutability
* All immutable objects are hashable, but all hashable objects are not immutable

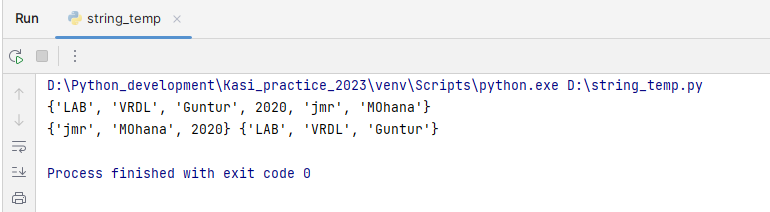
**Inbuilt methods:**

**Union():**

* Returns a set of all items from the original set & all items from the specified set
* It will return a set, not modifies the original or specified set
* Syntax: set1.uninon(set2)

**Examples:**

s1 = {'MOhana',2020,'jmr',2020}  
s2 = {'VRDL','Guntur','LAB'}  
print(s2.union(s1))  
print(s1,s2)

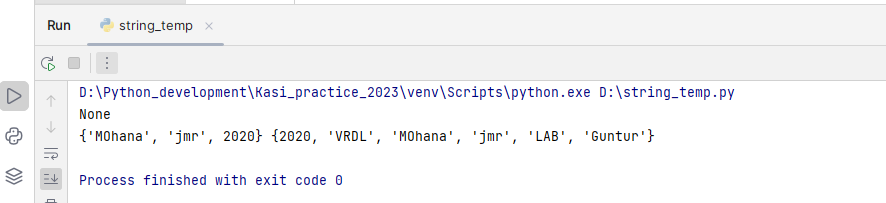


**Update():**

* Updates the current set by adding the items from another set
* It will modifies the specified set, returns none
* Synatx: set1.update(set2)

**Example:**

s1 = {'MOhana',2020,'jmr',2020}  
s2 = {'VRDL','Guntur','LAB'}  
  
print(s2.update(s1))  
print(s1, s2)

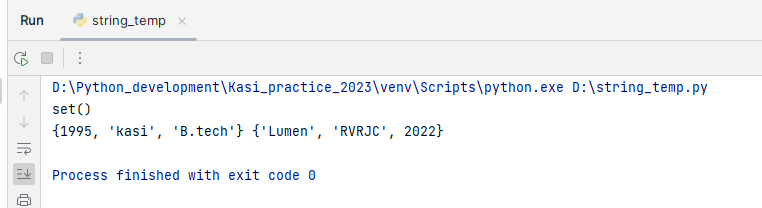
****

**Intersection():**

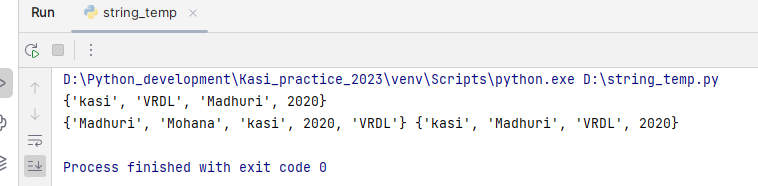
* Returns a set containing similarity between two or more sets
* Returns a set, not modifies the any of the set
* It will return empty set (set()) if no more common values between the sets
* Syntax: set1.intersection(set2)

**Examples:**

s1 = {1995, 'kasi', 'B.tech'}  
s2 = {'RVRJC', Lumen, 2022}  
print(s2.intersection(s1))  
print(s1,s2)

**4**

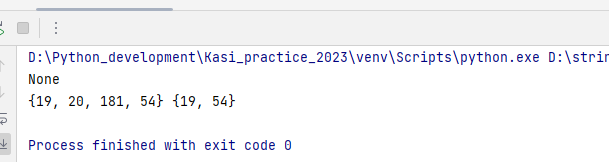
s1 = {'Mohana', 'kasi', 'Madhuri', 2020, 'VRDL'}  
s2 = {'kasi', 'Madhuri', 2020, 'VRDL'}  
print(s2.intersection(s1))  
print(s1, s2)

****

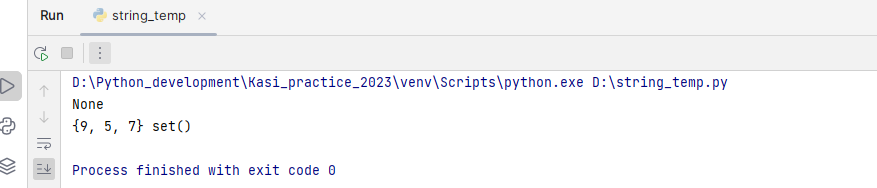
**Intersection\_update():**

* Modifies the set by retaining only common elements from the both the sets
* Modifiesthe set, returns None

s1 = {19,181,20, 19, 54,181}  
s2 = {54,19, 2020}  
print(s2.intersection\_update(s1))  
print(s1, s2)

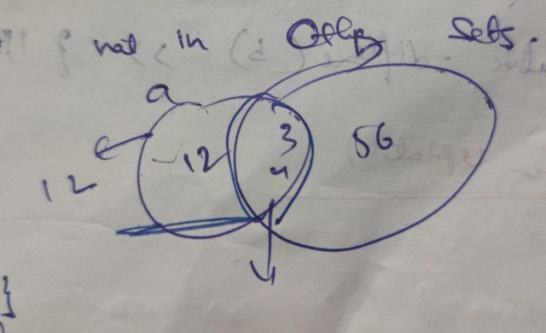
****

s1 = {7, 9, 5}  
s2 = {54,19, 2020}  
print(s2.intersection\_update(s1))  
print(s1, s2)

****

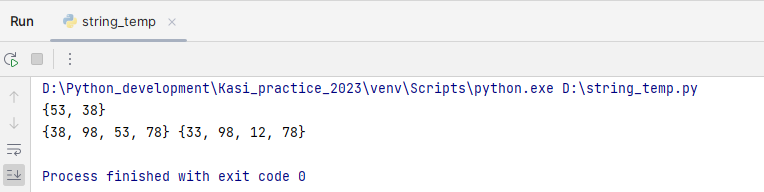
**Difference():**

* Returns a set containing the difference between two or more sets
* The returned set contains items that exist only in the base set, but not in other sets
* Returns new set, not modifies the original sets
* Syntax: set1.difference(set2)

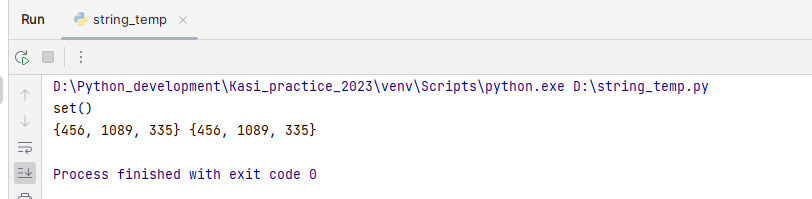
****

**Example:**

s1 = {53,38, 78,98}  
s2 = {12, 78, 33,98}  
print(s1.difference(s2))  
print(s1,s2)



s1 = {1089, 456, 335}  
s2 = {1089,335, 456}  
print(s1.difference(s2))  
print(s1,s2)



**difference\_update():**

* Updating the base set with the difference elements of base set not all other sets
* Syntax: set1.difference\_update(set2)

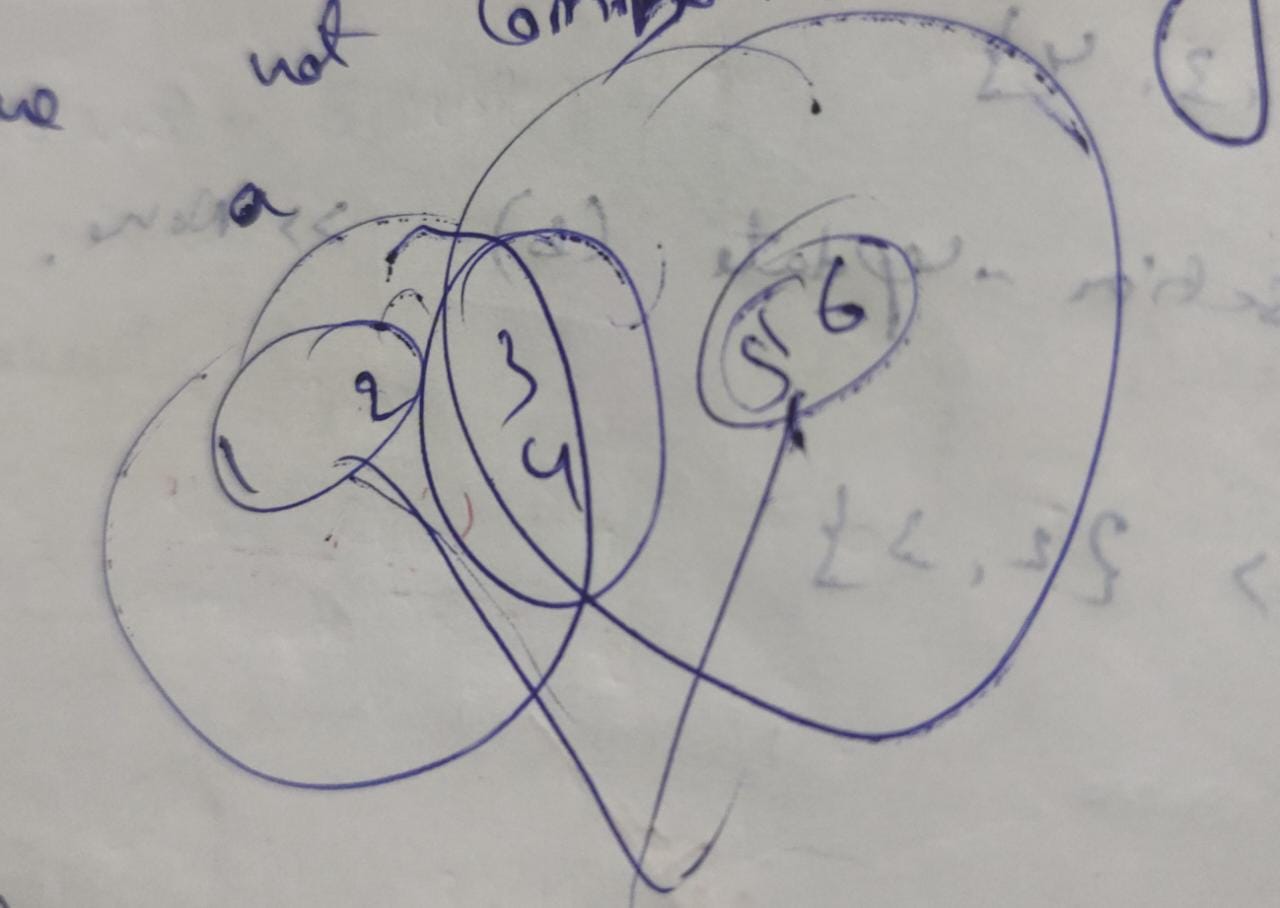
**Examples:**

s1 = {1089, 2020, 335, 663 }  
s2 = {1089,335, 456}  
print(s1.difference\_update(s2))  
print(s1, s2)

****

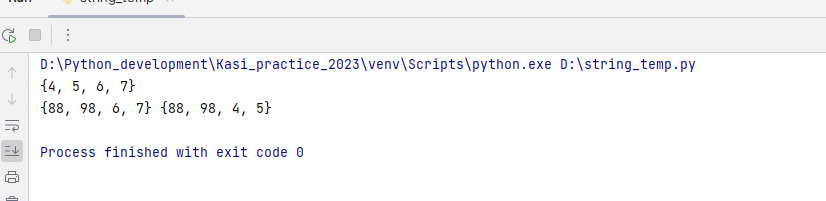
**Symmetric difference():**

* Returns a set that contains all the difference items among the all the sets
* Returns set, not modifies the base set
* Syntax: Set1.symmetric\_difference(set2)

****

**Example:**

s1 = {6, 7, 88, 98}  
s2 = {4, 5 , 88, 98}  
print(s1.symmetric\_difference(s2))  
print(s1, s2)



**symmetric\_difference\_update():**

* Update the base set with the difference elements of all sets
* Returns none, update the base set
* Syntax: set1.symmetric\_difference\_update(set2)

Example:

s1 = {6, 7, 88 , 98}  
s2 = {4, 5 , 88, 98}  
print(s1.symmetric\_difference\_update(s2))  
print(s1, s2)

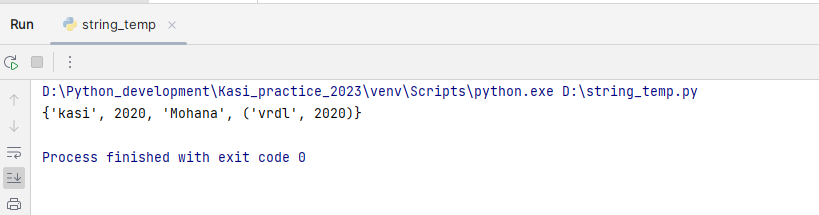


**Add():**

* Adds the element into the existing set
* If the element already exists, it will not add that element
* It will not support to add the mutable data types like list, set, dictionary
* Syntax: set.add(element)

**Example:**

set1\_ = {'Mohana', 2020, 'kasi',}  
set1\_.add(2020)  
set1\_.add(('vrdl', 2020))  
print(set1\_)

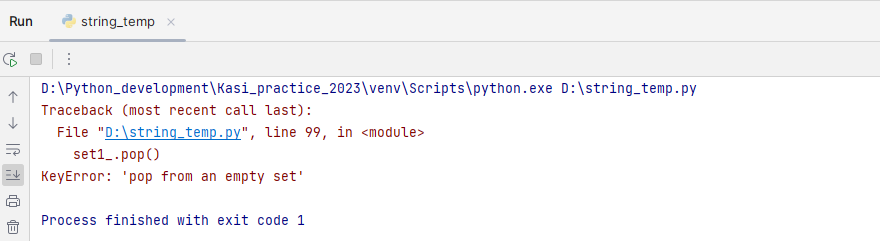


pop():

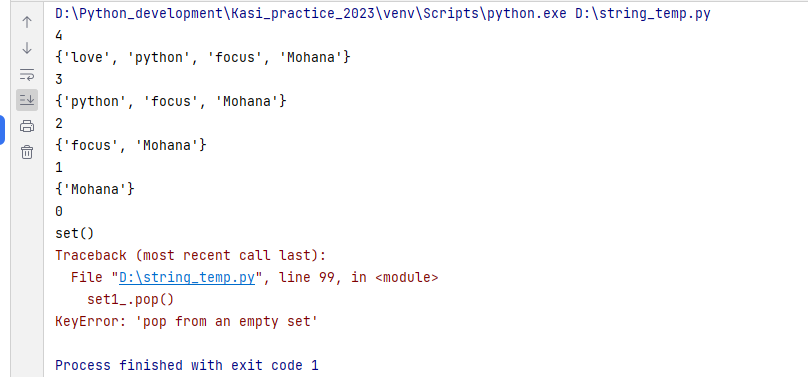
* Removes random element from the set
* It will not take any arguments, removes random key
* Returns the removed item
* If the set is empty, it will raise key error
* Syntax: set.pop()

Example:

set1\_ = {'Mohana', 'focus', 'python', 'love'}  
set1\_.pop()  
set1\_.pop()  
set1\_.pop()  
set1\_.pop()  
set1\_.pop()



set1\_ = {'Mohana', 'focus', 'python', 'love'}  
lenghth = len(set1\_)  
for i in range(lenghth+1):  
 print(len(set1\_))  
 print(set1\_)  
 set1\_.pop()

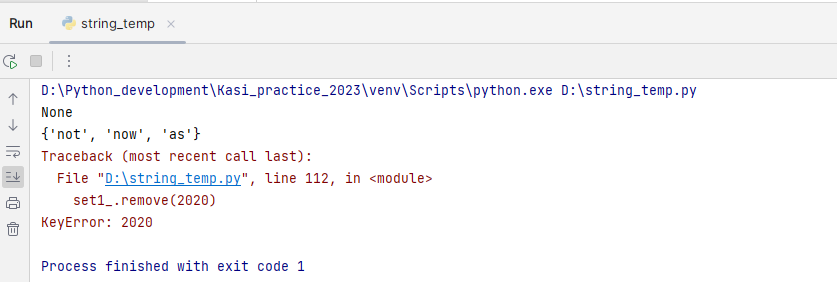


**remove():**

* Removes the specified element from the set
* If the element not found raises key error
* It will removes the element, returns none
* Syntax: set.remove(element)

Example:

set1\_ = {'not', 'as', 'of', 'now'}  
print(set1\_.remove('of'))  
print(set1\_)  
set1\_.remove(2020)

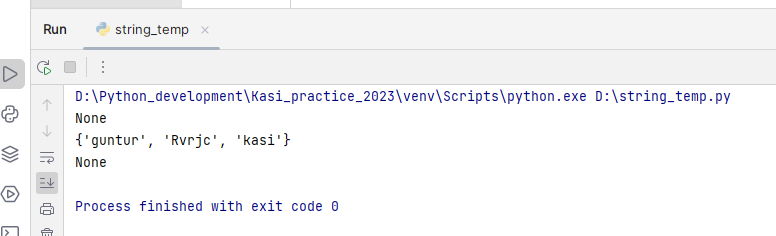


**discard():**

* Removes the specified item from the set
* It will not return anything, removes the item
* If the item not found it will not raise any error
* Syntax: set.discard(element)

**Example:**

set1\_ = {2015, 'kasi', 'Rvrjc', 'guntur'}  
print(set1\_.discard(2015))  
print(set1\_)  
print(set1\_.discard('Madhuri'))



**Dictionaries:**

* Collection of key value pairs
* Each element associated with unique key
* Separated by coma & bounded by {}
* Syntax: var\_name = {key: value, key: value ……..}
* It does not support indexing or slicing, but using keys we can fetch value like indexing

D = {} 🡪 empty dictionary

**Length:**

Len(var\_name) 🡪 return no of keys present in a dictionary

Different ways to construct dictionary:

d = dict() 🡪 empty dictionary

d = {‘a’: ‘kasi’, ‘b’: ‘guntur’}

d = dict(‘Bangalore’ = 25, ‘Guntur’ = 1008)

d = dict([(‘a’, 100), (‘b’, 180)]) 🡪 list containing tuples of key value pairs

**Characteristics of dictionaries:**

* Key cannot be duplicated
* Keys **will be a single** element
* Keys **always immutable data type** & Values can be of any data type

**Deallocation:**

1. **Clear():**
   * It will clear all elements in the dictionary. But empty dictionary will be present
   * Ex: d = {‘kasi’: 100, ‘Madhu’: 1000}
   * d.clear()
   * d🡪 {}
2. **del:**
   * It will deallocate the memory given to d & d will be deleted permanently
   * d = {‘kasi’:100, ‘Madhu’: 1000}
   * del d
   * d 🡪 ‘Name error’

**Accessing the values from a dictionary:**

**get():**

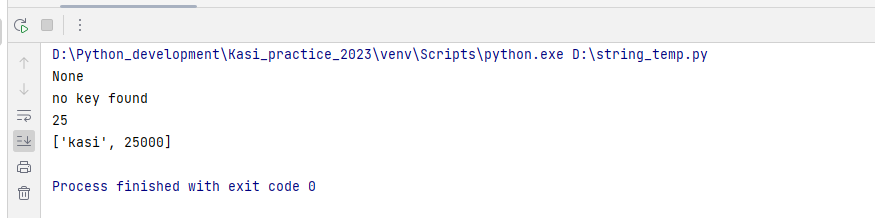
* It will give the value of the specified key in a dictionary
* Returns none if key not present
* Syntax: var\_name.get(‘key’, [default messege])

**Note: get() method returns none if the given key not found, if you give second parameter like any message it will print that message if the key not exist**

**Ex: var\_name.get(‘key’, ‘no key found’)**

**Example:**

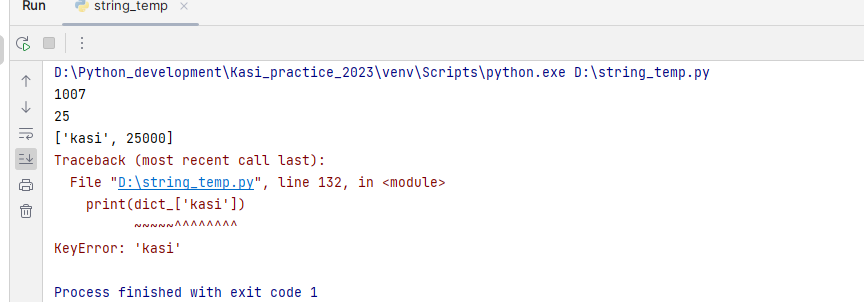
dict\_ = {'bangalore': 25, 'guntur': 1007, ('name', 1008): ['kasi', 25000]}  
print(dict\_.get('kasi')) *#returns none if key not present*print(dict\_.get('kasi', 'no key found')) *# returns the messege 'no key found' if the key not exist*print(dict\_.get('bangalore', 'no key found'))  
print(dict\_.get(('name', 1008)))

****

**Accessing value using key without inbuilt method:**

* It will returns key error if the key is not present

dict\_ = {'bangalore': 25, 'guntur': 1007, ('name', 1008): ['kasi', 25000]}  
print(dict\_['guntur'])  
print(dict\_['bangalore'])  
print(dict\_[('name', 1008)])  
print(dict\_['kasi'])

****

**Items():**

* It will returns a list containing tuples of key value pairs of a dictionary
* It will not modify the original dictionary & it will not take any arguments. It will just return a list of tuples having key value pairs
* Syntax: var\_name.items()

**Examples:**

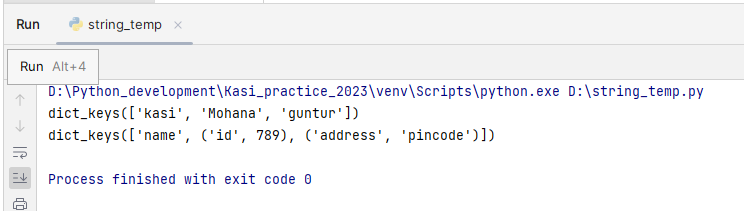
dict1\_ = {'name': 'Mohana', 'id': 8879, 'address': ('1b', '8th cross', 78185)}  
print(dict1\_.items())

**keys():**

* Keys() will returns a list containing keys
* It will not modify the original dictionary & it will not take any arguments. It will just return a list of keys
* Syntax: var\_name.keys()

**Examples:**

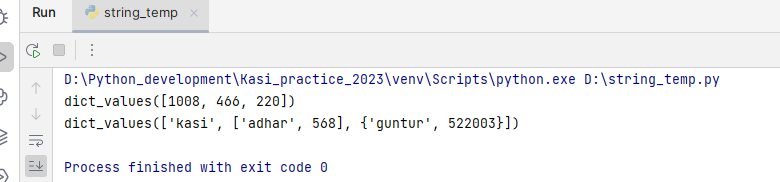
dict\_ = {'kasi': 1008, 'Mohana': 466, 'guntur': 220}  
print(dict\_.keys())  
dict1\_= {'name': 'kasi', ('id', 789): ['adhar', 568], ('address', 'pincode'): {'guntur', 522003}}  
print(dict1\_.keys())

****

**Values():**

* **Returns a list of values**
* **Syntax: var\_name.values()**

dict\_ = {'kasi': 1008, 'Mohana': 466, 'guntur': 220}  
print(dict\_.values())  
dict1\_= {'name': 'kasi', ('id', 789): ['adhar', 568], ('address', 'pincode'): {'guntur', 522003}}  
print(dict1\_.values())

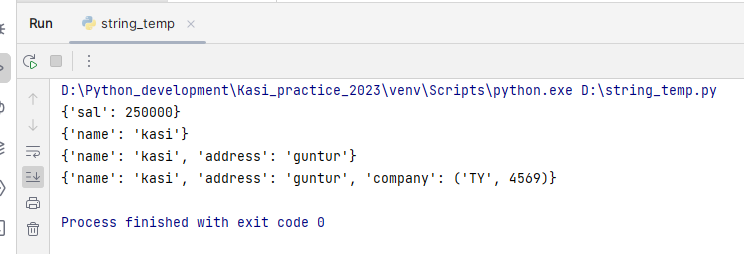
****

**Update():**

* Update the existing dictionary with the new items given
* Syntax: var\_name.update({‘key’: value......})
  + Var\_name.update(key=value}) 🡪 in this case you need to mention the key without quotes like variable creation
  + & you cannot give the composite keys in this way

**Example:**

d = {}  
d.update(sal=250000)  
print(d)  
d = {}  
d.update({'name': 'kasi'})  
print(d)  
d.update(address='guntur')  
print(d)  
d.update(company=('TY', 4569))  
print(d)

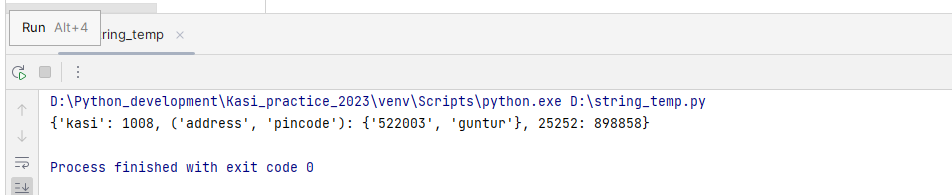


**Adding key & values normal syntax:**

Var\_name[key] = value

**Example:**

dict\_ = {}  
dict\_['kasi'] = 1008  
dict\_[('address', 'pincode')] = {'guntur', '522003'}  
dict\_[25252] = 898858  
print(dict\_)



**Fromkeys():**

* It will return a dictionary with the specified key & value
* Syntax: dict.fromkeys(keys, [value])
* Key is mandatory, An iterable specifying the keys of new dictionary
* Value is optional, the value of all keys. Default value is none

**Example:**

d = dict.fromkeys('kasi', 78) *# the key should be an iterable*print(d)  
list\_ = ['name', 'id', 'address']  
print(dict.fromkeys(list\_, 'sample'))  
list2\_ = ['name', 'company', 'address']  
print(dict.fromkeys(list2\_))

**pop():**

* It will removes the specified key & value from the dictionary & return the removed values
* Syntax: var\_name.pop(key, default value)
* It will return the default value if the key not found,
* If you not mention default message it will return the key error

**Example:**

dict\_ = {'name': 'kasi', ('id', 789): ['adhar', 568], ('address', 'pincode'): {'guntur', 522003}}  
print(dict\_.pop('name'))  
print(dict\_.pop(('id', 789)))  
print(dict\_.pop('kasi', 'not found'))  
print(dict\_.pop('Mohana'))

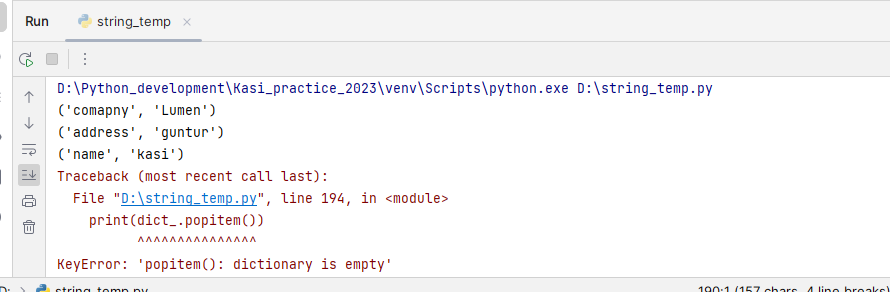


**popitem():**

* Removes the last inserted item into the dictionary
* It will not takes the any argument, it will return the tuple of key & values of removed
* Syntax: var\_name.popitem()

**Example:**

dict\_ = {'name': 'kasi', 'address': 'guntur', 'comapny': 'Lumen'}  
print(dict\_.popitem())  
print(dict\_.popitem())  
print(dict\_.popitem())  
print(dict\_.popitem())

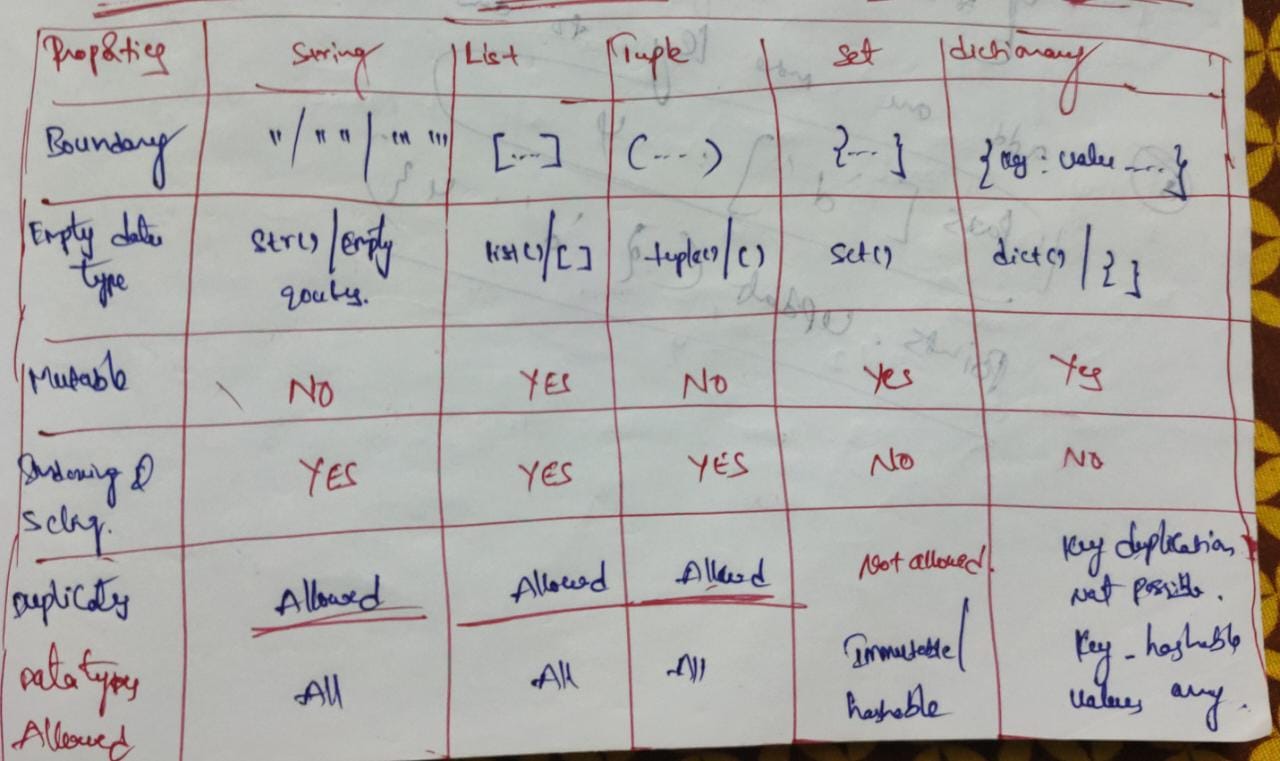


**Merging dictionaries:**

d1 = {'kasi': 25000, 'Madhu': 78000, 'loc': 'guntur'}  
d2 = {'company': 'Lumen'}  
d3 = {\*\*d1, \*\*d2}  
print(d3)  
  
d1 = {'kasi': 25000, 'Madhu': 78000, 'loc': 'guntur'}  
d2 = {'company': 'Lumen'}  
d3 = d1|d2  
print(d3)

**Sequence vs iterable:**

* A sequence is an object which can be indexed like string, list, tuple
* A iterable is an object which can be traversed . all collection data types are iterables

****

## Operators:

**Arithmetic operators:**

* Addition (+)
* Subtraction(-)
* Multiplication(\*)
* Division (/)
* Modulas(%)
* Exponent(\*\*)
* Floor division(//): Performs division operation & rounds of coefficient to the nearest value.

**Relational operators:**

* Greater than(>) 🡪 True if left operand is greater than right
* Lesser than(<) 🡪 True if left operand is lesser than right
* Equal to (==) 🡪 True if both the operands are equal
* Not equal to (!=) 🡪 True if both the operands are not equal
* Greater than or equal to (>=) 🡪 True if the left operand is greater or equal to right
* Lesser than or equal to (<=) 🡪 True if the left operand is lesser than or equal to right
* := #creating & assigning the data to variable at a time
* print(a:= 100)
* print(str(a))

**Logical operators:**

**and:** Returns true if the both the operands evaluate to true, returns false if any of the operand is evaluate to false

**or**: Returns true if any of the operands evaluate to true. Returns false if any of the operand is evaluate to false

**not:** reverse the result, returns false if the result is true

**all:** Returns true if all the elements in an iterable is non default values. Or else it will returns false

Ex: all([1,89,’kasi’])

* True

**any:** Returns true if an iterable has at least one non default value, or elase returns false

Ex: any([7,0,’’,True])

* True

**Assignment operator:** assign the value of right side of expression to the left side operand

Ex: a = 10

a = a+5 or a+=5

a= 10  
a+=23  
print(a)

**Identity operators:**

Used to check if the two values are located in same part of the memory.

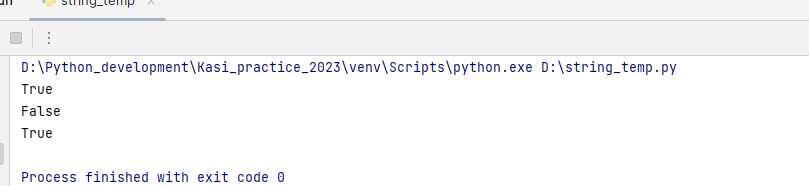
is: Returns true if both the values are pointing to same id

Ex: x is y

Is not: Returns true if the both the values are not pointing to same memory

Ex: x is not y

name = 'Mohana'  
name1 = name  
print(name is name1)  
print(name is not name1)  
a = 10  
b = 30  
print(a is not b)



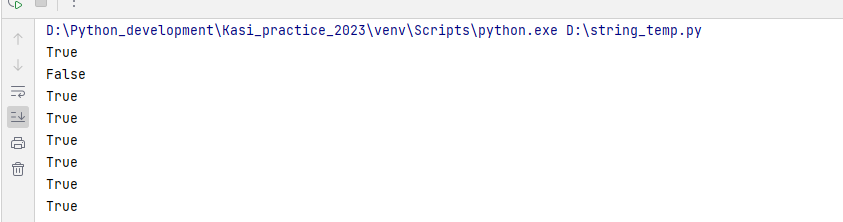
**Membership operator:**

Used to test if an element is present in an iterable or not

In: Evaluates to true if the element found in specified iterable other wise false

Not in: Evaluates to true if the element not found in specified iterable otherwise false

list\_ = ['kasi', 'jmr']  
print('jmr' in list\_)  
print('Mohana' in list\_)  
print('Mohana' not in list\_)  
string\_ = 'This is virtual world'  
print('is' in string\_)  
print('w' in string\_)  
set\_ = {522003, 'Guntur'}  
print('Guntur' in set\_)  
  
dict\_ = {'a': 1000, 'b':1008}  
print('b' in dict\_)  
x = 'kasi'  
print(x not in dict\_)



**Default values:**

* **Integer (int) 🡪 0**
* **Float 🡪 0.0**
* **Complex 🡪 0j**
* **Boolean(bool) 🡪 False**
* **String (str) 🡪 ‘’ or ‘’’’ or ‘’’’’’**
* **List (list) 🡪 []**
* **Tuple 🡪 ()**
* **Set 🡪 set()**
* **Dictionary (dict) 🡪 {}**

**Conditional statements:**

Conditional statements decide the **direction of the flow of execution** of a program.

**Types:**

If

If else

Elif

Nested if

**If statement:**

The block under the if, executes only when the condition satisfies

**Santax:**

If <condition>:

Statements

**Examples:**

*#WAP to check whether the user input number is even or not*number\_ = input(Enter the nummber)  
if int(number\_) %2 == 0:  
 print(fthe number {number\_} is even)  
  
  
*#WAP to check the given chracter is in lower case or not*char\_ = 'k'  
if char\_.islower():  
 print(the character is lower)  
  
if 'a'<= char\_ <= 'z':  
 print(the character is in lower case)  
  
*#write a program to check whether given element is present in collection or not*list\_ = [1,8, 'Mohana']  
num\_ = 8  
if num\_ in list\_:  
 print(its there)  
  
*#WAP to check given iterable is empty or not*list\_ = [8,'ksi']  
list\_ = []  
if list\_:  
 print(NOt empty)  
else:  
 print(empty)  
  
*#WAP to convert lowe case letters into upper and upper to lower*char\_ = 'M'  
if 'a'<= char\_<='z':  
 print(chr(ord(char\_)-32))  
else:  
 print(chr(ord(char\_)+32))  
  
if char\_.islower():  
 print(char\_.upper())  
else:  
 print(char\_.lower())  
  
*#WAP to check the string is starting with vowel or not*string\_ = 'iam Mohana kasi'  
if string\_[0] in 'aeiouAEIOU':  
 print('string starting with vowel')  
else:  
 print('string not starting with vowel')  
  
*#WAP to check the string is ending with vowel or not*string\_ = 'raising of the lord ele'  
if string\_[-1] in 'aeiouAEIOU':  
 print('string ending with vowel')  
else:  
 print('string not ending with vowel')  
  
*#WAP to check whether string is palindrome or not*string\_ = 'KASI'  
string\_ = 'DAD'  
if string\_ == string\_[::-1]:  
 print(palindrome)  
else:  
 print(not a palidnrome)  
  
*#WAP to check whether integer is a palindrome or not*number\_ = 1009  
if str(number\_) == str(number\_)[::-1]:  
 print(palindrome)  
else:  
 print(not a palindrome)  
  
*#WAP to check whether the iterable has even number of elements or not*list\_ = ['Mohana', 1008, 'guntur', 'rvr']  
  
if len(list\_) %2 == 0:  
 print('the list has even no of elements')  
else:  
 print('the list has odd number of elements')  
  
*#WAP to check if the first digit of the given number is odd or not*number\_ = 9945  
if int(str(number\_)[0]) %2 == 0:  
 print(the first digit is even)  
else:  
 print({the first digit is odd)

**In line if else conditional statements:**

**Syntax:** True statement if (condition) else false statement:

Example:

string\_ = theking  
  
print(the string has even characters if len(string\_)%2==0 else the string has odd characters)

**looping statements:**

**While loop ():** It is used when the number of iterations done is not known

**Steps to follow:**

* Keep a track of number of iterations with a reference variable
* Manually increment or decrement the reference variable

*#WAP to print -10 to to -1*n = -10  
while n<0:  
 print(n)  
 n+=1  
  
*#WAP to print even numbers from 1 to 50*n=1  
while n<=50:  
 if n%2==0:  
 print(n)  
 n+=1  
  
*#WAP to print all the characters in the string*string\_ = 'This is kasi nadh'  
n = 0  
while n<len(string\_):  
 print(string\_[n])  
 n+=1

**range():**

range(start, end, step):

* Returns a range object sequence of numbers, starting from 0 (by default), increments by 1 (by default) and ends at specified number
* To loop through code at specified number of times we use the range()

**for loop():**

* It is used when the number of iterations to be done is known
* There is no need to keep track of iterations manually, for loop() does it implicitly
* No need to increment the reference variable

**Enumerate(iterable):**

* It is a class which returns an enumerate object
* iterable can be a sequence, an iterator, or some other object which supports iteration
* returns a tuple containing index followed by value when we are appending in for loop.
* For dictionary it will return index & keys
* if you directly use enumerate on iterable it will return enumerate object

**reversed(sequence):**

returns a reversed iterator

sequence must be an object which has \_\_reversed\_\_() method

**loop control statements:**

1. **pass**
2. **break**
3. **continue**

**Pass:** performs no operation

**Break:** stops the execution of loop and break out of it

**Continue:** skips the current execution and continue with the next one

**Note:** If you want to empty the block without any operation at that time we use pass. Without using pass if we empty the block we will get error.

**for loop with else block:**

The else keyword in a for loop specifies a block of code to be executed when the loop is finished

**Syntax: for value in sequence:**

**Statements**

**Else:**

**Statements**

**Example:**

**Prime number check:**

A number which divided by 1 and it self is called prime number

**Zip():**

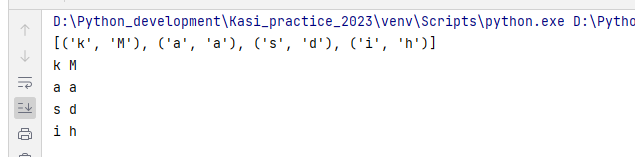
The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

If the passed iterables have different lengths, the **iterable with the least items** decides the length of the new iterator.

**Syntax: zip(iterator1, iterator2,.........)**

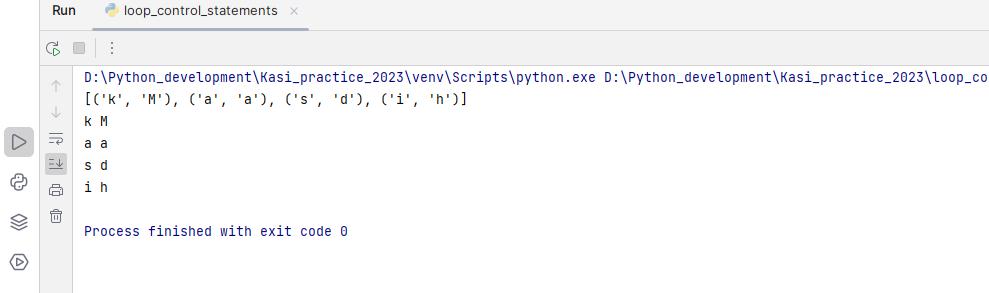
**Examples:**

s1 = 'kasi'  
s2 = 'Madh'  
print(list(zip(s1, s2)))  
for i1, i2 in zip(s1, s2):  
 print(i1, i2)

****

**If the iterators are of different lengths**

s1 = 'kasi'  
s2 = 'Madhuri'  
print(list(zip(s1, s2)))  
for i1, i2 in zip(s1, s2):  
 print(i1, i2)

****

**Zip\_longest:**

If we have different lengths in iterator will go for zip\_longest. Zip longest should be imported from itertools module

**Syntax:**

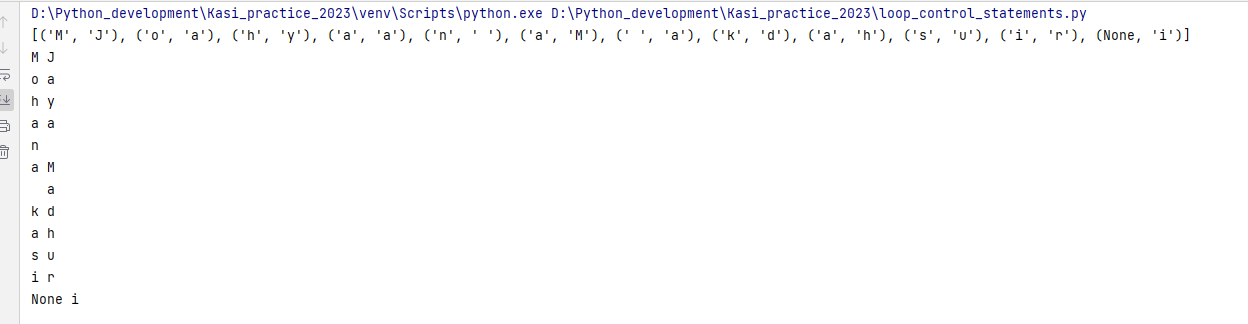
**From itertools import zip\_longest**

**Zip\_longest(iterator1, iterator2...............)**

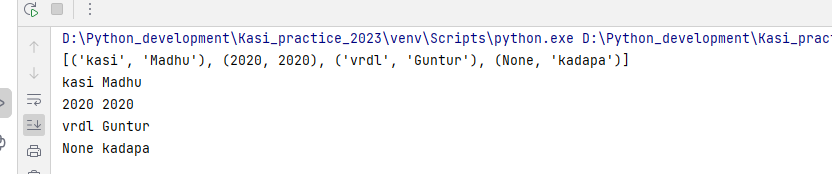
* **it will map with none if no items present in the any of the iterator**

**Examples:**

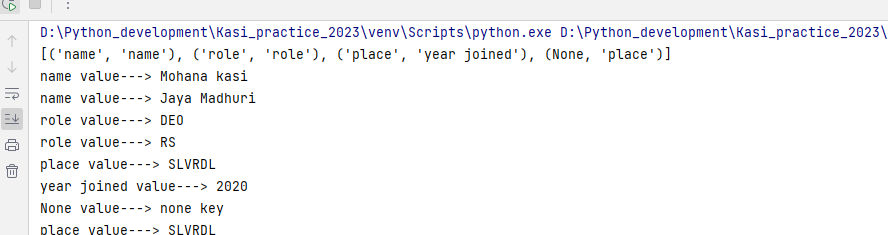
string1\_ = 'Mohana kasi'  
string2\_ = 'Jaya Madhuri'  
from itertools import zip\_longest  
print(list(zip\_longest(string1\_, string2\_)))  
for i1, i2 in zip\_longest(string1\_, string2\_):  
 print(i1, i2)

****

list1\_ = ['kasi', 2020, 'vrdl']  
list2\_ = ['Madhu', 2020, 'Guntur', 'kadapa']  
from itertools import zip\_longest  
print(list(zip\_longest(list1\_, list2\_)))  
for item1, item2 in zip\_longest(list1\_, list2\_):  
 print(item1, item2)

****

dict1\_ = {'name': 'Mohana kasi', 'role': 'DEO', 'place': 'SLVRDL'}  
dict2\_ = {'name': 'Jaya Madhuri', 'role': 'RS', 'year joined': 2020, 'place': 'SLVRDL'}  
from itertools import zip\_longest  
print(list(zip\_longest(dict1\_, dict2\_)))  
for key1, key2 in zip\_longest(dict1\_, dict2\_):  
 print(key1, value--->, dict1\_.get(key1, 'none key'))  
 print(key2, value--->, dict2\_.get(key2, 'None key'))

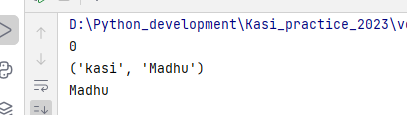
****

**Deep unpacking:**

It is a process where the elements in the nested data type will be unpacked based on the pattern in which they are stored

Example:

a = (0, ('kasi', 'Madhu'))  
index, item = a  
print(index)  
print(item)  
print(item[1])



*#WAP to create a dictionary with the word & its count pair in a string*string\_ = 'kasi coming from guntur and also kasi coming from bangalore to guntur'  
dict\_ = {}  
for word in string\_.split():  
 if word in dict\_:  
 dict\_[word] += 1  
 else:  
 dict\_[word] = 1  
print(dict\_)

**Default dict:**

* Default dict is used when the key has to initialize and update at the same time
* Unlike normal dictionary, here there is no need to check the condition for the presence of key
* Even if the key is not present default dict will create the key and initialize its default value, later the default value updated to its desired value

**Syntax: from collections import default dict**

**Var\_name = default dict(data type)**

**Comprehensions**

**List comprehension:**

List comprehension is an elegant way to define and create list based on existing list

**Syntax: [Expresssion for item in list]**

* It can be identifies when it receives a string or tuple and work on it like a list
* List comprehension can utilize conditional statements to modify the existing list

Only for loop syntax:

**[expression for var\_name in collection]**

For loop with only if block:

**[true block for var\_name in collection if <condition> ]**

For loop with if & else block:

**[true block if <condition> else false block for var\_name in collection]**

**Examples:**

*#WAP to create a list of tuples which is having index and its corresponding item  
#normal method*list\_ = ['Mohana', 'kasi', 'Guntur', 2020]  
res = []  
for index, item in enumerate(list\_):  
 res += [(index, item)]  
print(res)  
*#using list comprehension*list\_ = ['Mohana', 'kasi', 'Guntur', 2020]  
res = [(index, item) for index, item in enumerate(list\_)]  
print(res)  
  
*#WAP to create a list of even numbers from the below list  
#normal method*list\_ = [128, 226, 202, 6369, 789, 224, 3639, 447755]  
res = []  
for item in list\_:  
 if item%2==0:  
 res+=[item]  
*# print(res)  
#list comprehension*res = [item for item in list\_ if item%2==0] *#when having only if block*print(res)  
res = [item if item%2==0 else str(item)[::-1] for item in list\_] *#when having if & else block*print(res)  
  
  
*#WAP to create a list using the following list if the word is of even length store the word as it is, if it is of  
#odd length reverse the word & store it*list\_ = ['Mohana kasi', Madhu, 'Guntur', 'JMR', Rao]  
*#noraml method*res=[]  
for item in list\_:  
 if len(item)%2==0:  
 res += [item]  
 else:  
 res+=[item[::-1]]  
print(res)  
  
*#using list comprehension*res = [item if len(item)%2==0 else item[::-1] for item in list\_]  
print(res)  
  
*#WAP to create a list from the following list if the element are of type string keep them as it is else reverse it*list\_ = ['Mohana kasi', 2020, 57.89, True, madhuri, 'Guntur']  
*#Normal method*res = []  
for item in list\_:  
 if isinstance(item, str):  
 res += [item]  
 else:  
 res += [str(item)[::-1]]  
print(res)  
  
*#using list compreshension*res = [item if isinstance(item, str) else str(item)[::-1] for item in list\_]  
print(res)  
  
*#write a list comprehension to create strings starting with vowel*list\_ = ['engaged', 'irofa', 'an inter', 'Mohana', 'urs faith']  
res = [item for item in list\_ if item[0] in 'aeiouAEIOU']  
print(res)  
*#  
  
#set comprehension  
#WAP to create a set of squares from 1 to 10*res = {item\*\*2 for item in range(11)}  
print(res)  
  
*#Write a set comprehension to create a set of tuples of index, item*list\_ = ['kasi', 128, 1008, 'Guntrur']  
res = {(index, item) for index, item in enumerate(list\_)}  
print(res)  
  
*#write a set comprehension to create a set of tuples with its item & length pair*list\_ = ['java', 'Python kasi', 'Mohana kasi', madhu kasi]  
res = {(item, len(item)) for item in list\_}  
print(res)  
res = {(list\_[index], len(list\_[index])) for index in range(len(list\_))}  
print(res)  
  
*#write a set comprehension to create a set of even no's from the below list*list\_ = [12,89,77,665,223, 9969,648,7485,23148]  
res = {item for item in list\_ if item%2==0}  
print(res)  
  
*#write a set comprehension if the word is in a list of even length keep it as it is, if it is odd reverse it*list\_ = ['continuity', 'sequences', 'order of', 'england', 'india']  
res = {item if len(item)%2==0 else item[::-1] for item in list\_}  
print(res)  
  
*#write a set comprehension if the word in a list of type string keep it as it is, else reverse it*list\_ = ['rao', 'Madhuri', True, 'love', 'Mohana kasi', 2020]  
res = {item if isinstance(item, str) else str(item)[::-1] for item in list\_}  
print(res)  
  
*#write a program to create a dictionary with item & index pair  
#normal method*list\_ = ['recreate', 'Colors', 'Amazon', 'flipkart']  
res = {}  
for index, item in enumerate(list\_):  
 res[item] = index  
print(res)  
  
*#using dictionary comprehension*dict\_ = {item:index for index, item in enumerate(list\_)}  
print(dict\_)  
  
*#write a program to create a dictionary with word & its length pair*list\_ = ['Mohana kasi', 'Jaya Madhuri', 'porumamilla', 'Guntur']  
*#normal method*res = {}  
for word in list\_:  
 res[word] = len(word)  
print(res)  
*#dict comprehension*dict\_ = {word:len(word) for word in list\_}  
print(dict\_)  
  
*#write a program to create dictionary with character & its count pair*string\_ = 'python is a general purposs high level language'  
*#normal method*res = {}  
for char in string\_:  
 if char not in res:  
 res[char] = 1  
 else:  
 res[char] += 1  
print(res)  
*#using default dict*from collections import defaultdict  
res = defaultdict(int)  
for char in string\_:  
 res[char] +=1  
print(res)  
  
*#dict comprehension*res = {char:string\_.count(char) for char in string\_}  
print(res)  
  
*#write a program to create a dictioanry with word & count pair*list\_ = ['never', 'ever', 'do', 'and','ever', 'wrong', 'people', 'trust']  
dict\_ = {word:list\_.count(word) for word in list\_}  
print(dict\_)  
  
*#write a program to create a dictioanry with word & count pair only if the word is of even length*list\_ = ['never', 'ever', 'do', 'and','ever', 'wrong', 'people', 'trust']  
res = {}  
for item in list\_:  
 if len(item)%2==0:  
 res[item] = list\_.count(item)  
print(res)  
  
*#using default dict*from collections import defaultdict  
res = defaultdict(int)  
for item in list\_:  
 if len(item)%2==0:  
 res[item]+=1  
print(res)  
*#dict comprehension*dict\_ = {word:list\_.count(word) for word in list\_ if len(word)%2==0}  
print(dict\_)  
  
*#WAP to create a dictionary with index & word pair if the word is of odd length reverse it else keep it as it is*string\_ = 'This is Mohanakasi python world'  
*#normal method*res = {}  
for index, word in enumerate(string\_.split()):  
 if len(word)%2!=0:  
 res[index] = word[::-1]  
 else:  
 res[index] = word  
print(res)  
  
*#dict comprehension*dict\_ = {index: word[::-1] if len(word)%2!=0 else word for index, word in enumerate(string\_.split())}  
print(dict\_) *#in dict comprehension in else block no need to mention key only value is fine  
  
#WAP to create a dictionary with word and its length pair only if the word is starting with with vowel*string\_ = 'I am Mohana kasi I am from india even i am from Andhra pradesh also'  
*#normal method*res = {}  
for word in string\_.split():  
 if word[0] in 'aeiouAEIOU':  
 res[word] = len(word)  
print(res)  
  
*#dict comprehension*dict\_ = {word:len(word) for word in string\_.split() if word[0] in 'aeiouAEIOU'}  
print(dict\_)  
  
*#WAP to create a dictionary of item & its index pair if the item is of string data type reverse it else keep it as its*list\_ = ['Mohana kasi', True, 2020, 'Python world', 20+89j, 96.56]  
*#normal method*res = {}  
for index, item in enumerate(list\_):  
 if isinstance(item, str):  
 res[item[::-1]] = index  
 else:  
 res[item] = index  
print(res)  
  
*#dict comprehension*dict\_ = {index: item[::-1] if isinstance(item, str) else item for index, item in enumerate(list\_)}  
print(dict\_)  
  
*#Write dictionary comprehension to flip or swap keys & values in a dictioanary*d = {'a': 1, 'b':2, 'c':3}  
dict\_ = {value:key for key, value in d.items()}  
print(dict\_)  
dict\_ = {d[key]: key for key in d}  
print(dict\_)  
  
*#Write a dictionary comprehension of character & its ASCII value pair*string\_ ='Mohana kasi'  
dict\_ = {char:ord(char) for char in string\_}  
print(dict\_)

**Functions:**

* Function is a block of code which runs when it is called
* Functions helps break our program into smaller chunks
* As a program goes longer or larger, functions makes it as more organized & manageable
* Function avoids and make code reusable

**Types of functions:**

* + Built in functions
  + User-defined functions

**Built in functions:**

* Functions that are pre-defined in the software to perform some specific task

Ex: print(), input(), len(), dir(), id(), isinstance(), round()

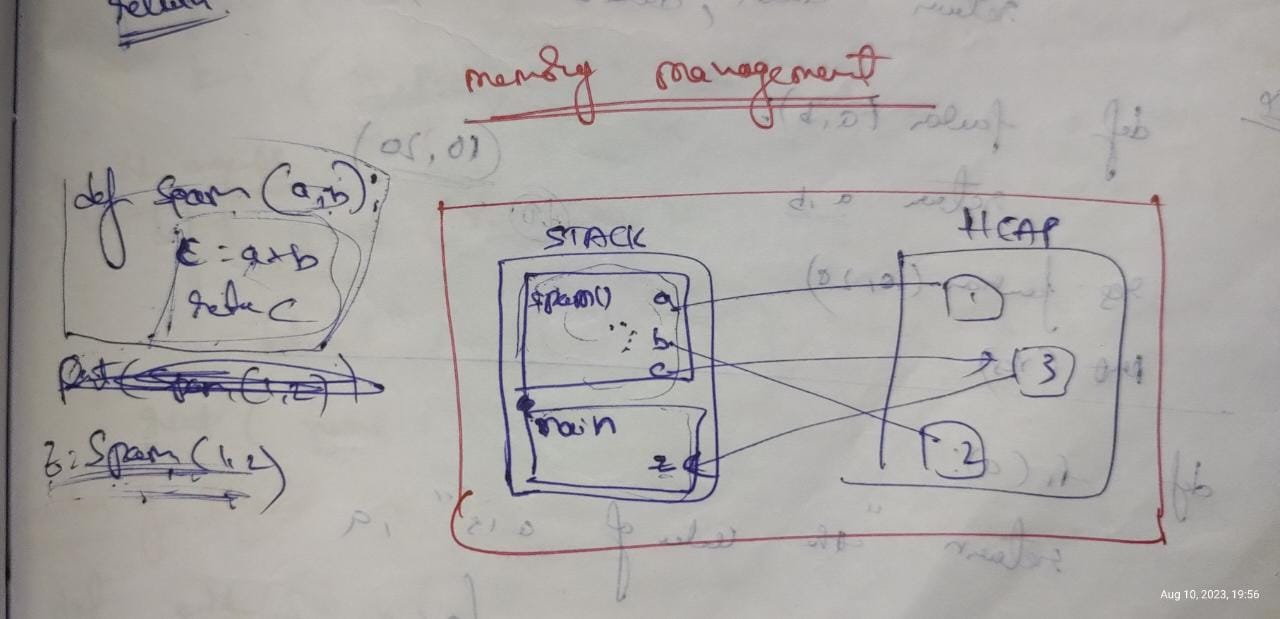
**User-defined functions:**

* Defined by the user in order to perform some specific task
* **Syntax : def function\_name(parameters):  
   statements1**

**Statement2**

**function\_name(values) 🡪 function call**

**Memory management:**

****

**Return statement:**

* return statement is used to get or return the values from a function to the caller
* The return statement is used to exit a function and go back to the place where it is called
* The statements after the return statement will not be executed once after return statement executed
* It can be return single or multiple data
* Syntax: return data

Or

return data1, dat2, data3.......

**Example:**

def add(a, b):  
 return a+b  
print(add(10,50))  
print(add(50,60))

**Type of arguments:**

* **Positional arguments**
* **Keyword arguments**
* **Combination of positional & keyword arguments**
* **Keyword & position only arguments**
* **Variable positional arguments --🡪 \*args**
* **Variable keyword arguments 🡪 \*\*kwargs**

**Position only arguments:**

def temp\_func(name, age):  
 print(f{name} is {age} years old)  
  
temp\_func('kasi', 27)

**Keyword arguments:**

def add(a, b):  
 print(fthe sum is {a+b})  
  
add(a=50, b=80)

**combination of position & keyword arguments:**

def sum(a, b, c, d):  
 print(fthe sum is {a+b+c+d})  
  
sum(18, 78, c=98, d=25)

**only posiotional:**

def add(a, b, c, d,/): 🡪 before / only positional arguments  
 return a+b+c+d  
print(add(10,20,50,90))

def add(a,b,/, c,d): 🡪 after / we can define keyword or positional both  
 print(a+b+c+d)   
  
add(10,20,c=80,d=6)

**keyword only arguments:**

def add(\*,a,b,c,d): 🡪 after \* only keyword argunents  
 print(a+b+c+d)  
  
add(a=10,b=90,c=60,d=60)

def add(a,b,\*,c,d): 🡪 before \* any type of arguments accepted but after \* only positional  
 print(a+b+c+d)  
  
add(a=52,b=96, c=10,d=65)  
add(10,20,c=50,d=60)

**Note:** / represents positional only arguments. If a function definition has / in the parameters place then the parameters defined before / only accept the positional arguments

But the parameters after / can be accept either positional or keyword arguments

\* represents keyword only arguments its specifies the parameters after \* mandatory accepts keyword arguments only. Where as the parameters after \* can accept both positional & keyword arguments

**Combination of both positional only, keyword only arguments:**

def add(a,b,/, \*, c, d):  
 print(a+b+c+d)  
  
add(1,2,c=6, d=45)  
add(a=63, b=69, 10,20) 🡪 wrong declaration

Examples:

def even\_num(start, end):  
 list\_ = []  
 for num in range(start, end+1):  
 if num%2==0:  
 list\_.append(num)  
 return list\_  
  
print(even\_num(10,30))

**Default parameters:**

* Default value indicates that function argument will take that value when no argument value passed during the function call
* The default value is assigned by using assignment (=) operator

Example:

def even\_print(end, start=0):  
 for num in range(start, end+1):  
 if num%2==0:  
 print(num, end=' ')  
  
even\_print(50)  
print()  
even\_print(50, 20)  
print()  
even\_print(50, start=30)

Note: Any global value can be assigned as a value to the default parameter.

If there is any modification done on that global variable after function definition then it will have no effect on the default value that has already been assigned

Example:

def add(a, b=x):  
 print(a+b)  
  
x = 100  
add(100)

**Variable positional arguments:**

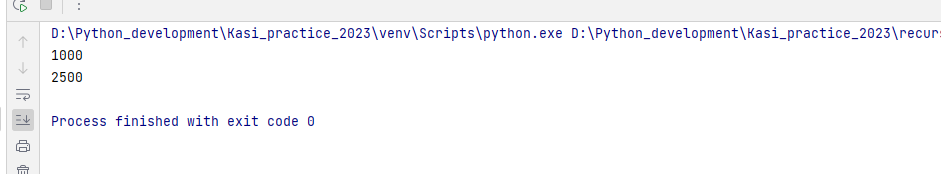
* \*args is used when the user doesn’t know how many arguments passed to a functiom
* \* args used in a function definition, it is called as packing
* By default packed data will be in the form of tuple

**Variable keyword arguments:**

* **\*\***kwargs is used as parameter in the function
* def var\_kw(\*\*kwargs):  
   for item in kwargs:  
   print(type(kwargs))  
   print(kwargs[item])  
    
  var\_kw(name='kasi', loc='Guntur')
* all keyword arguments will be packed in the form of dictionary by default

**Recursion:**

* **It is a function which call itself until the condition being satisfied**
* **Used to avoid the loop in some cases**
* **Python recursion limit can be found with a function from the sys module called getrecursionlimit()**
* from sys import getrecursionlimit  
  print(getrecursionlimit())
* We can change this value by using setrecursionlimit() function
* from sys import setrecursionlimit  
  setrecursionlimit(2500)  
  print(getrecursionlimit())



from sys import getrecursionlimit  
print(getrecursionlimit())  
from sys import setrecursionlimit  
setrecursionlimit(2500)  
print(getrecursionlimit())  
  
*#WAP to print 10 numbers  
# def numb(n):  
# if n<=10:  
# print(n)  
# n += 1  
# numb(n)  
# else:  
# return  
#  
# numb(1)  
  
#Write a recursion function to print numbers from 10 to 1*def nums(n):  
 if n>=1:  
 print(n)  
 n -=1  
 nums(n)  
 else:  
 return  
  
*# nums(10)  
  
#write a recursive function to find the sum of first n numbers*def sum\_n(n, sum=0):  
 *# print(sum)* if n>0:  
 *# print(n)  
 # print(sum)* sum += n  
 n -= 1  
 return sum\_n(n, sum)  
 else:  
 return sum  
print(sum\_n(20))  
  
*#wrirte a recursive function to count the number of digits in a number*def num\_count(num, count=0):  
 if num>0:  
 count += 1  
 num //= 10 #it will removes the last digit from the number  
 return num\_count(num, count)  
 else:  
 return count  
print(num\_count(1254))  
  
*#wrirte a recursive function to add the digits in a number*def num\_digit\_sum(num, sum=0):  
 if num>0:  
 last\_digit = num % 10 #it will gives last digit of a number  
 sum += last\_digit  
 num //= 10  
 return num\_digit\_sum(num, sum)  
 else:  
 return sum  
  
print(num\_digit\_sum(1234))  
  
*#write a recursive function to reverse a string*string\_ = 'Thi'  
def str\_reverse(string\_, res='', length=len(string\_)):  
 if length>0:  
 res += string\_[length-1]  
 length -= 1  
 return str\_reverse(string\_, res, length)  
 else:  
 return res  
  
print(str\_reverse(string\_))  
  
def reversed\_string(str\_, res='', index=0):  
 if index < len(str\_):  
 res = str\_[index]+ res  
 index += 1  
 return reversed\_string(str\_, res, index)  
 else:  
 return res  
  
print(reversed\_string('Mohana'))

**Lambda function:**

* Any anonymous function that defined without name is called as lambda function
* While in normal functions are defined using the def keyword, anonymous functions are defined using lambda keyword
* Syntax: lambda [arg1, arg2, arg3……..]: expression
* after creating lambda function you have to store it in a variable and we have to call that variable by passing arguments
* **in lambda function we can take n number of arguments**
* **but expression is only one to take**
* and only one **single---->if block** is **not acceptable** in lambda function
* if we want to take **if** block in lambda function we have to take else block also
* we can take and, or logical operators in a single expression
* pass break continue keywords are not used in lambda function
* we can take collection (one or any no of) as argument to the lambda function but expression is one only
* we **can’t do looping** in lambda functions
* to achieve looping we use **map and filter** functions along with the lambda functions

**Normal function:**

def add(a, b):  
 return a+b  
print(add(10,50))  
print(add(50,60))

**Using lambda function:**

add = lambda a, b: a+b  
print(add(15,15))

add = lambda a, b: a+b  
print(add(15,15))  
  
write a lambda expression that checks the given no is even or not  
even\_check = lambda num: num%2==0  
print(even\_check(10))  
print(even\_check(3))  
  
write a lambda expression that multiplies two numbers  
num\_mul = lambda num1, num2: num1\*num2  
print(num\_mul(10,20))  
  
write a lambda expression that return last elemts of the sequence  
last\_element = lambda sequence: sequence[-1]  
print(last\_element('Mohana kasi'))  
print(last\_element([1,9, 'Kasi']))  
  
write a lambda function that checks if the string is palindrome or not  
pal\_check = lambda string\_: string\_ == string\_[::-1]  
print(pal\_check('MALAYALAM'))  
print(pal\_check('kasi'))  
  
lambda expression to check string is a palindrome and to print it if it is a palindrome  
pal\_check = lambda string\_: string\_ if string\_ == string\_[::-1] else None  
print(pal\_check('kasi'))  
print(pal\_check('1001'))  
print(pal\_check('mom'))  
  
write a lambda express that print's whether the the number is even or odd  
evn\_odd\_check = lambda num: even if num %2==0 else odd  
print(evn\_odd\_check(12))  
print(evn\_odd\_check(3))  
  
lambda expression to check string is of even length and string stars with vowel  
ev\_vow = lambda string\_: len(string\_) %2 == 0 and string\_[0] in 'aeiouAEIOU'  
print(ev\_vow('kasi'))  
print(ev\_vow('Inox'))

**map():**

* map function applies a function to all the items in the iterable
* map takes a function and iterable as an arguments
* in map we can have n no of iterables but the no of iterables must match with the parameters in the function definition
* if the multiple iterables are of different length data loss will occurs
* in the function we must use return statement for result
* if we takes only one if block in function the map will implicitly returns none for the false results
* after completion of map function we have take it into a variable and type cast it (to list tuple or set only)
* Syntax: map(function\_to\_apply, list of inputs) 🡪 while mentioning function name no parenthesis needed

"""wap to convert all the strings in the list to upper case using map"""  
def upper\_conv(string\_):  
 return string\_.upper()  
  
list\_ = ['Mohana kasi', 'guntur']  
res = map(upper\_conv, list\_)  
print(list(res))  
  
"""wap to convert all the strings in the list to lower case using map"""  
list2\_ = ['MOHANA KASI', 'BENGALURU']  
def lower\_conv(string\_):  
 return string\_.lower()  
  
res = map(lower\_conv, list2\_)  
print(list(res))  
  
"""using lambda"""  
res = map(lambda string\_: string\_.lower(), list2\_)  
print(list(res))  
  
"""wap to convert allthe negeative numbers into positive using map"""  
list\_neg\_nums = [-8, -36, -22359]  
res = map(lambda num: abs(num), list\_neg\_nums)  
print(list(res))  
  
"using funtion"  
def pos\_conv(num):  
 return abs(num)  
  
res = map(pos\_conv, list\_neg\_nums)  
print(list(res))  
  
"""write a map function that return a list of even numbers inside a list using map"""  
list\_ = [12, 336, 3, 69]  
def even\_num\_check(num):  
 if num%2 == 0:  
 return num  
  
res = map(even\_num\_check, list\_)  
print(list(res))  
  
"""write a map function that takes two lists and return sum of each elements\n"""  
"""(sum of first index item of 1ist1 with first index item of list2 .....)"""  
  
  
def sum\_lists(item1, item2):  
 return item1 + item2  
  
list1\_ = [1,89,660]  
list2\_ = [23,969,6]  
res = map(sum\_lists, list1\_, list2\_)  
print(list(res))  
  
res = map(lambda item1, item2: item1+item2, list1\_, list2\_)  
print(list(res))  
'''write a map function that returns lenth of the each string present inside the list, tuple, set, and dictionary '''  
def lenght\_strings(string\_):  
 return len(string\_)  
  
list\_ = ['Guntur', 'python', 'city']  
res = map(lenght\_strings, list\_)  
print(list(res))  
  
res = map(lambda item: len(item), list\_)  
print(list(res))  
  
"""using dictionary as an iterable"""  
def len\_strings(key):  
 if isinstance(dict\_[key], str):  
 return len(dict\_[key])  
  
dict\_ = {'name': 'Mohana kasi', 'location': 'Guntur', 'age':"27"}  
res = map(len\_strings, dict\_)  
print(list(res))  
  
res = map(lambda key: len(dict\_[key]), dict\_)  
print(list(res))  
  
'''write a map function that returns the numeric data present inside the list, tuple, set, and dictionary '''  
def num\_check(item):  
 if isinstance(item, int):  
 return item  
list\_ = ['kasi', 2020, 6.6, 1995]  
res = map(num\_check, list\_)  
print(list(res))  
  
res = map(lambda item: item if isinstance(item, int) else None, list\_)  
print(list(res))  
  
def num\_check\_dict(key):  
 if isinstance(dict2\_[key], int):  
 return dict2\_[key]  
  
dict2\_ = {'name': 'kasi', 'age': 27}  
res = map(num\_check\_dict, dict2\_)  
print(list(res))  
  
res = map(lambda key: dict2\_[key] if isinstance(dict2\_[key], int) else None, dict2\_)  
print(list(res))  
  
"""write a map function that returns the strings having even lenth present inside a list"""  
def even\_string\_check(string\_):  
 if len(string\_) %2==0:  
 return string\_  
  
list\_ = ['Greater', 'wisdom', 'humorous','thriller']  
res = map(even\_string\_check, list\_)  
print(list(res))  
  
res = map(lambda string\_: string\_ if len(string\_) %2==0 else None, list\_)  
print(list(res))  
  
""""write a map function to create a dictionary of word and it's count pair in the following sentence"""  
def word\_count(word):  
 return word, string\_.split().count(word)  
  
string\_ = 'This is not a normal thing is it more a complexable'  
res = map(word\_count, string\_.split())  
print(dict(res))  
  
res = map(lambda word: (word, string\_.split().count(word)), string\_.split())  
print(dict(res))  
  
""""write a map function to return the words starting with vowels"""  
def vowl\_check(word):  
 if word[0] in 'aeiouAEIOU':  
 return word  
  
list\_ = ['kasi', 'Mohana', 'India', 'Inox', 'user id', 'england', 'outstanding']  
res = map(vowl\_check, list\_)  
print(list(res))  
  
res = map(lambda word: word if word[0] in 'aeiouAEIOU' else None, list\_)  
print(list(res))  
  
""""write a map function to return the palindrome strings inside a list"""  
def pal\_check(string\_):  
 if string\_ == string\_[::-1]:  
 return string\_  
  
list\_ = ['kasi', 'malayalam', '1001', '1008']  
res = map(pal\_check, list\_)  
print(list(res))  
res = map(lambda item: item if item == item[::-1] else None, list\_)  
print(list(res))  
  
"""write a map function that returns the list of numbers raised to the power of their indices using map"""  
"""we cant directly unpack the items in the argument position of lambda function"""  
def powers\_num(item):  
 return item \*\* list\_.index(item)  
  
list\_ = [12,3,4]  
res = map(powers\_num, list\_)  
print(list(res))  
res = map(lambda item: item \*\* list\_.index(item), list\_)  
print(list(res))  
  
def powers(tuple):  
 index, item = tuple  
 return item \*\* index  
  
list\_ = [12,89,33]  
res = map(powers, list(enumerate(list\_)))  
print(list(res))  
  
res = map(lambda tuple: tuple[1] \*\* tuple[0], list(enumerate(list\_)))  
print(list(res))  
  
  
"""write a map function that returns all the words in lower case in the given sentence"""  
def lower\_words(word):  
 return word.lower()  
  
sentence = 'THis IS PYTHON LANGUAGE'  
res = map(lower\_words, sentence.split())  
print(list(res))  
  
res = map(lambda item: item.lower(), sentence.split())  
print(list(res))

**Filter():**

* the filter function takes a function and an iterable as an argument
* it provides an elegant way to filter out all the elements of a sequence for which the function returns true
* filter creates a list of elements for which a function returns true
* filter takes only one iterable as an argument
* filter passes each element in the iterable through the function and returns elements which was true
* if we given any expression after return statement it doesn't execute that
* Syntax: filter(function, iterable)

"""filter out the even values in the list"""  
def even\_check(item):  
 if item %2==0:  
 return item  
  
list\_ = [12, 6, 9, 11, 98, 250, 38]  
res = filter(even\_check, list\_)  
print(list(res))  
  
res = filter(lambda item: item if item%2==0 else None, list\_)  
print(list(res))  
  
"""write a program that returns a list of strings with odd length using filter function"""  
def odd\_leng\_strings(item):  
 if len(item) %2 !=0:  
 return item  
  
list\_ = ['Mohana kasi', 'Python coder', 'Oracle', 'Guntur']  
res = filter(odd\_leng\_strings, list\_)  
print(list(res))  
res = filter(lambda item: item if len(item) %2!=0 else None, list\_)  
print(list(res))  
  
"""write a function that return a list of only even numbers inside a list using filter()"""  
def even\_check(item):  
 if item %2 == 0:  
 return item  
  
list\_ = [123, 663, 9, 120, 36, 8]  
res = filter(even\_check, list\_)  
print(list(res))  
  
'''write a function that returns only the numeric data present inside the list, tuple, set, and dictionary '''  
def nums\_check(item):  
 if isinstance(item, int):  
 return item  
list\_ = ['kasi', 2020]  
res = filter(nums\_check, list\_)  
print(list(res))  
  
res = filter(lambda item: item if isinstance(item, int) else None, list\_)  
print(list(res))

**Global variable:**

* A variable created in global name space i.e outside a function
* It can be accessed inside a function but can’t be modified
* In order to modify a global variable inside a function then we need to use global keyword

"accessing gloabal variable inside a function"  
a = 10  
def temp\_():  
 print(a)  
 *# a += 1 --> gives the error*temp\_()  
  
"accessing and modifying global variable inside a function"  
new\_var = 100  
def spam():  
 global new\_var  
 new\_var += 10  
  
spam()  
print(new\_var)

**Local variable():**

* A variable which is created inside a function is called local variable and it cant be accessed outside the function
* def temp\_():  
   var\_ = 108  
   var\_ += 10  
   print(var\_)  
    
  temp\_()  
  *# print(var\_) --> gives the error*

**Non local scopes:**

* Non local scopes are those which are neither local or global
* If a nested function has to modifies the variable of outer function, non local keyword should be used
* def outer\_func():  
   a = 12  
   def inner\_func():  
   print(a)  
   *# a += 1 --> gives you the error as cant be modified* inner\_func()  
  outer\_func()

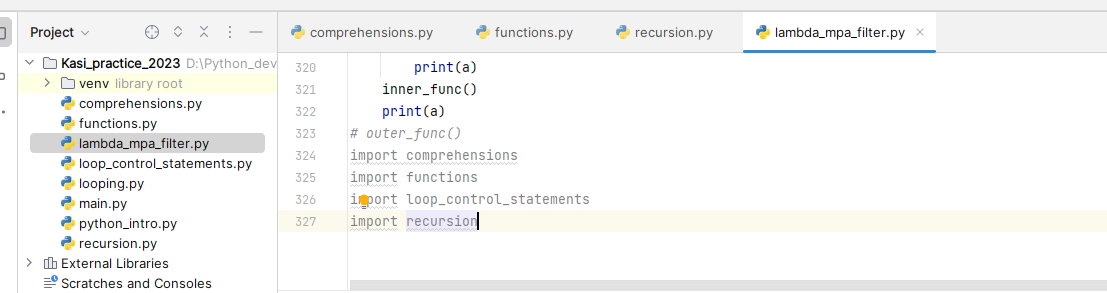
def outer\_func():  
 a = 12  
 def inner\_func():  
 nonlocal a  
 a += 10  
 print(a)  
 inner\_func()  
 print(a)  
outer\_func()

**Module:**

* A module is a file consisting of python code
* It can have functions, classes, variables
* Any python file can be referred as a module

**Types of modules:**

**User defined modules:**

* Created by user
* Can be accessed using import keyword
* 
* Syntax: import module\_name

**Built in module:**

* Pre-defined modules
* To install these we need to use pip command
* pip install module\_name

**import statement:**

* used to import modules in any program
* it can be written anywhere in the program
* any number of modules imported in a module
* Syntax: import module name

**From keyword:**

* It is used when we need to import some module from different package
* Syntax: from module\_name import function
  + from package\_name import module\_name
  + from package\_name.module\_name import function

Ex: from itertools import zip\_longest  
from collections import defaultdict

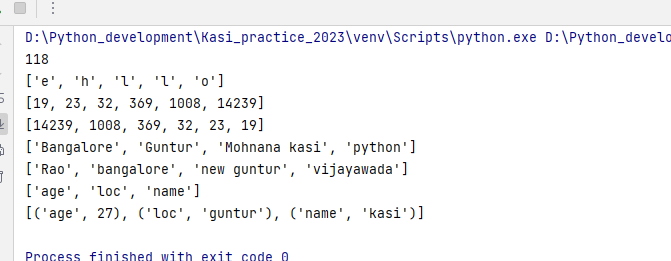
**As keyword:**

* used to given an alias name for the module as well as function
* Syntax: from module\_name import function as alias\_name
  + From package\_name import module\_name as alias name
  + From package\_name.module\_name import function as alias\_name

**sorted():**

* Returns a new sorted list from the items in the iterable
* Has two optional arguments which must be specified as keyword arguments
* Key specifies a function of one argument that is used to extract a comparison key from each element in the iterable. The default value is None
* Reverse is a Boolean value. If is true the elements are sorted in reversed order when it is in false the elements are sorted normal

string\_ = 'hello'  
print(sorted(string\_))  
list\_ = [369,32, 19, 23, 1008, 14239]  
print(sorted(list\_))  
print(sorted(list\_, reverse=True))  
t = ('Mohnana kasi', 'python', 'Guntur', 'Bangalore')  
print(sorted(t))  
set\_ = {'new guntur', 'vijayawada', 'Rao', 'bangalore'}  
print(sorted(set\_))  
dict\_ = {'name': 'kasi', 'age': 27, 'loc': 'guntur'}  
print(sorted(dict\_))  
print(sorted(dict\_.items()))

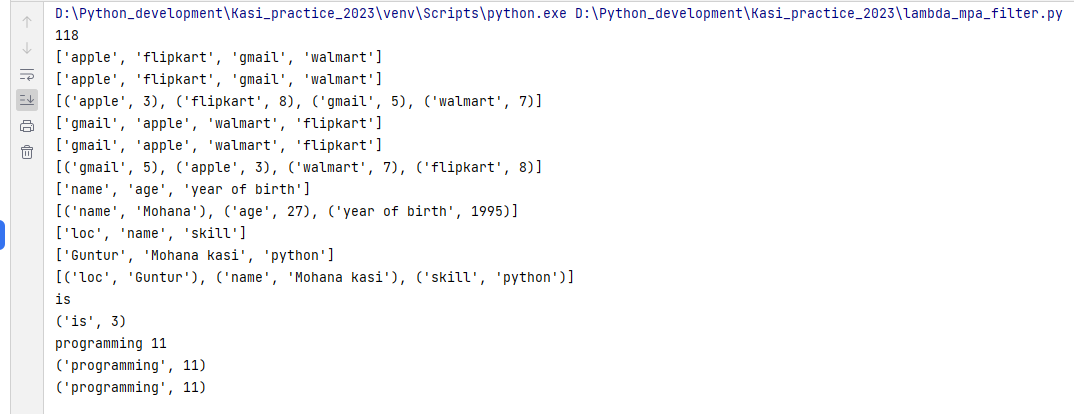


"write a program to sort the elements present in a list based on their length"  
list\_ = ['Mohana kasi', 'Guntur', 'python coding', 'never regrets']  
print(sorted(list\_, key=len, reverse=False))  
print(sorted(list\_, key=len, reverse=True))  
  
"write a program to find the largest and shortesd word in the following string"  
sentence\_ = 'python is a high level language'  
res = sorted(sentence\_.split(), key=len, reverse=False)  
print(res[0]) *#shortest word*print(res[-1]) *#largest word*"write a program to find the largest and shortesd word along with their lengths in the below sentence"  
sentence\_ = 'python is a high level language'  
res = sorted(sentence\_.split(), key=len)  
print(res[0], len(res[0]))  
print(res[-1], len(res[-1]))  
  
"write a program to sort the below list elements based on the last character of each string"  
list\_ = ['python', 'Mohana', 'kasi', 'testing', 'Bangalore']  
def last\_char(item):  
 return item[-1]  
  
print(sorted(list\_, key=last\_char))  
print(sorted(list\_, key= lambda item: item[-1]))  
  
"WAP to sort the below list based on the first character of each element"  
list\_ = ['python', 'Mohana', 'kasi', 'testing', 'Bangalore']  
def first\_char(item):  
 return item[0]  
  
print(sorted(list\_, key=first\_char))  
print(sorted(list\_, key = lambda item: item[0]))



Dictionaries sorting:

dict\_ = {'gmail': 5, 'walmart':7, 'apple':3, 'flipkart':8}  
print(sorted(dict\_)) *#sorts based on key ascii values*print(sorted(dict\_.keys())) *#sorts based on key ascii values*print(sorted(dict\_.items())) *#sorts based on keys & keeps values as it is  
  
#based on length of the keys*print(sorted(dict\_, key=len))  
print(sorted(dict\_.keys(), key=len))  
print(sorted(dict\_.items(), key = lambda item: len(item[0])))  
  
"write a program to sort the dictionary based on the keys last item"  
dict\_ = {'name': 'Mohana', 'year of birth': 1995, 'age': 27}  
print(sorted(dict\_, key= lambda item: item[-1]))  
print(sorted(dict\_.items(), key = lambda item: item[0][-1]))  
  
"write a program to sort the dictionary based on the values"  
dict\_ = {'name': 'Mohana kasi', 'loc': 'Guntur', 'skill': 'python'}  
print(sorted(dict\_, key = lambda key:dict\_[key]))  
print(sorted(dict\_.values()))  
print(sorted(dict\_.items(), key=lambda item: item[-1]))  
  
*#WAp to get the most repeated word*sentence\_ = 'is having is local you of you is not goog not bad'  
print(sorted(sentence\_.split(), key = lambda item: sentence\_.split().count(item))[-1])  
dict\_ = {key:sentence\_.split().count(key) for key in sentence\_.split()}  
print(sorted(dict\_.items(), key = lambda item: item[-1])[-1])  
*#WAP to print longest word with its length*sentence\_ = 'python is s high level general purpous programming language'  
res = sorted(sentence\_.split(), key = lambda item: len(item))  
print(res[-1], len(res[-1]))  
dict\_ = {key: len(key) for key in sentence\_.split()}  
res = sorted(dict\_.items(), key=lambda item:item[-1])  
print(res[-1])  
  
*#largest non repeated word*dict\_ = {key: len(key) for key in sentence\_.split() if sentence\_.split().count(key)==1}  
res = sorted(dict\_.items(), key = lambda item: item[-1])  
print(res[-1])



**File handling:**

**#Methods related to directory**

**1.Getcwd(): gets the current working directory**

**2.mkdir(): creating a new directory**

**3.chdir(): changes the current working directory**

**4.rmdir(): removes the directory**

**5.listdir(): used to get the list of all files and directories in a specified directory**

import os  
old\_dir = os.getcwd() *#it will gives the current working directory*print(old\_dir)  
os.chdir(r"D:\Kasi\_exp\_doc") *#changes the current working diectory to specified one*print(os.getcwd())  
os.chdir(old\_dir)  
print(os.getcwd())  
os.mkdir(r"D:\Kasi\_exp\_doc\kasi\_madhu\new6") *#it will create a new directory, when the directory already exists, it will throws error. Here it will create end folder i.e new6*os.rmdir(r"D:\Kasi\_exp\_doc\kasi\_madhu\new6") *#removes the directory*print(os.listdir(r"D:\Kasi\_exp\_doc"))

**Methods related to file:**

**1.popen(filename, mode):** Similar to open(), provides a pipe/gateway and access to file directory. It will works for only file not for directory

**2.rename(old\_name, new\_name):** renames the file with the new name, it will works for file as well as directory also

**3.remove():** Used to remove or delete a file path. This method cannot be be remove or delete a directory. If the specified path is a directory then OS error will be raised

os.popen(r"D:\Kasi\_exp\_doc\Q\_spiders\_certificate.pdf")*#opeing a file*os.rename(r"D:\Kasi\_exp\_doc\kasi\_madhuri\new10", r"D:\Kasi\_exp\_doc\kasi\_madhuri\new8") *#renaming folder*os.rename(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhu.txt", r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt") *#renames file*os.remove(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_file.docx")

**#methods related to path**

**Os.path.exists():** This method will check whether a file exists or not by passing the name of the file as parameter

Os module has a sub module named PATH by using it we can perform many more operations

**Os.path.getsize():** In this method python will give us the file size. To use this method we need to pass the path of the file

print(os.path.exists(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt")) *#checking for file path*print(os.path.exists(r"D:\Kasi\_exp\_doc\kasi\_madhu")) *#checks path is there or not*print(os.path.getsize(r"D:\Kasi\_python\_notes.docx")) *#gives the file*print(os.path.isfile(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt")) *#checks path is a file path or not*print(os.path.isdir(r"D:\Kasi\_exp\_doc")) *#checks passed path is directory path or not*

**files:**

**files are name locations on disk to store related information**

**when we want to read from a file or write from a file we need to open it first**

**when we are done we need to close it first**

**\*open a file**

**\*Read / write a into a file**

**\*Close the file**

**Opening a file in python:**

**In python file can be opened in two methods**

* Without context manager
* With using context manager

**Without using context manger:**

File\_obj = open(file\_name, mode)

**With context manager:**

with open(file\_name, mode) as file\_obj:

pass

**Modes:**

There are four different methods for opening a file

‘r’ 🡪 read opens a file for reading, throws error if file not exist

‘a’ 🡪 opens a file for appending, creates the file if the file doesn’t exist

‘w’ 🡪 opens a file for writing, Creates the file if doesn’t exist (overrides the content of the file if already exists)

‘x’ 🡪 Creates the specified file, returns an error if the file exist

**Note:** When a file is opened without using context manager (Without with block) file remains open unless we close it manually

When a file is opened using context manager (using with block) the file remains open unless the control is in the with block. Once the control is outside the with block the file will be automatically closed

**#without using context manger:**

f = open(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt", 'r')  
print(f) *#it will print the file object, if we need to print the data in the file we need to use for loop*for line in f:  
 print(line)  
print(f.closed)  
f.close()  
print(f.closed)

**#With context manger:**

with open(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt") as file\_1:  
 for line in file\_1:  
 print(line)

**file object attributes:**

**file.closed:** returns true if the file is closed

**file.mode:** returns access mode with the with the which file is opened

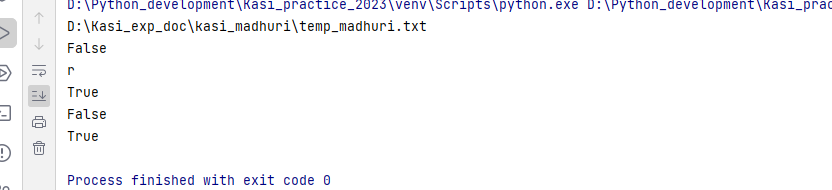
**file.name:** Returns name of the file

**file.readable():** Returns true if the file is opened in read mode

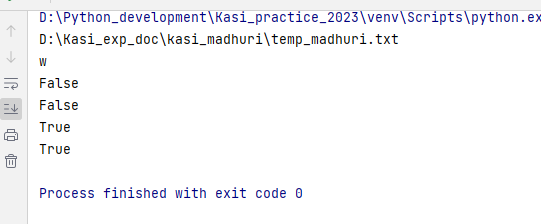
**file.writable():** Returns true if the file is opened in write mode

**file.close(): closes the file**

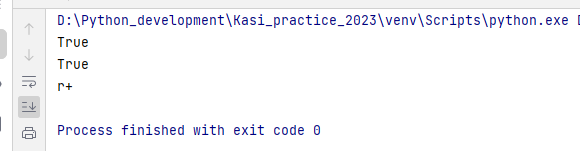
file = open(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt")  
print(file.name)  
print(file.closed)  
print(file.mode)  
print(file.readable())  
print(file.writable())  
file.close()  
print(file.closed)

****

with open(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt", 'w') as file1:  
 print(file1.name)  
 print(file1.mode)  
 print(file1.closed)  
 print(file1.readable())  
 print(file1.writable())  
print(file1.closed)

****

with open(r"D:\Kasi\_exp\_doc\kasi\_madhuri\temp\_madhuri.txt", 'r+') as file:  
 print(file.readable())  
 print(file.writable())  
 print(file.mode)

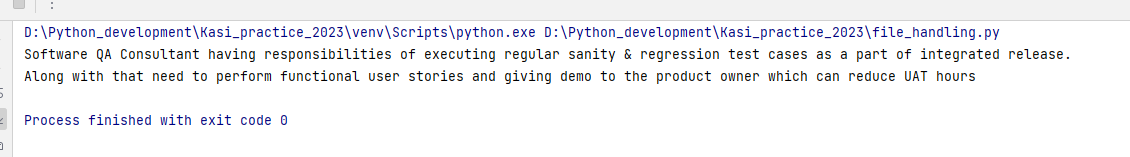
****

"functions to be poerform on read operaration"  
"read()"  
"readlin()"  
"readlines()"

**Read(): reads the data from starting to ending of the file**

**Note: read() can have integer as an argument. It specifies the number of characters to be read from the starting**

with open(r"D:\data1.txt", 'r') as file1:  
 print(file1.read())

****

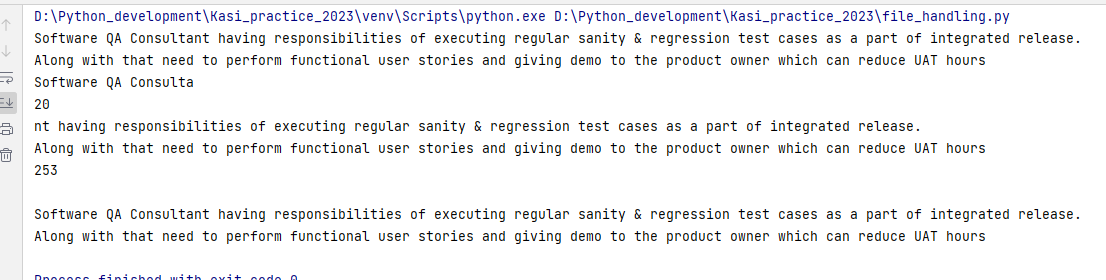
**Note: once the file has been read the cursor can not go to the starting position to read again we need to move the cursor using seek method**

**Tell() & seek() method:**

**Tell():** returns the current position of the cursor in the file

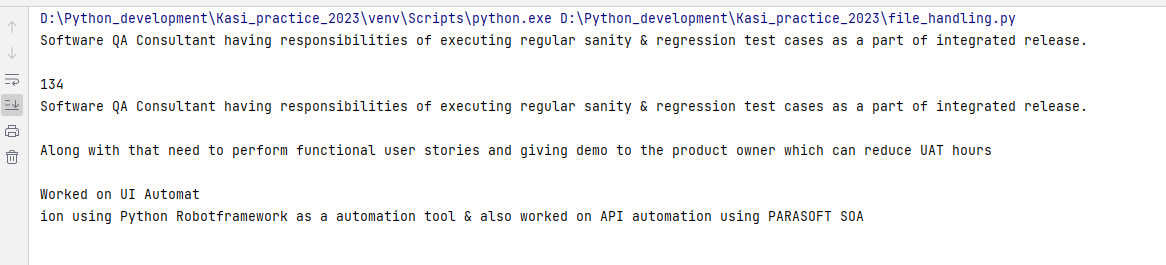
**Seek():** navigates to the specified position

with open(r"D:\data1.txt", 'r') as file1:  
 print(file1.read(20))  
 print(file1.tell())  
 print(file1.read()) *#reads remainging lines from 20 pos as cursor at 20th pos* print(file1.tell())  
 print(file1.read()) *#it cant read from starting by it self as cursor will not go to first pos* file1.seek(0) *#it will go to 0th position as specified* print(file1.read())

****

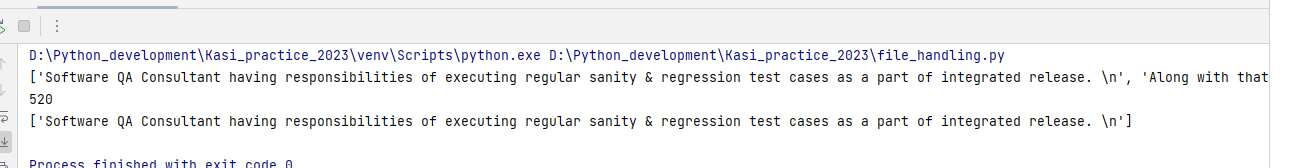
**Readline():** reads a single line from the file

with open(r"D:\data1.txt", 'r') as file1:  
 print(file1.readline()) *#reads the first line* print(file1.tell())  
 file1.seek(0)  
 print(file1.readline()) *#reads the first line* print(file1.readline()) *#reads the second line* print(file1.readline(20)) *#reads the first 20 chars from the third line* print(file1.readline()) *#prints the continuation char's from the third line bcoz cursor will be at 20th pos of third line*

****

**Readlines():** returns entire text in the form of list separating each line as an element

with open('D:\data1.txt', 'r') as file1:  
 print(file1.readlines())  
 print(file1.tell())  
 file1.seek(0)  
 print(file1.readlines(100)) *#prints first 100 chars in the form of list*



Examples on normal file handling:

"""wap to count the no of lines present in the file"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file2:  
 count = 0  
 for line in file2:  
 if line.strip():  
 count += 1  
 print(count)  
  
"""wap to print line and line no from the file"""  
"""normal method"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file3:  
 count = 0  
 for line in file3:  
 print(count, line)  
 count += 1  
"""using enumerate"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file4:  
 for line\_no, line in enumerate(file4):  
 print(line\_no, line)  
  
"""wap to count the no of words in a given file"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file4:  
 count = 0  
 for line in file4:  
 if line.strip():  
 words = line.split()  
 count += len(words)  
 print(count)  
  
"""wap to print the file from the last oif the file"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file4:  
 for line in reversed(list(file4)):  
 print(line)  
  
"""wap to count the no of words that are starting with vowels"""  
with open(r"C:\Kasi\_python\_moolya\sample\_2.txt", 'r') as file4:  
 count\_words\_vow = 0  
 for line in file4:  
 if line.strip():  
 words = line.split()  
 for word in words:  
 if word[0].lower() in 'aeiou':  
 count += 1  
 print(count)

“””writing into normal files”””  
  
with open(r"temp23.txt", 'w') as w\_file:  
 w\_file.write('This is Mohana kasi')  
  
with open(r"temp23.txt", 'a') as w\_file2:  
 w\_file2.writelines(['\nits a python world', '\ncontains all objects', '\nhaving intrepretor'])

“””CSV file handling”””

import csv  
  
"reading in csv"  
with open(r"sample\_data.csv", 'r') as csv\_file1:  
 r\_obj = csv.reader(csv\_file1)  
 r\_obj.\_\_next\_\_() *#skips the header* for row in r\_obj:  
 print(row)  
  
with open(r"sample\_data.csv", 'r') as csv\_file1:  
 r\_obj = csv.DictReader(csv\_file1)  
 for row in r\_obj:  
 print(row)  
  
  
"""write a program to read all the employees in employee table"""  
with open(r"sample\_data.csv", 'r') as csv\_file2:  
 r\_obj = csv.DictReader(csv\_file2)  
 for row in r\_obj:  
 print(row['name'])  
  
"""write program to print only the names with salaries > 7000"""  
with open(r'employees.csv', 'r') as csv\_file1:  
 r\_obj = csv.DictReader(csv\_file1)  
 for row in r\_obj:  
 if int(row['sal']) > 7000:  
 print(row['FIRST\_NAME'])  
  
"""wap to group male and female names""" "(using default dict)"  
from collections import defaultdict  
dict\_ = defaultdict(list)  
with open(r"employees.csv", 'r') as csv\_file1:  
 r\_obj = csv.DictReader(csv\_file1)  
 for row in r\_obj:  
 if row['Gender'] == 'Male':  
 dict\_['male'] += [row['FIRST\_NAME']]  
 elif row['Gender'] == 'FeMale':  
 dict\_['female'] += [row['FIRST\_NAME']]  
  
print(dict\_)

"""writing into csv\_files"""  
"using writer()"  
with open(r"practice\_write.csv", 'w') as csv\_file1:  
 w\_obj = csv.writer(csv\_file1)  
 w\_obj.writerow(['name', 'age', 'loc']) *#writer row takes only one iterable as input* w\_obj.writerows([('kasi', 26, 'guntur'), ('Mohana', 28, 'guntur')]) *#write rows will a collection of iterables*"using Dictwriter()"  
with open('practice\_write.csv', 'a') as csv\_file2:  
 w\_obj = csv.DictWriter(csv\_file2, ['empid', 'loc', 'desig'])  
 w\_obj.writeheader()  
 w\_obj.writerow({'empid':1234, 'loc':'guntur', 'desig':'sdet'})  
 w\_obj.writerows([{'empid':1986, 'loc':'bangalore', 'desig':'qa1'}, {'empid':10990, 'loc':'guntur', 'desig':'DEV'}])

“””JSON & Pickle”””

*"""json"""*import json  
  
*# Python dictionary*data = {  
 'name': 'John Doe',  
 'age': 30,  
 'city': 'New York'  
}  
  
*# Serialize to JSON (python to json)*json\_data = json.dumps(data)  
print(json\_data)  
  
*#save to a file*with open(r"data.json", 'w') as js\_file:  
 json.dump(data, js\_file)  
  
*#deserilization (json to python)*with open(r"data.json", 'r') as js\_file1:  
 ds\_data = json.load(js\_file1)  
 print(type(ds\_data))  
 for key in ds\_data:  
 print(key)  
  
  
  
"""pickele"""  
import pickle  
data = {  
 'name': 'John Doe',  
 'age': 30,  
 'city': 'New York'  
}  
  
*#serilizing python object to pickle*pk\_data = pickle.dumps(data)  
print(pk\_data)  
  
*#deserialize pickle data to python object*print(pickle.loads(pk\_data))  
  
*#serializing python data to pickle into a file*with open(r"pk\_data1.pkl", 'wb') as pk\_file1:  
 pickle.dump(data, pk\_file1)  
  
*#deserializing the pickle data from pkl file*with open(r"pk\_data1.pkl", 'rb') as pk\_data1:  
 print(pickle.load(pk\_data1))

**Decorators:**

[Decorators](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of a function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it. But before diving deep into decorators let us understand some concepts that will come in handy in learning the decorators.

### First Class Objects In Python, functions are [first class objects](https://www.geeksforgeeks.org/first-class-functions-python/) which means that functions in Python can be used or passed as arguments. Properties of first class functions:

* A function is an instance of the Object type.
* You can store the function in a variable.
* You can pass the function as a parameter to another function.
* You can return the function from a function.
* You can store them in data structures such as hash tables, lists, …

Example1: Treating the function as objects

# Python program to illustrate functions

# can be treated as objects

def temp1(string):  
 return string.split()  
  
print(temp1('Kasi from guntur'))  
f1 = temp1  
print(f1('Mohana kasi'))

#Example2: Passing the function as an argument

def rev\_string(string\_):  
 return string\_[::-1]  
  
def string\_split(string\_):  
 return string\_.split()  
  
def greet(func):  
 print(f"{func.\_\_name\_\_} iam called by {greet.\_\_name\_\_}")  
 print(func('This is sample data for decorators'))  
  
greet(rev\_string)  
greet(string\_split)

]

#Example3: Returning functions from another function.

# Python program to illustrate functions

# Functions can return another function

def create\_adder(x):

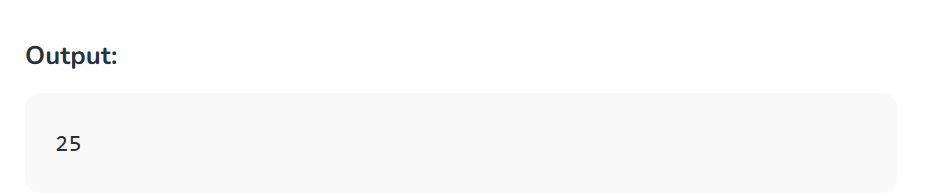
def adder(y):

return x+y

return adder

add\_15 = create\_adder(15)

print(add\_15(10))



def greeting(greet\_msg):  
 def msg\_convy(actual\_msg):  
 return greet\_msg+actual\_msg  
  
 return msg\_convy  
temp1 = greeting('Welcome to moolya')  
print(temp1('Mohana kasi'))

**Regular expressions:**

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

## RegEx Module:

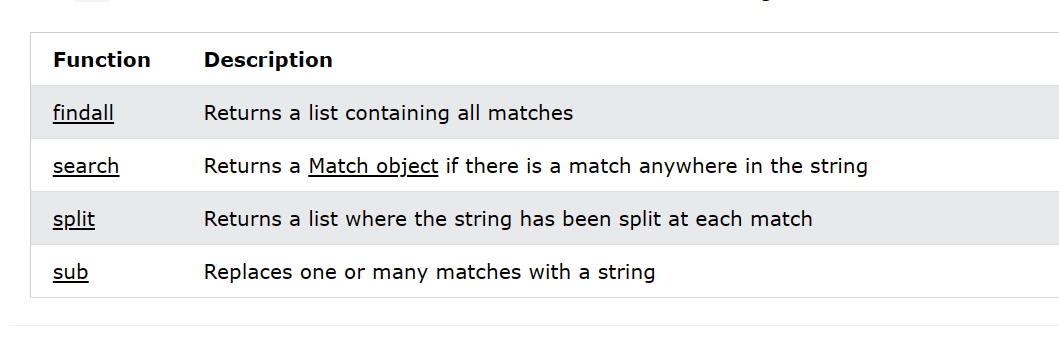
Python has a built-in package called re, which can be used to work with Regular Expressions.

**Import the re module:**

Import re

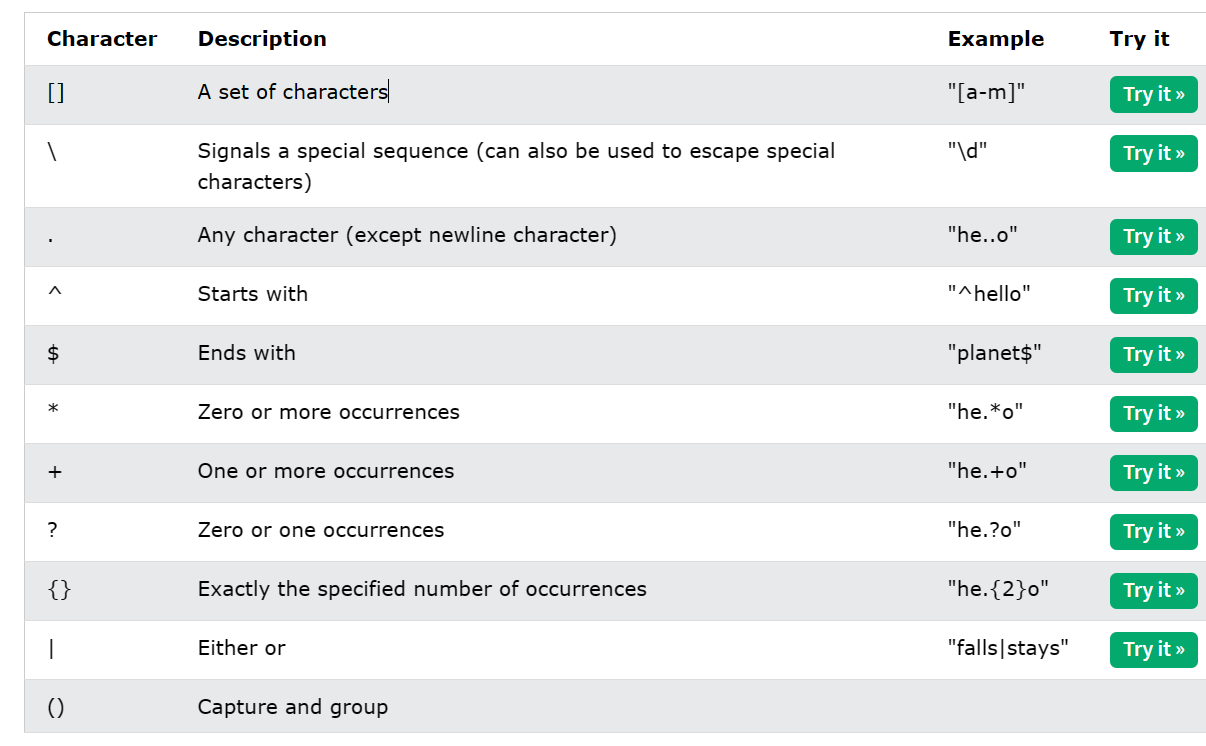
## RegEx Functions:

The re module offers a set of functions that allows us to search a string for a match:



## Metacharacters:

Metacharacters are characters with a special meaning.



[] 🡪 set of characters

\ 🡪 special sequence

. Any character (except new line)

\* 🡪 zero or more

+ 🡪 one or more

? 🡪 zero or one

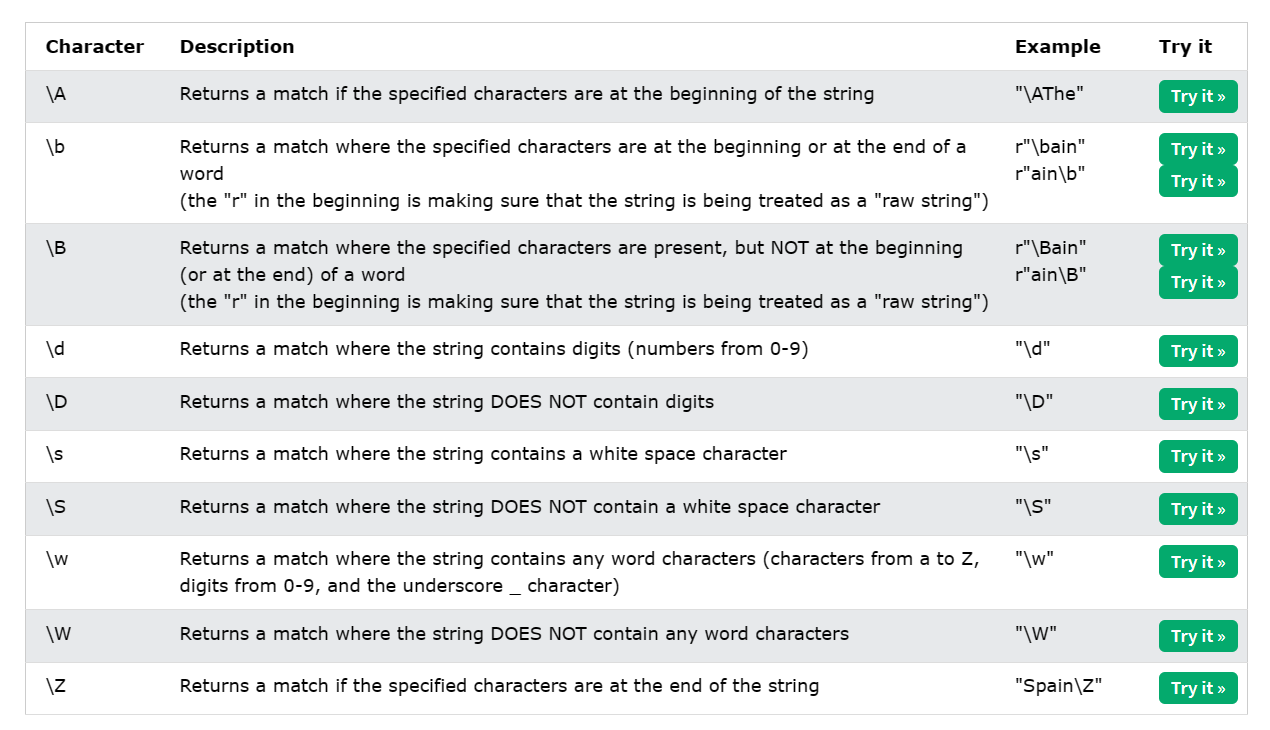
{} 🡪 exactly specified no of occurances

| 🡪 either

() 🡪 capture and group

**Special sequences:**

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:



\A 🡪 matches the pattern at the beginning of the string

\b 🡪 matches the pattern at the beginning or end of the string

\B 🡪 matches at anywhere in the string

\d matches the digits from 0-9

\D 🡪 matches all other than digits

\s matches white spaces

\S matches all other than white spaces

\w matches all word characters (a-z, A-Z, 0-9)

\W matches other than word characters

\Z matches characters at the end of the string

Findall:

*"""findall  
The findall() function returns a list containing all matches.  
"""*"""Example"""  
import re  
  
txt = "The rain in Spain"  
x = re.findall("ai", txt)  
print(x)  
  
"sets"  
print(re.findall('[a-m]', "The rain in Spain"))  
print(re.findall('^Th', 'The rain in Spain'))  
print(re.findall('he..o', 'hello planet'))  
print(re.findall('i$', 'This is Mohana Kasi'))  
print(re.findall('kasi$', 'This is Mohana kasi'))  
print(re.findall('he.\*o', 'hello hai'))  
print(re.findall('Moh.+a', 'Hello Mohana'))  
print(re.findall('Moh.?a', 'Hello Mohana')) *#gives only one occurance*print(re.findall(r'\d{2}', 'THis is kasi 12 and the 45 and not 987'))  
*#Return an empty list if no match was found:*print(re.findall('india', 'iam from guntur'))  
print(re.findall(r'\A[aeiou]', 'is this Mohana Kasi'))  
print(re.findall(r'\A[aeiou]', 'This this Mohana Kasi'))  
print(re.findall(r'\A\d', '19 sector Hello kasi'))  
  
pattern = re.compile(r'\Ath')  
print(pattern.findall('this is not the correct of way of doing life prospective'))  
  
pattern = re.compile(r"\A\w")  
print(pattern.findall('\_1 sector kasi'))  
print(pattern.findall('99 sector'))  
print(pattern.findall('Aeronatical sector'))  
print(pattern.findall('$hello hai python'))  
  
r"""\b finds the match at the begining or ending"""  
pattern2 = re.compile(r'\bth')  
print(pattern2.findall('this is kasi theoritical world of thomas'))  
pattern3 = re.compile(r'an\b')  
print(pattern3.findall('its an energitic performance by the shami as a moolyan i can enjoy it'))  
pattern4 = re.compile(r"\bat|at\b")  
print(pattern4.findall('throat not appropriat at the time of attnendance swat'))  
  
r"""\B find the batch anywhere but not starting or ending of the string"""  
pattern = re.compile(r'\Bin')  
print(pattern.findall('all becmin ai driven upcoming generating proping moduling and integration toing'))  
pattern2 = re.compile(r'\B[0-9]')  
print(pattern2.findall('this tro23p conv45terminalogy $5po'))  
pattern2 = re.compile(r'\B')  
  
  
r"""\d finds the all matched digits"""  
print(re.findall(r'\d', 'its a great 20fullof23 fullof20and 23terminologu'))  
print(re.findall(r'\D', 'its a great 20fullof23 fullof20and 23terminologu'))  
print(re.findall(r'\d$', 'oh my god full23 is 8th wonde5'))  
print(re.findall(r'^\d', '19th sector banned for34 fine on 18'))  
  
print(re.findall(r'\w', 'hello $rambo$\_ is 59 is fine?'))  
print(re.findall(r'\W', 'hello $rambo$\_ is 59 is fine?'))  
  
found\_match = re.findall(r'\s', 'today iam not cominng')  
print(found\_match)  
print(len(found\_match))  
  
print(re.findall(r'\S', 'Iam the')) *#mathces other than white spaces*pattern = re.compile(r'on\Z')  
print(pattern.findall('Iam an ocean on afternoon temperory person'))  
  
print(re.findall(r"\B[+]\d{2} \d{3}-\d{3}-\d{4}", 'this is kasi +91 888-621-3059 number'))  
  
print(re.findall(r'\b\w{2}\b','560001 is a 09 country '))

Practice programs:

import re  
"matchina a series or a particular characters"  
print(re.findall(r"[0-9]", 'This is kasi 19th sector 0 division line'))  
"\d equivalent to [0-9]"  
print(re.findall(r'\d','This is kasi 19th sector 0 division line'))  
print(re.findall(r'[a-z]', 'is This python3$ language1'))  
print(re.findall(r'[A-Z]', 'hey Hello HAI'))  
print(re.findall(r"\w", 'This is MOHANA \_99 cordor #$()')) *#matches alphabests, numbers, \_*print(re.findall(r"[a-zA-z0-5]", 'Its not the way of 8886213059'))  
print(re.findall(r"\W", 'THis is $kasi 99)()')) *#matches only special characters*print(re.findall(r"\A[aeiou]", 'is this way to bangalore'))  
print(re.findall(r"[etio]", 'india is a nation of prime'))  
print(re.findall(r"888", 'this is kasi 8886213059 number'))  
"^ inside a starting of the character set"  
"nothing but negation"  
print(re.findall(r'[^a-zA-Z0-9]', 'this is $way of ~09')) *#matches only special characters*print(re.findall(r"[^0-9]", 'this 99sector@')) *#matches othethan numbers*print(re.findall(r"&{2}", 'Hello & Mohana &&'))  
  
"[]+"  
"gives you entire sub string matched in between the range given"  
print(re.findall(r"[0-9]+", 'hey hai 67 sector'))  
print(re.findall(r"[a-z]+", 'Hey hello man23'))  
  
  
"word boundary"  
"transition between non word to word at front & word to non word transition at back side of the substring\n" \  
"then it is called as word boundary"  
  
print(re.findall(r"\b\d{6}\b", 'hey man 522003 is your pincode'))  
print(re.findall(r"\b\d{2}\b", 'hey it is +23 degree right'))  
  
"finding the letters startwith k"  
print(re.findall(r"\bk\w\*\b", 'this is kasi kingdom of kingyard'))  
  
"sum of individual numbers in a string using regular expresssions"  
total = 0  
string\_ = 'hey 23 and sector 29 is full'  
list\_ = re.findall(r'\d', string\_)  
print(total := sum([int(item) for item in list\_]))  
  
  
"sum of multiple numbers in a string using regular expressions"  
total = 0  
found\_nums = re.findall(r"\d+", 'this is 34th street 99sector')  
print(total:=sum([int(item) for item in found\_nums]))  
  
"to find total spaces in a string using regular expressions"  
total = []  
spaces = re.findall(r"\s", 'this is kasi hectic not yours')  
print(total:=len(spaces))  
  
"to count the number of occurences of each non special characters (only alphabets)"  
"usinng default dict"  
from collections import defaultdict  
dict\_ = defaultdict(int)  
found\_chars = re.findall(r"[a-zA-Z]", 'this is is sample23 string$%')  
for char in found\_chars:  
 dict\_[char] += 1  
  
print(dict\_)  
  
"count the upper case characters in string using regular expressions"  
found\_ups = re.findall(r"[A-Z]", 'This is NOT proper LOGIC #$7$')  
print(len(found\_ups))  
  
*# "finding valid phone numbers from a list"*l = ['123-4567-890', '888-621-3059']  
valid\_nums = re.findall(r"\d{3}-\d{3}-\d{4}", str(l))  
print(valid\_nums)  
  
  
"finding phone numbers startiwith 8 or 9 followed by 00 ex(800 or 900)"  
l\_ = ['123-4567-890', '800-621-3059', '901-789-4568','900-745-3214']  
match\_des = re.findall(r"[89]00-\d{3}-\d{4}", str(l\_))  
print(match\_des)  
  
l\_ = ['888-621-3059', '949-129-4540', '989-840-3459']  
match = re.findall(r"\d{3}-\d{3}-\d{2}59\b", str(l\_))  
print(match)  
  
  
  
"finding vowel words in a list"  
l = ['iam', 'king', 'user']  
match = []  
for word in l:  
 found = re.findall(r"[aeiou]", word)  
 if found:  
 match += word  
print(match)

**Exception handling:**

Errors and exceptions can lead to program failure or unexpected behavior, and Python comes with a robust set of tools to improve the stability of the code.

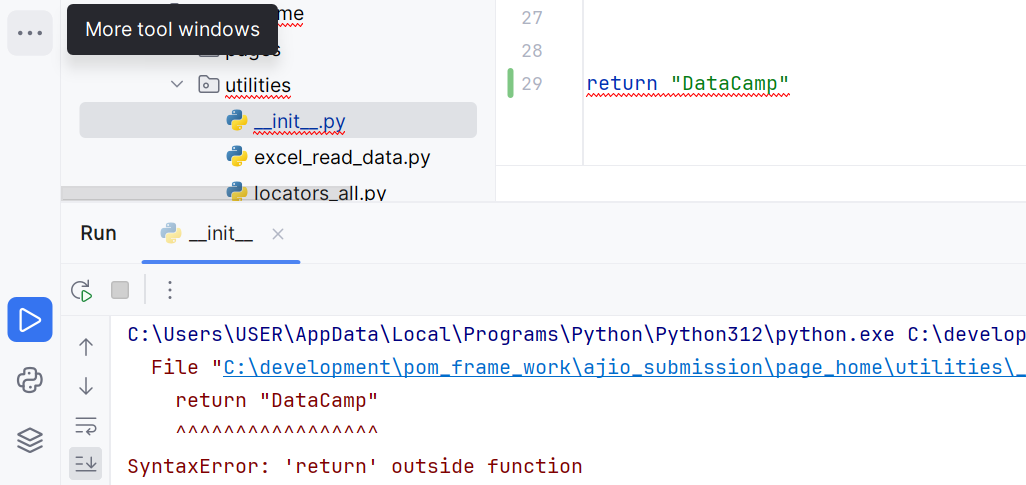
An error is an issue in a program that prevents the program from completing its task.

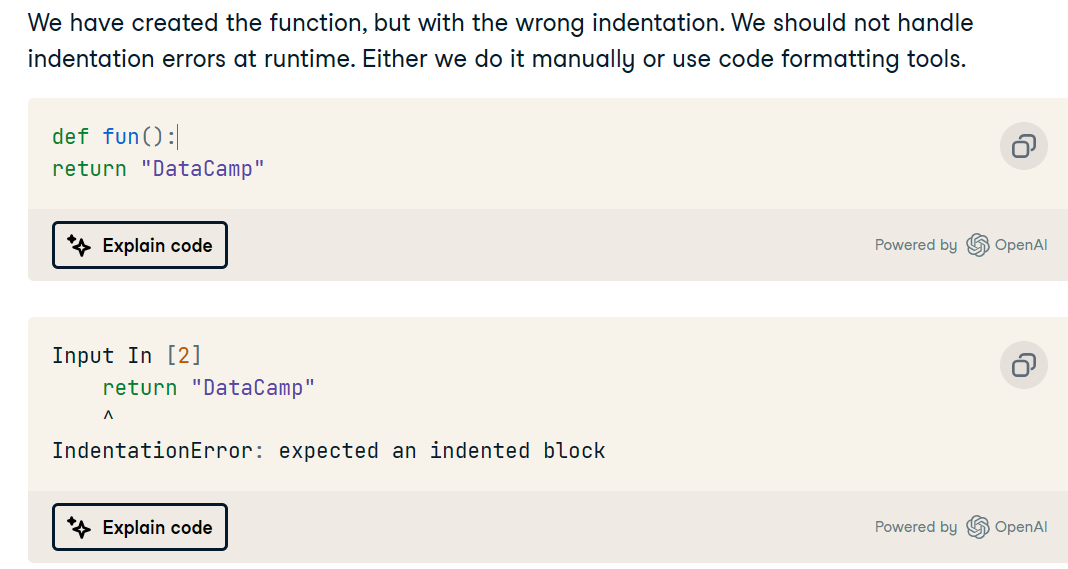
In comparison, an exception is a condition that interrupts the normal flow of the program. Both errors and exceptions are a type of runtime error, which means they occur during the execution of a program.

In simple words, the error is a critical issue that a normal application should not catch, while an exception is a condition that a program should catch.

## Errors in Python

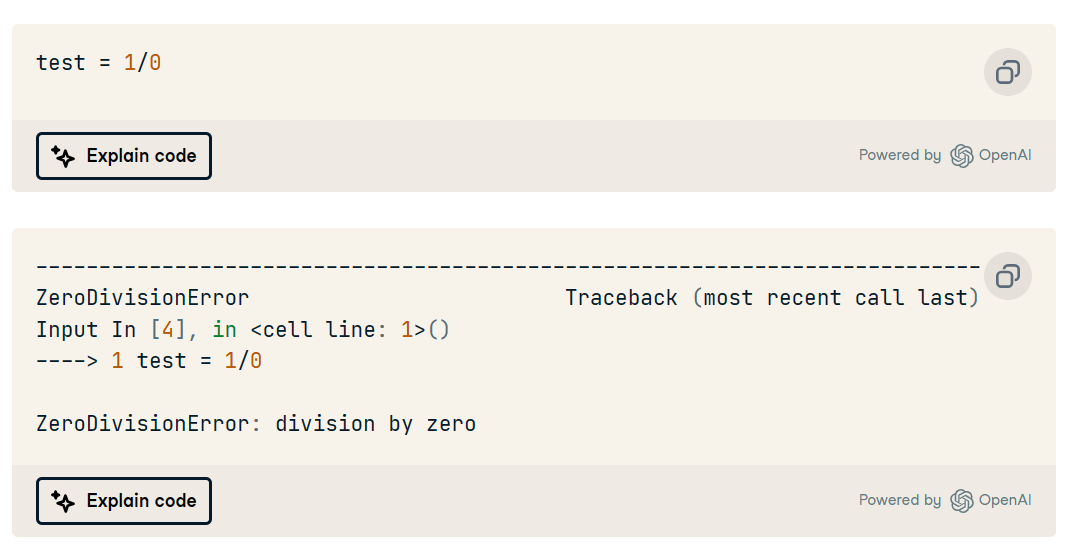
Here is an example of a syntax error where a return outside the function means nothing. We should not handle errors in a program. Instead, we must create a function that returns the string.

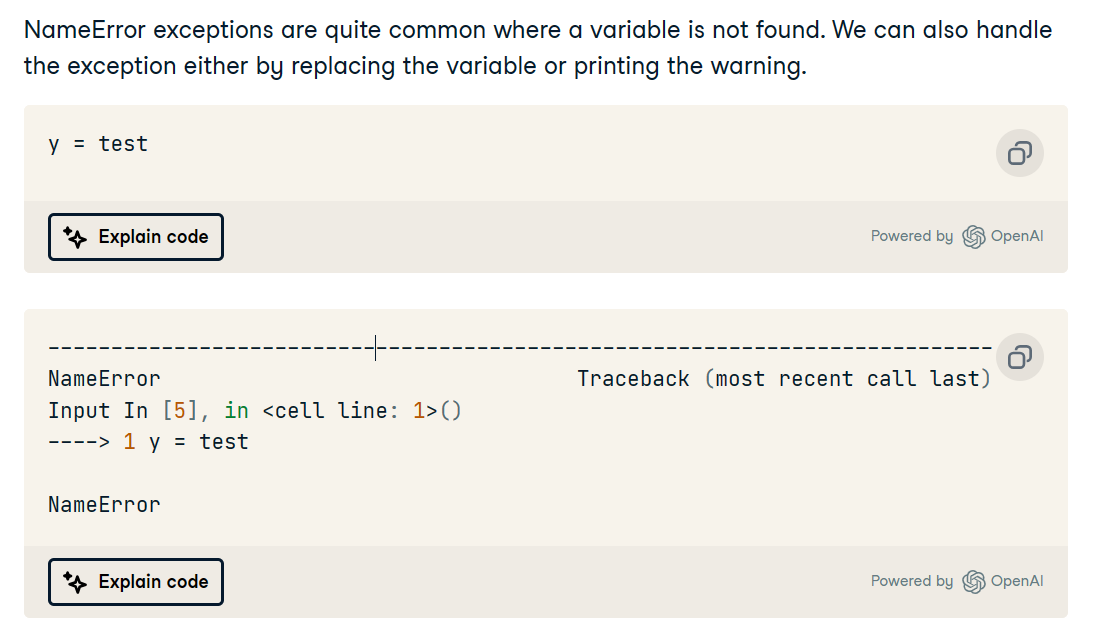




## Exceptions in Python

We have encountered a ZeroDivisionError (Exception). We can handle it at runtime by using `try` and `except` blocks.





## Built-in Python Exceptions:

Here is the list of default Python exceptions with descriptions:

1. **AssertionError:**raised when the assert statement fails.
2. **EOFError:** raised when the input() function meets the end-of-file condition.
3. **AttributeError:** raised when the attribute assignment or reference fails.
4. **TabError:** raised when the indentations consist of inconsistent tabs or spaces.
5. **ImportError:** raised when importing the module fails.
6. **IndexError:** occurs when the index of a sequence is out of range
7. **KeyboardInterrupt:**raised when the user inputs interrupt keys (Ctrl + C or Delete).
8. **RuntimeError:**occurs when an error does not fall into any category.
9. **NameError:**raised when a variable is not found in the local or global scope.
10. **MemoryError:**raised when programs run out of memory.
11. **ValueError:**occurs when the operation or function receives an argument with the right type but the wrong value.
12. **ZeroDivisionError:** raised when you divide a value or variable with zero.
13. **SyntaxError:**raised by the parser when the Python syntax is wrong.
14. **IndentationError:**occurs when there is a wrong indentation.
15. **SystemError:**raised when the interpreter detects an internal error.

## Exception Handling with try, except, else, and finally:

Encapsulation:

Public – anywhere

Protected – inside class, outside class but you should use \_before the variable

Private – inside class but not outside class