# **Python IDE’s:**

Pycharm

IDLE → setup

### **Interpreter:** Program that executes line by line

### **Compiled language:** it executes entire program at a time

Supports dynamic memory allocation: memory can allocated, deallocated during execution of program

Features: Readable syntax , easy to learn, free and open source large standard library, easy to write

### **Identifiers:** It is a name given to entities like variable, list, tuple, class, array, etc.

If you assign ay name to a programmable entity in python is called identifier

Syntax: isidentifier() → to check identifier or not

Eg: word = “bharat”

word.isidentifier() → True

6word = “bharat” → syntax error

Rules:

* Names should be start with alphabets and \_
* It contains alphanumeric but not start with numeric
* Special symbols are not allowed
* Not exceed 32 characters

### **Variables:** used to store values, no need to declare variables in python

Eg: Name = “ram” A = 10

If there are more than one word we have to use ‘\_’ (underscore)

Eg: a = “jai\_sree\_ram”

Rules:

* Can contains [A-Z, a-z, 0-9, \_ ]
* No special characters

### **Keywords:**

Special words or reserved words

It have own meaning → universal standard words → 32 keywords

“We can not assign any value to the keywords”

Syntax: help(“keywords”) → it shows in orange color

### **Constants :**

it is like a variable that never change and They are written in capitals letters

More Than one word we use underscore ‘\_’ and No syntax for constants

Rules:

* Names should be start with alphabets and \_
* not start with numeric
* Special symbols are not allowed

# 2. MEMORY MANAGEMENT IN PYTHON

Memory allocation can be defined as allocating a block of space in the computer memory to a program. In Python, memory allocation and deallocation (deleting a memory space) method is automatic as the Python developers created a **garbage collector** for Python so that the user does not have to do manual garbage collection.

**There are two parts of memory:**

stack memory: It is organized memory which stores references inside stack frames. heap memory: It is an unorganized memory which stores the objects.

### **Reference counting**

Reference counting works by counting the number of times an object is referenced by the references in the system. When references to an object are removed, the reference count for an object is decremented. When the reference count becomes zero, the object is deallocated.

E.g.,

a = "python" b = "java" c = "python"

In the above example, the reference count of "python" is 2 as it is referred to by both a and c. Similarly, the reference count of "java" is 1 as it is pointed only by b.

### **Garbage Collection**

Garbage collection is a process in which the interpreter frees up the memory when not in use to make it available for other objects.

Assume a case where no reference is pointing to an object in memory i.e. it is not in use so, the python virtual machine (pvm) has a garbage collector that automatically deletes that object from the heap memory

### **id()**

The id() function is a built-in function that returns the unique identifier of an object. The identifier is an integer, which represents the memory address of the object. The id() function is commonly used to check if two variables or objects refer to the same memory location.

Let us see an example to explain memory management: Consider the following code,

# Print and input function file

rule to be followed while using print() → cannot have sep and end before the values to be printed

print(value1, value2....., end="", sep="")

conventions

- there should be a space after "," in print statement

- the file should end in a new line.

### **print() →** Standard function used to print the output to the console.

Syntax : print(value1, value2, …, end=”\n”, sep=” “)

Parameters:

##### 1. **end →** used to append (add at the end) any string at the end of the output of the print statement in python. Or used to separate two print statements. → Default value: “\n”

Syntax: print(values, end=“string”)

Eg:

print("hello world", end=" -> ")

print("welcome to python class")

# o/p: "hello world -> welcome to python class"

##### 2. **sep →** adds the specified string in between the values to be printed. Or used to separate the values

present in the same print statement. → Default value: whitespace(“ “)

Syntax: print(values , sep=“string”)

Eg :

name = "John"

sentence = "My name is"

greet = "haiiii"

print(sentence, name, greet) # My name is John haiiii

print(sentence, name, greet, sep=",") # My name is,John,haiiii

print(sentence, name, greet, sep="\*") # My name is \* John \* haiiii

### **input() :** Takes the user's input at the runtime.

The input() function always reads the input as a string, even if it comprises digits.

Syntax : input() or input(“string”)

E.g.,

name = input("enter a name: ")

print("My name is", name)

### 

# Operators in python

which are performing some operation either on values or variable

### Arithmetic operator:

1. + → operator which represents addition
2. - → operator which represents subtraction
3. \* → operator which represents multiplication
4. / → operator which represents division
5. % → operator which represents modulus(it will give remainder)

for example 5 % 2

remainder = 1 → in python if we are giving

d = 5

e = 2

print(d/e) → 5/2 (always the division operator gives floating

point)

a = 4

b = 2

print(a/b)

// - represents floor division(integer number)

print(d//e)

\*\* - exponent → represent power

print(a\*\*b)

if in one expression we are having more than one operator

like 5 + 2 \* 4

print(5 + 2 \* 4)

*In arithmetic operators first come - Precedence*

1. () → parenthesis
2. \*\* → exponent
3. \* → multiplication
4. / → division
5. // → floor division
6. % → modulus
7. + → addition
8. - → sub

### **Assignment operator**

we are going to assign some values to a variable

== → it is a assignment operator

a = 10

print(a)

a = 10

a +=2 → a = a + 2 → compound assignment operator

a -=5

a /= 2

a \*\*=2

print(a)

### **comparison (or) relational operator**

→ relation between two operators

1. > → greater than
2. < → less than
3. >= → greater than or equal to
4. <= →less than or equal to
5. == → double equal to
6. != → not equal to

these are used to check some conditions and result would be either "True" or "False"

a = 6

print(a > 6)

print(a == 6)

print(a != 5)

### **logical operator**

used to combine two or more conditions or conditional statements

three types of logical operator

1. and
2. or
3. not

##### **and(operator)**

a, b = 5, 4

→ if i want to check 'a' should be greater than 4 and 'b' should be less than 10

→ like a > 4 and b < 10

print(a > 4 and b < 10) → 5>4 and 4 < 10

*for* ***and*** *truth table is*

**and** operator will give output as true if both operators are true and if any one is false it gives output as false

True - True = True

True - False = False

False- True = False

False- False= False

##### **or (operator)**

means either one condition should be true

a = 5

b = 10

print(a>4 or b>6)# true

print(a<4 or b<3)#false

print(a<4 or b<13)#True

print(a<4 or b>13)#false

***or*** *operator truth table*

True - True = True

True - False = True

False - True = True

False - False = False

##### **not(operator)**

it will reverse the result

age = 20

adult = age >= 18

print(adult)

print(not adult)

x = True

print(x)

print(not x)

### **Identity(operator)**

##### is → evaluates true if the variable on either side of the operator point to same object memory

x = [1,2,3]

y = [1,2,3]

print(id(x))

print(id(y))

print(x is y)

name1 = "steve"

name2 = "steve"

print(name1 is name2)

print(id(name1))

print(id(name2))

##### is not

x = [1,2,3]

y = [1,2,3]

print(id(x))

print(id(y))

print(x is not y)

name1 = "steve"

name2 = "steve"

print(name1 is not name2)

### **Membership operators**

##### in → if value present in data it returns True

##### not in → if value is not present it return True

word = "python class every morning"

print("p" in word)

print("every" in word)

print("world" not in word)

print("python" not in word)

### **Ternary operator**

in python it is a shortest way of writing an if-else statement in a single line

x = 5

if x > 10:

result = "greater than 10"

else:

result = "less than or equal to 10"

print(result)

syntax: value 1 if condition else value2

result = "greater than 10" if x > 10 else "less than or equal to

10"

print(result)

# Datatype

→ Data type represents the type of the data that is stored in a memory location

→ variable can hold values and every value has a datatype

##### **type()**

return the type of the variable passed

a = 10

print(type(a))

b = 2.89

print(type(b))

##### **dir()**

returns list of attribute that are attached to the object

categories in data types, In python data types are classified into two types

1. individual data type
2. collection /container data types

### **1.individual data types:** used to hold numeric values

Integer | float | complex | boolean

### **2.collection/container data types:**

string | list | tuple | set | dictionary

\*\* We can’t Use ‘0’ as prefix in python → octal integer representation

##### **integer:** it specifies the data stored inside the memory location to be an integer

it can be positive or negative

a = 5

a = int(15)

*operation on integer*

every arithmetic operations we can

###### **divmod() →** returns the quotient and remainder of the division

###### **abs() →** converts negative numbers into positive

print(abs(-18))

##### **float** → specifies that data is of type decimal

it can be positive or negative

b = 5.8

b = float(9.2)

##### **complex number** → written in the form of a+bj

a is the real part and b is imaginary part

it can be positive or negative

a = 2+3j

a = 2

b = 3

c = complex(2,3)

print(c)

### **Booleans →** useful in conditional expressions statements

defined by the True and False keyword

a = int(True)

print(a)

in python boolean values can be represented by integers

where '1' represents "True"

'0' represents "False"

print(divmod(4,3))

number = 03

print(number)

# Strings

collection of characters

strings in python are surrounded by either single quotes(''),double quotes(""),or triple("''")

it is an immutable data types → once they are created ,they cannot be changed and any operation

that appears to modify the original string it creates new string

syntax:var\_name = "string"

*Different ways to construct the string:*

using single quote(' '): → word = 'hello world'

using double quote(" ") → word = "hello world"

using triple quotes("''" “””, ‘’’ ‘’’) → message = (‘’’hello world, hi how are you !gudmrng’’’)

using string constructor: → word = str("hello world")

##### **empty string:** → word = " "

##### **regular strings:** → considers backslash as a special character

s = "Hello \task from pytho\n world"

print(s)

##### **raw strings:** → they are the string literals prefixed with 'r' or 'R'

do not treat backslash as a part of escape sequence

s =r"Hello \task from pytho\n world"

print(s)

##### **indexing a string** → process of extracting single character at a time

indexing can be positive(start with o) or negative(starts with -1)

syntax:var\_name[index]

a = "hello"

-5 -4 -3 -2 -1

h e l l o

0 1 2 3 4

print(a[0])

print(a[-3])

##### **slicing a string** → process of extracting multiple characters at a time

syntax:var\_name[start index:end index:step value]

start index:default value-0(optional)

end index: default value -length of the string(optional)

step value:default value-1(optional)

word = "Hello World"

-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

H e l l o W o r l d

0 1 2 3 4 5 6 7 8 9 10

positive:

print(word[0])-->prints the character present at the 0th index

print(word[10])-->d

print(word[0:5])-->Hello(prints up to 5th character bot not included the 5th character)

print(word[:5])-->Hello

print(word[6:])-->world

Negative

print(word[-1])-->d

print(word[-11])-->h

print(word[-4:])-->orld

print(word[0:-6])-->Hello

print(word[2:-3])-->llo Wo

step value

print(word[::2])-->prints every alternate character

print(word[::-2])-->prints every alternate character in reverse order

print(word[::-1])-->prints the string in reverse order

print(word[::-1]) → print in reverse order

### Activity

Print extension of the filename

name = 'Youtube.txt'

print(name[-3:])

Print only filename

name = 'Youtube.txt'

print(name[:-3])

Printing only protocol in url

url = 'https://google.com'

print(url[:5])

Print only domain

url = 'https://google.com'

print(url[6:])

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

##### **type()** → which return datatype of the variable or an object

word = "Hello World"

print(type(word))

print(type(url))

##### **len()** → returns the length of the string

print(len(word))

print(len(url))

##### **dir()** → returns the list of attributes that attached to the object

print(dir(word))

##### **help()** → return information about attributes

print(help("Hello".upper))

print(help("Hello".split))

### **string methods**

##### **lower()** → converts the string into lowercase

Syntax: string.lower()

s = "PyTHon ProgRAMMING iS Fun"

print(s.lower())

word = "Hello World"

print(word.lower())

##### **upper()** → converts the string into uppercase

syntax:string.upper()

s = "PyTHon ProgRAMMING iS Fun"

print(s.upper())

word = "Hello World"

print(word.upper())

print(s.upper(),end=(""))

print(word.upper())

print(s.upper(), word.upper())

##### **swapcase()** → converts lowercase into uppercase and vice versa

syntax:string.swapcase()

student = "His RoLL Number iS SomeThing"

print(student.swapcase())

s = "PyTHon ProgRAMMING iS Fun"

print(s.swapcase())

##### **count()** → returns the number of occurrences of a substring in original string

syntax:string.count("substring",si,ei)

my\_message = "Hello world, Hello Universe"

print(my\_message.count("l"))

print(my\_message.count("Hello",0,10))

### find() and rfind()

##### **find()** → searching the string for a specified value and return the position where it is found

→ return -1 if the value is not find

→ it give the first occurrences of the value

syntax: string.find(value,si,ei)

my\_message = "Hello World"

print(my\_message.find("l"))

s = "Today is beautiful day"

print(s.find("day"))

##### **rfind()** → it return the last position of the value

my\_message = "Hello World"

print(my\_message.rfind("l"))

s = "Today is beautiful day"

print(s.rfind("day"))

##### **index()** → searching the string for a specified value and return the position where it is found

→ it give the first occurrences of the value

→ return the value error if the value is not found

syntax: string.index(value,[start],[end])

##### **rindex() →** it return the last position of the value

my\_message = "Hello World"

print(my\_message.index("l"))

print(my\_message.rindex("l"))

# print(my\_message.index("m"))

##### **replace()** → replace specifies string with another specified string

syntax:string.replace(oldvalue,newvalue,[count])

s1 = "Malayalam"

print(s1.replace("a","q"))

print(s1.replace("a","q",1))

s3 = "Hello World"

print(s3.replace("World", "universe"))

print(s3.replace("Hello", "World"))

##### **startswith()** → returns true if the string starts with the specified value

syntax = string.startswith(value,si,ei)

sentence = "how are you are you fine"

print(sentence.startswith("how"))

print(sentence.startswith("world"))

##### **endswith()** → returns true if the string ends with the specified value

syntax = string.endswith(value,si,ei)

sentence = "how are you are you fine"

print(sentence.endswith("fine"))

print(sentence.endswith("you"))

##### **spilt()** → split the string at the specified separator and return list

syntax: string.split([separator],[maxsplit])

sentence1 = "This is my string"

print(sentence1.split("s"))

print(sentence1.split(" "))

print(sentence1.split("T"))

##### **strip()** → removes leading and trailing(spaces at the end)character

##### **rstrip()** → removes the trailing(spaces at the end) characters

my\_string = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Hello

world===================="

print(my\_string.rstrip("="))

##### **lstrip()** → removes the leading space character

my\_string = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Hello

world===================="

print(my\_string.lstrip("\*"))

my = "python Hello world programming "

print(my.rstrip("programming"))

##### **isalnum()** → Return True if all the characters in the string are alphanumeric

sentence4 = "world2345"

print(sentence4.isalnum())

sentence5 = "world2345.7"

print(sentence5.isalnum())

##### **isalpha()** → Return True if all the character in the string are alphabet

##### **isdigit()** → returns True if all the characters in the string are digits

##### **islower()** → returns True if all the characters in the string are lower

##### **isupper()** → returns True if all the characters in the string are upper

##### **isspace()** → returns True if all the characters in the string are spaces

print(sentence4.isspace())

s9 = " "

print(s9.isspace())

### **f-strings:formatted strings**

→ they provide a concise and readable syntax for strings

→ f-strings are prefixed with letter"f" or 'F' before opening the quotes

→ with in the string expressions are enclosed in curly braces"{}"

name = "Alice"

age = 30

result = f"My name is {name} and i am {age} years old"

print(result)

### **Activity**

my\_string = "apple, banana, orange, mango"

→ split a string into list of substring

→ construct f-string

x = 10

y = 20

result = x + y

result1=f"the result of{x} and {y} is {result}"

print(result1)

→ count the number of occurrences of the letter "i"

string = "python programming is fun and we like it"

print(string.count("i"))

# LISTS

List → collection of homogeneous and heterogeneous

##### Homogeneous → all items are same data types

##### Heterogeneous → it can accommodate different data types

→ we are checking here why strings are homogeneous

→ checking why lists are both hetero and homogeneous

→ elements in list are ordered

→ separate it by comma

→ list can hold duplicate values

→ it is mutable datatype

→ Boundary = [\_\_\_]

syntax:var\_name = [ele1, ele2, ele3,------]

##### length of a list= len(var\_name)

*different ways to construct a string*

my\_list = []

my\_list = [1, 2, 3, 4, 5]

my\_list = list([1, 2, 3, 4, 5])

##### **Indexing a list**

→ process of extracting single element at a time

→ indexing can be positive(starts with 0) or negative(starts with -1)

→ it can’t be print in reverse of index

syntax = var\_name[index]

my\_list = [10, 1.25, "hello", [1, 2, 3, "hai"], 4 + 6j]

# 0 1 2 3

print(my\_list[0]) → 10

print(my\_list[3]) → [1, 2, 3, "hai"]

print(my\_list[3][2]) → 3

print(my\_list[2][3]) → l

print(my\_list[4][2]) → typeError: 'complex' object is not subscriptable

print(my\_list[1][1]) → typeError: 'float' object is not subscriptable

##### **slicing a list →** process of extracting multiple elements at a time

Syntax:var\_name[startindex:endindex:stepvalue]

→ si - default value i s "0"

→ ei - default value length of the list

→ sv - default value is 1

print(word[::2]) → prints every alternate character

print(word[::-2]) → prints every alternate characters in

###### **reverse order**

print(word[::-1]) → prints the list in reverse order

→ note: elements at the end index will not be added in the slicing

### **activity**

-7 -6 -5 -4 -3 -2 -1

names = ["apple", "google", "yahoo", "amazon", "facebook", "instagram", "microsoft"]

0 1 2 3 4 5 6

→ reverse the list

print(names[::-1])

→ what is the output

print(names[2])

print(names[2][3])

→ what is the output

print(names[-2:3]) → it will not iterate in reverse order

print(names[-6:5])

print(names[1 + 4])

print(names[::2])

print(names[::-2])

### **list methods**

→ adding elements in the list we have three methods

##### **append()** → Adding the element at the end of the list(both individual and collection data types)

→ element can be any data type

syntax: list.append(element)

l = [1,2]

l.append([3,4])

print(l)

l.append(["google","microsoft"])

print(l)

l.append(3, 4) → TypeError: list.append() takes exactly one

argument (2 given)

print(l)

l.append(3)

print(l)

##### **extend()** → extends the existing list with the items of the given sequence

→ it will take only collection data types

syntax: list.extend(iterable)

a= [1,2]

a.extend([3,4])

print(a)

a.extend(3)

print(a) → TypeError: 'int' object is not iterable

a.extend(["google"])

print(a)

##### **insert()** → add an element at the specified position(both individual and collection data type)

syntax:list.insert(position,element)

list = ["google", "microsoft", "apple", "akamai"]

list.insert(0, "facebook")

print(list)

list.insert(12, "Tcs")

print(list)

list.insert(1,[1,4.5,"python", (2,3)])

print(list)

\*\*\*\*\* removing elements from list \*\*\*\*\*\*\*

##### **pop()** → removes the element at the specified position/

syntax:list.pop([pos])

→ by default pop() removes and returns the last element in the list

→ returns removed value

list1 = ["google", "microsoft", "apple", "akamai"]

print(list1.pop(0))

print(list1)

print(list1.pop())

print(list1)

print(list1.pop(12)) → IndexError: pop index out of range

##### **remove()** → Removes the first occurrences of the element specified

syntax: list.remove(element)

→ if the value is not present--> value error

names = ["john", "bob", "steve", "john", "steve"]

names.remove("john")

print(names)

names.remove("hannah")

print(names)

##### **clear()** → it is used to clear the entire list without deleting the list

syntax: list.clear()

name1 = ["john", "bob", "steve", "john", "steve"]

name1.clear()

print(name1)

##### **del keyword →** if you want to delete particular value and access with index

name =["apple", "banana", "grapes", "mango", "kiwi"]

# 0 1 2 3 4

del name[0]

print(name)

del name[1:4]

print(name)

del name[::2]

print(name)

##### **copy()** → returns the copy of the list

syntax:list\_var.copy()

##### **shallow copy** → In this method the element of the list will copied to the new list as it is and memory locations will be different

→ unless the main has a nest list

→ without nested list

a = [1, 2, 3, 4, 5]

b = a.copy()

print(a)

print(b)

print(id(a))

print(id(b))

→ with nested list

c = [6, 7, 8,"python", ["hello", "hai"]]

d = c.copy()

print(c)

print(d)

print(id(c))

print(id(d))

print(id(c[3]))

print(id(d[3]))

##### **Deepcopy()** → in this method the element of the main list will be copied to the new list as it is

→ memory address will be different even for the nested list

→ we need to import deepcopy()

syntax: from copy import deepcopy

list = deepcopy()

from copy import deepcopy

num = [1, 2, 3, [4, 5, "hai"]]

num2 = deepcopy(num)

print(num)

print(num2)

print(id(num))

print(id(num2))

print(id(num[3]))

print(id(num2[3]))

##### **sort() →** sort the list

→ in order to sort the list ,a list should be homogeneous

→ modifies the original list it self

syntax:list.sort(key = function,reverse = True)

names2 = ["apple", "google", "yahoo", "amazon", "facebook"]

names2.sort() → gives output based on ASCII Value:

print(names2)

char = "A"

print(ord(char))

char1 = "a"

print(ord(char1))

char2 = "z"

print(ord(char2))

char3 = "@"

print(ord(char3))

names2.sort(reverse = True)

print(names2)

l2 = [1, 2, 6, 9, 3, 1, 0]

l2.sort(reverse = True)

print(l2)

names2.sort(key = len)

print(names2)

##### **custom sorting**

names2.sort(key = len,reverse = True)

print(names2)

##### Merge two lists

l1 = [1, 2, 3]

l2 = [4, 5]

print(l1 + l2) → [1, 2, 3, 4, 5] → correct method

print(\*l1, \*l2) → 1, 2, 3, 4, 5 → not correct method

# TUPLE

### Tuple → tuples are immutable data type

→ enclosed with () or parenthesis

→ tuple is both homogeneous and heterogeneous

→ ways to construct tuple

t = ()

t = tuple()

t = (1,)

t = (1, 2, 3, "hello")

### **tuple methods**

print(dir(t))

##### **count()** → returns number of items a specified value occurs

syntax: tuple.count(value)

a = (1, 12, 4, 6, 7, 4, 7, 9, 10)

print(a.count(4))

print(a.count(7))

print(a.count(1))

##### **index()** → returns the first occurrences of the element specified

syntax:tuple.index(value)

a = (1, 12, 4, 6, 7, 4, 7, 9, 10)

print(a.index(4))

print(a.index(10))

print(a.index(11)) → value error

# SETS

→ sets are unordered data type

→ elements inside the sets are unique

→ sets are mutable ,but elements in the set must be immutable \*\*

→ sets cannot be indexed or sliced \*\*

→ Boundary: {\_\_\_\_}

syntax: var\_name = {1, 2.5, (1, 2, 3), "hai"}

##### **length of the set: len(var\_name)**

s = {1, "apple", (1, 2, 3)}

print(s)

print(dir(s))

### **sets methods**

##### **union()** → returns a set that contain all items from the original set and all items from the specified set

a = {1, 2, 3, 4}

b = {3, 4, 5, 6}

a.union(b)

print(a.union(b))

##### **update()** → it update the current set by adding items from any iterable

c = {1, 2, 3, 4}

b = {3, 4, 5, 6}

print(a.update(b))

print(a)

##### **intersection()** → returns a set that contains the similarity between two or more sets

a = {1, 2, 3, 4}

b = {3, 4, 5, 6}

print(a.intersection(b))

c = {3, 4, 5, 6, 7}

print(a.intersection(b, c))

##### **difference()** → returns a set containing the difference between two or more sets

→ the returns the set contains items that exist only in the base set but not in other set

a = {1, 2, 3, 4}

b = {3, 4, 5, 6}

print(a.difference(b))

print(a - b)

##### **difference\_update()** → update the base set with the elements that are present in base set but not in other set

t1 = {1, 2, 3}

t2 = {2 }

print(t1.difference\_update(t2))

print(t1)

##### **symmetric\_difference()** → returns a set that contains all items that are not common among all the sets

t3 = {1, 2, 3, 4}

t4 = {3, 4, 5, 6}

print(t3.symmetric\_difference(t4))

t5 = {1, 2, 3, 4, "hello", (1, 2, 3)}

t6 = {10, 20, 30, 4}

print(t5.symmetric\_difference(t6))

##### **symmetric\_difference\_update()** → updates the base set with the items that are not common among all the sets

t5 = {1, 2, 3, 4, "hello", (1, 2, 3)}

t6 = {10, 20, 30, 4}

print(t5.symmetric\_difference\_update(t6))

print(t5)

### ***adding element to set***

##### **add()** → adds an element to the existing set, if the element already exist, the add() does not add the element

s1 = {"apple", "google", "microsoft"}

print(s1.add("facebook"))

print(s1)

### **remove element from the set**

##### **remove()** → removes the specified element from the set raises the keyerror if not present

s1 = {"apple", "google", "microsoft"}

print(s1.remove("google"))

print(s1)

# print(s1.remove("facebook"))

# print(s1) → key error

##### **discard()**:removes the specified element from the set

→ does not throw any error if the specified element is not present

s2 = { "yahoo", "sony", "tcs", "akamai"}

print(s2.discard("sony"))

print(s2)

print(s2.discard("apple"))

print(s2)

##### **pop()** → it removes random item

→ return the removed item

s3 = { "yahoo", "sony", "tcs", "akamai"}

print(s3.pop())

print(s3)

s4 = {}

print(s4.pop())

print(s4) → TypeError: pop expected at least 1 argument, got 0

##### **clear()** → removes all the elements from the set

t5 = {"apple", 1, (1, 2, 3, 4)}

print(t5.clear())

print(t5)

##### **issubset()** → returns True if all the items in the base set exits in the reference set otherwise returns false

x = {1, 2, 3}

y = {1, 2, 3, 19, 20}

print(x.issubset(y))

z = {2, 3, 5}

print(x.issubset(z))

##### **issuperset()** → returns true if all the items in reference set exists in the base set, otherwise it returns false

x = {1, 2, 3}

y = {1, 2, 3, 19, 20}

print(x.issuperset(y))

print(y.issuperset(x))

##### **isdisjoint()** → returns true if items are present in the both sets are uncommon, otherwise it returns false

x1 = {1, 2, 3}

x2 = {4, 5, 6}

print(x1.isdisjoint(x2))

x3 = {2, 3, 4}

print(x1.isdisjoint(x3))

# Dictionaries

### dictionary:

* collection of key-value pair
* each element is associated with unique key
* separated by comma operator
* boundary = {-----}
* it does not support indexing and slicing

syntax : var\_name = {key1: value1, key2: value2\_\_\_\_\_\_\_\_\_\_\_\_}

→ length:returns numbers of key present in the dictionary

→ len(dict\_varibale)

##### different ways to create a dictionary:

1. d = {} #empty dictionary
2. d = dict()

##### using dictionary constructor:

1. d = dict(Bangalore = 25, chennai = 35, delhi = 30)
2. d = dict([("Banglore",25), ("chennai",35), ("delhi", 30)])
3. d = dict({"bangalore" : 25, "chennai": 30, "delhi": 20})

#### *characteristics of dictionary*

1 → key cannot be duplicate

2 → keys should be single element

3 → values can be of any datatype

4 → values can be accessed through keys only

5 → keys must be immutable data types

##### composite keys: → dictionaries can have composite keys i.e tuple as keys

student\_grades = {("alice", "math"): 85, ("alice", "science"): 90}

#### *Accessing values from a dictionary*

##### get() → it returns value of the item with the specifies key

d = {"Banglore" : 25, "Chennai": 35, "Delhi":30}

print(d.get("Banglore"))

print(d.get("india"))#none

print(d.get("india", "the key is not found in the dict"))#the key is not found in the dict

#### *Adding/updating the dictionary with new key and value pair*

using key and value syntax: d[key] = value

d = {"Banglore" : 25, "Chennai": 35, "Delhi":30}

d["mysore"] = 80

print(d)

##### update() → updates the existing dictionary with specified item

d.update(noida = 26)

print(d)

d.update({"vizag": 34, "chochin": 28})

print(d)

##### setdefault() → returns the value of the item with the specified key(keyname, value)

→ if value is not specified → it add’s ‘none’ as a value

d = {"Banglore" : 25, "Chennai": 35, "Delhi":30}

print(d.setdefault("kerala"))

print(d)

print(d.setdefault("kolkata", 60))

print(d)

#### *constructing a dictionary from other iterable*

##### iterable: (collection data types) containing multiple elements

##### fromkeys() → default value will be none

syntax:dict.fromkeys(keys,value)

names = ["apple", "google", "yahoo", "gmail", "google", "apple"]

print(dict.fromkeys(names,0))

print(dict.fromkeys(names,1))

print(dict.fromkeys(names))

##### items():returns the key-values pairs of dictionary as a tuple in a list

syntax:dictionary.items()

my\_dict = {"a": 1, "b": 2, "c": 3}

print(my\_dict.items())

print(("b", 2) in my\_dict.items())#checking specified key-value pair exists in the dictionary

##### keys():it returns a list of all keys present in the dictionary

syntax:dictionary.keys()

d1 = {"bangalore": 25, "kolkata": 26}

print(d1.keys())

##### values():returns list of all the values

syntax: dictionary.values()

d1 = {"bangalore": 25, "kolkata": 26}

print(d1.values())

#### *deleting the key and value*

##### pop():removes the specified items in the dictionary returns removed values as output

syntax:dictionary.pop(keyname, defaultvalue)

employee = {"name": "Ram", "age": 30}

print(employee.pop("name"))

print(employee)

print(employee.pop("python"))#key error

##### popitem() → removes the item that was inserted at the last into the dictionary

→ returns the removed key-value pair as tuple

syntax:dictionary.popitem()

names1 = {"name": "Ram", "age": 30, "salary": 2000}

print(names1.popitem())

print(names1)

##### clear() → if you want to clear entire dictionary

print(names1.clear())

print(names1)

##### merging the dictionaries

d1 = {"fname": "steve", "lname": "jobs"}

d2 = {"age": 56, "company": "apple"}

d3 = {\*\*d1, \*\*d2}

print(d3)

d4 = d1 | d2

print(d4)

#### *sequence and iterable:*

→ sequence is an object which can be indexed

eg: list, tuple, string

→ **iterable:**(collection data types) containing multiple elements

→ all sequences are iterables but all iterable are not sequences

eg: sets and dictionaries(they are interable including string,list,tuples)

# Conditional statements

conditional statements in programming language decides the direction of flow of program execution

### if statement:

syntax: if(condition):

statement 1

statement 2

---------

statement n

a = 3

if a > 2:

print("enter the block")

### if-else condition:

syntax:

if(condition):

true block

else:

false block

b = int(input())

if b == 9:

print("equal")

else:

print("not equal")

### elif conditional statement:when we have to multiple condition

syntax:

if (condition):

statement

elif(condition):

statement

else:

statement

x = 11

if x > 10:

print(" x is greaterthan 10")

elif x < 10:

print("x is lessthan 10")

else:

print("x is equal to 10")

##### ord():returns ASCII values of the specified characters

print(ord("A"))

##### chr():returns character of specified ascii value

print(chr(82))

→ checking given number is even or odd

num = 16

if num % 2 == 0:

print("Number is even")

else:

print("Number is odd")

→ the greatest of two numbers

c = 10

d = 8

if c > d:

print(" c is the greater number")

else:

print("b is greater")

→ if the string is starting with vowel or not

string = "Aye!!!! hello world"

if string[0] in "aeiouAEIOU":

print("string is starting with a vowel")

else:

print("string is not starting with a vowel")

→ check if the given value is a sequence or not

value = "hello"

if isinstance(value, (str, list, tuple)):#isinstance check if the object "value" is an instance(occurred)

print("value is a sequence")

else:

print("value is not a sequence")

→ given value is an iterable (collection data type)

sentence = True

if isinstance(sentence,(str, list, tuple, set, dict)):

print("sentence is an iterable")

else:

print("sentence is not an iterable")

→ iterable have even number of elements inside them or no

s = {"hello", (1,2), 10, 20, 30}

a = 1

b = 2

c = 3

if (a>b) and (a>c):#

print("a is greater")

elif (b>c) and (b>a):

print("b is greater")

else:

print("c is greater")

→ key is present in dictionary

s = {"a": 1, (1,2): 2, "c" : 3}

key = (1,2)

if key in s:

print("key is present")

else:

print("key is not present")

→ *palindrome* or not--> (level) if you reverse same value should come

string = "level"

reversed\_string = string[::-1]

# print(string)

# print(reversed\_string)

if(string == reversed\_string): #if string == string[::-1]

print(f"{string} is a palindrome")

else:

print(f"{string} is not a palindrome")

→ if key is present --> update its value by 1

→ if key is not present--> create the key with 1 as its value

d = {"a": 1, "b": 2, "c": 3}

key = "x"

if key in d:

d[key] = d[key] + 1 #d[key] += 1 # if i take key as "a"

d[a] = 2

else:

d[key] = 1 #d[x] = 1 if i take key as "x"

print(d)

# LOOPs

### while loop:

it is used when the number of iterations(repetition of process)to be done is not know

→ steps to follow:

1. keep a track of number of iterations with a reference

variable

2. manually increment/decrement the reference variable

syntax: while condition:

statements

count = 1

while count <= 5:

print(count, end = " ") #1, 2, 3, 4, 5

count += 1

print the numbers from 1 to 10

start = 1

end = 10

while start <= end:

print(start)

start += 1

print "hello word" 5 times

start = 1

while start <= 5:

print("hello world")

start += 1

→ print numbers from 10 to 1

start = 10

end = 1

while start >= end:

print(start)

start -= 1

→ print even numbers from 1 to 10

start = 1

end = 10

while start <= end:

if start % 2 == 0:

print(start)

start += 1

→ iterate over string

s = "hello"

index = 0

while index <len(s): #0<5, 1<5, 2<5, 3<5, 4<5,

print(s[index])

index += 1

### For loop

* it is used when the number of iterations to be done is known
* there is no need to keep a track of iterations , for loop does it implicitly
* no need to increment/decrement manually
* here in for loop we use range() range(start,end,step)

##### range(): this function is used to generate a sequence of numbers.

for i in range(5):

print(i)

for j in range(2, 8):

print(j)

for num in range(1, 10, 2):

print(num)

syntax: for ref\_var in range(si,ei,step):

statements

syntax: for ref\_var in iterable:

statements

→ i want to print "hello world" 5 times

for start in range(0, 5):

print("hello world")

→ print numbers from 1 to 10

for num in range(1, 11):

print(num)

→ print numbers from 10 to 1

for num in range(10,0,-1):

print(num)

→ print numbers from -1 to -10

for num in range(-1,-11,-1):

print(num)

→ print numbers from -10 to -1

for num in range(-10, 0):

print(num)

→ print even numbers from 1 to 20

for num in range(1, 21):

if num % 2 == 0:

print(num)

→ print odd numbers from 10 to 20

for num in range(10, 21):

if num % 2 != 0:

print(num)

s = "hello world"

for index in range(0, len(s)):

print(s[index]) #s[0] = h, s[1],s[2]

l = [10, 20, 30, 40, 23]

for index in range(0, len(l)):

print(l[index])

→ get index value pair

s = "python"

for index in range(0, len(s)):

print(index, s[index])

for index in s:

print(index) #p y t h o n

l = [10, 20, 30]

for index in range(len(l)):

print(index, l[index])

for element in l:

print(element)

### Activity

Activity

1. print numbers from -10 to -1

act1 = -10

while act1 <= -1:

print(act1)

act1 -= -1

0utput :

-10

-9

-8

-7

-6

-5

-4

-3

-2

-1

2. print numbers from -1 to -10

act2 = -1

while act2 >= -10:

print(act2, end=" ")

act2 += -1

Output:

-1

-2

-3

-4

-5

-6

-7

-8

-9

-10

3. iterate over list

act3 = [10, 20, 30, 40, 50]

index = 0

while index < len(act3):

print(act3[index])

index += 1

Output:

10

20

30

40

50

4. iterate over tuple

act4 = (10, 20, 30, 40, 50)

index = 0

while index < len(act3):

print(act3[index])

index += 1

Output:

10

20

30

40

50

5. Get index value

act5 = ("hi", 10, 12, "true")

for tup\_index in range(len(act5)):

print(act5[tup\_index])

Output:

hi

10

12

true

6. get index value pair

act6 = (1, 2, 3, 4, 5)

for tup\_index in range(len(act6)):

print(tup\_index, act6[tup\_index])

Output:

0 1

1 2

2 3

3 4

4 5