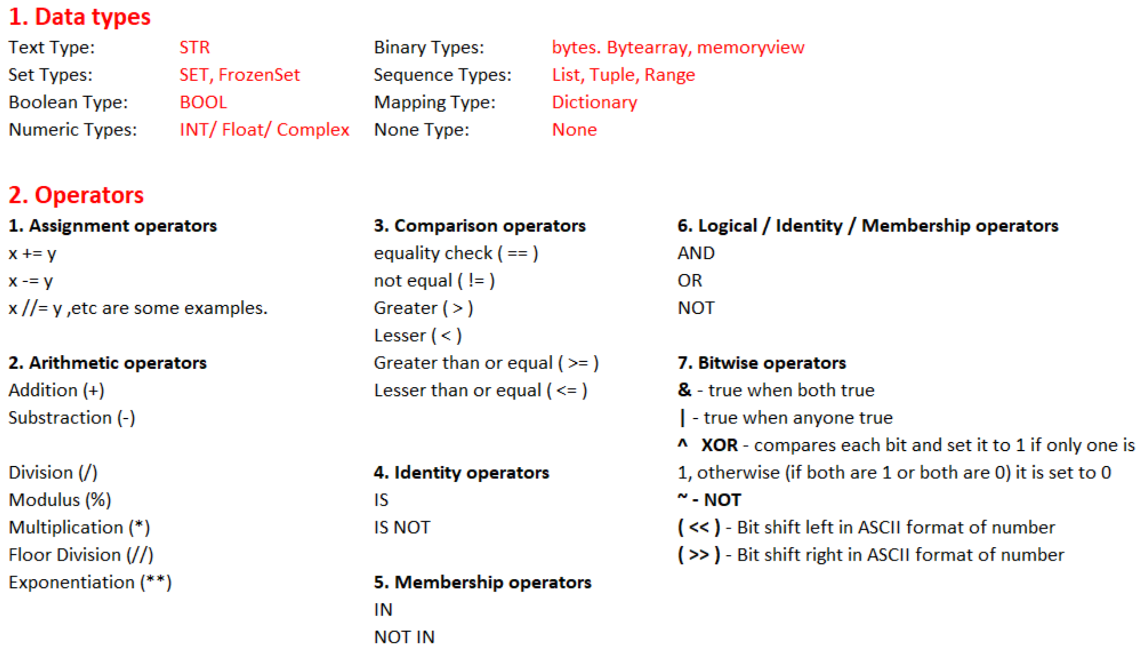


1. Data types

Text Type:	STR	Binary Types:	bytes. Bytearray, memoryview
Set Types:	SET, FrozenSet	Sequence Types:	List, Tuple, Range
Boolean Type:	BOOL	Mapping Type:	Dictionary
Numeric Types:	INT/ Float/ Complex	None Type:	None

2. Operators

1. Assignment operators
x += y
x -= y
x //= y ,etc are some examples.

2. Arithmetic operators
Addition (+)
Substraction (-)

Division (/)
Modulus (%)
Multiplication (*)
Floor Division (//)
Exponentiation (**)

3. Comparison operators
equality check (==)
not equal (!=)
Greater (>)
Lesser (<)
Greater than or equal (>=)
Lesser than or equal (<=)

4. Identity operators
IS
IS NOT

5. Membership operators
IN
NOT IN

6. Logical / Identity / Membership operators
AND
OR
NOT

7. Bitwise operators
& - true when both true
| - true when anyone true
^ XOR - compares each bit and set it to 1 if only one is 1, otherwise (if both are 1 or both are 0) it is set to 0
~ - NOT
(<<) - Bit shift left in ASCII format of number
(>>) - Bit shift right in ASCII format of number

In [1]:

```python
# 1. Assignment operators
a = 12
b = 5
c = 'Pneumonoultramicroscopicsilicovolcanoconiosis'
d = 'tree'
```

In [2]:

```python
# 1. Arithematic operators examples
# addition
print(a+b, c+d)

# exponentiation('a' raised to 'b' power)
print(a**b)

# division
print(a/b)

# Modulus
print(a%b)

# floor division
print(a//b)
```

```
17 Pneumonoultramicroscopicsilicovolcanoconiosistree
248832
2.4
2
2
```

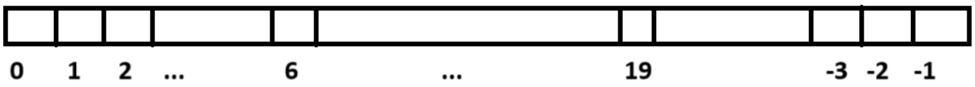
3. Inbuilt Data Structures

Following are the builtin data structures in python which can be classified in 2 classes as :

A. Ordered Data Structures

1. Strings

- **indexing**



- **slicing** - slices the string based on index.
  slice(stop)
  slice(start, stop, step)
- **stripping**
  - lstrip()
  - rstrip()
- **split()** - splits the string based on a particular character.
- **count()** - returns the number of times a specified value appears in the string.

In [3]:

```python
# slicing
print(c[-15:])

# Accessing mainstring till 10th place & stepvalue 3.
print(c[:10:3])

# reversing string
print(c[::-1])

#checking lenght of a string
print(len(c))

# checking count of a particular substring with main string object.
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple", 10, 24)

print(x)
```

```
volcanoconiosis
Punl
sisoinoconaclovociliscipocsorcimartluonomuenP
45
1
```

2. Lists

- declared using squared brackets[ ] or explicit type defining.
- Changable/ Mutable in nature, allows duplicate values of mixed data types.
- dtype : <class 'list'>
- List function : https://www.w3schools.com/python/python_ref_list.asp (https://www.w3schools.com/python/python_ref_list.asp)

3. Tuples

- declared using round brackets ( ) or explicit type defining.
- unchangable/ Imutable in nature, allows duplicate values of mixed data types.
- dtype : <class 'tuple'>
- Tuple functions : https://www.w3schools.com/python/python_ref_tuple.asp (https://www.w3schools.com/python/python_ref_tuple.asp)

List operations

In [4]:

```python
# declaration
L1 = [True, "fish", 28, 6]
L2 = [14, 1, 19]

# takes 1 element and adds the same at end of appeneded list.
L1.append([1, 2, 3])

'''takes 1 argument and adds same to end of extended list only
difference here is it unpacks internal elements of taken argument'''
L1.extend([4, 5, 6])
print("L1=", L1)

# remove function
L1.remove(28)

# pop function
L1.pop(3)

# delete operation
del L2[0]

print(L1, L2)

L2.clear()
print(L2)

L3 = [73, 55, 42, 14, 82]
L5 = [19, 54, 3, 10, 91]

# normal copying
L4 = L3
L3[1] = 6
print('L3 is',L3,'L4 is', L4)

# shallow copying
L4 = L3[:]
L3[1] = 15
print('L3 is', L3, 'L4 is',L4)
```

```
L1= [True, 'fish', 28, 6, [1, 2, 3], 4, 5, 6]
[True, 'fish', 6, 4, 5, 6] [1, 19]
[]
L3 is [73, 6, 42, 14, 82] L4 is [73, 6, 42, 14, 82]
L3 is [73, 15, 42, 14, 82] L4 is [73, 6, 42, 14, 82]
```

Tuple operations

In [5]:

```python
'''All operations perform same as List in tuples
too lets discuss some that are specific to tuple'''
# Packing & Unpacking
stud1 = (19, 82, 53, 24)
(roll_no, art, maths, science) = stud1
print(art)

stud2 = (5, 60, 57, 48, 89)
(roll_no, art, maths, science) = stud2
# unpacking elements require same number of arguments as in input
print(art)

'''error : ValueError - too many values to unpack'''
```

```
82

---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_15344\3345782133.py in <module>
      7
      8 stud2 = (5, 60, 57, 48, 89)
----> 9 (roll_no, art, maths, science) = stud2
     10 # unpacking elements require same number of arguments as in input
     11 print(art)

ValueError: too many values to unpack (expected 4)
```

B. Unordered Data Structures

4. Sets

- declared using curly brackets { } or explicit type defining.
- Changable/ Mutable in nature, does not allow duplicate values but allows mixed data types.
- dtype : <class 'sets'>
- SET functions : https://www.w3schools.com/python/python_ref_set.asp (https://www.w3schools.com/python/python_ref_set.asp)

5. Dictionary

- declared using curly brackets{} or explicit type defining with key:values.
- Keys are as an index and are Immutable, whereas values are mutable.
- dtype : <class 'Dictionary'>
- Dictionary functions : https://www.w3schools.com/python/python_ref_dictionary.asp (https://www.w3schools.com/python/python_ref_dictionary.asp)

Sets operations

In [6]:

```python
S1 = set(('apple', 'banana', 'cherry'))
S2 = {1, 2, 5, 6}

# adding new elements to sets
S1.add('orange')
S1.update(S2)
print(S1)

# removing items from set
S1.remove(1)
S1.discard('cherry')
S1.pop()
# nothing to mention in paranthesis it pops top/first element
print(S1)

S1.clear()
print(S1)

#completely deletes the set
del S1
print(S1)

''' #error : NameError--name 'S1' is not defined
-(S1 is deleted completed )'''
```

```
{'cherry', 1, 2, 5, 6, 'banana', 'orange', 'apple'}
{5, 6, 'banana', 'orange', 'apple'}
set()

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_15344\752527326.py in <module>
     19 #completely deletes the set
     20 del S1
---> 21 print(S1)
     22
     23 ''' #error : NameError--name 'S1' is not defined

NameError: name 'S1' is not defined
```

Dictionary operations

In [7]:

```python
Dict = {'A':1, 'B':2, 'C':3, 'D':4}

# extracting items.....
print(Dict.keys())
print(Dict.values())

# changing values for a key
Dict['A']= 19

#Inserting new Key value pair
Dict['E']=5

# deleting key-value
del Dict['C']

print(Dict)
```

```
dict_keys(['A', 'B', 'C', 'D'])
dict_values([1, 2, 3, 4])
{'A': 19, 'B': 2, 'D': 4, 'E': 5}
```

C. User-Defined OR Derived Data Structures

Arrays, Stack, Queue, Trees, Linked Lists, Graphs, HashMaps

4. Control Statements

In [8]:

```
# 1. if loop  ************************
if <condition> :
    <<statement_1>>

# 2. if...else loop ******************
if <condition> :
    <<statement_1>>

elif <condition_2> :
    <<statement_2>>

else:
    <<statement_1>>

# 3. for loop ************************
for in range(start, end, step):
    << statement >>

# 4. while loop **********************
while (condition_1):
    << statement_1 >>

# 5. switch case *********************
def one():
    return "one"
def two():
    return "two"
def three():
    return "three"
def default():
    return "no spell exist"

numberSpell = {
    1: one,
    2: two,
    3: three
    }
def spellFunction(number):
    return numberSpell.get(number, default)()

print(spellFunction(3))
print(spellFunction(10))
```

```
  File "C:\Users\hp\AppData\Local\Temp\ipykernel_15344\982656547.py", line
2
    if <condition> :
       ^
SyntaxError: invalid syntax
```

5. Programming concepts

1. Functions

In [9]:

```
def my_function():
    print("Hello from a function")

my_function()
```

```
Hello from a function
```

2. Lambda Functions

In [10]:

```
x = lambda a, b: a * b
print(x(5, 6))
```

```
30
```

3. Comprehensions

Generator Comprehensions - are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don’t allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient. Let’s look at the following example to understand generator comprehension:

In [11]:

```
# -------------------------- List comprehensions
list_using_comp = [var**2 for var in range(1, 10)]
print("Output List comprehension:", list_using_comp)

# -------------------------- Dictionary comprehension
input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}
print("Output Dictionary comprehensions:",dict_using_comp)

# -------------------------- Set comprehension
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:",set_using_comp)

# -------------------------- Generator comprehension
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_gen = (var for var in input_list if var % 2 == 0)
print("Output values using generator comprehensions:", end = ' ')
for var in output_gen:
    print(var, end = ' ')
```

```
Output List comprehension: [1, 4, 9, 16, 25, 36, 49, 64, 81]
Output Dictionary comprehensions: {1: 1, 3: 27, 5: 125, 7: 343}
Output Set using set comprehensions: {2, 4, 6}
Output values using generator comprehensions: 2 4 4 6
```

4. Map

In [12]:

```python
# example_1
def myfunc(a):
    return len(a)
x = map(myfunc, ('apple', 'banana', 'cherry'))
print(list(x))

# example_2
countries = ['india', 'auSTralia', 'JaPaN', 'Iceland']
res = list(map(lambda x: x.upper(), countries))
print(res)
```

```
[5, 6, 6]
['INDIA', 'AUSTRALIA', 'JAPAN', 'ICELAND']
```

5. Filter

In [13]:

```python
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
    if x < 18:
        return False
    else:
        return True

adults = filter(myFunc, ages)
for x in adults:
    print(x)
```

```
18
24
32
```

6. Reduce

In [14]:

```python
from functools import reduce
'''example_1: Using the Reduce function, concatenate a list of
words in input_list and print the output as a string.'''

input_list = ['All','you','have','to','fear','is','fear','itself']
string1 = str(reduce(lambda x,y: x + " " +y, input_list))
print(string1)
```

```
All you have to fear is fear itself
```

In [15]:

```python
# example_2: Create a list of numbers
nums = [1, 2, 3, 4, 5]

#Use reduce() with a lambda function to find
#the product of all numbers in the list

result = reduce(lambda x, y: x * y, nums)
print(result)
```

```
120
```

6. Object Oriented Programming

a) Classes & Objects

Object is any entity in real world variable a is an object a string "str" storing value "Hello world" is also an object similarly can store an integer or float values or even complex data structures.

Class is a blueprint on which instances of same class are bind and used. basically a class is a set of rules which ae followed by objects including variable and functions also class defines the attributes(behaviour).

In [16]:

```python
class Fruit:
    def __init__(self):
        self.name = "apple"
        self.color = "red"
# ------- scope of class declaration was only untill here

# ------ creating and assigning instances/objects to class
# driver code
my_fruit = Fruit()
my_fruit.color = "green"
my_fruit.name = "kiwi"

print(my_fruit.color)
print(my_fruit.name)
```

```
green
kiwi
```

Constructors

Constructors are generally used for instantiating an object the main task for constructor is to initialize / assign values to the data memeber of class whn an object is created and are of 2 types.

- **Defaut contructor** A simple constructor with no arguments, it only has a single default argument which is a referrence to the instance being constructed.
- **Parameterized constructor** A constructor is with arguments where first one is always taken as reference to instance being constructed known as self and rest of arguments are provided by coder.

In [17]:

```
class Fruit:
    def __init__(self, name, clr):
        self.name = name
        self.color = clr
# parameterized contructor example a better approach

# creating and assigning instances/objects to class
apple = Fruit("apple", "red")
banana = Fruit("banana", "yellow")

print(apple.name)
print(banana.color)
```

```
apple
yellow
```

Init method & self parameter

Init_method

- is a special method in a class.
- Automatically executed with every new class instance(object).
- **`__init__`** is a reserved keyword, programmers cannot use it thus this method does not needs calling to execute.

self_parameter

- self is a reference to current instance of class and is used to access variables belonging to class.
- It does not necessarily have to be word self user can use any name for it.

In [18]:

```
 1  class Students:
 2      def __init__(self, roll, name):
 3          self.roll_no = roll
 4          self.name = name
 5          house = "Yellow"
 6      def details(self):
 7          print(roll+ "," +name+" is from " +house+ " house")
 8  
 9  stu1 = Students(28, 'Ram')
10  stu2 = Students(19, 'Amit')
11  stu1.details()
12  stu2.details()
13  
14  """As we know we are going to get 'NAME' error for all the variables
15  as we havent mentioned the keyword "self"before them in Line 7,
16  Two possible ways to solve the error.
17  
18  - Making house variable an attribute by adding 'self' keyword to it
19  also dont forget to add 1 more argument parameter for house
20  - Moving house variable from 'init' to 'details' method """
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15344\429922766.py in <module>
      9 stu1 = Students(28, 'Ram')
     10 stu2 = Students(19, 'Amit')
---> 11 stu1.details()
     12 stu2.details()
     13 

~\AppData\Local\Temp\ipykernel_15344\429922766.py in details(self)
      5         house = "Yellow"
      6     def details(self):
----> 7         print(roll+ "," +name+" is from " +house+ " house")
      8 
      9 stu1 = Students(28, 'Ram')

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

In [19]:

```
# Solution 1
class Students:
    def __init__(self, roll, name):
        self.roll_no = roll
        self.name = name
        self.house = "Yellow"

    def details(self):
        print(str(self.roll_no) + ", " + self.name + " is from "
              + self.house + " house")

stu1 = Students(28, 'Ram')
stu2 = Students(19, 'Anjaneya')
stu1.details()
stu2.details()
```

```
28, Ram is from Yellow house
19, Anjaneya is from Yellow house
```

In [20]:

```
# Solution 2
class Students:
    def __init__(self, roll, name):
        self.roll_no = roll
        self.name = name

    def details(self):
        house = "Yellow"
        print(str(self.roll_no) + ", " + self.name + " is from "
              + house + " house")

stu1 = Students(28, 'Ram')
stu2 = Students(19, 'Anjaneya')
stu1.details()
stu2.details()
```

```
28, Ram is from Yellow house
19, Anjaneya is from Yellow house
```

Access Specifiers

Just like other OOP supporting languages Python also supports Access Specifiers to enable access restriction rules to a certain extent.
The access specifiers supported by python is as follows:

1. **Public** Data members without any preventions of access to any method or object are called public and declared using without any underscore.
   ex . self.name=name
2. **Protected** Data members declared as protected can only be accessible in current class and derived subclasses, We use a single underscore **"_"** in the beginning of their variable name to declare them as private.
   ex. self._name=name
3. **Private** Variabes accessible only by the present class are **private** declared using double underscore **"__"** before the variable name.
   ex. self.__name=name.

Functions & Methods in OOP

- **Functions** : are lines of code that can be used multiple times by simply calling independently.
- **Methods** : on other hand is a set of code simillar to functions but can only be inside the class instance - object.
  - Unlike funxctions, methods cannot work with zero parameters It atleast has 1-parameter(self).
  - Method is a concept of OOP.

Functions	Methods
Defined outside class	Defined inside class
can be executed just be calling name	cannot execute on itself needs object
Has '0' zero parameters	Need atleast 1 parameter(self, cls)
Cannot modify class attributes	can modify but is dependent on classes & objects

Methods in python can be of 3 types

- **Instance methods** A method that is used by using an object(instance) of a certain class.
- **Class methods** declared using a decorator "@classmethod", defining within the class but outside `__init__` hence does not uses "self" / instances.
- **Static methods** declared using a decorator "@staticmethod", a static method is bound to a class but not require an instance to run and execute as shown in the example below.

In [21]:

```python
class Student:
    school = 'TVB'                                              # stati
                                                                # & can
    def __init__(self, m1, m2, m3):                             # instar
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1 + self.m2 + self.m3)/ 3

    @classmethod                                                # class
    def getSchool(cls):                                         # begins
        return cls.school                                       # define

    @staticmethod                                               # class
    def info():                                                 # and do
        print("Hi there everyone... Lets learn python !!!")    # its co

s1= Student(44, 58, 21)
s2= Student(86, 52, 73)

print(s2.avg())                                                 # ----
print(Student.getSchool())                                      # ----
Student.info()                                                  # ----
```

```
70.33333333333333
TVB
Hi there everyone... Lets learn python !!!
```

b) Inheritance and Overriding

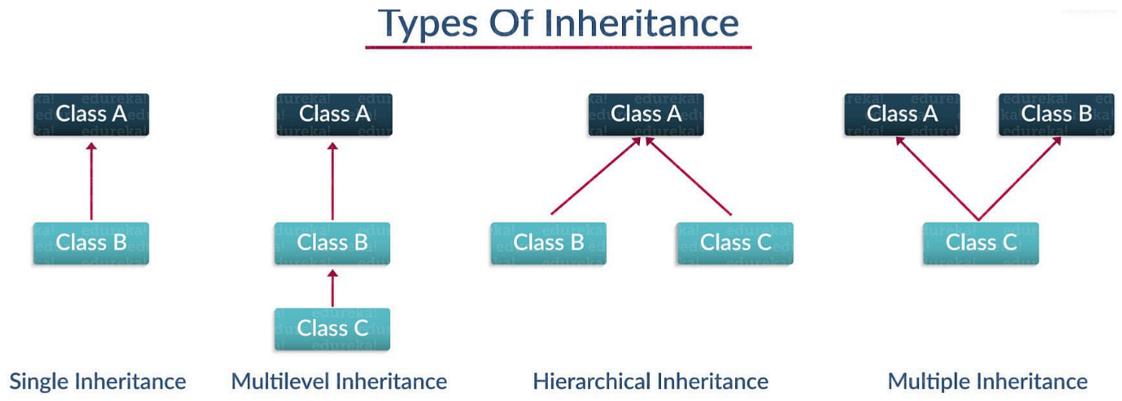
- Inheritance is powerfull feature in OOP, using this child class acquire properties of parent class.
- Positioning variables inside `__init__` method has an advantage i.e. we dont need to call it seperately as its auto-initialized by python using this we can use members / methods in parent class.

Super()
keyword in python is a builtin functionality & returns a proxy object of superclass that allows us to access methods of parent class.

Uses of super()

- Allows us to avoid using base class name explicitly.
- Working with multiple inheritance

Types of inheritance



In [22]:

```python
# example of inheritance and overriding...
import math

class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)

# Creating instances
rectangle = Rectangle(5, 4)
circle = Circle(3)

# Calculating area
print("Area of Rectangle:", rectangle.area())                 # Outpu
print("Area of Circle:", circle.area())                       # Outpu

```

```
Area of Rectangle: 20
Area of Circle: 28.274333882308138
```

Example involving in all concepts...

In [23]:

```
# Class with encapsulation, constructor ( __init__ method), and method
class Animal:
    def __init__(self, name):
        self.__name = name  # Encapsulation
    def make_sound(self):
        print("Animal sound")
    def display_name(self):
        print(f"Animal name: {self.__name}")
# --------------------------------------------------------------------
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.__breed = breed
    def make_sound(self):
        print("Bark")
    def display_details(self):
        print(f"Dog name:{self._Animal__name}, Breed:{self.__breed}")
# ------------------------------------------------- Abstraction and Exc
class Calculator:
    def add(self, num1, num2):
        try:
            result = num1 + num2
            return result
        except TypeError as e:
            print(f"Error: {e}")
            return None
# ------------------------------------------------- Access Specifiers
class MyClass:
    def __init__(self):
        self._protected_variable = 10
        self.__private_variable = 20

    def get_private_variable(self):
        return self.__private_variable
# ------------------------------------------------ Creating instances c
animal = Animal("Generic Animal")
dog = Dog("Buddy", "Golden Retriever")
animal.make_sound()
dog.make_sound()
animal.display_name()
dog.display_name()
dog.display_details()
# ------------------------------------------------- Abstraction and Exc
calculator = Calculator()
result = calculator.add(5, "10")                   # Output: Error: uns
print(result)                                      # Output: None
# ------------------------------------------------- Access Specifiers
my_instance = MyClass()
print(my_instance._protected_variable)             # Output: 10
print(my_instance.get_private_variable())          # Output: 20
```

```
Animal sound
Bark
Animal name: Generic Animal
Animal name: Buddy
Dog name:Buddy, Breed:Golden Retriever
Error: unsupported operand type(s) for +: 'int' and 'str'
None
10
20
```

Method Resolution Order (MRO)

In [24]:

```python
class Phone:
    def __init__(self):
        self.ver = 14
        self.summary()
    def summary(self):
        print("This is an Android Phone")

class MotoG32(Phone):
    def __init__(self):
        super().__init__()
        self.ver = 10
    def childsummary(self):
        print("This is an Android Phone".upper())

my_phn = MotoG32()
print("child class version:", my_phn.ver)
print("parent class version:", Phone().ver)
print(MotoG32.mro())
my_phn.childsummary()
```

```
This is an Android Phone
child class version: 10
This is an Android Phone
parent class version: 14
[<class '__main__.MotoG32'>, <class '__main__.Phone'>, <class 'object'>]
THIS IS AN ANDROID PHONE
```

c) Polymorphism

The Literal meaning of polymorphism is a condition of occurence in different forms. which refers to use of single type entity (method, operator or object) to represent different types depending on scenario.

types

- Operator polymorphism
- Functional polymorphism
- Class polymorphism

In [25]:

```python
# Polymorphism with Inheritance:
class Bird:
    def intro(self):
        print("There are many types of birds.")
    def flight(self):
        print("Most of the birds can fly but some cannot.")
class sparrow(Bird):
    def flight(self):
        print("Sparrows can fly.")
class ostrich(Bird):
    def flight(self):
        print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()
obj_bird.intro()
obj_bird.flight()
obj_spr.intro()
obj_spr.flight()
obj_ost.intro()
obj_ost.flight()
```

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

d) Encapsulation

Encapsulation is one of the most fundamental concept of object-oriented programming. In OOPs, we need to wrap more than one data type and method together. This type of wrapping is called encapsulation. Encapsulation puts some restrictions on data variables and methods to access directly and can prevent accidental change. For this, we use access specifiers which we have read earlier.

A class is an example of encapsulation in which we wrap some data types and methods together.

In [26]:

```python
'''Consider a real-life example of encapsulation, let assume there
is a Car with has a name, reg_no., owner’s name, mobile number.'''
class Car:
    def __init__(self):
        self.name='MG12'
        self.reg_no= 123
        self.ownername='kailash'
        self.mobile = 9237428321
```

e) Data Abstraction

It hides unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came. Data Abstraction in Python can be achieved by creating abstract classes.

f) Exception Handling

An error / exception is an event that disrupts the normal flow of an execution of the code. these exceptions can be of multipe types such as.

SyntaxError: Interpreter encounters a syntax error, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
TypeError: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
NameError: This exception is raised when a variable or function name is not found in the current scope.
IndexError: This exception is raised when an index is out of range for a list, tuple, or other sequence types.
KeyError: This exception is raised when a key is not found in a dictionary.
ValueError: Raised when a function/ method is called with an invalid argument or input, such as trying to convert a string to an INT, when the string does not represent a valid INT.
AttributeError: Raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.
IOError: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
ZeroDivisionError: This exception is raised when an attempt is made to divide a number by zero.
ImportError: This exception is raised when an import statement fails to find or load a module.

In [27]:

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero!")
    else:
        print("Result:", result)
    finally:
        print("End of division function")

# Example usage
divide(10, 2)    # Output: Result: 5.0 \n End of division function
divide(10, 0)    # Output: Error: Division by 0! \n End of division func
```

```
Result: 5.0
End of division function
Error: Division by zero!
End of division function
```

In shape area code example above:

- The Shape class serves as an abstract base class defining the common interface (area() method) for its subclasses.
- The Rectangle and Circle classes are concrete implementations of the Shape class, providing specific implementations for the area() method.
- Users of the Rectangle and Circle classes don't need to know the internal details of how the area() method is implemented; they only need to know that they can call it to get the area of the shape.

- This demonstrates how data abstraction in Python allows you to create abstract data types with well-defined interfaces, hiding the implementation details from the users of the class.

NumPy

In [28]:

```
import numpy as np
import pandas as pd
```

A. Creating a Numpy Array

In [29]:

```
ar1 = np.array(6)
ar1
```

Out[29]:

```
array(6)
```

In [30]:

```
ar2 = np.array([74,43,29,64,552])
ar3 = np.array([[74,43,29,64,552],[74,43,29,64,552]])
ar4 = np.array([[[74,43,29,64,552],[74,43,29,64,552]],
                [[74,43,29,64,552],[74,43,29,64,552]],
                [[74,43,29,64,552],[74,43,29,64,552]]])
```

In [31]:

```
print("1D-array:-",ar2)
print("2D-array:-")
print(ar3)
```

```
1D-array:- [ 74  43  29  64 552]
2D-array:-
[[ 74  43  29  64 552]
 [ 74  43  29  64 552]]
```

In [32]:

```
print("3D-array:-")
print(ar4)
```

```
3D-array:-
[[[ 74  43  29  64 552]
  [ 74  43  29  64 552]]

 [[ 74  43  29  64 552]
  [ 74  43  29  64 552]]

 [[ 74  43  29  64 552]
  [ 74  43  29  64 552]]]
```

There are other ways in which you can create arrays. The following ways are commonly used when you know the size of the array beforehand:

- **`np.ones()`** : It is used to create an array of 1s.

In [33]:

```
arr = np.ones(5)
print(arr.dtype)
arr
```

```
float64
```

Out[33]:

```
array([1., 1., 1., 1., 1.])
```

Notice that, by default, numpy creates data type = float64, but we can change the type by declaring it explicitly

In [34]:

```
arr = np.ones((2,3), dtype=int)
arr
```

Out[34]:

```
array([[1, 1, 1],
       [1, 1, 1]])
```

- **`np.zeros()`** : It is used to create an array of 0s.

In [35]:

```
np.zeros(5)
```

Out[35]:

```
array([0., 0., 0., 0., 0.])
```

- **`np.arange` ()**: It is used to create an array with increments of fixed step size.

In [36]:

```
np.arange(3,35,2)
```

Out[36]:

```
array([ 3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33])
```

- **`np.linspace()`** : It is used to create an array of fixed length.

In [37]:

```
'''inserts blank spaces at given locations mentioned in the
argument bracket; and size of array as next argument'''
np.linspace(1, 10, 8, 20)
```

Out[37]:

```
array([ 1.        ,  2.28571429,  3.57142857,  4.85714286,  6.14285714,
        7.42857143,  8.71428571, 10.        ])
```

- **`np.random.random()`** : It is used to create an array of random numbers.

In [38]:

```
np.random.random([2,4])
```

Out[38]:

```
array([[0.69876388, 0.04257375, 0.06032463, 0.34834433],
       [0.16405482, 0.00781048, 0.03640433, 0.38647299]])
```

- **`np.random.randint()`** : It is used to create an array of random numbers.

In [39]:
```
np.random.randint(1056, size=10)
```

Out[39]:
```
array([676, 350, 368, 586, 989, 418, 777, 881, 226,  17])
```

- **`np.full()`** : Create a constant array of any number ‘n’

In [40]:
```
'''np.full(arg1, arg2) : arg1= size of array,
arg2= element u want to be the array of'''
np.full(7,5)
```

Out[40]:
```
array([5, 5, 5, 5, 5, 5, 5])
```

- **`np.tile()`** : Create an identity matrix of any dimension

In [41]:
```
np.tile(7,5)
```

Out[41]:
```
array([7, 7, 7, 7, 7])
```

- **`np.eye()`** : Create an identity matrix of any dimension

In [42]:
```
np.eye(2,5)
```

Out[42]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

In [43]:
```
np.eye(3,3)
```

Out[43]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

B. Operations on numpy arrays

1. type()

In [44]:
```
type(ar4)
```

Out[44]:
```
numpy.ndarray
```

2. len()

In [45]:
```
len(ar3)
```

Out[45]:
```
2
```

3. .size

In [46]:
```
ar4.size
```

Out[46]:
```
30
```

4. .shape

In [47]:
```
ar4.shape
```

Out[47]:
```
(3, 2, 5)
```

5. .ndim

In [48]:
```
print(ar4.ndim)
print(ar1.ndim)
```

```
3
0
```

6. .itemsize

In [49]:
```
# Returns Length of one array element in bytes.
print(ar3.itemsize)
ar3=np.array([[74,64,552],[29,64,552]], dtype=np.float64)
print("float64 size",ar3.itemsize)
ar3=np.array([[74,43,29,64],[74,43,29,64]],dtype=np.complex128)
print("complex128 size",ar3.itemsize)
```

```
4
float64 size 8
complex128 size 16
```

7. Arithematic operations

In [50]:
```
print(ar2 * 2)
print("multiplying 2 arrays=")
print(ar2 * np.array([3,1,5,7,2]))
```

```
[ 148   86   58  128 1104]
multiplying 2 arrays=
[ 222   43  145  448 1104]
```

In [51]:
```
print("dividing the array by single number:", ar2/5)
print("dividing the array by an array:", ar2/np.array([3,1,5,7,2]))
```

```
dividing the array by single number: [ 14.8   8.6   5.8  12.8 110.4]
dividing the array by an array: [ 24.66666667  43.           5.8
9.14285714 276.        ]
```

8. Array indexing & subsetting

In [52]:
```python
print(ar2)
print(ar2[0])
print(ar2[-1])
```

```
[ 74  43  29  64 552]
74
552
```

9. Conditional subsetting

In [53]:
```python
# Conditional subsetting with 2 examples
print(ar2 < 50)
ar2[ar2<50]
```

```
[False  True  True False False]
```

Out[53]:
```
array([43, 29])
```

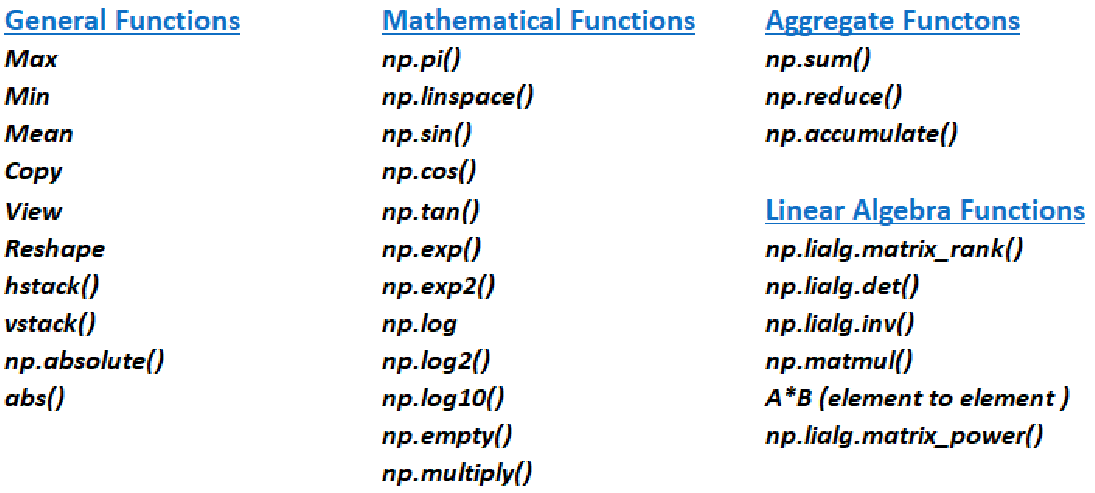
9. Slicing

In [54]:
```python
# Slicing
print(ar2[-3:])

# Comparative Slicing
EorO = np.array(['even', 'odd', 'odd', 'even', 'even'])
ArrTyp = ar2[EorO =='even']
print(" Compared elements/values of even numbers:",ArrTyp)
```

```
[ 29  64 552]
 Compared elements/values of even numbers: [ 74  64 552]
```

C. Array Functions



General Functions
Max
Min
Mean
Copy
View
Reshape
hstack()
vstack()
np.absolute()
abs()

Mathematical Functions
np.pi()
np.linspace()
np.sin()
np.cos()
np.tan()
np.exp()
np.exp2()
np.log
np.log2()
np.log10()
np.empty()
np.multiply()

Aggregate Functons
np.sum()
np.reduce()
np.accumulate()

Linear Algebra Functions
np.lialg.matrix_rank()
np.lialg.det()
np.lialg.inv()
np.matmul()
*A*B (element to element)*
np.lialg.matrix_power()

1. General Functions

- **Max**
- **Min**
- **Mean**

In [55]:

```
print(ar2.max())
print(ar2.min())
print(ar2.mean())
print(ArrTyp.mean())
```

```
552
29
152.4
230.0
```

- **Copy & View functions**

In [56]:

```
a1 = np.array([1, 2, 3, 4, 5])
x = a1.copy()
a1[0] = 42
print("Copy function:-")
print(a1)
print(x)
print("")
print("View function")
a2 = np.array([6, 7, 8, 9, 10])
x = a2.view()
a2[0] = 42
print(a2)
print(x)
```

```
Copy function:-
[42  2  3  4  5]
[1 2 3 4 5]

View function
[42  7  8  9 10]
[42  7  8  9 10]
```

- **Reshape**

In [57]:

```python
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
x = arr.reshape(4, 3)
y = arr.reshape(2,3,2)
print("2D array form: ")
print(x)
print("")
print("3D array form: ")
print(y)
```

```
2D array form: 
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

3D array form: 
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

- **Stacking Arrays (vstack | hstack)**

`np.hstack()` and `np.vstack()`

Stacking is done using the `np.hstack()` and `np.vstack()` methods. For horizontal stacking, the number of rows should be the same, while for vertical stacking, the number of columns should be the same.

In [58]:

```python
a = np.array([1, 2, 3])
b = np.array([2, 3, 4])
print("H-STACK format",np.hstack((a,b)))
print("V-STACK format",np.vstack((a,b)))
```

```
H-STACK format [1 2 3 2 3 4]
V-STACK format [[1 2 3]
 [2 3 4]]
```

- **Absolute() and abs()**

In [59]:

```python
arr = np.array([[ 0,  -1,  2,  -3],[ 4,  -5,  -6,  7]])
print("abs() function")
print(abs(arr))
print("absolute() function")
np.absolute(arr)
```

```
abs() function
[[0 1 2 3]
 [4 5 6 7]]
absolute() function
```

Out[59]:

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

B. Mathematical Functions

Trignometric functions

9. **np.pi()**

In [60]:
```
np.pi
```

Out[60]:
```
3.141592653589793
```

10. **np.linspace()**

In [61]:
```
theta = np.linspace(0, np.pi, 5)
theta
```

Out[61]:
```
array([0.        , 0.78539816, 1.57079633, 2.35619449, 3.14159265])
```

11. **np.sin()**

In [62]:
```
np.sin(theta)
```

Out[62]:
```
array([0.00000000e+00, 7.07106781e-01, 1.00000000e+00, 7.07106781e-01,
       1.22464680e-16])
```

12. **np.cos()**

In [63]:
```
np.cos(theta)
```

Out[63]:
```
array([ 1.00000000e+00,  7.07106781e-01,  6.12323400e-17, -7.07106781e-01,
       -1.00000000e+00])
```

13. **np.tan()**

In [64]:
```
np.tan(theta)
```

Out[64]:
```
array([ 0.00000000e+00,  1.00000000e+00,  1.63312394e+16, -1.00000000e+00,
       -1.22464680e-16])
```

Example - 6 (Exponential and logarithmic functions)

In [65]:
```
x = [1, 2, 5, 10]
x = np.array(x)
```

14. **np.exp()**

```
In [66]: np.exp(x) # e = 2.718...

Out[66]: array([2.71828183e+00, 7.38905610e+00, 1.48413159e+02, 2.20264658e+04])
```

15. **np.exp2()**

```
In [67]: # 2^1, 2^2, 2^3, 2^10
         np.exp2(x)

Out[67]: array([   2.,    4.,   32., 1024.])
```

16. **np.power()**

```
In [68]: np.power(x,3)

Out[68]: array([   1,    8,  125, 1000], dtype=int32)
```

17. **np.log()**

```
In [69]: np.log(x)

Out[69]: array([0.        , 0.69314718, 1.60943791, 2.30258509])
```

18. **np.log2()** --- log to base 2

```
In [70]: np.log2(x)

Out[70]: array([0.        , 1.        , 2.32192809, 3.32192809])
```

19. **np.log10()** --- log to base 10

```
In [71]: np.log10(x)

Out[71]: array([0.     , 0.30103, 0.69897, 1.     ])
```

```
In [72]: np.log

Out[72]: <ufunc 'log'>
```

20. **np.empty()**

```
In [73]: y = np.empty(4)
         y

Out[73]: array([0.     , 0.30103, 0.69897, 1.     ])
```

21. **np.multiply()**

In [74]:
```
np.multiply(x, 12, out=y)
```

Out[74]:
```
array([ 12.,  24.,  60., 120.])
```

In [75]:
```
y = np.zeros(8)
y
```

Out[75]:
```
array([0., 0., 0., 0., 0., 0., 0., 0.])
```

22. **np.power()**

In [76]:
```
np.power(2, x, out = y[::2])
```

Out[76]:
```
array([   2.,    4.,   32., 1024.])
```

C. Aggregate Functions

In [77]:
```
x = np.arange(1,9)
x
```

Out[77]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

23. **np.sum()**

In [78]:
```
np.sum(x)
```

Out[78]:
```
36
```

24. **np.reduce()**

In [79]:
```
np.add.reduce(x)
```

Out[79]:
```
36
```

25. **np.accumulate()**

In [80]:
```
np.add.accumulate(x)
```

Out[80]:
```
array([ 1,  3,  6, 10, 15, 21, 28, 36], dtype=int32)
```

In [81]:
```
np.multiply.accumulate(x)
```

Out[81]:
```
array([    1,     2,     6,    24,   120,   720,  5040, 40320],
      dtype=int32)
```

D. Linear Algebra Functions

NumPy provides the `np.linalg` package to apply common linear algebra operations

In [82]:
```
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])
print(A)
```

```
[[ 6  1  1]
 [ 4 -2  5]
 [ 2  8  7]]
```

26. **matrix_rank()**

In [83]:
```
np.linalg.matrix_rank(A)
```

Out[83]:
```
3
```

27. **np.linalg.det()** --- determinant of a matrix

In [84]:
```
np.linalg.det(A)
```

Out[84]:
```
-306.0
```

28. **np.linalg.inv()** --- Inverse of a matrix

In [85]:
```
np.linalg.inv(A)
```

Out[85]:
```
array([[ 0.17647059, -0.00326797, -0.02287582],
       [ 0.05882353, -0.13071895,  0.08496732],
       [-0.11764706,  0.1503268 ,  0.05228758]])
```

29. **np.matmul()** --- Matrix Multiplication

In [86]:
```
B = np.linalg.inv(A)
np.matmul(A,B) #actual matrix multiplication
```

Out[86]:
```
array([[1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.22044605e-16, 1.00000000e+00, 0.00000000e+00],
       [1.11022302e-16, 2.22044605e-16, 1.00000000e+00]])
```

30. **A * B** --- Element to element multiplication of matrix

In [87]:
```
A * B
```

Out[87]:
```
array([[ 1.05882353, -0.00326797, -0.02287582],
       [ 0.23529412,  0.26143791,  0.4248366 ],
       [-0.23529412,  1.20261438,  0.36601307]])
```

31. **np.linalg.matrix_power()** --- Matrix raised to a certain power

In [88]:

```
np.linalg.matrix_power(A,3) # matrix multiplication A A A
```

Out[88]:

```
array([[336, 162, 228],
       [406, 162, 469],
       [698, 702, 905]])
```

Pandas

A. Pandas Series

Pandas series are basically 1 dimensional arrays that can hold any data types

In [89]:

```
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

Creating a Data Frame Students using raw data stored in a dictionary

In [90]:

```
Roll_No = [5,12,22,23,34,50]
Name  = ['Adam', 'Sham', 'Sharon', 'Deepak', 'Seema', 'Vijay']
Div = ['A','B','A','A','B','B']
Eng = [54, 73, 68, 40, 92, 88]
Maths = [45, 35, 18, 80, 60, 55]
Sci = [82, 85, 30, 100, 50, 74]
data = {"Roll_No": Roll_No, "Name": Name, "Div":Div, "English": Eng,
       "Maths":Maths, "Science":Sci}

Div = ['A','B']
Total = [128, 92]
Teacher = ['Rashmika', 'Rita']
data1 = {"Div":Div, "Total":Total, "Teacher":Teacher}

Name  = ['Adam', 'Sham', 'Sharon', 'Deepak', 'Seema', 'Vijay']
Roll_No = [5,12,22,23,34,50]
Blood_grp = ["O+","A+","A-","B+","A+","O+"]
data2 = {"Name":Name, "roll":Roll_No, "Blood_grp":Blood_grp}
```

In [91]:

```
# using the above dictionary for making a data frame.
Students = pd.DataFrame(data)
Division = pd.DataFrame(data1)
Profile = pd.DataFrame(data2)
Students.head(3)
```

Out[91]:

	Roll_No	Name	Div	English	Maths	Science
0	5	Adam	A	54	45	82
1	12	Sham	B	73	35	85
2	22	Sharon	A	68	18	30

In [92]:

```
type(Students)
```

Out[92]:

```
pandas.core.frame.DataFrame
```

B. Importing data from different file formats and creat Df

1. **`df_name = pd.read_csv('File_name.csv')`** : For importing csv file (comma separated values) with arguments such as,
   filepath_or_buffer = file_name OR file_address
   sep = '|' OR ',' OR ' ' OR '.'
   header = None.

2. **`df_name = pd.read_json('File_name.json')`** : For importing json files.
   filepath_or_buffer = file_name OR file_address
   sep = '|' OR ',' OR ' ' OR '.'
   header = None.

3. **`df_name = pd.read_excel('File_name.xlsx')`** : For importing excel files.
   filepath_or_buffer = file_name OR file_address
   sep = '|' OR ',' OR ' ' OR '.'
   header = None.

In [93]:

```
sales = pd.read_excel('sales.xlsx')
sales.head(3)
```

Out[93]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

C. Formatting the DataFrame

1. Setting Header of DataFrame as "None" (which will set column headers with index numbers)

example:
Sample_df = pd.read_csv('file_name.csv', sep ='|', header = None)

In [94]:

```
sales = pd.read_excel('sales.xlsx', header = None)
sales.head(3)
```

Out[94]:

	0	1	2	3	4
0	Market	Region	No_of_Orders	Profit	Sales
1	Africa	Western Africa	251	-12901.51	78476.06
2	Africa	Southern Africa	85	11768.58	51319.5

2. Replacing Column Headers with userdefined values

example:
Sample_df = pd.read_csv('file_name.csv', sep ='|', header = None)
Sample_df.columns = ['Roll_no', 'Name', 'Address', 'contact', 'Class']

3. Indexing in pandas DataFrame

- Setting a column as indexing (temporarily after loading data).
- Setting a column as indexing (while importing data by read file).
- Setting a column as indexing permanently & keeping ot in place for furter process.
- MultiIndexing

In [95]:

```
sales = pd.read_excel('sales.xlsx')
sales.head()
```

Out[95]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89
3	Africa	Eastern Africa	110	8013.04	44182.60
4	Africa	Central Africa	103	15606.30	61689.99

In [96]:

```
#Setting a column as indexing (temporarily after loading data).
sales.set_index("Region").head(3)
```

Out[96]:

	Market	No_of_Orders	Profit	Sales
Region				
Western Africa	Africa	251	-12901.51	78476.06
Southern Africa	Africa	85	11768.58	51319.50
North Africa	Africa	182	21643.08	86698.89

In [97]:

```
sales.head(3)
```

Out[97]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

In [98]:

```
#Setting a column as indexing permanently & keeping ot in place for fur
sales.set_index("Profit", inplace = True)
```

In [99]:

```
sales.head(3)
```

Out[99]:

	Market	Region	No_of_Orders	Sales
Profit				
-12901.51	Africa	Western Africa	251	78476.06
11768.58	Africa	Southern Africa	85	51319.50
21643.08	Africa	North Africa	182	86698.89

In [100]:

```
#Setting a column as indexing (while importing data by read file).
sales = pd.read_excel('sales.xlsx', index_col = [0,1,2])
sales.head(3)
```

Out[100]:

			Profit	Sales
Market	**Region**	**No_of_Orders**		
Africa	**Western Africa**	**251**	-12901.51	78476.06
	Southern Africa	**85**	11768.58	51319.50
	North Africa	**182**	21643.08	86698.89

D. Data Transformations

1. Dataframe referrencing OR Accessing mechanism

Data referencing in pandas is done by 2 type of functions that is:

`.loc[]` : is used to refer a single value or a set of values in the dataframe using the **Label** of the rows and columns.
df.loc[row , column] ------ labels means the exact names and not he index number

`.iloc[]` : is used to refer a single value or a set of values in the dataframe using indexes of the rows and columns.
df.loc[row_index , column_index]

In [101]:

```
sales = pd.read_excel('sales.xlsx')
sales.head(3)
```

Out[101]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

loc[ ]

In [102]:

```
sales.loc[0:3, "No_of_Orders": "Sales"]
```

Out[102]:

	No_of_Orders	Profit	Sales
0	251	-12901.51	78476.06
1	85	11768.58	51319.50
2	182	21643.08	86698.89
3	110	8013.04	44182.60

In [103]:

```
sales.loc[1:3,"Profit"]
```

Out[103]:

```
1    11768.58
2    21643.08
3     8013.04
Name: Profit, dtype: float64
```

In [104]:

```
sales.loc[5,"Profit"]
```

Out[104]:

```
-16766.9
```

In [105]:

```
sales.loc[5,:]
```

Out[105]:

```
Market            Asia Pacific
Region            Western Asia
No_of_Orders               382
Profit                -16766.9
Sales                124312.24
Name: 5, dtype: object
```

iloc[ ]

In [106]:

```
sales.iloc[5,3:6]
```

Out[106]:

```
Profit      -16766.9
Sales      124312.24
Name: 5, dtype: object
```

In [107]:

```
sales.iloc[5:8,:]
```

Out[107]:

	Market	Region	No_of_Orders	Profit	Sales
5	Asia Pacific	Western Asia	382	-16766.90	124312.24
6	Asia Pacific	Southern Asia	469	67998.76	351806.60
7	Asia Pacific	Southeastern Asia	533	20948.84	329751.38

2. Data Slicing

In [108]:

```
sales[["Market","Sales","Profit"]].head(5)
```

Out[108]:

	Market	Sales	Profit
0	Africa	78476.06	-12901.51
1	Africa	51319.50	11768.58
2	Africa	86698.89	21643.08
3	Africa	44182.60	8013.04
4	Africa	61689.99	15606.30

In [109]:

```
# Data Slicing using iloc() function
sales.iloc[:,[0,3,2]].head(3)
```

Out[109]:

	Market	Profit	No_of_Orders
0	Africa	-12901.51	251
1	Africa	11768.58	85
2	Africa	21643.08	182

In [110]:

```
# Data Slicing using iloc() function
sales.set_index("Region")
sales.iloc[[0,1,2],:]
```

Out[110]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

3. Data - Filtering

In [111]:

```
sales[sales["Sales"]>300000]
```

Out[111]:

	Market	Region	No_of_Orders	Profit	Sales
6	Asia Pacific	Southern Asia	469	67998.76	351806.60
7	Asia Pacific	Southeastern Asia	533	20948.84	329751.38
8	Asia Pacific	Oceania	646	54734.02	408002.98
9	Asia Pacific	Eastern Asia	414	72805.10	315390.77
11	Europe	Western Europe	964	82091.27	656637.14
16	LATAM	Central America	930	74679.54	461670.28

In [112]:

```
sales[ (sales["Market"].isin(["LATAM", "Europe"]))
      & (sales["Sales"] > 250000) ]

# condition 1:==== (sales["Market"].isin(["LATAM", "Europe"]))
# condition 2:==== (sales["Sales"] > 250000)
```

Out[112]:

	Market	Region	No_of_Orders	Profit	Sales
11	Europe	Western Europe	964	82091.27	656637.14
13	Europe	Northern Europe	367	43237.44	252969.09
16	LATAM	Central America	930	74679.54	461670.28

* examples of data transformations______________________________*

Replace the sales values in the form of thousands eg. 300000 - 300K

In [113]:

```
sales.Sales=sales.Sales.floordiv(1000)
sales.head(3)
```

Out[113]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78.0
1	Africa	Southern Africa	85	11768.58	51.0
2	Africa	North Africa	182	21643.08	86.0

In [114]:

```
#Rename the column
sales.rename(columns={'Sales': 'Sales in Thousands'}, inplace=True)
sales.head()
```

Out[114]:

	Market	Region	No_of_Orders	Profit	Sales in Thousands
0	Africa	Western Africa	251	-12901.51	78.0
1	Africa	Southern Africa	85	11768.58	51.0
2	Africa	North Africa	182	21643.08	86.0
3	Africa	Eastern Africa	110	8013.04	44.0
4	Africa	Central Africa	103	15606.30	61.0

Adding new table to the dataframe "Profit % of Total"

In [115]:

```
total_sum = sales.Profit.sum()
sales['Profit %'] = sales.Profit.apply(lambda x: x/total_sum*100)
sales.head()
```

Out[115]:

	Market	Region	No_of_Orders	Profit	Sales in Thousands	Profit %
0	Africa	Western Africa	251	-12901.51	78.0	-1.943646
1	Africa	Southern Africa	85	11768.58	51.0	1.772967
2	Africa	North Africa	182	21643.08	86.0	3.260587
3	Africa	Eastern Africa	110	8013.04	44.0	1.207185
4	Africa	Central Africa	103	15606.30	61.0	2.351130

In [116]:

```
sales = pd.read_excel('sales.xlsx')
sales.head(3)
```

Out[116]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

E. Reading and Understanding Data

1. head() : To display rows from top.
2. tail() : To display rows from bottom.

In [117]:

```
sales.head(3)
```

Out[117]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89

In [118]:

```
sales.tail(3)
```

Out[118]:

	Market	Region	No_of_Orders	Profit	Sales
20	USCA	Eastern US	443	47462.04	264973.98
21	USCA	Central US	356	33697.43	170416.31
22	USCA	Canada	49	7246.62	26298.81

3. info()

In [119]:

```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Market        23 non-null     object 
 1   Region        23 non-null     object 
 2   No_of_Orders  23 non-null     int64  
 3   Profit        23 non-null     float64
 4   Sales         23 non-null     float64
dtypes: float64(2), int64(1), object(2)
memory usage: 1.0+ KB
```

4. describe()

In [120]:

```
sales.describe()
```

Out[120]:

	No_of_Orders	Profit	Sales
count	23.000000	23.000000	23.000000
mean	366.478261	28859.944783	206285.108696
std	246.590361	27701.193773	160589.886606
min	37.000000	-16766.900000	8190.740000
25%	211.500000	12073.085000	82587.475000
50%	356.000000	20948.840000	170416.310000
75%	479.500000	45882.845000	290182.375000
max	964.000000	82091.270000	656637.140000

5. Displaying float values with specified decimal values

In [121]:

```
pd.options.display.float_format = '{:,.2f}'.format
sales.describe()
```

Out[121]:

	No_of_Orders	Profit	Sales
count	23.00	23.00	23.00
mean	366.48	28,859.94	206,285.11
std	246.59	27,701.19	160,589.89
min	37.00	-16,766.90	8,190.74
25%	211.50	12,073.08	82,587.48
50%	356.00	20,948.84	170,416.31
75%	479.50	45,882.85	290,182.38
max	964.00	82,091.27	656,637.14

F. Other Pandas Functions / Operations of Pandas

In [122]:

```
Students
```

Out[122]:

	Roll_No	Name	Div	English	Maths	Science
0	5	Adam	A	54	45	82
1	12	Sham	B	73	35	85
2	22	Sharon	A	68	18	30
3	23	Deepak	A	40	80	100
4	34	Seema	B	92	60	50
5	50	Vijay	B	88	55	74

In [123]:

```
Division
```

Out[123]:

	Div	Total	Teacher
0	A	128	Rashmika
1	B	92	Rita

In [124]:

```
Profile
```

Out[124]:

	Name	roll	Blood_grp
0	Adam	5	O+
1	Sham	12	A+
2	Sharon	22	A-
3	Deepak	23	B+
4	Seema	34	A+
5	Vijay	50	O+

1. merge() (https://www.w3schools.com/python/pandas/ref_df_merge.asp)

In [125]:

```
newdf = Students.merge(Profile,
    how='right', on ="Name").merge(Division,how="left", on= "Div")

newdf
```

Out[125]:

	Roll_No	Name	Div	English	Maths	Science	roll	Blood_grp	Total	Teacher
0	5	Adam	A	54	45	82	5	O+	128	Rashmika
1	12	Sham	B	73	35	85	12	A+	92	Rita
2	22	Sharon	A	68	18	30	22	A-	128	Rashmika
3	23	Deepak	A	40	80	100	23	B+	128	Rashmika
4	34	Seema	B	92	60	50	34	A+	92	Rita
5	50	Vijay	B	88	55	74	50	O+	92	Rita

2. melt() (https://www.w3schools.com/python/pandas/ref_df_melt.asp) -- makes the df from wide to narrow/ long

In [126]:

```
newdf.melt().head(3)
```

Out[126]:

	variable	value
0	Roll_No	5
1	Roll_No	12
2	Roll_No	22

2. pivot() (https://www.geeksforgeeks.org/python-pandas-pivot/) -- makes a pivot of given df

In [127]:
```
table = pd.pivot_table(sales, values ='A',
        index =['B', 'C'], columns =['B'],aggfunc = np.sum)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15344\2177263313.py in <module>
----> 1 table = pd.pivot_table(sales, values ='A',
      2         index =['B', 'C'], columns =['B'],aggfunc = np.sum)

E:\Anaconda\lib\site-packages\pandas\core\reshape\pivot.py in pivot_table(data, values, index, columns, aggfunc, fill_value, margins, dropna, margins_name, observed, sort)
     93         return table.__finalize__(data, method="pivot_table")
     94
---> 95     table = __internal_pivot_table(
     96         data,
     97         values,

E:\Anaconda\lib\site-packages\pandas\core\reshape\pivot.py in __internal_pivot_table(data, values, index, columns, aggfunc, fill_value, margins, dropna, margins_name, observed, sort)
    139         for i in values:
    140             if i not in data:
--> 141                 raise KeyError(i)
    142
    143         to_filter = []

KeyError: 'A'
```

add() Adds the values of a DataFrame with the specified value(s)
add_prefix() Prefix all labels
add_suffix() Suffix all labels
agg() Apply a function or a function name to one of the axis of the DataFrame
align() Aligns two DataFrames with a specified join method
all() Return True if all values in the DataFrame are True, otherwise False
any() Returns True if any of the values in the DataFrame are True, otherwise False
append() Append new columns
applymap() Execute a function for each element in the DataFrame
apply() Apply a function to one of the axis of the DataFrame
assign() Assign new columns
astype() Convert the DataFrame into a specified dtype **at** Get or set the value of the item with the specified label **axes** Returns the labels of the rows and the columns of the DataFrame
bfill() Replaces NULL values with the value from the next row
combine() Compare the values in two DataFrames, and let a function decide which values to keep
combine_first() Compare two DataFrames, and if the first DataFrame has a NULL value, it will befilled with the respective value from the second DataFrame
compare() Compare two DataFrames and return the differences
convert_dtypes() Converts the columns in the DataFrame into new dtypes
corr() Find the correlation (relationship) between each column
count() Returns the number of not empty cells for each column/row
cov() Find the covariance of the columns

cummax() Calculate the cumulative maximum values of the DataFrame
cummin() Calculate the cumulative minmum values of the DataFrame
cumprod() Calculate the cumulative product over the DataFrame
cumsum() Calculate the cumulative sum over the DataFrame
diff() Calculate the difference between a value and the value of the same column in the previous row
div() Divides the values of a DataFrame with the specified value(s)
dot() Multiplies the values of a DataFrame with values from another array-like object, and add the result
drop() Drops the specified rows/columns from the DataFrame
drop_duplicates() Drops duplicate values from the DataFrame
droplevel() Drops the specified index/column(s)
dropna() Drops all rows that contains NULL values
dtypes Returns the dtypes of the columns of the DataFrame
duplicated() Returns True for duplicated rows, otherwise False
empty Returns True if the DataFrame is empty, otherwise False
eq() Returns True for values that are equal to the specified value(s), otherwise False
equals() Returns True if two DataFrames are equal, otherwise False
eval Evaluate a specified string
explode() Converts each element into a row
ffill() Replaces NULL values with the value from the previous row
fillna() Replaces NULL values with the specified value
filter() Filter the DataFrame according to the specified filter
first() Returns the first rows of a specified date selection
floordiv() Divides the values of a DataFrame with the specified value(s), and floor the values
ge() Returns True for values greater than, or equal to the specified value(s), otherwise False
get() Returns the item of the specified key
groupby() Groups the rows/columns into specified groups
gt() Returns True for values greater than the specified value(s), otherwise False
iat Get or set the value of the item in the specified position
idxmax() Returns the label of the max value in the specified axis
idxmin() Returns the label of the min value in the specified axis
infer_objects() Change the dtype of the columns in the DataFrame
insert() Insert a column in the DataFrame
interpolate() Replaces not-a-number values with the interpolated method
isin() Returns True if each elements in the DataFrame is in the specified value
isna() Finds not-a-number values
isnull() Finds NULL values
items() Iterate over the columns of the DataFrame
iteritems() Iterate over the columns of the DataFrame
iterrows() Iterate over the rows of the DataFrame
itertuples() Iterate over the rows as named tuples
join() Join columns of another DataFrame
last() Returns the last rows of a specified date selection
le() Returns True for values less than, or equal to the specified value(s), otherwise False
lt() Returns True for values less than the specified value(s), otherwise False
keys() Returns the keys of the info axis
kurtosis() Returns the kurtosis of the values in the specified axis
mask() Replace all values where the specified condition is True
median() Return the median of the values in the specified axis

memory_usage() Returns the memory usage of each column
mod() Modules (find the remainder) of the values of a DataFrame
mode() Returns the mode of the values in the specified axis
mul() Multiplies the values of a DataFrame with the specified value(s)
ne() Returns True for values that are not equal to the specified value(s), otherwise False
nlargest() Sort the DataFrame by the specified columns, descending, and return the specified number of rows
notna() Finds values that are not not-a-number
notnull() Finds values that are not NULL
nsmallest() Sort the DataFrame by the specified columns, ascending, and return the specified number of rows
nunique() Returns the number of unique values in the specified axis
pct_change() Returns the percentage change between the previous and the current value
pipe() Apply a function to the DataFrame
pivot_table() Create a spreadsheet pivot table as a DataFrame
pow() Raise the values of one DataFrame to the values of another DataFrame
prod() Returns the product of all values in the specified axis
product() Returns the product of the values in the specified axis
quantile() Returns the values at the specified quantile of the specified axis
query() Query the DataFrame
radd() Reverse-adds the values of one DataFrame with the values of another DataFrame
rdiv() Reverse-divides the values of one DataFrame with the values of another DataFrame
reindex() Change the labels of the DataFrame
reindex_like() ??
rename() Change the labels of the axes
rename_axis() Change the name of the axis
reorder_levels() Re-order the index levels
replace() Replace the specified values
reset_index() Reset the index
rfloordiv() Reverse-divides the values of one DataFrame with the values of another DataFrame
rmod() Reverse-modules the values of one DataFrame to the values of another DataFrame
rmul() Reverse-multiplies the values of one DataFrame with the values of another DataFrame
rpow() Reverse-raises the values of one DataFrame up to the values of another DataFrame
rsub() Reverse-subtracts the values of one DataFrame to the values of another DataFrame
rtruediv() Reverse-divides the values of one DataFrame with the values of another DataFrame
sample() Returns a random selection elements
sem() Returns the standard error of the mean in the specified axis
select_dtypes() Returns a DataFrame with columns of selected data types
set_axis() Sets the index of the specified axis
set_flags() Returns a new DataFrame with the specified flags
skew() Returns the skew of the values in the specified axis
sort_index() Sorts the DataFrame according to the labels
sort_values() Sorts the DataFrame according to the values
squeeze() Converts a single column DataFrame into a Series
std() Returns the standard deviation of the values in the specified axis
sub() Subtracts the values of a DataFrame with the specified value(s)
swaplevel() Swaps the two specified levels
take() Returns the specified elements
to_xarray() Returns an xarray object

transform() Execute a function for each value in the DataFrame

transpose() Turns rows into columns and columns into rows

truediv() Divides the values of a DataFrame with the specified value(s)

truncate() Removes elements outside of a specified set of values

update() Update one DataFrame with the values from another DataFrame

value_counts() Returns the number of unique rows

values Returns the DataFrame as a NumPy array

var() Returns the variance of the values in the specified axis

where() Replace all values where the specified condition is False

xs() Returns the cross-section of the DataFrame

iter() Returns an iterator of the info axes

1. Write a pandas query to find the top 5 customers with the highest total purchase amount. assume you have two dataframes: customers (CustomerID, Name) and orders (OrderID, CustomerID, Amount).

```
merged_df = pd.merge(customers, orders, on='CustomerID')
top_customers = merged_df.groupby(['CustomerID', 'Name'])['Amount'].sum().reset_index()
top_customers = top_customers.sort_values(by='Amount', ascending=False).head(5)
print(top_customers)
```

2. Write a pandas query to find the nth highest salary from a dataframe employees with columns EmployeeID, Name, and Salary.

```
n = 2 # replace with desired rank
nth_highest_salary = employees['Salary'].drop_duplicates().nlargest(n).iloc[-1]
print(nth_highest_salary)
```

3. Given a dataframe sales with columns SaleID, ProductID, SaleDate, and Quantity, write a pandas query to find the total quantity sold for each product per month.

```
assuming sales dataframe is already defined
sales['Month'] = sales['SaleDate'].dt.to_period('M')
total_quantity_per_month = sales.groupby(['ProductID', 'Month'])
['Quantity'].sum().reset_index()
print(total_quantity_per_month)
```

4. Write a pandas query to find all employees who have more than one manager. assume you have a dataframe employees (EmployeeID, Name, ManagerID).

```
multiple_managers = employees.groupby(['EmployeeID', 'Name'])['ManagerID'].nunique()
multiple_managers = multiple_managers[multiple_managers > 1].reset_index()
print(multiple_managers)
```

5. Given a dataframe orders with columns OrderID, CustomerID, OrderDate, and a dataframe orderdetails with columns OrderID, ProductID, Quantity, write a pandas query to find the top 3 products with the highest sales quantity.

```
merged_df = pd.merge(orderdetails, orders, on='OrderID')
top_products = merged_df.groupby('ProductID')['Quantity'].sum().reset_index()
top_products = top_products.sort_values(by='Quantity', ascending=False).head(3)
print(top_products)
```

6. Write a pandas query to find the second most recent order date for each customer from a dataframe orders (OrderID, CustomerID, OrderDate).

```
second_recent_order = orders.sort_values(['CustomerID',
'OrderDate']).drop_duplicates('CustomerID', keep='last').shift(1) print(second_recent_order)
```

7. Given a dataframe products with columns ProductID, Name, Price, and a dataframe sales with columns SaleID, ProductID, Quantity, write a pandas query to find the product with the highest revenue.

```
merged_df = pd.merge(sales, products, on='ProductID')
merged_df['Revenue'] = merged_df['Price'] * merged_df['Quantity']
top_product = merged_df.groupby('ProductID')
['Revenue'].sum().reset_index().sort_values(by='Revenue',
ascending=False).head(1)
```