PYTHON

Python Timeline/History

- Python was conceived in the late 1980s.
  - Guido van Rossum, Benevolent Dictator For Life
  - Rossum is Dutch, born in Netherlands, Christmas brea bored, big fan of Monty python's Flying Circus
  - Descendant of ABC, he wrote glob() func in UNIX
  - M.D. @ U of Amsterdam, worked for CWI, NIST, CNRI, Google
  - Also, helped develop the ABC programming language
- In 1991 python 0.9.0 was published and reached the masses through alt.sources
- In January of 1994 python 1.0 was released
  - Functional programming tools like lambda, map, filter, and reduce
  - comp.lang.python formed, greatly increasing python's userbase

Python Timeline/History

- In 1995, python 1.2 was released.
- By version 1.4 python had several new features
  - Keyword arguments (similar to those of common lisp)
  - Built-in support for complex numbers
  - Basic form of data-hiding through name mangling (easily bypassed however)
- Computer Programming for Everybody (CP4E) initiative
  - Make programming accessible to more people, with basic “literacy” similar to those required for English and math skills for some jobs.
  - Project was funded by DARPA
  - CP4E was inactive as of 2007, not so much a concern to get employees programming “literate”

Python Timeline/History

- In 2000, Python 2.0 was released.
  - Introduced list comprehensions similar to Haskells
  - Introduced garbage collection
- In 2001, Python 2.2 was released.
  - Included unification of types and classes into one hierarchy, making pythons object model purely Object-oriented
  - Generators were added(function-like iterator behavior)
- Python 3.11. 0, was released on 24 October 2022
- Standards
  - http://www.python.org/dev/peps/pep-0008/

PYTHON

- Python is a high-level, interpreted, interactive and object oriented-scripting language, designed to be highly readable, commonly uses English keywords.

  - Python is Interpreted: This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.
  - Python is Interactive: This means that you can actually sit at a Python prompt and interact it directly to write your programs.
  - Python is Object-Oriented: This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
  - Python is Beginner's Language: Python is a great language for the beginner programmers and supports the development of a wide range of applications

FEATURES OF PYTHON

- Python is object-oriented
- Structure supports such concepts as polymorphism, operation overloading, and multiple inheritance
- It's free (open source)
- Downloading and installing Python is free and easy
- Source code is easily accessible
- Free doesn't mean unsupported! Online Python community is huge
- It's portable
- Python runs virtually every major platform used today
- As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform

- It's powerful
- Dynamic typing
- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, SciPy)
- Automatic memory management
- It's mixable
- Python can be linked to components written in other languages easily
    - Linking to fast, compiled code is useful to computationally intensive problems
    - Python is good for code steering and for merging multiple programs in otherwise conflicting languages

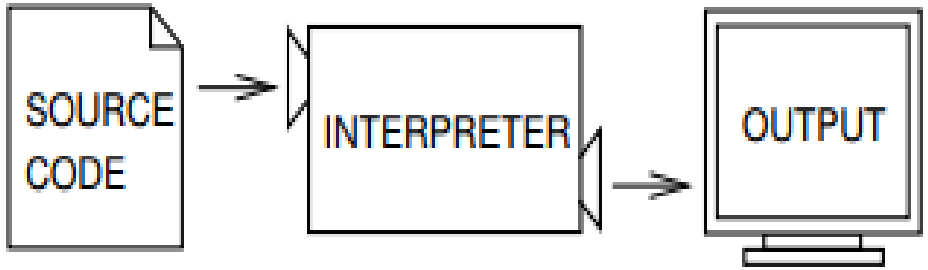
- It's easy to use
- Rapid turnaround: no intermediate compile and link steps as in C or C++
- Python programs are compiled automatically to an intermediate form called *bytecode*, which the interpreter then reads
- This gives Python the development speed of an interpreter without the performance loss inherent in purely interpreted languages

- It's easy to learn
- Structure and syntax are pretty intuitive and easy to grasp

- Scalable:
- Python provides a better structure and support for large programs than shell scripting.

PYTHON INTERPRETER AND INTERACTIVE MODE

An interpreter processes the program a little at a time, alternately reading lines and performing computations.



The interpreter reads Python programs and commands, and executes them. The Python programs are compiled automatically before being scanned into the interpreter. The fact that this process is hidden makes Python faster than a pure interpreter.

PYTHON INTERPRETER AND INTERACTIVE MODE

Python is considered an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter:

Interactive Mode

In interactive mode, we can type Python programs and the interpreter displays the result: >>> 1 + 1

>>> 2

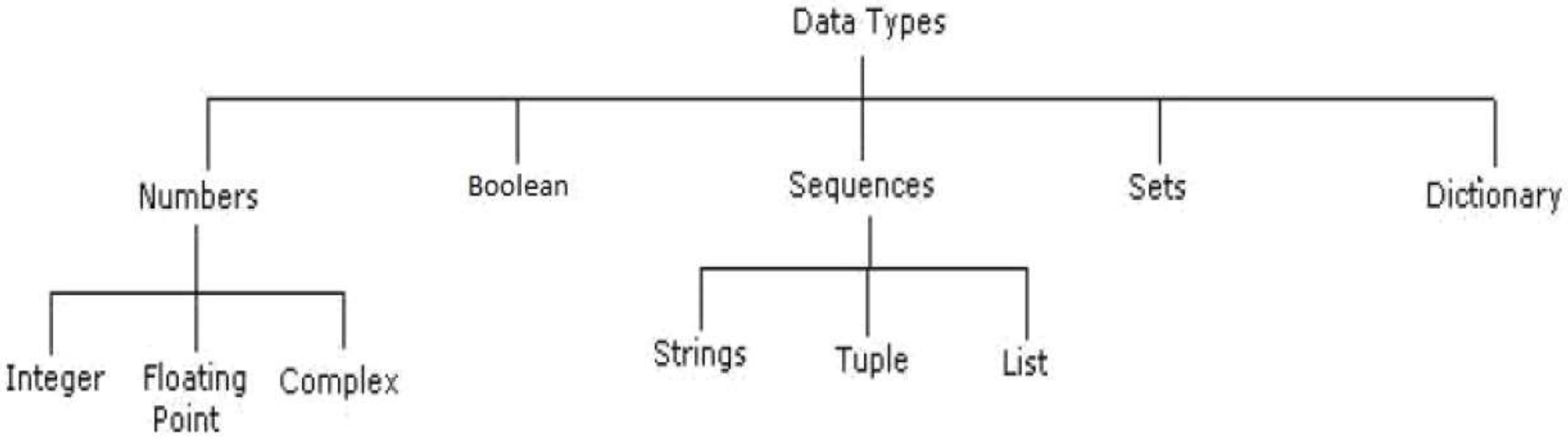
Script Mode

Storing the code in a file and use the interpreter to execute the contents of the file, which is called a script.

By convention, Python scripts have names that end with .py.

Data type

- Every value in Python has a data type.
- It is a set of values, and the allowable operations on those values.



Numbers

- Number data type stores **Numerical Values.**
- This data type is immutable [i.e. values/items cannot be changed].
- Python supports following number types
    - 1. integers
    - 2. floating point numbers
    - 3. Complex numbers.

Integers

- Integers are the whole numbers consisting of +ve or –ve sign.
- **Example:**
- a=10
- a=eval(input(“enter a value”))
- a=int(input(“enter a value”))

Float

- it has decimal part and fractional part.
- **Example**:
- a=3.15
- a=eval(input(“enter a value”))
- a=float(input(“enter a value”))

Complex

- It is a combination of real and imaginary part.
- **Example:**
- 2+5j
- a+bi

Number conversions

- Python converts numbers internally in an expression containing mixed types to a common type for evaluation.
- But sometimes, we'll need to convert a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

  - Type int(x) to convert x to a plain integer.
  - Type long(x) to convert x to a long integer.
  - Type float(x) to convert x to a floating-point number.
  - Type complex(x) to convert x to a complex number with real part x and imaginary part zero.
  - Type complex(x, y) to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Sequence

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.
- There are three types of sequence data type available in Python, they are
    - **1. Strings**
    - **2. Lists**
    - **3. Tuples**

Strings

- String is defined as a continues set of characters represented in quotation marks (either single quotes (‘) or double quotes (“).
- An individual character in a string is accessed using a subscript (index).
- The subscript should always be an integer (positive or negative).
- A subscript starts from **0 to n-1**.
- Strings are **immutable** i.e. the contents of the string **cannot** be changed after it is created.
- Python will get the input at run time by default as a string.
- Python does not support character data type. A string of size 1 can be treated as characters.
  - **1.** single quotes (' ')
  - **2.** double quotes (" ")
  - **3.** triple quotes(“”” “””””)

Immutability:

- Python strings are immutable as they cannot be changed after they are created.
- Therefore [ ] operator cannot be used on the left side of an assignment.

operations	Example	output
element assignment	a="PYTHON" a[0]='x'	TypeError: 'str' object does not support element assignment
element deletion	a= PYTHON del a[0]	TypeError: 'str' object doesn't support element deletion
delete a string	a= PYTHON del a print(a)	NameError: name 'a' is not defined

Operations on string:

- 1. Indexing
- 2. Slicing
- 3. Concatenation
- 4. Repetitions
- 5. Membership
- Example A="HELLO"

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

Indexing

- **Positive indexing** - helps in accessing the string from the **beginning**
- >>>a=”HELLO”
- >>>print(a[0])
- >>>H
- Negative **indexing** - helps in accessing the string from the **end**.
- >>>print(a[-1])
- >>>O

Slicing

- Subsets of strings can be taken using the slice operator ([ ] and [:])
- The **[n : m]** operator extracts substring from the strings.
  - Whereas n is included and m is excluded in the substring
- >>> a=”HELLO”
- >>> print(a[0:4]) → HELL
- >>> print(a[:3]) → HEL
- >>> print(a[0:]) → HELLO

Concatenation

- The + operator joins the text on both sides of the operator.

  - a=”save”
  - b=”earth”
  - print(a+b)
  - saveearth

Repetitions

- The * operator repeats the string on the left hand side times the value on right hand side.
  - *>>> str = 'Hello World!'*
  - *>>> print (str * 2) # Prints string two times*
  - *Hello World!Hello World!*

Membership

- Using membership operators to **check a particular character** is in string or not.
- Returns **true if present**
- >>> s="good morning"
- >>>"m" in s
- True
- >>> "a" not in s
- True

Lists

- List is an ordered sequence of items. Values in the list are called elements / items.
- It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- Items in the lists can be of **different data types**.

Operations on list:

- 1. Indexing
- 2. Slicing
- 3. Concatenation
- 4. Repetitions
- 5. Updating
- 6. Membership
- 7. Comparison

Lists Creation

- Listname=[item1,item2,……]
- Eg: a=[1,2,5,8]
- List can comprise elements of different data types
- Eg: b=[1,2.3,”hello”,3] → comprises integers, string, floating point values
- List is **mutable** which means contents of list can be altered.
- a[2]=10
- print (a) → [1,2,10,8]

operations	examples	description

create a list	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> print(a)
	[2, 3, 4, 5, 6, 7, 8, 9, 10]
		in this way we can create a list at compile time

Indexing	>>> print(a[0])
	2
	>>> print(a[8])
	10
	>>> print(a[-1])
	10
		Accessing the item in the position 0
		Accessing the item in the position 8
		Accessing a last element using negative indexing.

Slicing	>>> print(a[0:3])
	[2, 3, 4]
	>>> print(a[0:])
	[2, 3, 4, 5, 6, 7, 8, 9, 10]
		Printing a part of the list.

Concatenation	>>>b=[20,30] >>> print(a+b) [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30]	Adding and printing the items of two lists.
Repetition	>>> print(b*3) [20, 30, 20, 30, 20, 30]	Create a multiple copies of the same list.

Updating	>>> print(a[2])
	4
	>>> a[2]=100
	>>> print(a)
	[2, 3, 100, 5, 6, 7, 8, 9, 10]
		Updating the list using index value.

Membership	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in list. Otherwise returns false.

Comparison	>>> a=[2,3,4,5,6,7,8,9,10]
	>>>b=[2,3,4]
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

List slices

- List slicing is an operation that extracts a subset of elements from an list and packages them as another list.

Syntax:

Listname[start:stop]

Listname[start:stop:steps]

- default start value is 0
- default stop value is n-1
- [:] this will print the entire list
- [2:2] this will create a empty slice

slices	example	description

a[0:3]	>>> a=[9,8,7,6,5,4]
	>>> a[0:3]
	[9, 8, 7]
		Printing a part of a list from 0 to 2.

a[:4]	>>> a[:4] [9, 8, 7, 6]	Default start value is 0. so prints from 0 to 3
a[1:]	>>> a[1:] [8, 7, 6, 5, 4]	default stop value will be n-1. so prints from 1 to 5
a[:]	>>> a[:] [9, 8, 7, 6, 5, 4]	Prints the entire list.
a[2:2]	>>> a[2:2] [ ]	print an empty slice
a[0:6:2]	>>> a[0:6:2] [9, 7, 5]	Slicing list values with step size 2.
a[::-1]	>>> a[::-1] [4, 5, 6, 7, 8, 9]	Returns reverse of given list values

List methods

- Methods used in lists are used to manipulate the data quickly.
- These methods work only on lists.
- They do not work on the other sequence types that are not mutable, that is, the values they contain cannot be changed, added, or deleted.

syntax:

list name.method name(element/index/list)

	syntax	example	description

1	a.append(element)	>>> a=[1,2,3,4,5]
		>>> a.append(6)
		>>> print(a)
		[1, 2, 3, 4, 5, 6]
			Add an element to the end of the list

2	a.insert(index,element)	>>> a.insert(0,0)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6]
			Insert an item at the defined index

3	a.extend(b)	>>> b=[7,8,9]
		>>> a.extend(b)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8,9]
			Add all elements of a list to the another list

4	a.index(element)	>>> a.index(8) 8	Returns the index of the first matched item

5	a.sort()	>>> a.sort()
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8]
			Sort items in a list in ascending order

6	a.reverse()	>>> a.reverse()
		>>> print(a)
		[8, 7, 6, 5, 4, 3, 2, 1, 0]
			Reverse the order of items in the list

7	a.pop()	>>> a.pop() 0	Removes and returns an element at the last element
8	a.pop(index)	>>> a.pop(0) 8	Remove the particular element and return it.

9	a.remove(element)	>>> a.remove(1)
		>>> print(a)
		[7, 6, 5, 4, 3, 2]
			Removes an item from the list

10	a.count(element)	>>> a.count(6) 1	Returns the count of number of items passed as an argument

11	a.copy()	>>> b=a.copy()
		>>> print(b)
		[7, 6, 5, 4, 3, 2]
			Returns a shallow copy of the list

12	len(list)	>>> len(a) 6	return the length of the length
13	min(list)	>>> min(a) 2	return the minimum element in a list
14	max(list)	>>> max(a) 7	return the maximum element in a list.

15	a.clear()	>>> a.clear()
		>>> print(a)
		[ ]
			Removes all items from the list.

Mutability

- Lists are mutable. (can be changed)
- Mutability is the ability for certain types of data to be changed without entirely recreating it.
- An item can be changed in a list by accessing it directly as part of the assignment statement.
- Using the indexing operator (square brackets[ ]) on the left side of an assignment, one of the list items can be updated.

Example	description

>>> a=[1,2,3,4,5]
>>> a[0]=100
>>> print(a)
[100, 2, 3, 4, 5]
	changing single element

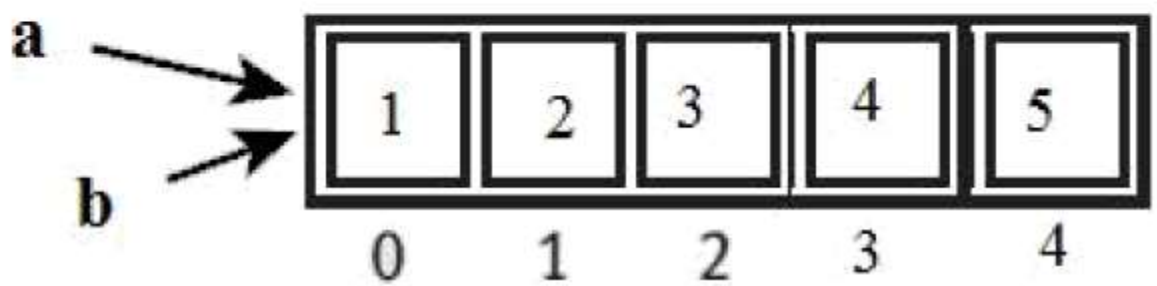
>>> a=[1,2,3,4,5]
>>> a[0:3]=[100,100,100]
>>> print(a)
[100, 100, 100, 4, 5]
	changing multiple element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[ ]
>>> print(a)
[4, 5]
	The elements from a list can also be removed by assigning the empty list to them.

Aliasing(copying)

- Creating a copy of a list is called aliasing. When you create a copy both list will be having same memory location. **Changes in one list will affect another list.**
- **Alaising refers to having different names for same list values**
- In this a single list object is created and modified using the subscript operator.
- When the first element of the list named **“a”** is replaced, the first element of the list named “**b”** is also replaced.
- This type of change is what is known as a **side effect**. This happens because after the assignment **b=a**, the variables **a** and **b** refer to the exact same list object.
- They are **aliases** for the same object. This phenomenon is known as **aliasing**.
- To prevent aliasing, a new object can be created and the contents of the original can be copied which is called **cloning**

Example	Output:
a= [1, 2, 3 ,4 ,5]	
b=a	
print (b)	[1, 2, 3, 4, 5]
a is b	True
a[0]=100	
print(a)	[100,2,3,4,5]
print(b)	[100,2,3,4,5]



Cloning

- To avoid the disadvantages of copying we are using cloning. creating a copy of a same list of elements with two different memory locations is called cloning.
- Changes in one list will not affect locations of another list.
- Cloning is a process of making a copy of the list without modifying the original list.
- Cloning can be performed by:
    - **1. Slicing**
    - **2. list()method**
    - **3. copy() method**

clonning using Slicing

```
>>>a=[1,2,3,4,5]
>>>b=a[:]
>>>print(b)
[1,2,3,4,5]
>>>a is b
False
```

clonning using copy() method

```
a=[1,2,3,4,5]
>>>b=a.copy()
>>> print(b)
[1, 2, 3, 4, 5]
>>> a is b
False
```

clonning using List( ) method

```
>>>a=[1,2,3,4,5]
>>>b=list(a)
>>>print(b)
[1,2,3,4,5]
>>>a is b
false
>>>a[0]=100
>>>print(a)
>>>a=[100,2,3,4,5]
>>>print(b)
>>>b=[1,2,3,4,5]
```

List as parameters

- In python, arguments are passed by reference.
- If any changes are done in the parameter which refers within the function, then the changes also reflects back in the calling function.
- When a list to a function is passed, the function gets a reference to the list.
- Passing a list as an argument actually passes a reference to the list, not a copy of the list.
- Since lists are mutable, changes made to the elements referenced by the parameter change the same list that the argument is referencing.

Example 1`:	**Output**
def remove(a): a.remove(1) a=[1,2,3,4,5] remove(a) print(a)	[2,3,4,5]
Example 2:	**Output**

def inside(a):
  for i in range(0,len(a),1):
  a[i]=a[i]+10
  print("inside",a)
a=[1,2,3,4,5]
inside(a)
print("outside",a)
	inside [11, 12, 13, 14, 15]
	outside [11, 12, 13, 14, 15]

Example 3	output
def insert(a): a.insert(0,30) a=[1,2,3,4,5] insert(a) print(a)	[30, 1, 2, 3, 4, 5]

Tuple

- Tuple is same as list, except that the set of elements is **enclosed in parentheses**
- instead of square brackets.
- **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add

elements to a tuple or remove elements from the tuple.

methods	example	description

list( )	>>> a=(1,2,3,4,5)
	>>> a=list(a)
	>>> print(a)
	[1, 2, 3, 4, 5]
		it convert the given tuple into list.

tuple( )	>>> a=[1,2,3,4,5]
	>>> a=tuple(a)
	>>> print(a)
	(1, 2, 3, 4, 5)
		it convert the given list into tuple.

Benefit of Tuple:

- Tuples are faster than lists.
- If the user wants to protect the data from accidental changes, tuple can be used.
- Tuples can be used as keys in dictionaries, while lists can't.

Operations on Tuples:

- **1. Indexing**
- **2. Slicing**
- **3. Concatenation**
- **4. Repetitions**
- **5. Membership**
- **6. Comparison**

Operations	examples	description
Creating a tuple	>>>a=(20,40,60,"apple","ball")	Creating the tuple with elements of different data types.
Indexing	>>>print(a[0]) 20 >>> a[2] 60	Accessing the item in the position 0 Accessing the item in the position 2
Slicing	>>>print(a[1:3]) (40,60)	Displaying items from 1st till 2nd.
Concatenation	>>> b=(2,4) >>>print(a+b) >>>(20,40,60,"apple","ball",2,4)	Adding tuple elements at the end of another tuple elements
Repetition	>>>print(b*2) >>>(2,4,2,4)	repeating the tuple in n no of times

Membership	>>> a=(2,3,4,5,6,7,8,9,10)
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in tuple. Otherwise returns false.

Comparison	>>> a=(2,3,4,5,6,7,8,9,10)
	>>>b=(2,3,4)
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

Tuple methods

methods	example	description

a.index(tuple)	>>> a=(1,2,3,4,5)
	>>> a.index(5)
	4
		Returns the index of the first matched item.

a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple
min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

Tuple Assignment

- Tuple assignment allows, variables on the left of an assignment operator and values of tuple on the right of the assignment operator.

- Multiple assignment works by creating a tuple of expressions from the right hand side, and a tuple of targets from the left, and then matching each expression to a target.

- Because multiple assignments use tuples to work, it is often termed tuple assignment.

- It is often useful to swap the values of two variables.

Example:

Swapping using temporary variable:	Swapping using tuple assignment:
a=20 b=50 temp = a a = b b = temp print("value after swapping is",a,b)	a=20 b=50 (a,b)=(b,a) print("value after swapping is",a,b)

Multiple assignments:

Multiple values can be assigned to multiple variables using tuple assignment.

```
>>>(a,b,c)=(1,2,3)
>>>print(a)
1
>>>print(b)
2
>>>print(c)
3
```

Tuple as return value

- A Tuple is a comma separated sequence of items.
- It is created with or without ( ).
- A function can return one value. if you want to return more than one value from a function. we can use tuple as return value.

Example1:	**Output:**

def div(a,b):
  r=a%b
  q=a//b
  return(r,q)
a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
r,q=div(a,b)
print("reminder:",r)
print("quotient:",q)
	enter a value:4
	enter b value:3
	reminder: 1
	quotient: 1

Example2:	**Output:**

def min_max(a):
  small=min(a)
  big=max(a)
  return(small,big)
a=[1,2,3,4,6]
small,big=min_max(a)
print("smallest:",small)
print("biggest:",big)
	smallest: 1
	biggest: 6

Tuple as argument

- The parameter name that begins with * gathers argument into a tuple.

Example:	**Output:**
def printall(*args): print(args) printall(2,3,'a')	(2, 3, 'a')

Boolean

- Boolean data type have two values either it can take 0 or 1.
- 0 means False
- 1 means True
- True and False is a keyword.

Example

- >>> 3==5
- False
- >>> True+True
- 2
- >>> False+True
- 1
- >>> False*True
- 0

Dictionaries

- Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- Dictionary is created by using **curly brackets**. i,e. { }
- All elements in dictionary are placed inside the curly braces i.e. { }
- Dictionary comprises of elements of different data types.
- Dictionaries **are accessed via keys** and not via their position.
- Keys must be immutable data type (numbers, strings, tuple)
- A dictionary is an associative array. Any key of the dictionary is associated (or mapped) to a value.
- The values of a dictionary can be any Python data type. So dictionaries are **keyvalue - pairs**
- Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.

Operations on dictionary

- 1. Accessing an element
- 2. Update
- 3. Add element
- 4. Membership

Operations	Example	Description

Creating a dictionary	>>> a={1:"one",2:"two"}
	>>> print(a)
	{1: 'one', 2: 'two'}
		Creating the dictionary with elements of different data types.

accessing an element	>>> a[1]
	'one'
	>>> a[0]
	KeyError: 0
		Accessing the elements by using keys.

Update	>>> a[1]="ONE"
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Assigning a new value to key. It replaces the old value by new value.

add element	>>> a[3]="three"
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
		Add new element in to the dictionary with key.

membership	a={1: 'ONE', 2: 'two', 3: 'three'}
	>>> 1 in a
	True
	>>> 3 not in a
	False
		Returns True if the key is present in dictionary. Otherwise returns false.

Methods in dictionary

Method	Example	Description

a.copy( )	a={1: 'ONE', 2: 'two', 3: 'three'}
	>>> b=a.copy()
	>>> print(b)
	{1: 'ONE', 2: 'two', 3: 'three'}
		It returns copy of the dictionary. here copy of dictionary 'a' get stored in to dictionary 'b'

a.items()	>>> a.items() dict_items([(1, 'ONE'), (2, 'two'), (3, 'three')])	Return a new view of the dictionary's items. It displays a list of dictionary's (key, value) tuple pairs.
a.keys()	>>> a.keys() dict_keys([1, 2, 3])	It displays list of keys in a dictionary
a.values()	>>> a.values() dict_values(['ONE', 'two', 'three'])	It displays list of values in dictionary

a.pop(key)	>>> a.pop(3)
	'three'
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Remove the element with *key* and return its value from the dictionary.

setdefault(key,value)	>>> a.setdefault(3,"three")
	'three'
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
	>>> a.setdefault(2)
	'two'
		If key is in the dictionary, return its value. If key is not present, insert key with a value of dictionary and return dictionary.

a.update(dictionary)	>>> b={4:"four"}
	>>> a.update(b)
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three', 4: 'four'}
		It will add the dictionary with the existing
		dictionary

fromkeys()	>>> key={"apple","ball"}
	>>> value="for kids"
	>>> d=dict.fromkeys(key,value)
	>>> print(d)
	{'apple': 'for kids', 'ball': 'for kids'}
		It creates a dictionary from key and values.

len(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>>lena(a) 3	It returns the length of the list.
clear()	a={1: 'ONE', 2: 'two', 3: 'three'} >>>a.clear() >>>print(a) >>>{}	Remove all elements form the dictionary.
del(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>> del(a)	It will delete the entire dictionary.

List	Tuples	Dictionary
A list is mutable	A tuple is immutable	A dictionary is mutable
Lists are dynamic	Tuples are fixed size in nature	In values can be of any data type and can repeat, keys must be of immutable type
List are enclosed in brackets[ ] and their elements and size can be changed	Tuples are enclosed in parenthesis ( ) and cannot be updated	Tuples are enclosed in curly braces { } and consist of key:value
Homogenous	Heterogeneous	Homogenous
Example: List = [10, 12, 15]	Example: Words = ("spam", "egss") Or Words = "spam", "eggs"	Example: Dict = {"ram": 26, "abi": 24}
Access: print(list[0])	Access: print(words[0])	Access: print(dict["ram"])

Can contain duplicate elements	Can contain duplicate elements. Faster compared to lists	Cant contain duplicate keys, but can contain duplicate values
Slicing can be done	Slicing can be done	Slicing can't be done
Usage: ❖ List is used if a collection of data that doesnt need random access. ❖ List is used when data can be modified frequently	Usage: ❖ Tuple can be used when data cannot be changed. ❖ A tuple is used in combination with a dictionary i.e.a tuple might represent a key.	Usage: ❖ Dictionary is used when a logical association between key:value pair. ❖ When in need of fast lookup for data, based on a custom key. ❖ Dictionary is used when data is being constantly modified.

Advanced list processing: List Comprehension

- List comprehensions provide a concise way to apply operations on a list.
- It creates a new list in which each element is the result of applying a given operation in a list.
- It consists of brackets containing an expression followed by a “for” clause, then a list.
- The list comprehension always returns a result list.
- **Syntax**

list=[expression for item in list if conditional]

List Comprehension	Output
>>>L=[x**2 for x in range(0,5)] >>>print(L)	[0, 1, 4, 9, 16]
>>>[x for x in range(1,10) if x%2==0]	[2, 4, 6, 8]
>>>[x for x in 'Python Programming' if x in ['a','e','i','o','u']]	['o', 'o', 'a', 'i']
>>>mixed=[1,2,"a",3,4.2] >>> [x**2 for x in mixed if type(x)==int]	[1, 4, 9]
>>>[x+3 for x in [1,2,3]]	[4, 5, 6]
>>> [x*x for x in range(5)]	[0, 1, 4, 9, 16]
>>> num=[-1,2,-3,4,-5,6,-7] >>> [x for x in num if x>=0]	[2, 4, 6]

>>> str=["this","is","an","example"]
>>> element=[word[0] for word in str]
>>> print(element)
	['t', 'i', 'a', 'e']

Nested list

- List inside another list is called nested list

Example:

```
>>> a=[56,34,5,[34,57]]
>>> a[0]
56
>>> a[3]
[34, 57]
>>> a[3][0]
34
>>> a[3][1]
57
```

Programs on matrix:

Matrix addition	Output

a=[[1,1],[1,1]]
b=[[2,2],[2,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(b)):
  c[i][j]=a[i][j]+b[i][j]
for i in c:
  print(i)
	[3, 3]
	[3, 3]

Matrix subtraction	Output

a=[[1,1],[1,1]]
b=[[2,2],[2,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(b)):
  c[i][j]=a[i][j]+b[i][j]
for i in c:
  print(i)
	[-1, -1]
	[-1, -1]

Matrix multiplication	Output

a=[[1,1],[1,1]]
b=[[2,2],[2,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(b)):
  for k in range(len(b)):
  c[i][j]=a[i][j]+a[i][k]*b[k][j]
for i in c:
  print(i)
	[3, 3]
	[3, 3]

Matrix transpose	Output

a=[[1,3],[1,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(a)):
  c[i][j]=a[j][i]
for i in c:
  print(i)
	[1, 1]
	[3, 2]

Sets

- Set is an unordered collection of values of any data type with no duplicate entry.
- In set every element is unique.
- Elements are separated by”,” enclosed with { }

- **Example:**
  - a={1,2,3,4,5,6,7,6,7}
  - print(a)
  - {1,2,3,4,5,6,7} → it automatically eliminates duplicated

VARIABLES

- A variable allows us to store a value by assigning it to a name, which can be used later.
- Named memory locations to store values.
- Programmers generally choose names for their variables that are meaningful.
- It can be of any length. No space is allowed.
- We **don't need to declare a variable** before using it. In Python, we simply assign a value to a variable and it will exist.
- **Assigning values to variables:**

  = sign used to assign values to a variables.
- **Example:**

```
a=5 // single assignment
a,b,c=2,3,4 // multiple assignment
a=b=c=2 //assigning one value to multiple variables
```

KEYWORDS

- Keywords are the reserved words in Python.
- We cannot use a keyword as variable name, function name or any other identifier.
- They are used to define the syntax and structure of the Python language.
- Keywords are case sensitive.

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

IDENTIFIERS

- Identifier is the name given to entities like class, functions, variables etc. in Python.
- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like !, @, #, $, % etc. in our identifier.
- Identifier can be of any length.

Valid identifiers	Invalid identifiers
num	Number 1
Num	num 1
Num1	addition of program
_NUM	1Num
NUM_temp2	Num.no
IF	if
Else	else

INPUT AND OUTPUT

INPUT:

- Input is a data entered by user in the program.
- In python, **input () function** is available for input.
- **Syntax:**
  - variable = input (“data”)
- **Example:**
  - a=input("enter the name:")

  Output

  Enter the name: Hello

OUTPUT:

- Output can be displayed to the user using Print statement.

Syntax:

- print (expression/constant/variable)

Example:

- print ("Hello") → Hello
- print(5) → 5
- print(3+5) → 8
- C=10
- print(c) → 10

COMMENTS

- Comments are used to provide more details about the program like name of the program, author of the program, date and time, etc.
- Types of comments:
    - 1. single line comments
    - 2. multi line comments

Single line comments

- A hash sign (#) is the beginning of a comment.
- Anything written after # in a line is ignored by interpreter.

Example:

- Percentage = (minute * 100) / 60 **# calculating percentage of an hour**

Multi line comments:

- Multiple line comments can be written using triple quotes “””
  …….”””

Example:

“”” This program is created by Robert
in Dell lab on 12/12/12

based on newtons method”””

OPERATORS

- Operators are the constructs which can manipulate the value of operands.
- Consider the expression *4 + 5 = 9.* Here, 4 and 5 are called operands and + is called operator

Types of Operators:

- 1. Arithmetic Operators
- 2. Comparison (Relational) Operators
- 3. Assignment Operators
- 4. Logical Operators
- 5. Bitwise Operators
- 6. Membership Operators
- 7. Identity Operators

Arithmetic operators

Operator	Description	Example a=10,b=20
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Comparison (Relational) Operators

Operator	Description	Example a=10,b=20
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c

		= c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Logical Operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise Operators

Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Membership Operators

- Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- Let, **x=[5,3,6,4,1].** To check particular item in list or not**, in and not in** operators are used.

Example:

- x=[5,3,6,4,1]
- >>> 5 in x
- True
- >>> 5 not in x
- False

Identity Operators

- They are used to check if two values (or variables) are located on the same part of the memory.

Example

- x = 5
- y = 5
- a = 'Hello'
- b = 'Hello'
- print(x is not y) → False
- print(a is b) → True

Operator Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

FUNCTIONS

- Function is a sub program which consists of set of instructions used to perform a specific task.
- A large program is divided into basic building blocks called function.
- When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.
- Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.
- Functions are used to avoid rewriting same code again and again in a program.
- Function provides code re-usability
- The length of the program is reduced.

BUILT IN FUNCTION

Built in function	description
>>>max(3,4) 4	# returns largest element
>>>min(3,4) 3	# returns smallest element
>>>len("hello") 5	#returns length of an object
>>>range(2,8,1) [2, 3, 4, 5, 6, 7]	#returns range of given values
>>>round(7.8) – 8.0	#returns rounded integer of the given number
>>>chr(5) \x05'	#returns a character (a string) from an integer
>>>float(5) 5.0	#returns float number from string or integer
>>>int(5.0) 5	# returns integer from string or float
>>>pow(3,5) 243	#returns power of given number
>>>type(5.6) <type 'float'>	#returns data type of object to which it belongs
>>>t=tuple([4,6.0,7]) (4, 6.0, 7)	# to create tuple of items from list

USER DEFINED FUNCTIONS

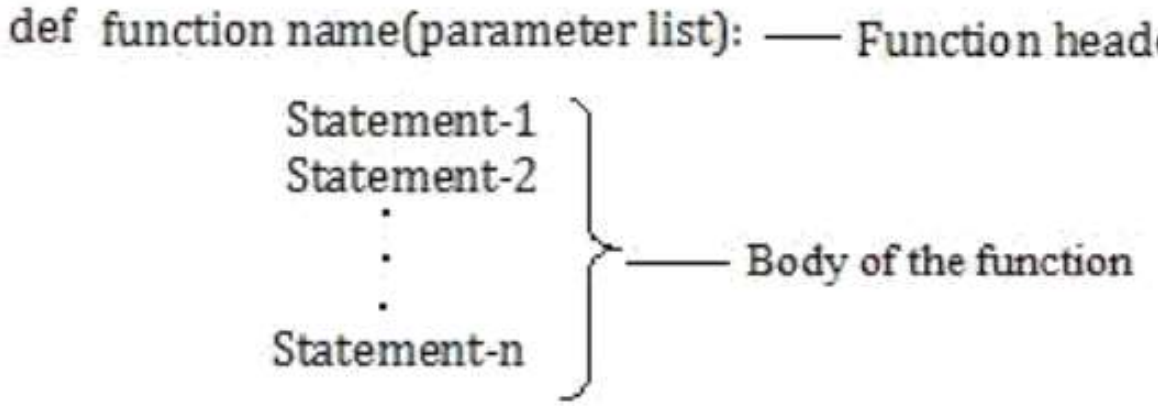
- User defined functions are the functions that programmers create for their requirement and use.
- These functions can then be combined to form module which can be used in other programs by importing them.
- The programmer can write their own functions which are known as user defined function. These functions can create by using **def** keyword.

ELEMENTS OF USER DEFINED FUNCTIONS

- There are two elements in user defined function.
    - 1. function definition
    - 2. function call

FUNCTION DEFINITION

- A **function definition specifies** the name of a new function and the sequence of statements that execute when the function is called.
- **Syntax**



- **def** is a keyword that indicates that this is a function definition.
- The rules for function names are the same as for variable names
- The first line of the function definition is called the **header**; the rest is called **body of function**.
- **The header has to end with a colon and the body has to be indented**. The function contains any number of statements.
- parameter list contains list of values used inside the function.

example

```
def add():
    a=eval(input(“enter a value”))
    b=eval(input(“enter a value”))
    c=a+b
    print(c)
```

Function call()

- function is called by using the function name followed by parenthesis().
- Argument is a list of values provided to the function when the function is called
- The statements inside the function does not executed until the function is called.
- Function can be called n number of times

Syntax:

- Function name(argument list)

Example:

- add()

FUNCTION PROTOTYPES

- Based on arguments and return type functions are classified into 4 types.

  - 1. Function without arguments and without return type
  - 2. Function with arguments and without return type
  - 3. Function without arguments and with return type
  - 4. Function with arguments and with return type

without return type	
Without argument	**With argument**

def add():
  a=int(input("enter a"))
  b=int(input("enter b"))
  c=a+b
  print(c)
add()
	def add(a,b):
	  c=a+b
	  print(c)
	a=int(input("enter a"))
	b=int(input("enter b"))
	add(a,b)

OUTPUT: enter a 5 enter b 10 15	**OUTPUT:** enter a 5 enter b 10 15

with return type	
Without argument	**With argument**

def add():
  a=int(input("enter a"))
  b=int(input("enter b"))
  c=a+b
  return c
c=add()
print(c)
	def add(a,b):
	  c=a+b
	  return c
	a=int(input("enter a"))
	b=int(input("enter b"))
	c=add(a,b)
	print(c)

OUTPUT: enter a 5 enter b 10 15	**OUTPUT:** enter a 5 enter b 10 15

ARGUMENTS TYPES

- You can call a function by using the following types of formal arguments:

    - 1. Required arguments
    - 2. Keyword arguments
    - 3. Default arguments
    - 4. Variable-length arguments

Required Arguments

- The number of arguments in the function call should match exactly with the function definition.

Example	Output:
def student(name, roll): print(name,roll) student("george",98)	George 98
def student(name, roll): print(name,roll) student(101,"rithika")	101 rithika
def student(name, roll): print(name,roll) student(101)	student() missing 1 required positional argument: 'name'
def student(name, roll): print(name,roll) student()	student() missing 2 required positional arguments: 'roll' and 'name'

Keyword Arguments

- Python interpreter is able to use the keywords provided to match the values with parameters even though if they are arranged in out of order

Example	Output:
def student(name, roll,mark): print(name,roll,mark) student (mark=90,roll=102,name="bala")	bala 102 90
def student(name, roll,mark): print(name,roll,mark) student (name="bala", roll=102,mark=90)	bala 102 90

Default Arguments

- Assumes a default value if a value is not provided in the function call for that argument.
- It allows us to miss one or more arguments or place out of order in function call.

Example	Output:
def student(name="raja", roll=101,mark=50): print(name,roll,mark) student (mark=90,roll=102,name="bala")	bala 102 90
def student(name="raja", roll=101,mark=50): print(name,roll,mark) student (name="bala", roll=102)	bala 102 50
def student(name="raja", roll=101,mark=50): print(name,roll,mark) student ()	raja 101 50

Variable length Arguments

- If we want to specify more arguments than specified while defining the function, variable length arguments are used. It is denoted by * symbol before parameter.

Example	Output:
def student(name,*mark): print(name,mark) student (“bala”,90)	bala (90,)
def student(name, roll,*mark): print(name,roll,mark) student ("raja",90,70,80)	raja 90 (70, 80)

MODULES

- A module is a file containing Python definitions, functions, statements and instructions.
- Standard library of Python is extended as modules.
- To use these modules in a program, programmer needs to import the module.
- Once we import a module, we can reference or use to any of its functions or variables in our code.
- There is large number of standard modules also available in python.
- Standard modules can be imported the same way as we import our user-defined modules.
    - 1. import keyword
    - 2. form keyword

import keyword

- import keyword is used to get all functions from the module.
- Every module contains many function.
- To access one of the function , you have to specify the name of the module and
- the name of the function separated by dot . This format is called dot notation.

Syntax:

- **import module_name**

Importing builtin module:	**importing user defined module:**
import math x=math.sqrt(25) print(x)	import cal x=cal.add(5,4) print(x)

from keyword

- from keyword is used to get only particular function from the module.

Syntax:

- **from** module_name **import** function_name

Importing builtin module:	Importing user defined module:
frommath importsqrt x=sqrt(25) print(x)	fromcal importadd x=add(5,4) print(x)

math-mathematical module

math functions	description
math.ceil(x)	Return the ceiling of *x*, the smallest integer greater than or equal to *x*
math.floor(x)	Return the floor of *x*, the largest integer less than or equal to *x*.
math.factorial(x)	Return *x* factorial
math.sqrt(x)	Return the square root of *x*
math.log(x)	return the natural logarithm of *x*
math.log10(x)	returns the base-10 logarithms
math.log2(x)	Return the base-2 logarithm of
math.sin(x)	returns sin of x radians
math.cos(x)	returns cosine of x radians
math.tan(x)	returns tangent of x radians
math.pi	The mathematical constant π = 3.141592
math.gcd(x,y)	the greatest common divisor of the integers *x* and *y*
math.e	returns The mathematical constant e = 2.718281

random-Generate pseudo-random numbers

random function	*description*
random.**randrange**(*stop*)	return random numbers from 0
random.**randrange**(*start*, *stop*[, *step*])	return the random numbers in a range
random.**uniform**(*a*, *b*)	Return a random floating point number

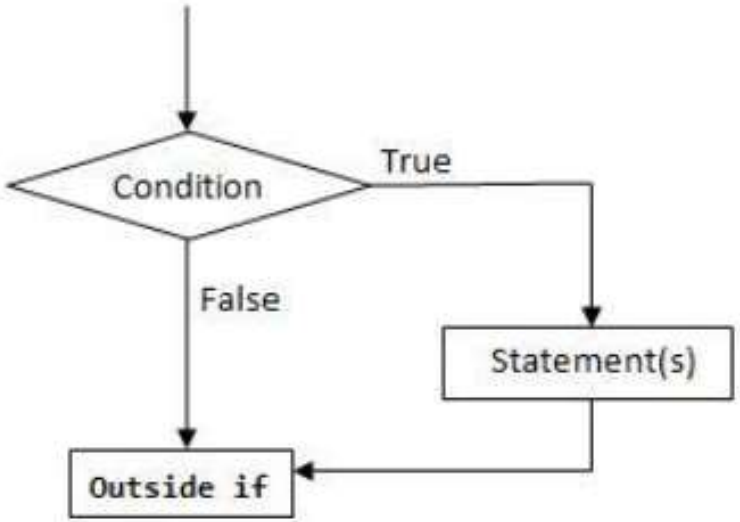
Conditional (if)

- conditional (if) is used to test a condition, if the condition is true the statements inside if will be executed.

Syntax:

```
if(condition 1):
    Statement 1
```

Flowchar



Example:

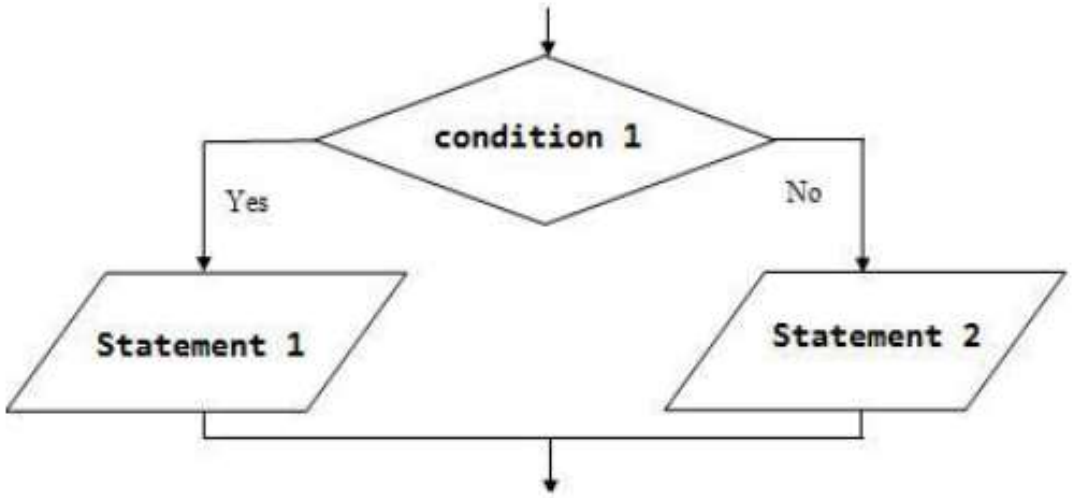
1. Program to provide flat rs 500, if the purchase amount is greater than 2000.
2. Program to provide bonus mark if the category is sports.

Program to provide flat rs 500, if the purchase amount is greater than 2000.	**output**
purchase=eval(input(enter your purchase amount)) if(purchase>=2000): purchase=purchase-500 print(amount to pay ,purchase)	enter your purchase amount 2500 amount to pay 2000
Program to provide bonus mark if the category is sports	**output**
m=eval(input(enter ur mark out of 100)) c=input(enter ur categery G/S) if(c== S): m=m+5 print(mark is ,m)	enter ur mark out of 100 85 enter ur categery G/S S mark is 90

Alternative (if-else)

- In the alternative the condition must be true or false. In this **else** statement can be combined with **if** statement.
- The **else** statement contains the block of code that executes when the condition is false.
- If the condition is true statements inside the if get executed otherwise else par
- **synt**

```
if(condition 1):
    Statement 1
else:
    Statement 2
```



Odd or even number	**Output**

n=eval(input("enter a number"))
if(n%2==0):
  print("even number")
else:
  print("odd number")
	enter a number4
	even number

positive or negative number	**Output**

n=eval(input("enter a number"))
if(n>=0):
  print("positive number")
else:
  print("negative number")
	enter a number8
	positive number

leap year or not	**Output**

y=eval(input("enter a yaer"))
if(y%4==0):
  print("leap year")
else:
  print("not leap year")
	enter a yaer2000
	leap year

greatest of two numbers	**Output**

a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
if(a>b):
  print("greatest:",a)
else:
  print("greatest:",b)
	enter a value:4
	enter b value:7
	greatest: 7

eligibility for voting	**Output**

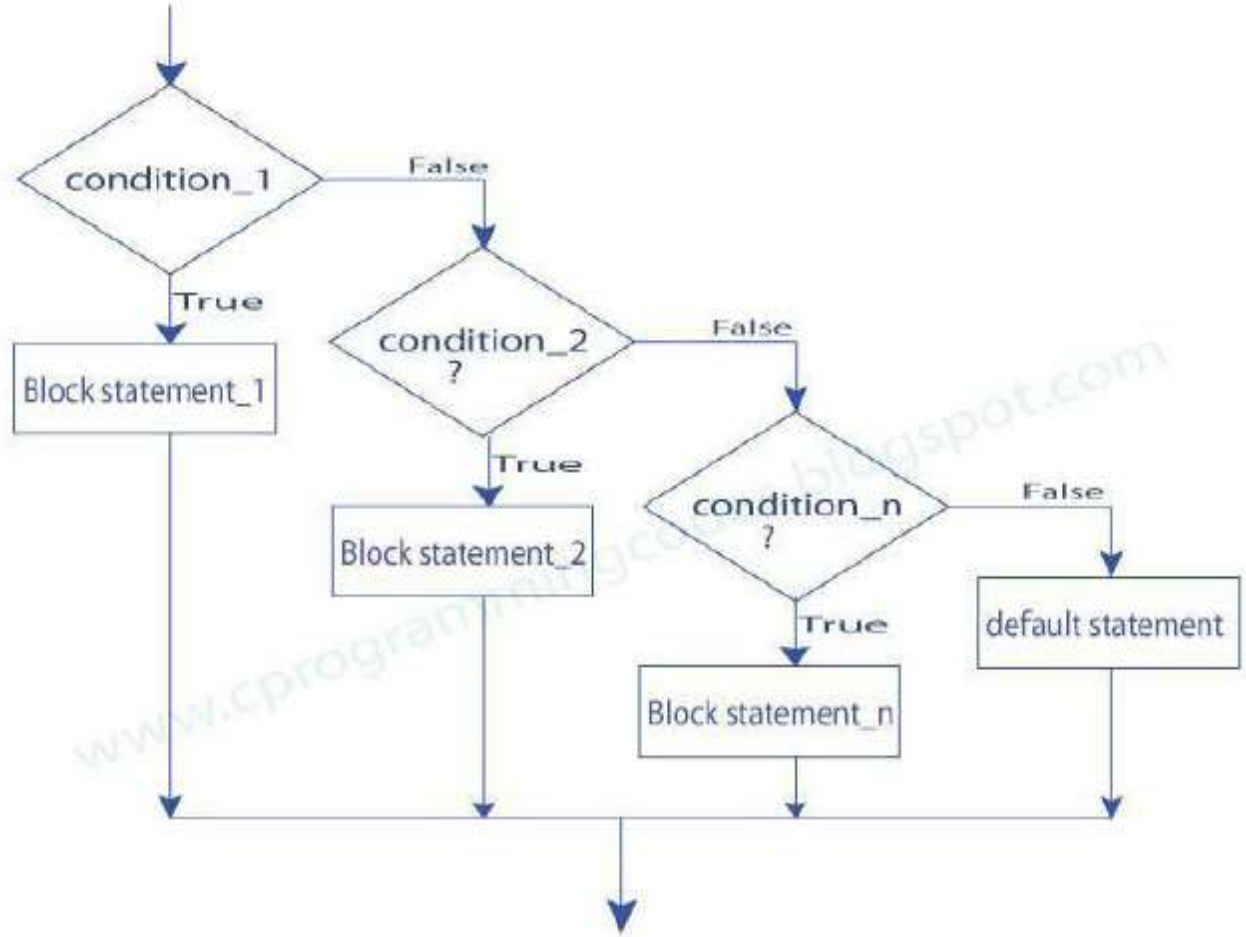
age=eval(input("enter ur age:"))
if(age>=18):
  print("you are eligible for vote")
else:
  print("you are eligible for vote")
	enter ur age:78
	you are eligible for vote

Chained conditionals(if-elif-else)

- The elif is short for else if.
- This is used to check more than one condition.
- If the condition1 is False, it checks the condition2 of the elif block. If all the conditions are False, then the else part is executed.
- Among the several if...elif...else part, only one part is executed according to the condition.
- The if block can have only one else block. But it can have multiple elif blocks.
- The way to express a computation like that is a chained conditional.

syntax

```
if(condition 1):
      statement 1
elif(condition 2):
      statement 2
elif(condition 3):
      statement 3
else:
     default statement
```



Examples

student mark system	Output

mark=eval(input("enter ur mark:"))
if(mark>=90):
  print("grade:S")
elif(mark>=80):
  print("grade:A")
elif(mark>=70):
  print("grade:B")
elif(mark>=50):
  print("grade:C")
else:
  print("fail")
	enter ur mark:78
	grade:B

traffic light system	**Output**

colour=input("enter colour of light:")
if(colour=="green"):
  print("GO")
elif(colour=="yellow"):
  print("GET READY")
else:
  print("STOP")
	enter colour of light:green
	GO

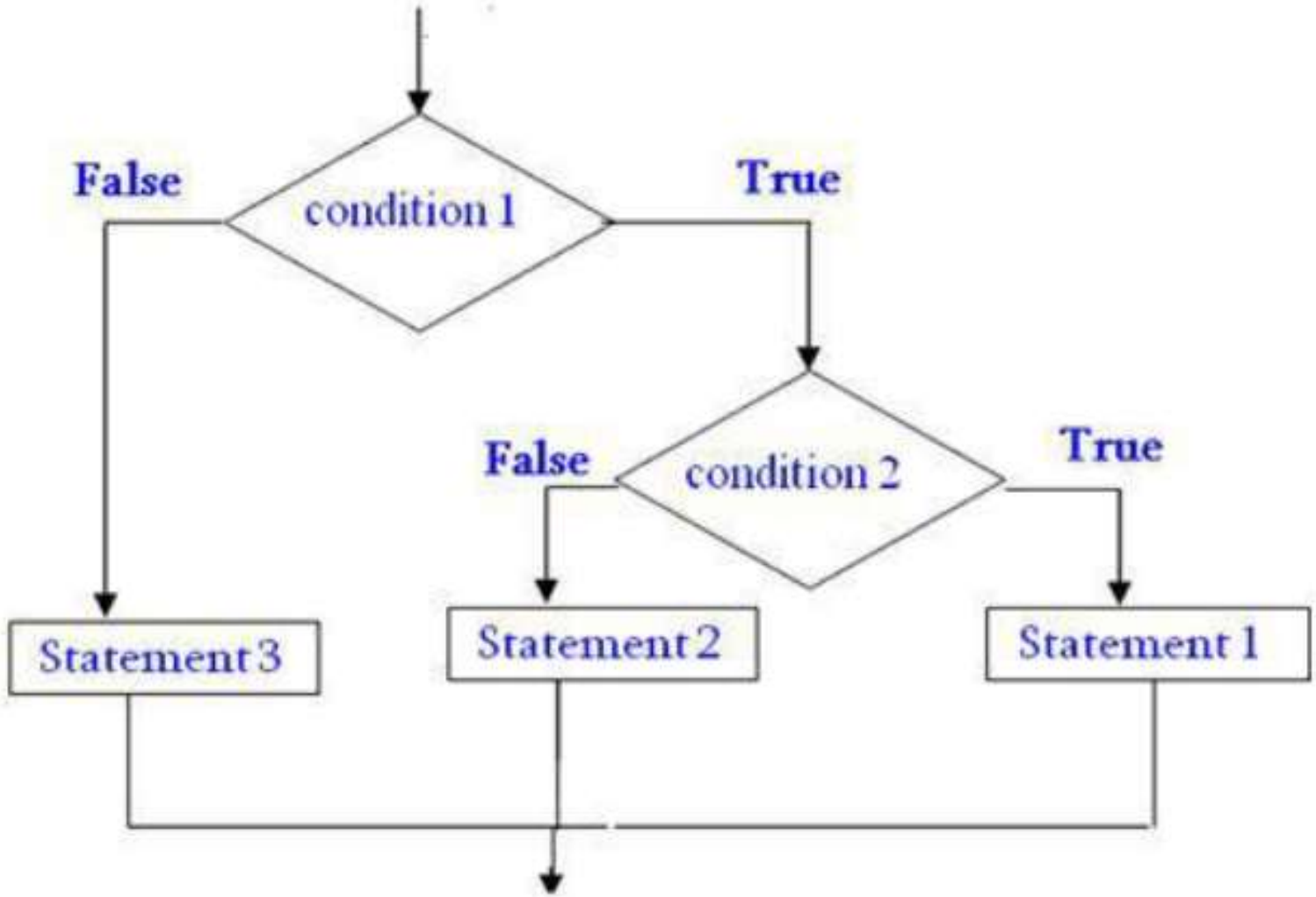
compare two numbers	**Output**

x=eval(input("enter x value:"))
y=eval(input("enter y value:"))
if(x == y):
  print("x and y are equal")
elif(x < y):
  print("x is less than y")
else:
  print("x is greater than y")
	enter x value:5
	enter y value:7
	x is less than y

Nested conditionals

- One conditional can also be nested within another.
- Any number of condition can be nested inside one another. In this, if the condition is true it checks another if condition1.
- If both the conditions are true statement1 get executed otherwise statement2 get execute.
- if condition uted
- ***Syntax:***

```
if (condition):
        if(condition 1):
            statement 1
        else:
            statement 2
else:
        statement 3
```


False
condition 1
True
False
condition 2
True
Statement 3
Statement 2
Statement 1

Example

greatest of three numbers	output

a=eval(input(enter the value of a))
b=eval(input(enter the value of b))
c=eval(input(enter the value of c))
if(a>b):
  if(a>c):
  print(the greatest no is ,a)
  else:
  print(the greatest no is ,c)
else:
  if(b>c):
  print(the greatest no is ,b)
  else:
  print(the greatest no is ,c)
	enter the value of a 9
	enter the value of a 1
	enter the value of a 8
	the greatest no is 9

ITERATION

State

- Transition from one process to another process under specified condition with in a time is called state.

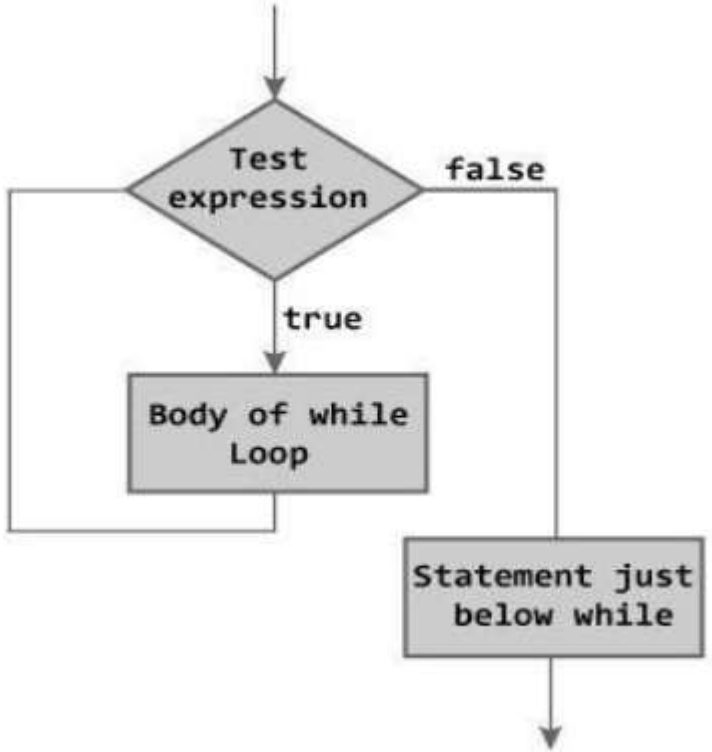
While loop

- While loop statement in Python is used to **repeatedly executes set of statement** as long as a given condition is true.
- In while loop, test expression is checked first. The body of the loop is entered only if the test expression is True.
- After one iteration, the test expression is checked again.
- This process continues until the test expression evaluates to False.
- In Python, the body of the while loop is determined through **indentation**.
- The statements inside the while start with indentation and the first **unindented line marks the end**.

- **Syntax**

```
inital value
while(condition):
    body of while loop
    increment
```

- **Flowchart**



else in while loop

- If else statement is used within while loop , **the else part will be executed when the condition become false.**
- The statements inside for loop and statements inside else will also execute
- Else statement will not be executed if the loop terminated by using **break** statement.

Program	output

i=1
while(i<=5):
  print(i)
  i=i+1
else:
  print("the number greater than 5")
	1
	2
	3
	4
	5
	the number greater than 5

For loop

- ***for in range:***
- We can generate a sequence of numbers using range() function.
- range(10) will generate numbers from 0 to 9 (10 numbers).
- In range function have to define the start, stop and step size as **range(start,stop,step size).**
- step size defaults to **1** if not provided.

Syntax

```
for i in range(start,stop,steps):
    body of for loop
```

For in sequence

- The for loop in Python is used to iterate over a sequence (list, tuple, string).
- Iterating over a sequence is called traversal. Loop continues until we reach the last element in the sequence.
- The body of for loop is separated from the rest of the code using indentation

Syntax

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

s.no	sequences	example	output
1.	For loop in string	for i in "Ramu": print(i)	R A M U

s.no	sequences	example	output
2.	For loop in list	for i in [2,3,5,6,9]: print(i)	2 3 5 6 9
3.	For loop in tuple	for i in (2,3,1): print(i)	2 3 1

else in for loop

- If else statement is used in for loop, the else statement is executed when the loop has reached the limit.
- The statements inside for loop and statements inside else

example	output

for i in range(1,6):
  print(i)
else:
  print("the number greater than 6")
	1
	2
	3
	4
	5 the number greater than 6

Examples

print nos divisible by 5 not by 10	**output**

n=eval(input("enter a"))
for i in range(1,n,1):
  if(i%5==0 and i%10!=0):
  print(i)
	enter a:30
	5
	15
	25

Fibonacci series	**output**

a=0
b=1
n=eval(input("Enter the number of terms: "))
print("Fibonacci Series: ")
print(a,b)
for i in range(1,n,1):
  c=a+b
  print(c)
  a=b
  b=c
	Enter the number of terms: 6
	Fibonacci Series:
	0 1
	1
	2
	3
	5
	8

find factors of a number	**Output**

n=eval(input("enter a number:"))
for i in range(1,n+1):
  if(n%i==0):
  print(i)
	enter a number:10
	1
	2
	5
	10

check the no is prime or not	**output**

n=eval(input("enter a number"))
for i in range(2,n):
  if(n%i==0):
  print("The num is not a prime")
  break
else:
  print("The num is a prime number.")
	enter a no:7
	The num is a prime number.

LOOP CONTROL STRUCTURES

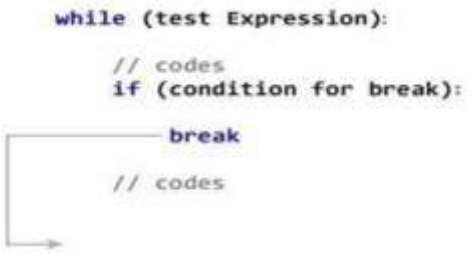
- **Break**
- **continue**
- **pass**

break

- Break statements can alter the flow of a loop.
- It terminates the current
- loop and executes the remaining statement outside the loop.
- If the loop has else statement, that will also gets terminated and come out of the loop completely.

Syntax:

break



Example

example	Output

for i in "welcome":
  if(i=="c"):
  break
  print(i)
	w
	e
	l

CONTINUE

- It terminates the current iteration and transfer the control to the next iteration in
- the loop.

Syntax:

continue

Example

Example:	Output

for i in "welcome":
  if(i=="c"):
  continue
  print(i)
	w
	e
	l
	o
	m
	e

PASS

- It is used when a statement is required syntactically but you don t want any code to execute.
- It is a null statement, nothing happens when it is executed

Syntax:

pass

Example	Output

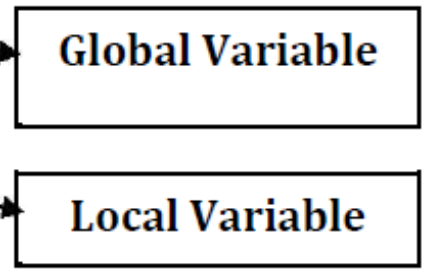
for i in welcome :
  if (i == c):
  pass
  print(i)
	w
	e
	l
	c
	o
	m
	e

Global Scope

- The *SCOPE* of a variable refers to the places that you can see or access a variable.
- A variable with global scope can be used anywhere in the program.
- It can be created by defining a variable outside the function

Example	output

a=50
def add():
  b=20
  c=a+b
  print(c)
def sub():
  b=30
  c=a-b
  print(c)
print(a)
	70
	20
	50

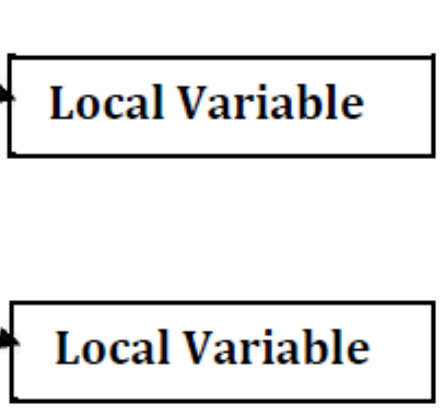


Local Scope

- A variable with local scope can be used only within the function

Example	output

def add():
  b=20
  c=a+b
  print(c)
def sub():
  b=30
  c=a-b
  print(c)
print(a)
print(b)
	
	70
	
	20
	
	error
	error



Function Composition

- Function Composition is the ability to call one function from within another
- function
- It is a way of combining functions such that the result of each function is passed as the argument of the next function.

Example:	**Output:**

def add(a,b):
  c=a+b
  return c
def mul(c,d):
  e=c*d
  return e
c=add(10,20)
e=mul(c,30)
print(e)
	900

find sum and average using function composition	**output**

def sum(a,b):
  sum=a+b
  return sum
def avg(sum):
  avg=sum/2
  return avg
a=eval(input("enter a:"))
b=eval(input("enter b:"))
sum=sum(a,b)
avg=avg(sum)
print("the avg is",avg)
	enter a:4
	enter b:8
	the avg is 6.0

Recursion

- A function calling itself till it reaches the base value - stop point of function call.
- Example: factorial of a given number using recursion

Factorial of n	Output

def fact(n):
  if(n==1):
  return 1
  else:
  return n*fact(n-1)
 	
n=eval(input("enter no. to find fact:"))
fact=fact(n)
print("Fact is",fact)
	enter no. to find fact:5
	Fact is 120

String built in functions and methods

- **Syntax to access the method**

Stringname.method()

a= “happy birthday” # here, a is the string name.

	syntax	example	description
1	a.capitalize()	>>> a.capitalize() ' Happy birthday	capitalize only the first letter in a string
2	a.upper()	>>> a.upper() 'HAPPY BIRTHDAY	change string to upper case
3	a.lower()	>>> a.lower() ' happy birthday	change string to lower case
4	a.title()	>>> a.title() ' Happy Birthday '	change string to title case i.e. first characters of all the words are capitalized.

	syntax	example	description
5	a.swapcase()	>>> a.swapcase() 'HAPPY BIRTHDAY'	change lowercase characters to uppercase and vice versa
6	a.split()	>>> a.split() ['happy', 'birthday']	returns a list of words separated by space
7	a.center(width, fillchar)	>>>a.center(19, *) '***happy birthday***'	pads the string with the specified fillchar till the length is equal to width
8	a.count(substring)	>>> a.count('happy') 1	returns the number of occurrences of substring
9	a.replace(old,new)	>>>a.replace('happy', 'wishyou happy')	replace all old substrings with new substrings
		'wishyou happy birthday'	

10	a.join(b)	>>> b="happy"
		>>> a="-"
		>>> a.join(b)
		'h-a-p-p-y'
			returns a string concatenated with the elements of an iterable. (Here a is the iterable)

11	a.isupper()	A= happy birthday >>> a.isupper() False	checks whether all the case-based characters (letters) of the string are uppercase.
12	a.islower()	>>> a.islower() True	checks whether all the case-based characters (letters) of the string are lowercase.
13	a.isalpha()	>>> a.isalpha() False	checks whether the string consists of alphabetic characters only.
14	a.isalnum()	>>> a.isalnum() False	checks whether the string consists of alphanumeric characters.
15	a.isdigit()	>>> a.isdigit() False	checks whether the string consists of digits only.
16	a.isspace()	>>> a.isspace() False	checks whether the string consists of whitespace only.

17	a.istitle()	>>> a.istitle() False	checks whether string is title cased.
18	a.startswith(substring)	>>> a.startswith("h") True	checks whether string starts with substring
19	a.endswith(substring)	>>> a.endswith("y") True	checks whether the string ends with the substring
20	a.find(substring)	>>> a.find("happy") 0	returns index of substring, if it is found. Otherwise -1 is returned.
21	len(a)	>>>len(a) >>>14	Return the length of the string
22	min(a)	>>>min(a) >>>	Return the minimum character in the string
23	max(a)	max(a) >>> y	Return the maximum character in the string

String modules:

- A module is a file containing Python definitions, functions, statements.
- Standard library of Python is extended as modules.
- To use these modules in a program, programmer needs to import the module
- Once we import a module, we can reference or use to any of its functions or variables in our code.
- There is large number of standard modules also available in python.
- Standard modules can be imported the same way as we import our user-defined modules.

Syntax:

```
import module_name
```

Example	output
import string print(string.punctuation) print(string.digits) print(string.printable) print(string.capwords("happy birthday")) print(string.hexdigits) print(string.octdigits)	 !"#$%&'()*+,-./:;<=>?@[\]^_`{\|}~ 0123456789 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~ Happy Birthday 0123456789abcdefABCDEF 01234567

Escape sequences in string

Escape Sequence	Description	example
\n	new line	>>> print("hai \nhello") hai hello
\\	prints Backslash (\)	>>> print("hai\\hello") hai\hello
\'	prints Single quote (')	>>> print("'") '
\"	prints Double quote (")	>>>print("\"") "
\t	prints tab sapace	>>>print(hai\thello) hai hello
\a	ASCII Bell (BEL)	>>>print(\a)

List as array

- **Array:**
- Array is a collection of similar elements. Elements in the array can be accessed by index.
- Index starts with 0. Array can be handled in python by module named array.
- To create array have to import array module in the program.

Syntax :

import array

Syntax to create array:

array_name = module_name.function_name(datatype ,[elements])

Example:

a=array.array(i ,[1,2,3,4])

a- array name

array- module name

i- integer datatype

Program to find sum of array elements	**Output**
import array sum=0 a=array.array('i',[1,2,3,4]) for i in a: sum=sum+i print(sum)	10

Methods in array

- a=[2,3,4,5]

NO	Syntax	example	Description
1	array(data type, value list)	array(i ,[2,3,4,5])	This function is used to create an array with data type and value list specified in its arguments.
2	append()	>>>a.append(6) [2,3,4,5,6]	This method is used to add the at the end of the array.
3	insert(index,element)	>>>a.insert(2,10) [2,3,10,5,6]	This method is used to add the value at the position specified in its argument.
4	pop(index)	>>>a.pop(1) [2,10,5,6]	This function removes the element at the position mentioned in its argument, and returns it.

NO	Syntax	example	Description
5	index(element)	>>>a.index(2) 0	This function returns the index of value
6	reverse()	>>>a.reverse() [6,5,10,2]	This function reverses the array.
7	count()	a.count() 4	This is used to count number of elements in an array

File

- File is a named location on disk to store related information. It is used to
- permanently store data in a memory (e.g. hard disk).

File Types:

- **1. text file**
- **2. binary file**

Text File	Binary file
Text file is a sequence of characters that can be sequentially processed by a computer in forward direction.	A binary files store the data in the binary format (i.e. 0 s and 1 s)
Each line is terminated with a special character, called the EOL or End of Line character	It contains any type of data (PDF , images , Word doc ,Spreadsheet, Zip files,etc)

Operations on Files

- In Python, a file operation takes place in the following order,
  - **1. Opening a file**
  - **2. Reading / Writing file**

How to open a file:

Syntax:	Example:
file_object=open(file_name.txt , mode)	f=open(sample.txt , w)

How to create a file:

Syntax:	Example:
file_object=open(file_name.txt , mode) file_object.write(string) file_object.close()	f=open(sample.txt , w) f.write(hello) f.close()

Modes in file

modes	description
r	read only mode
w	write only
a	appending mode
r+	read and write mode
w+	write and read mode

Differentiate write and append mode:

write mode	append mode
It is use to write a string into a file.	It is used to append (add) a string into a file.
If file is not exist it creates anew file.	If file is not exist it creates anew file.
If file is exist in the specified name, the exixting content will overwrite in a file by the given string.	It will add the string at the end of the old file.

S.No	Syntax	Example	Description
1	f.write(string)	f.write("hello")	Writing a string into a file.
2	f.writelines(sequence)	f.writelines(1st line \n second line)	Writes a sequence of strings to the file.
3	f.read(size)	f.read( ) #read entire file f.read(4) #read the first 4 charecter	To read the content of a file.
4	f.readline( )	f.readline( )	Reads one line at a time.
5	f.readlines( )	f.readlines( )	Reads the entire file and returns a list of lines.
6	f.seek(offset,whence) whence value is optional.	f.seek(0)	Move the file pointer to the appropriate position. It sets the file pointer to the starting of the file.
	whence =0 from begining	f.seek(3,0)	Move three character from the beginning.
	whence =1 from current position	f.seek(3,1)	Move three character ahead from the current position.
	whence =2 from last position	f.seek(-1,2)	Move to the first character from end of the file

S.No	Syntax	Example	Description
7	f.tell( )	f.tell( )	Get the current file pointer position.
8	f.flush( )	f.flush( )	To flush the data before closing any file.
9	f.close( )	f.close( )	Close an open file.
10	f.name( )	f.name( )	Return the name of the file.
11	f.mode( )	f.mode( )	Return the Mode of file.
12	os.rename(old name,new name)	import os os.rename(1.txt , 2.txt)	Renames the file or directory.
13	os.remove(file name)	import os os.remove("1.txt")	Remove the file.

Format operator

Convert no into string:	output
>>> x = 52 >>> f.write(str(x))	52
Convert to strings using format operator, %	**Example:**
print (format string %(tuple of values)) file.write(format string %(tuple of values)	>>>age=13 >>>print(The age is %d %age) The age is 13
Program to write even number in a file using format operator	**OutPut**

f=open("t.txt","w")
n=eval(input("enter n:"))
for i in range(n):
  a=int(input("enter number:"))
  if(a%2==0):
  f.write(a)
f.close()
	enter n:4
	enter number:3
	enter number:4
	enter number:6
	enter number:8
	
	result in file t.txt
	4
	6
	8

Format operator

- The argument of write( ) has to be a string, so if we want to put other values in a file, we have to convert them to strings.
- The first operand is the format string, which specifies how the second operand is formatted.
- The result is a string. For example, the format sequence '%d' means that the
- second operand should be formatted as an integer (d stands for decimal)

Format character	Description
%c	Character
%s	String formatting
%d	Decimal integer
%f	Floating point real number

Command line argument

- The command line argument is used to pass input from the command line to your program when they are started to execute.
- Handling command line arguments with Python need sys module.
- sys module provides information about constants, functions and methods of the python interpreter
- argv[ ] is used to access the command line argument. The argument list starts from 0.
- argv[ ] the first argument always the file name
- sys.argv[0]= gives file name
- sys.argv[1]=provides access to the first input

Example 1	**output**
import sys print(“the file name is %s” %(sys.argv[0]))	python demo.py the file name is demo.py
addition of two num	**output**
import sys a= sys.argv[1] b= sys.argv[2] sum=int(a)+int(b) print("sum is",sum)	sam@sam~$ python sum.py 2 3 sum is 5

Word count using comment line arg:	**Output**

from sys import argv
a = argv[1].split()
dict = {}
for i in a:
  if i in dict:
  dict[i]=dict[i]+1
  else:
  dict[i] = 1
print(dict)
print(len(a))
	C:\Python34>python word.py
	"python is awesome lets program in python"
	{'lets': 1, 'awesome': 1, 'in': 1, 'python': 2, 'program': 1, 'is': 1}
	7

Errors and exception

Errors

- Errors are the mistakes in the program also referred as bugs.
- They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:
    - **1. Syntax errors**
    - **2. Runtime errors**
    - **3. Logical errors**

Syntax errors

- Syntax errors are the errors which are displayed when the programmer do mistakes when writing a program.
- When a program has syntax errors it will not get executed.
- Common Python syntax errors include:
    - **1. leaving out a keyword**
    - **2. putting a keyword in the wrong place**
    - **3. leaving out a symbol, such as a colon, comma or brackets**
    - **4. misspelling a keyword**
    - **5. incorrect indentation**
    - **6. empty block**

Runtime errors

- If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter.
- However, the program may exit unexpectedly during execution if it encounters a runtime error.
- When a program has runtime error I will get executed but it will not produce output.
- Common Python runtime errors include:
  - **1. division by zero**
  - **2. performing an operation on incompatible types**
  - **3. using an identifier which has not been defined**
  - **4. accessing a list element, dictionary value or object attribute which doesn't exist**
  - **5. trying to access a file which doesn't exist**

Logical errors

- Logical errors are the most difficult to fix.
- They occur when the program runs without crashing, but produces an incorrect result.
- Common Python logical errors include:
    - **1. using the wrong variable name**
    - **2. indenting a block to the wrong level**
    - **3. using integer division instead of floating-point division**
    - **4. getting operator precedence wrong**
    - **5. making a mistake in a Boolean expression**

Exceptions

- An exception(runtime time error)is an error, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates or quit.

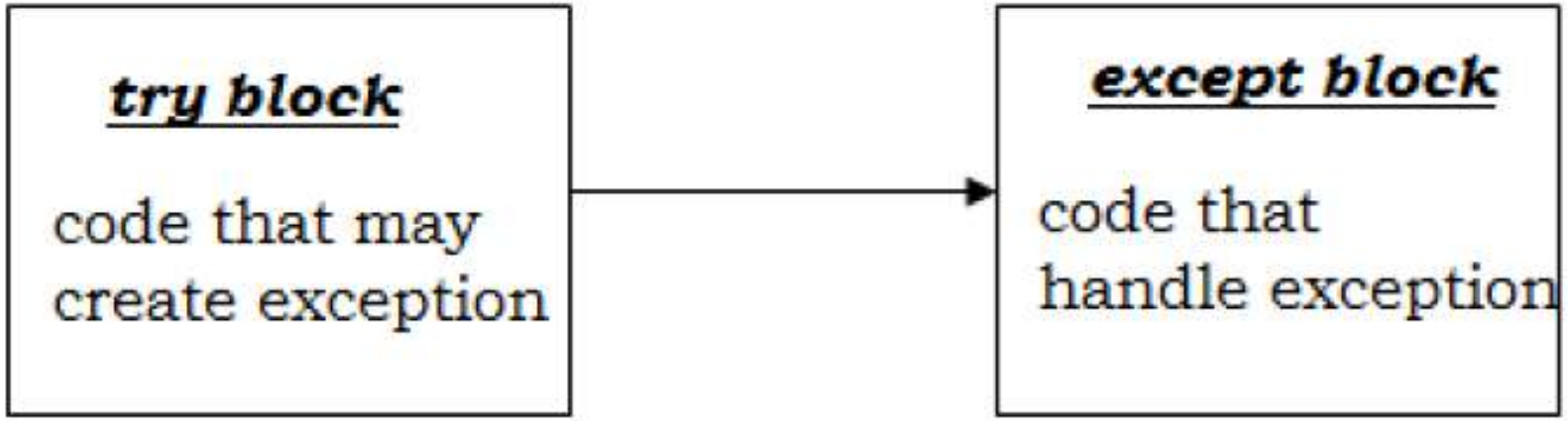
S.No.	Exception Name	Description
1	FloatingPointError	Raised when a floating point calculation fails.
2	ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
3	AttributeError	Raised in case of failure of attribute reference or assignment.
4	ImportError	Raised when an import statement fails.
5	KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.

6	IndexError	Raised when an index is not found in a sequence
7	KeyError	Raised when the specified key is not found in the dictionary.
8	NameError	Raised when an identifier is not found in the local or global namespace
9	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
10	SyntaxError	Raised when there is an error in Python syntax.
11	IndentationError	Raised when indentation is not specified properly.
12	SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
13	SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.

14	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
15	ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
16	RuntimeError	Raised when a generated error does not fall into any category.

Exception Handling

- Exception handling is done by try and catch block.
- Suspicious code that may raise an exception, this kind of code will be placed in try block.
- A block of code which handles the problem is placed in except block.



Catching Exceptions

1. try...except
2. try...except...inbuilt exception
3. try... except...else
4. try...except...else....finally
5. try.. except..except..
6. try...raise..except..

try ... except

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- It is up to us, what operations we perform once we have caught the exception.
- **Syntax**

```
try:
        code that create exception
except:
        exception handling statement
```

Example:	Output

try:
  age=int(input("enter age:"))
  print("ur age is:",age)
except:
  print("enter a valid age")
	enter age:8
	ur age is: 8
	enter age:f
	enter a valid age

try…except…inbuilt exception

- **Syntax**

  try:

  code that create exception

  except inbuilt exception:

  exception handling statement

Example:	Output

try:
  age=int(input("enter age:"))
  print("ur age is:",age)
except ValueError:
  print("enter a valid age")
	enter age:d
	enter a valid age

try ... except ... else clause

- Else part will be executed only if the try block does’t raise an exception.
- Python will try to process all the statements inside try block.
- If value error occurs, the flow of control will immediately pass to the except block and remaining statement in try block will be skipped.

Syntax

```
try:
        code that create exception
except:
        exception handling statement
else:
        statements
```

Example program	Output

try:
  age=int(input("enter your age:"))
except ValueError:
  print("entered value is not a number")
else:
  print("your age :",age)
	enter your age: six
	entered value is not a number
	enter your age:6
	your age is 6

try ... except...finally

- A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

Syntax

```
try:
        code that create exception
except:
        exception handling statement
else:
        statements
finally:
        statements
```

Example program	**Output**

try:
  age=int(input("enter your age:"))
except ValueError:
  print("entered value is not a number")
else:
  print("your age :",age)
finally:
  print("Thank you")
	enter your age: six
	entered value is not a number
	Thank you
	enter your age:5
	your age is 5
	Thank you

try...multiple exception

Syntax

```
try:
        code that create exception
except:
        exception handling statement
except:
        statements
```

Example	Output:

a=eval(input("enter a:"))
b=eval(input("enter b:"))
try:
  c=a/b
  print(c)
except ZeroDivisionError:
  print("cant divide by zero")
except ValueError:
  print("its not a number")
	enter a:2
	enter b:0
	cant divide by zero
	enter a:2
	enter b: h
	its not a number

Raising Exceptions

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise
- **Syntax:**

>>> raise error name

Example:	**Output:**

try:
  age=int(input("enter your age:"))
  if (age<0):
  raise ValueError("Age can't be negative")
except ValueError:
  print("you have entered incorrect age")
else:
  print("your age is:",age)
	enter your age:-7
	Age can't be negative