Python for Data Analysis and Engineering
Class 3

Ashutosh Ranjan
Data Analyst

❖ Agenda for Today’s Class

- ❖ Recap of previous class learnings
- ❖ New Topics for Today’s class
  - ➢ ETL Process
    - ▪ ETL using Python
  - ➢ File Handling
  - ➢ Map-Reduce-Filter Function
  - ➢ Exception Handling
    - ▪ Mechanism of Exception Handling
  - ➢ Web Scraping with Python
    - ▪ Demo: Web data scraping

❖ Recap of Previous Class Learnings

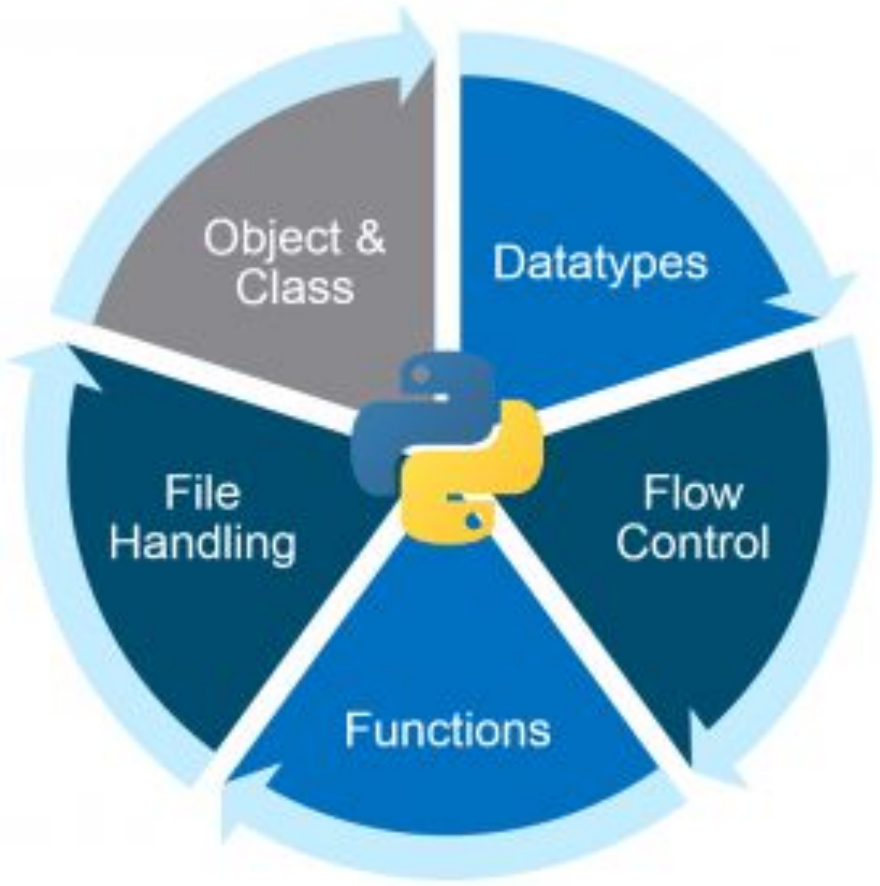
- **Python:**
  - ✓ Python Fundamentals
  - ✓ Data Structure
  - ✓ OOPs Concepts
    - Class
    - Objects
    - Methods
    - Encapsulations
    - Polymorphism
    - Data Abstractions
  - ✓ Loading Python Libraries

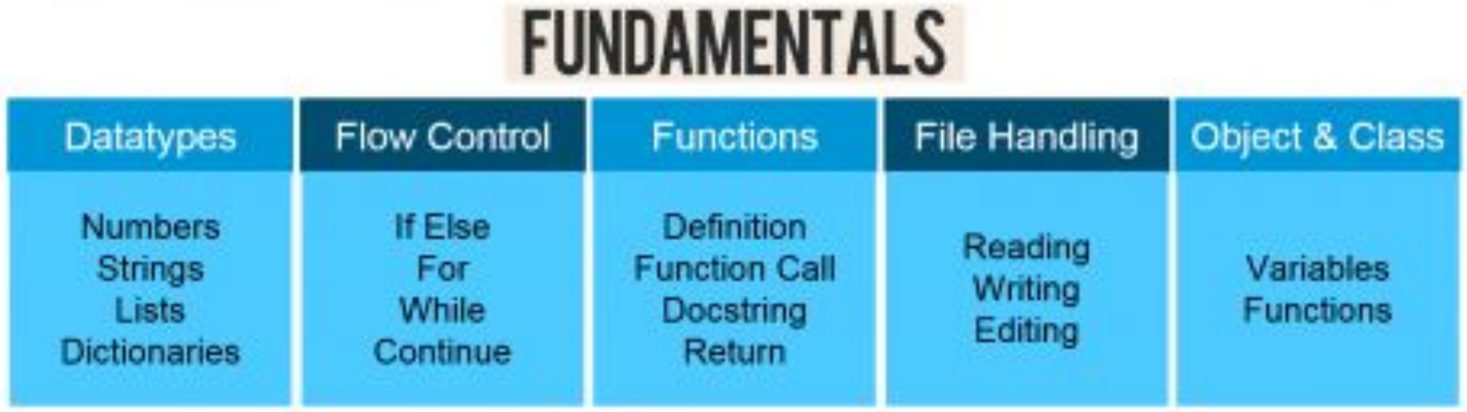
- **Pandas:**
  - ✓ Reading data using pandas
  - ✓ Reading data using pandas
  - ✓ Dataframe data types
  - ✓ Data Frames methods
  - ✓ Filters and Sorters
  - ✓ Dataframe slicing
  - ✓ Aggregation, Group by, Concatenation, Merge, Pivot, Cross Tab
  - ✓ Missing Values

- **Numpy**
  - ✓ Arrays
  - ✓ Matrix
  - ✓ Sampling
  - ✓ Sorting
  - ✓ Merge
  - ✓ Basic of Mathematical and Statistical operations

- **Visualization**
  - ✓ Working with Matplotlib
  - ✓ Working with Seaborn

Python Fundamentals



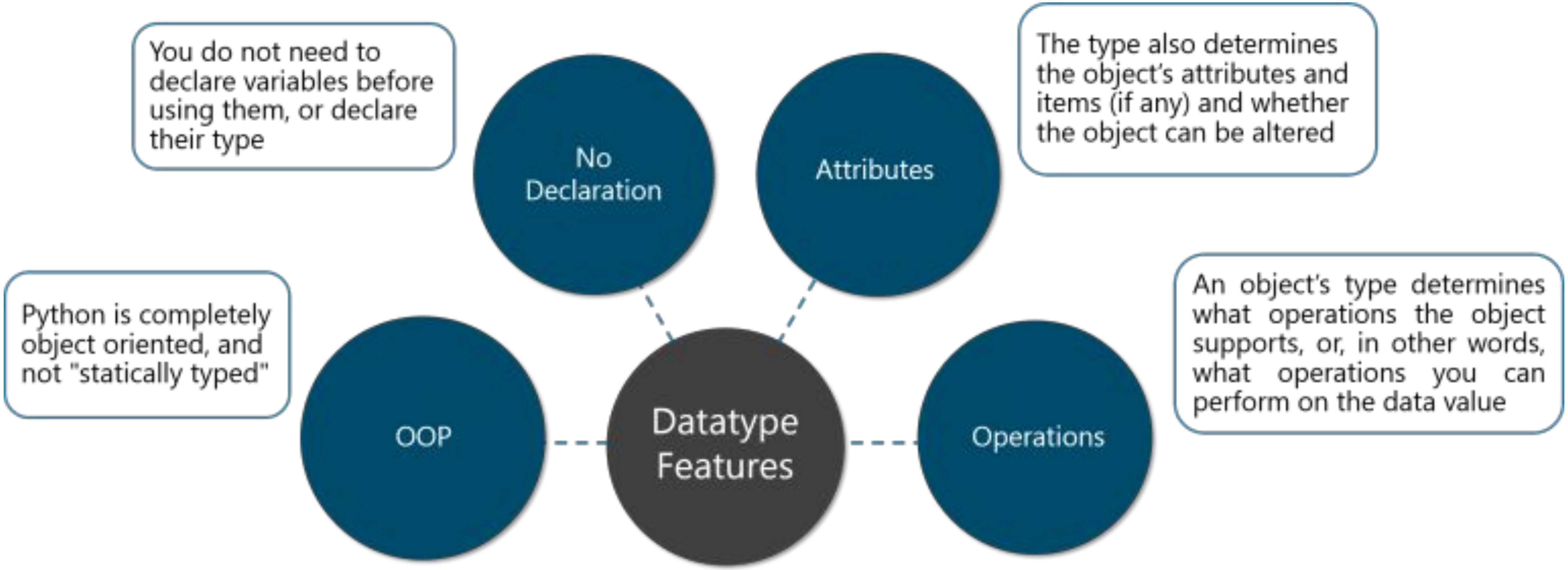


FUNDAMENTALS

Datatypes	Flow Control	Functions	File Handling	Object & Class
Numbers Strings Lists Dictionaries	If Else For While Continue	Definition Function Call Docstring Return	Reading Writing Editing	Variables Functions

Data Types

All data values in Python are represented by objects and each object or value has a datatype.


You do not need to declare variables before using them, or declare their type
No Declaration
Attributes
The type also determines the object's attributes and items (if any) and whether the object can be altered
Python is completely object oriented, and not "statically typed"
OOP
Datatype Features
Operations
An object's type determines what operations the object supports, or, in other words, what operations you can perform on the data value

Native Data Types in Python

Boolean

True / False

```
if (number % 2) = 0:
        even = True
else:
        even = False
```

Numbers

Integers, Floats, Fractions and Complex Numbers

```
a = 5
b = 7.3
c = 2 + 3j
```

Strings

Sequences of Unicode Characters

```
s = "This is a string"
```

Bytes & ByteArray

Contain Single Bytes

```
b = 'A\nB\nC'
```

Lists

Ordered sequences of values

```
a = [ 1, 2.2, "Python"]
```

Tuples

Ordered immutable sequences of values

```
t = [ 2, "Tuple", "95"]
```

Sets

Unordered bags of values

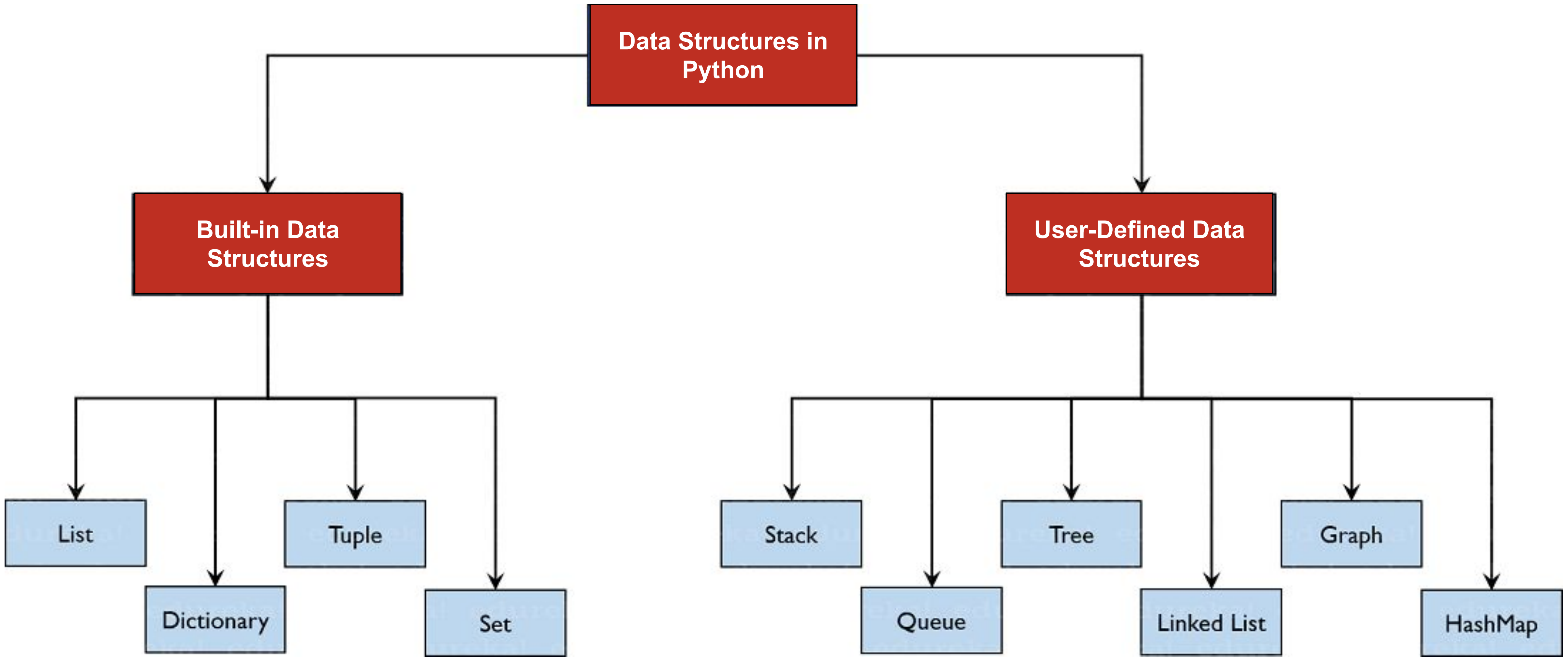
```
week = {'Mon', 'Tue',
'Wed', 'Thu', 'Fri', 'Sat',
'Sun'}
```

Dictionaries

Unordered bags of key-value pairs

```
d = {'value':5, 'key':125}
```

Data Structures In Python



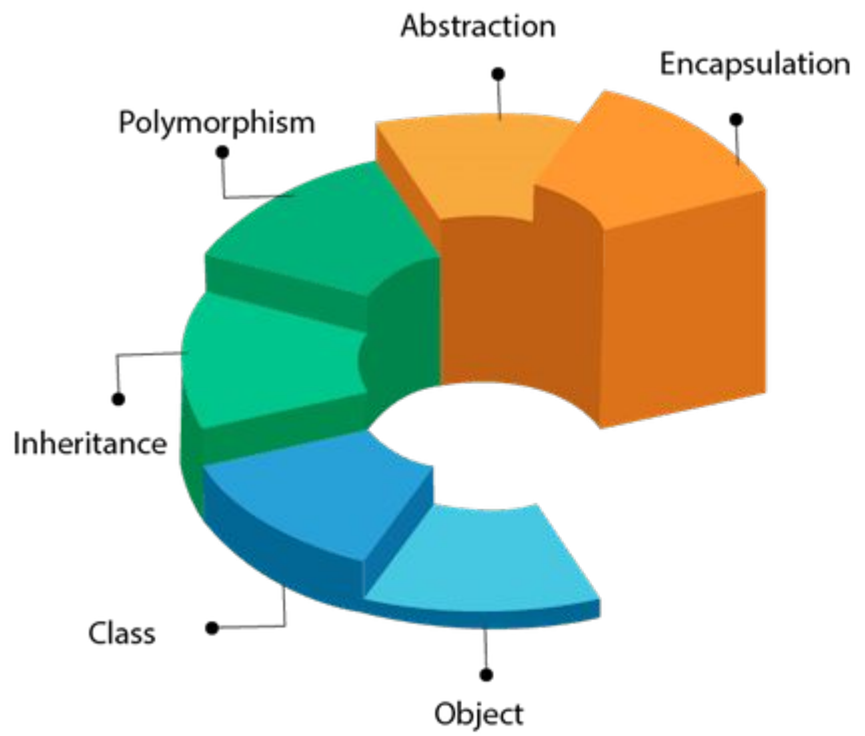
OOPs Concept in Python

Object-oriented programming (OOPs) is a method of designing a program by binding there related properties and behaviours into individual objects.

Principles of OOPs:

- Class
- Object
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

OOPs (Object-Oriented Programming System)



Class:

- The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods.
- We can also say that class is a blueprint of an object i.e. if any person is an object, class will have all the information about the structure or behaviour that person.
- OOPs, reduce the size of our code.

```
#python Program
class antwaker:
    def __init__(self, name , userid,password):
        self.name=name
        self.userid=userid
        self.password=password
```

Object:

- Object is a simple entity which posses some property or behaviour.
- Everything in Python is an object, and almost everything has attributes and methods

```
#python Program
class antwaker:
    def __init__(self, name , userid,password):
        self.name=name
        self.userid=userid
        self.password=password

ob1=antwaker(‘student’,‘Abc’,‘abc@123’) ##object
print(ob1.name)
```

Method:

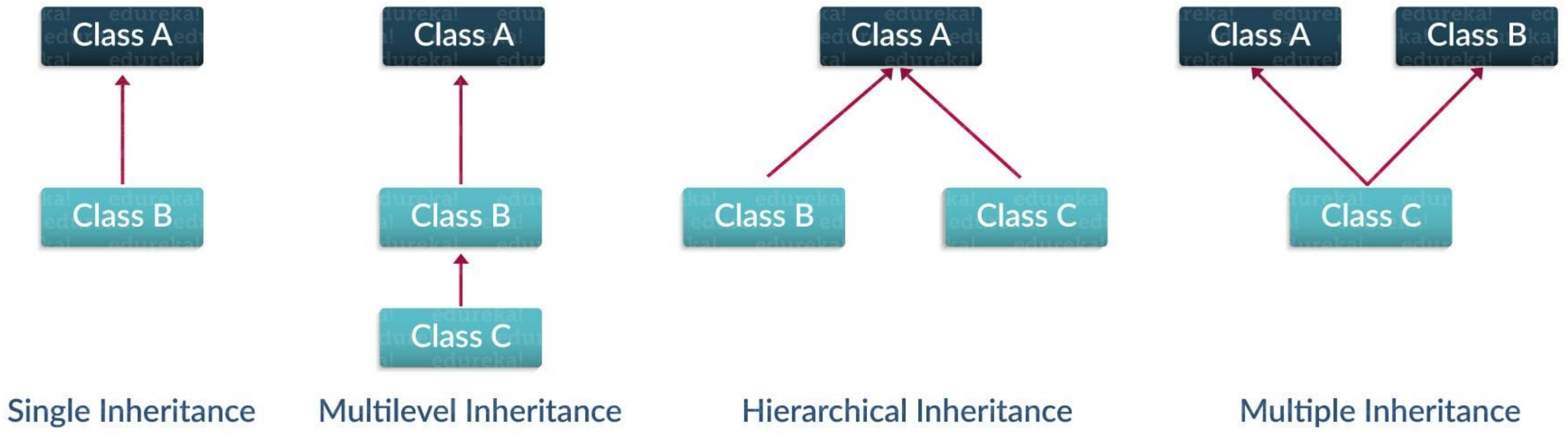
- The method is a function that is associated with an object.
- In Python, a method is not unique to class instances. Any object type can have methods.

```
#python Program
class antwaker:
    def __init__(self, name , userid,password):
        self.name=name
        self.userid=userid
        self.password=password
    def display(self): ##method of an class
        print(self.name)
        print(self.userid)

ob1=antwaker(‘student’,‘Abc’,‘abc@123’) ##object
ob1.display()
```

Inheritance:

- Inheritance is the capability of one class to derive or inherit the properties from another class.
- The benefits of inheritance are:
  a. Reusability of code.
  b. We can use the features of the other class, without modifying it.
  c. It is transitive in nature i.e. A->B->C == A->C and B->C.
  d. The new class is known as derived class or child class.

Types of Inheritance



Polymorphism:

- Polymorphism contains two words “poly” and “morphs”. Poly means many, and morph means shape.
- Polymorphism states that we can perform a task in more than one ways.

Example:

```
class Python:
   def description(self):
      print("call Python.")

class SQL:
   def description(self):
      print("call SQL.")

a = Python()
b = SQL()
for x in (a,b):
   x.description()
```

```
#python Program
li=[1,2,3,4]
def add(lis):
    s=0
    for i in lis:
        s+=i
    return s

print(‘First Method’,add(li))
print(‘2nd Method:’,sum(li))
```

When the function is called using the object **“a”** then the function of class **Python** is called and when it is called using the object **“b”** then the function of class **SQL** is called.

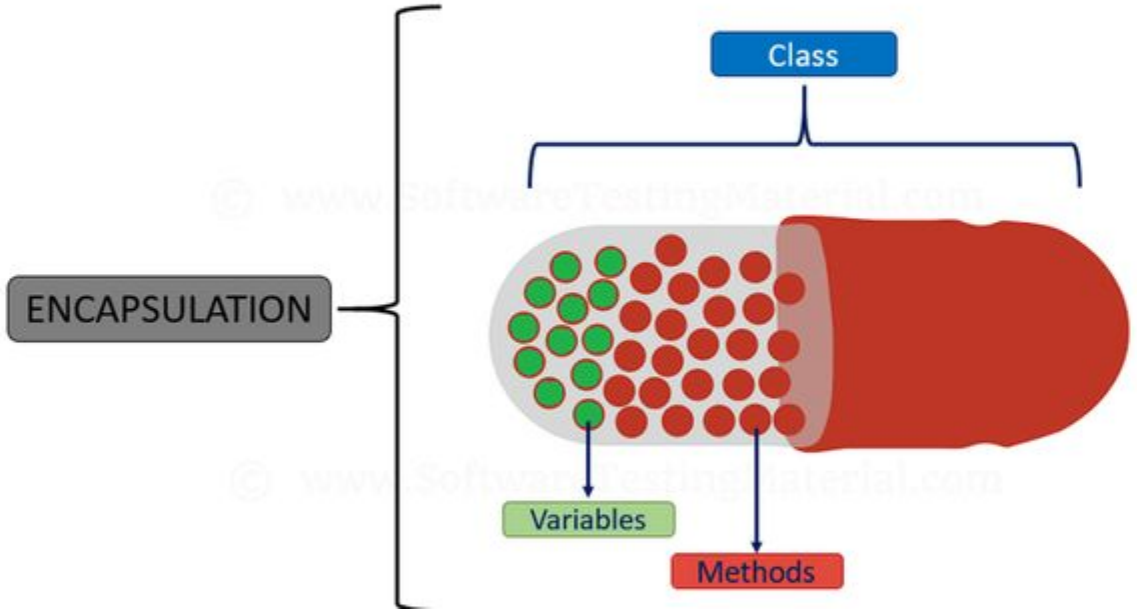
Data Abstraction:

- Data Abstraction is the process of hiding the real implementation of an application from a user and emphasizing only on usage of the application.

- Objects are the building blocks of Object-Oriented Programming. An object contains some properties and methods.
    - We can hide them from the outer world through access modifiers.
    - We can provide access only for required functions and properties to the other programs.
    - This is the general procedure to implement abstraction in OOPS.

Encapsulation:

- It is a process of wrapping of data, variable, and methods into a single entity in a program.
- Encapsulation acts as a protective layer by ensuring that, access to wrapped data is not possible by any code defined outside the class in which the wrapped data are defined.
- In Python, Encapsulation can be achieved by declaring the data members of a class either as private or protected.
- In Python, 'Private' and 'Protected' are called Access Modifiers, as they modify the access of variables or methods defined in a class.



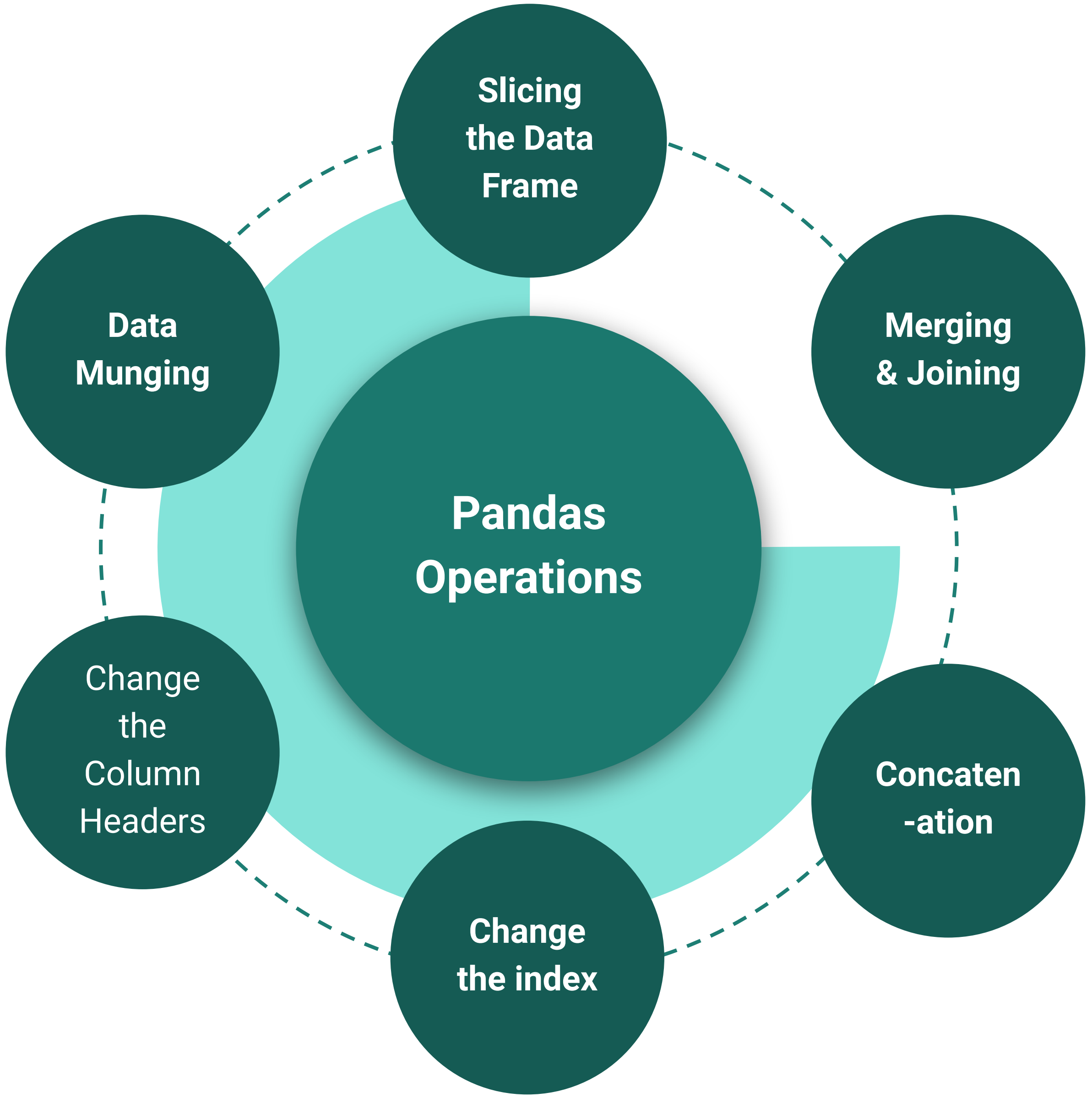
❖ Pandas

Pros:

- Widely used for data manipulation
- Simple, intuitive syntax
- Integrated well with other Python tools including visualization libraries
- Support for common data formats (read from SQL databases, CSV files, etc.)

Drawbacks:

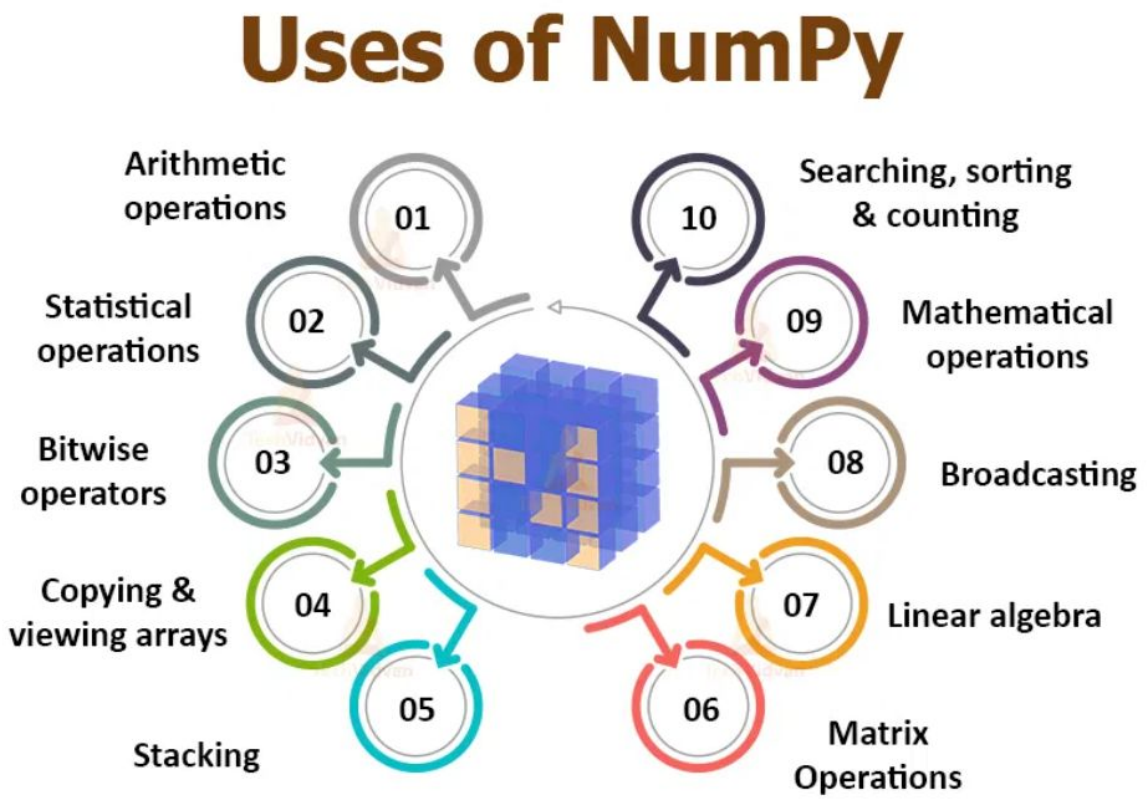
- Since it loads all data into memory, it isn't scalable and can be a bad choice for very large (larger than memory) datasets



❖ Numpy

NumPy stands for ‘Numerical Python’.

- It is a package in Python to work with arrays.
- It is a basic scientific library.
- Its most important feature is the n-dimensional array object.
- It has uses in statistical functions, linear algebra, arithmetic operations, bitwise operations, etc.
- We perform all the operations on the array elements. We can initialize these arrays in several ways.



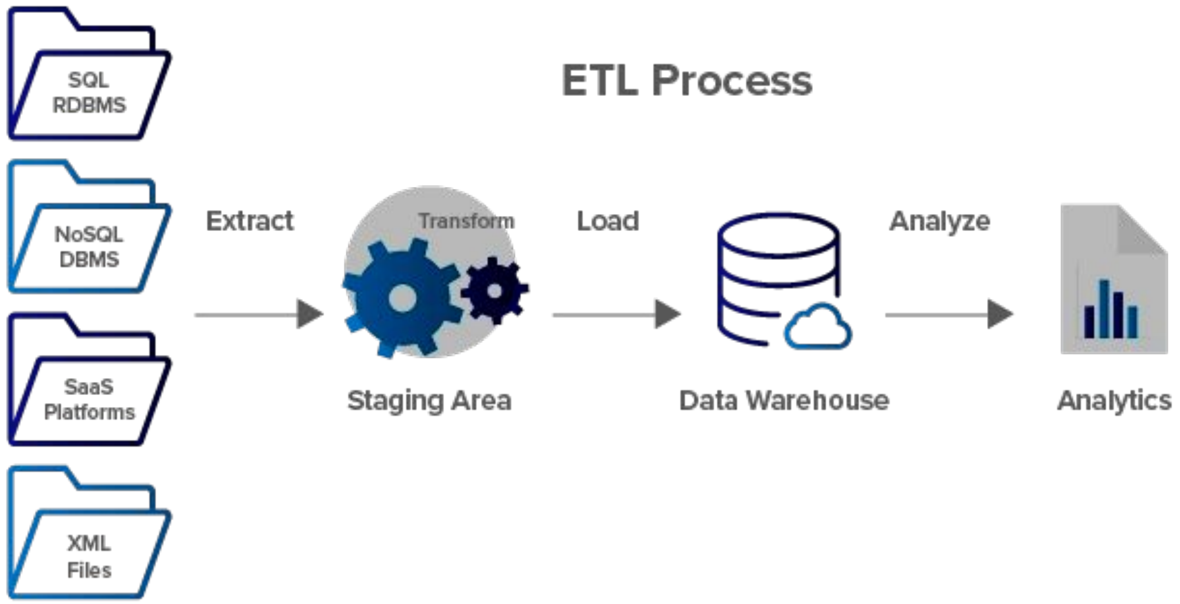
❖ Agenda for Today's Class

- ❖ Recap of previous class learnings
- ❖ New Topics for Today's class
  - ➢ ETL Process
    - ■ ETL using Python
  - ➢ File Handling
  - ➢ Map-Reduce-Filter Function
  - ➢ Exception Handling
    - ■ Mechanism of Exception Handling
  - ➢ Web Scraping with Python
    - ■ Demo: Web data scraping

❖ ETL Process

- Extract, Transform and Load – ETL, forms the fundamental process behind any kind of data management.

- ETL – Extraction, Transformation and Loading, is a trilogy of processes that collects varied source data from heterogeneous databases and transforms them into disparate data warehouses.



❖ ETL Process

- Extract:
  - Reads data from multiple data sources and extracts required set of data
  - Recovers necessary data with optimum usage of resources
  - Should not disturb data sources, performance and functioning

- Transform:
  - Filtration, cleansing and preparation of data extracted, with lookup tables
  - Authentication of records, refutation and integration of data
  - Data to be sorted, filtered, cleared, standardized, translated or verified for consistency

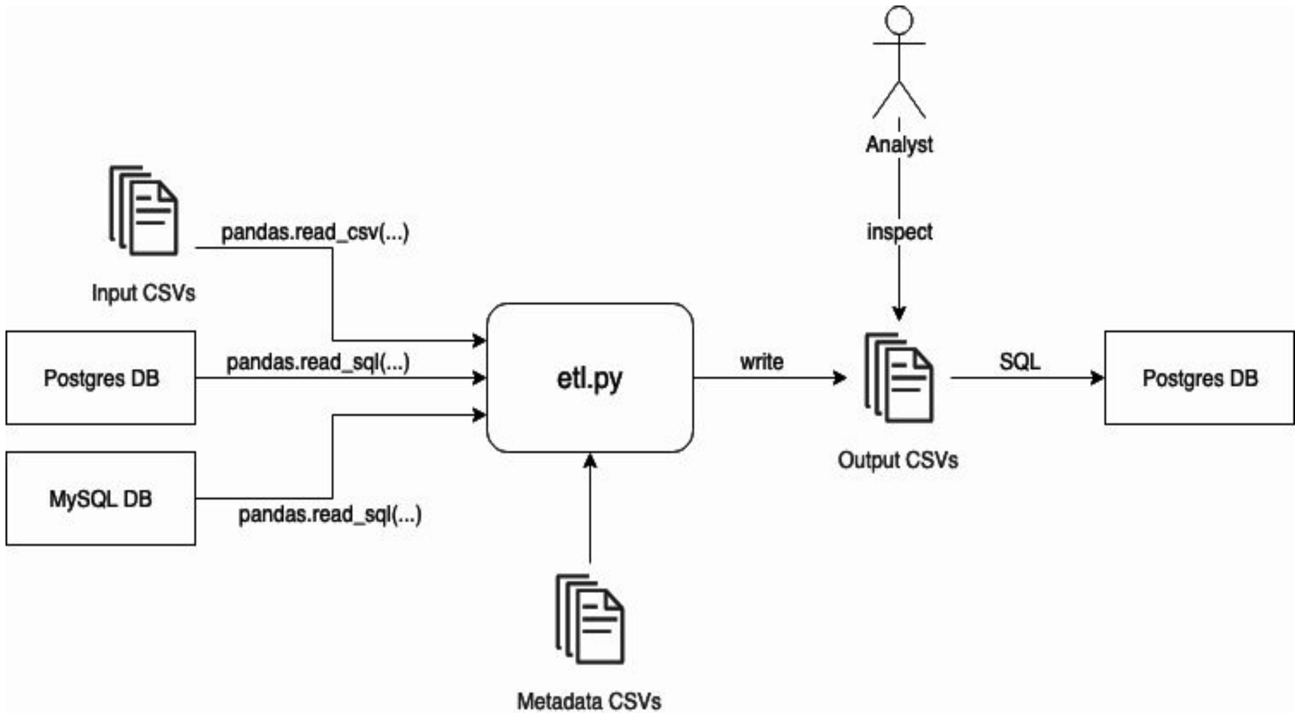
- Load:
  - Writing data output, after extraction and transformation to a data warehouse
  - Either physical insertion of record as a new row in database table or link processes for each record from the main source

❖ ETL using Python

❖ Useful Pandas functions

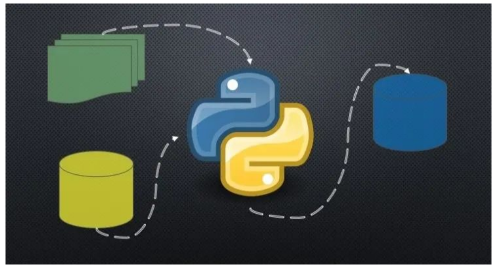
Most of ETL code revolve around using the following functions (not limited to below):

- `drop_duplicates`
- `dropna`
- `replace` / `fillna`
- `df[df['column'] != value]:` filtering
- `apply:` transform, or adding new column
- `merge:` SQL like inner, left, or right join
- `groupby`
- `read_csv` / `to_csv`
- Functions like `drop_duplicates` and `drop_na` are nice abstractions and save tens of SQL statements.
- `replace` / `fillna` is a typical step to manipulate the data array.



❖ Tools for ETL Using Python

- **Petl** (Python ETL):



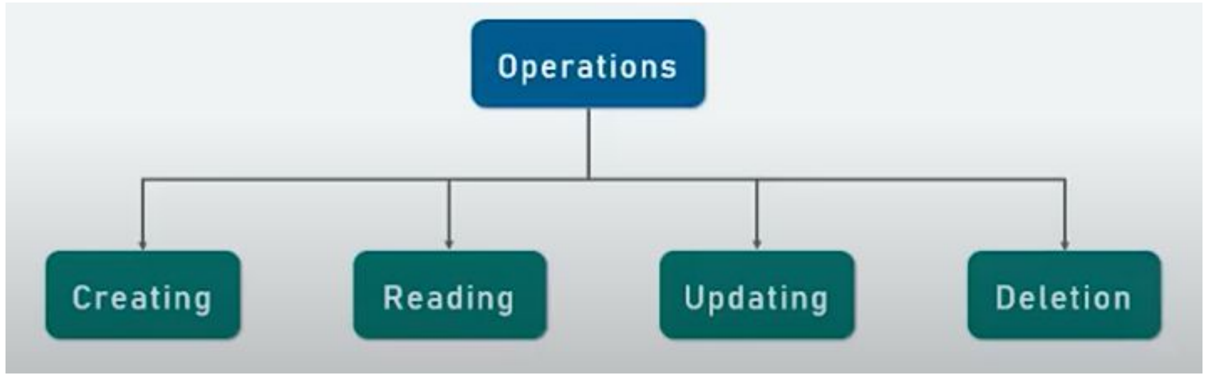
https://petl.readthedocs.io/en/stable/

- **Pandas**

- **Apache Airflow**

❖ File Handling in Python

- ❖ A file is a source of storing information permanently in the form of a sequence of bytes on a disk.
- ❖ The operations performed like creating a file, opening a file, reading a file or manipulating data inside a file is referred to as file handling.
- ❖ Steps used during File Handling in Python.
  - o Opening a file to write
  - o Appending and writing to a file
  - o Closing a file
  - o Creating a file



❖ File Handling in Python

❖ **Syntax**:

```
Object = open(file_name, mode)
```

The open function returns the instance of the file.

It takes 2 primarily arguments: `file_name` and `mode`

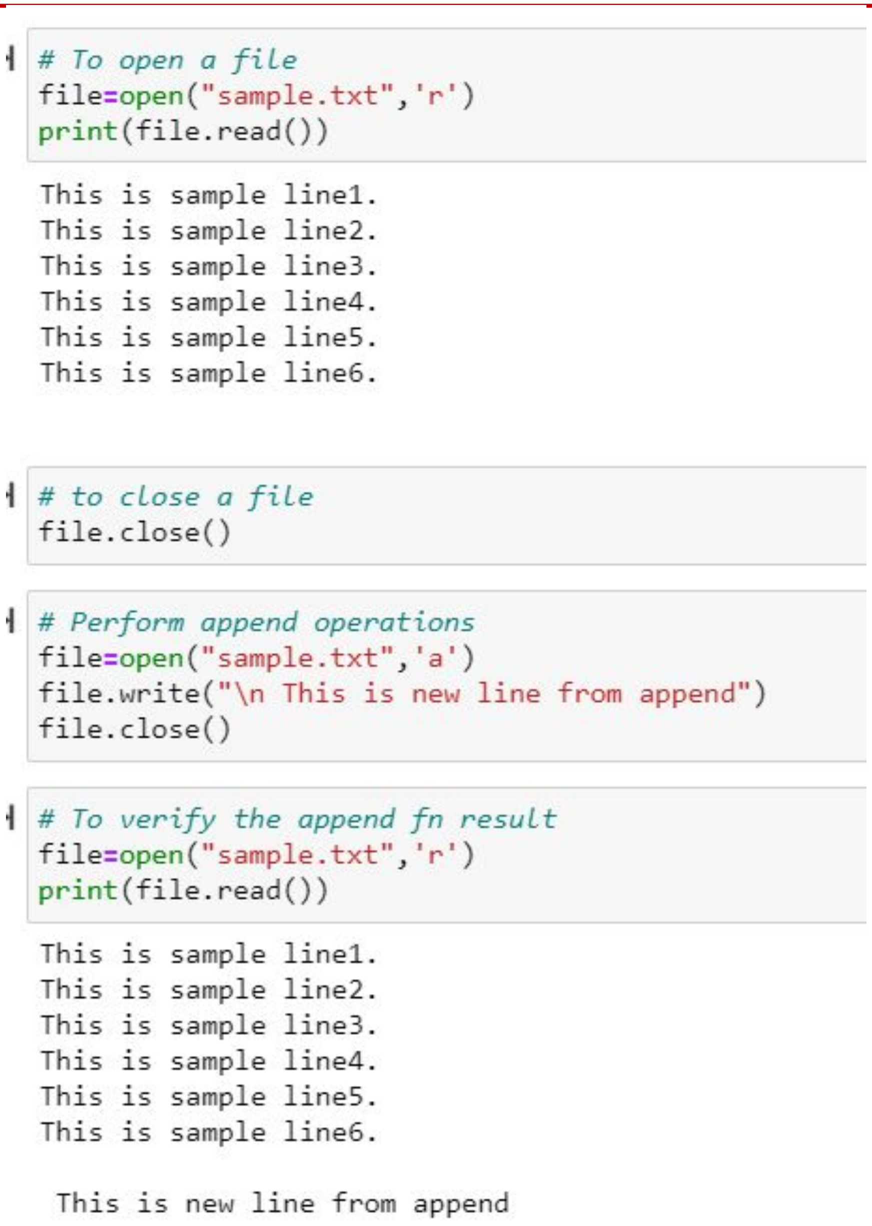
❖ **Argument Mode:**

"r" = Read from a file.
"w" = Writing to a file with erasing previous data.
"a" = Append to previously written file.
"x" = Create a file
"t" = Text file, Default value. – Used to specify type of file
"b" = binary file. For eg. Images - Used to specify type of file

```
file=open("sample2.txt",'x')         # To create a file
file=open("sample2.txt")             # To open a file in read mode
print(file.read())                   # Print the content of file
file=open("sample2.txt",'a')         # To open a file in append mode
file.write("\n This is new line2")   # To append content to file
file.close()                         # To close the file
file=open("sample2.txt",'w')         # To open file in write mode
file.write("This is write cmd")      # To write the content
file.close()
image=open("image2.png",'rb')        # To open a binary file
print(image.read())                  # To read content
image.close()
```

❖ File Handling in Python

Example code:



```python
# To open a file
file=open("sample.txt",'r')
print(file.read())
```

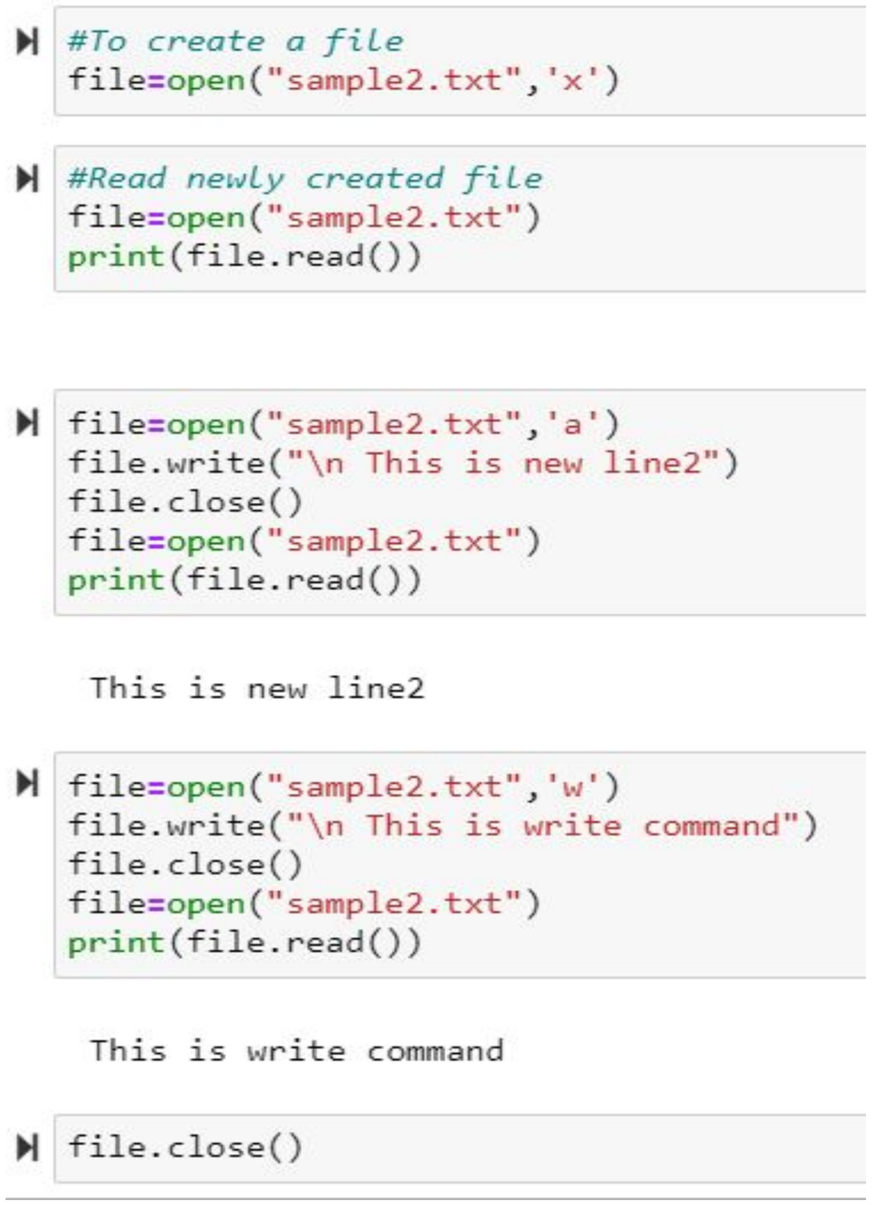
```
This is sample line1.
This is sample line2.
This is sample line3.
This is sample line4.
This is sample line5.
This is sample line6.
```

```python
# to close a file
file.close()
```

```python
# Perform append operations
file=open("sample.txt",'a')
file.write("\n This is new line from append")
file.close()
```

```python
# To verify the append fn result
file=open("sample.txt",'r')
print(file.read())
```

```
This is sample line1.
This is sample line2.
This is sample line3.
This is sample line4.
This is sample line5.
This is sample line6.
 This is new line from append
```



```python
#To create a file
file=open("sample2.txt",'x')
```

```python
#Read newly created file
file=open("sample2.txt")
print(file.read())
```

```python
file=open("sample2.txt",'a')
file.write("\n This is new line2")
file.close()
file=open("sample2.txt")
print(file.read())
```

```

 This is new line2
```

```python
file=open("sample2.txt",'w')
file.write("\n This is write command")
file.close()
file=open("sample2.txt")
print(file.read())
```

```

 This is write command
```

```python
file.close()
```

❖ Map-Reduce-Filter Functions

- Map, Filter, and Reduce are paradigms of functional programming. They allow the you to write simpler, shorter code, without necessarily needing to bother about intricacies like loops and branching.

- All three of these are convenience functions that can be replaced with List Comprehensions or loops, but provide a more elegant and short-hand approach to some problems.

❖ Map-Reduce-Filter Functions

- Before continuing, one this you should be familiar with before reading about the Map-Reduce-Filter methods:
- *What is an anonymous function/method or lambda?*
-
```
lambda arguments: expression
```
- Think of lambdas as one-line methods without a name. They work practically the same as any other method in Python, for example:
-
```
lambda x, y: x + y
```
- Lambdas differ from normal Python methods because they can have only one expression, can't contain any statements and their return type is a `function` object. So the line of code above doesn't exactly return the value `x + y` but the function that calculates `x + y.`

❖ Map-Reduce-Filter Functions

- *Why are lambdas relevant to `map()`, `filter()` and `reduce()`?*

- All three of these methods expect a `function` object as the first argument. This function object can be a pre-defined method with a name (like `def add(x,y)`).

- Though, more often than not, functions passed to `map()`, `filter()`, and `reduce()` are the ones you'd use only once, so there's often no point in defining a referenceable function.

- To avoid defining a new function for your different `map()/filter()/reduce()` needs - a more elegant solution would be to use a short, disposable, anonymous function that you will only use once and never again - a lambda.

❖ Map-Reduce-Filter Functions

- MAP(): The basic function of map() is to manipulate iterables. Map executes all the conditions of a function on the items in the iterable

```
Syntax: map(function_to_apply, iterables) list(map(lambda x: x*2, range(0,5)))
```

- FILTER(): It is used to filter the iterables as per the conditions and It creates a list of elements for which a function returns true.

```
Syntax: filter(function_to_apply, list_of_inputs) list(filter(lambda x:x>2, range(-1,5)))
```

- Reduce(): It is a useful function for performing some computation on a list and returning the result. It applies a rolling computation to sequential pairs of values in a list.

```
Syntax: reduce(function_to_apply, list_of_inputs) list(reduce(lambda a,b: a+b,
range(0,6)))
```

Example:

```
data = reduce(lambda x,y: x+y,map(lambda x:2*x,filter(lambda x: (x>=1), range(-5,5))))
print(data)
```

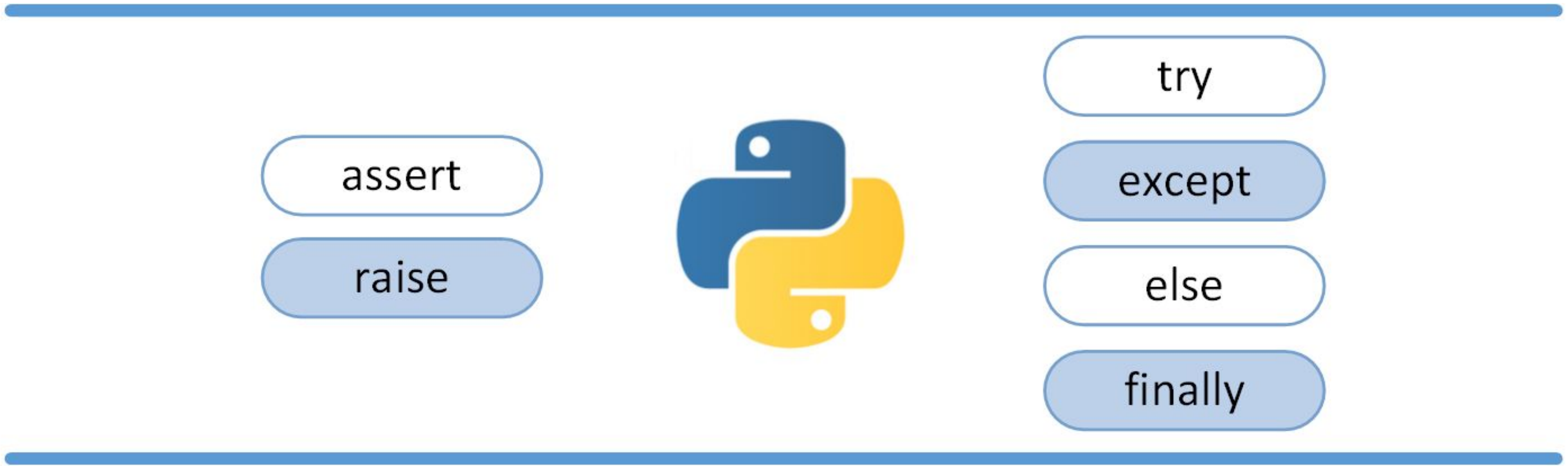
❖ Exception Handling

- A Python program terminates as soon as it encounters an error. In Python, an error can be a syntax error or an exception.

- In today's class, we will see what an exception is and how it differs from a syntax error.

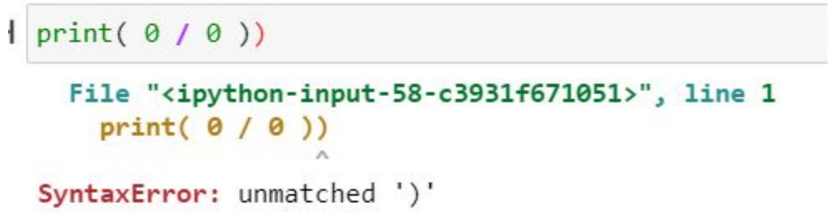
- After that, we will learn about raising exceptions and making assertions.

- Then, we'll finish with a demonstration of the try and except block.



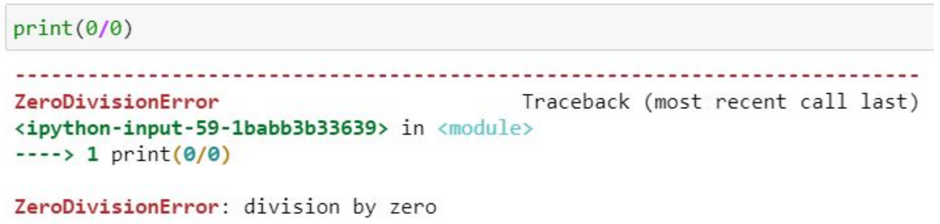
❖ Exception Handling

- **Syntax Errors** : occurs when the parser detects an incorrect statement.



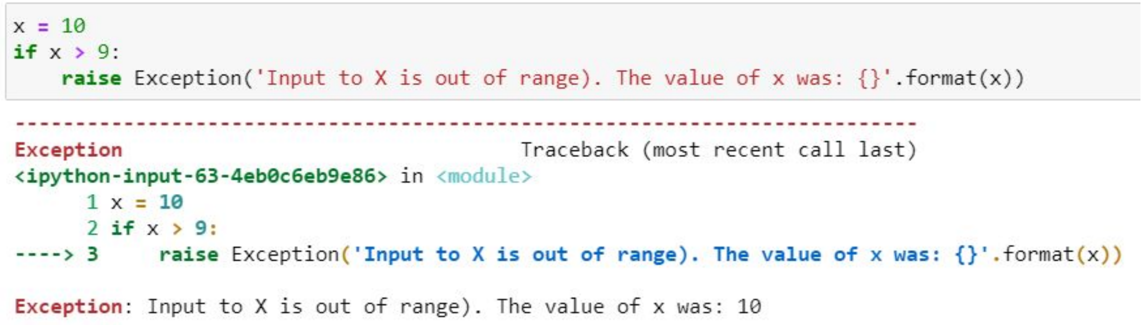
```
print( 0 / 0 ))

  File "<ipython-input-58-c3931f671051>", line 1
    print( 0 / 0 ))
                  ^
SyntaxError: unmatched ')'
```

- **Exception Error**: Error occurs whenever syntactically correct Python code results in an error



```
print(0/0)

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-59-1babb3b33639> in <module>
----> 1 print(0/0)

ZeroDivisionError: division by zero
```

- **Raising an Exception**
  use raise to throw an exception if a condition occurs. The statement can be complemented with a custom exception.



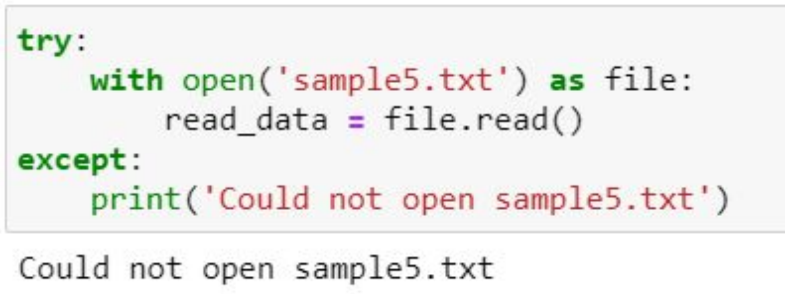
```
x = 10
if x > 9:
    raise Exception('Input to X is out of range). The value of x was: {}'.format(x))

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-63-4eb0c6eb9e86> in <module>
      1 x = 10
      2 if x > 9:
----> 3     raise Exception('Input to X is out of range). The value of x was: {}'.format(x))

Exception: Input to X is out of range). The value of x was: 10
```

❖ Mechanism of Exception Handling

❖ Try and Except

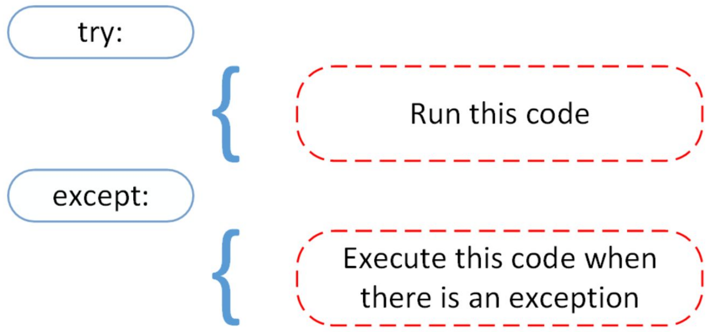
The try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a “normal” part of the program. The code that follows the except statement is the program’s response to any exceptions in the preceding try clause.

Example:



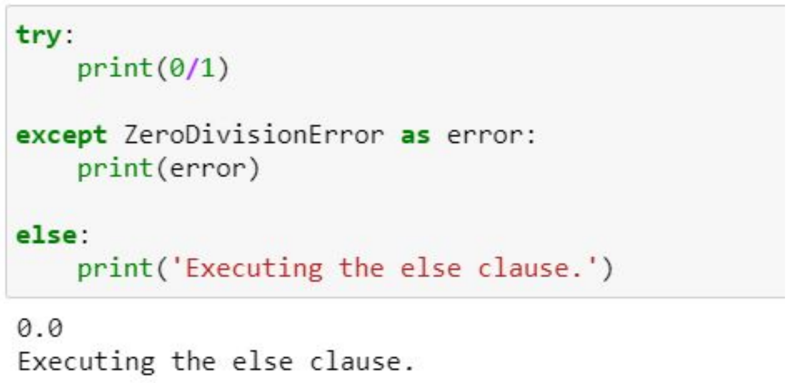
```
try:
    with open('sample5.txt') as file:
        read_data = file.read()
except:
    print('Could not open sample5.txt')
```

```
Could not open sample5.txt
```



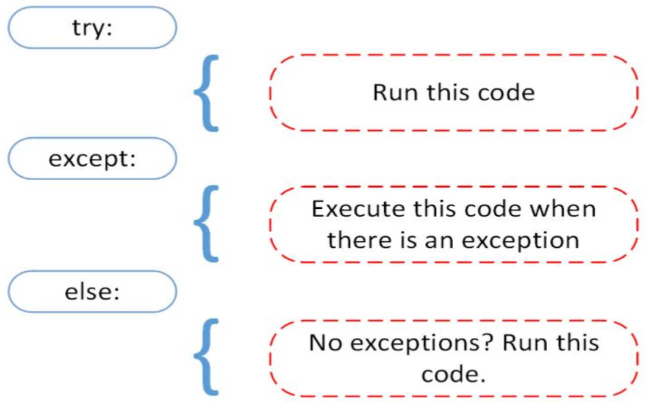
❖ The Else Clause

Using the else statement, you can instruct a program to execute a certain block of code only in the absence of exceptions.



```
try:
    print(0/1)

except ZeroDivisionError as error:
    print(error)

else:
    print('Executing the else clause.')
```

```
0.0
Executing the else clause.
```



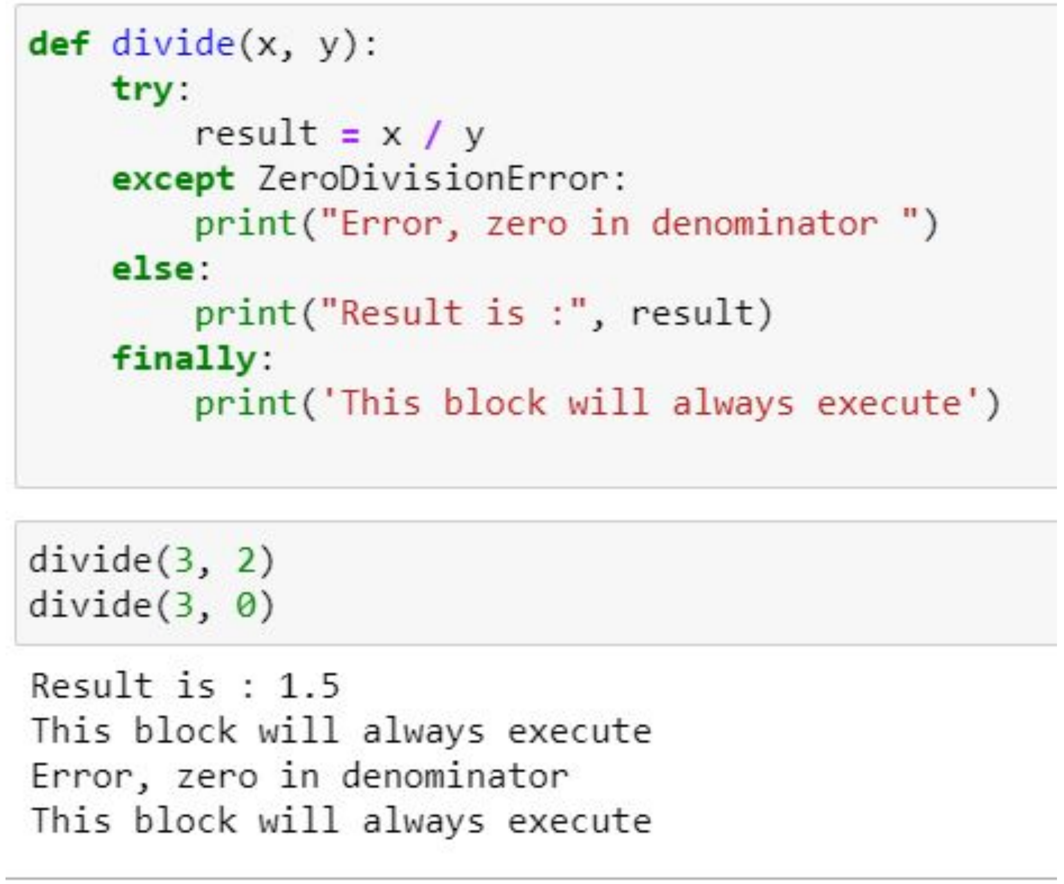
try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.

❖ Mechanism of Exception Handling

❖ **Using finally:**

Finally block always gets executed either exception is generated or not.

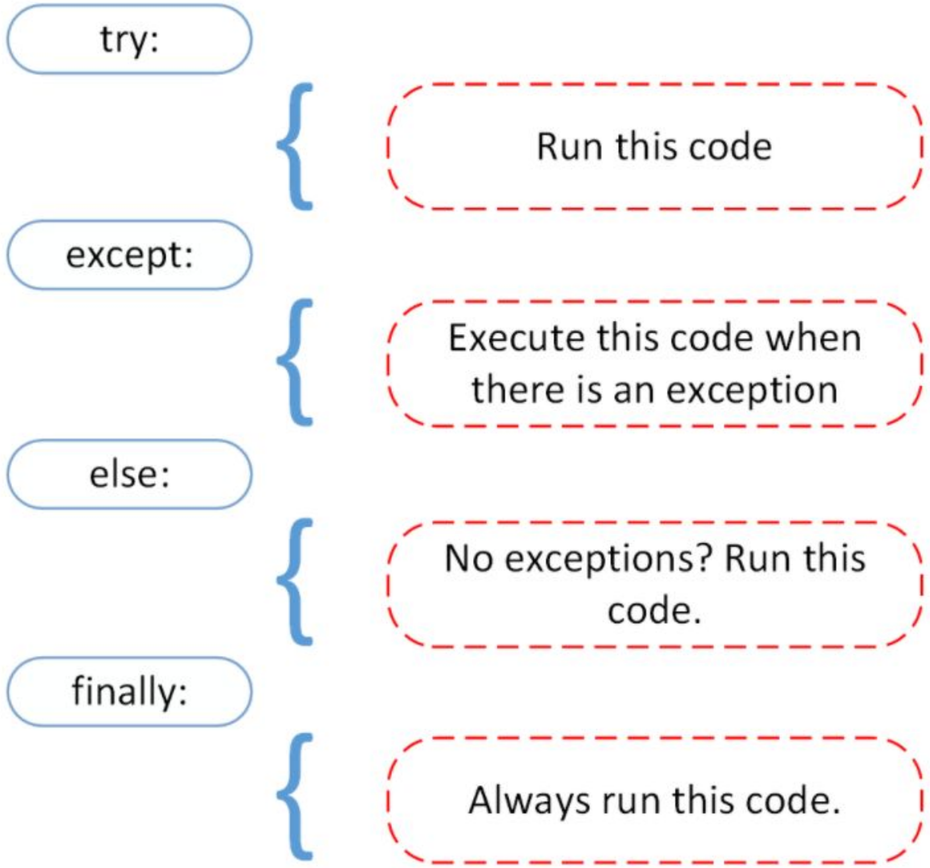
Example:



```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error, zero in denominator ")
    else:
        print("Result is :", result)
    finally:
        print('This block will always execute')
```

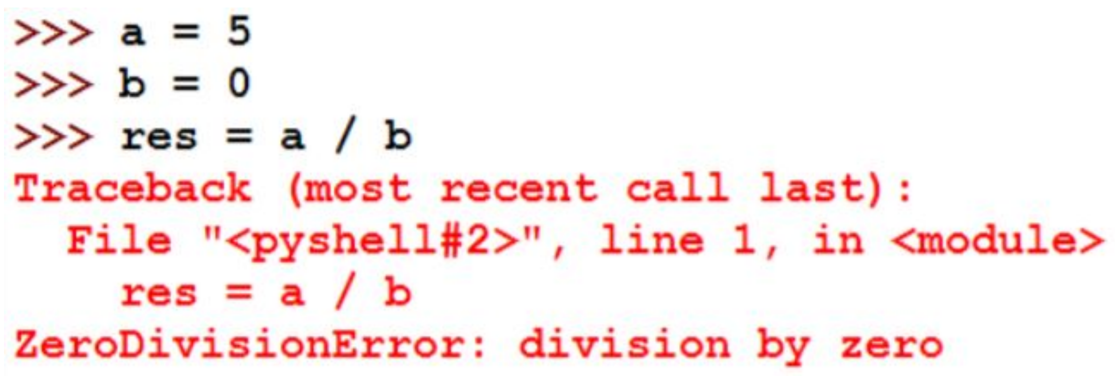
```
divide(3, 2)
divide(3, 0)
```

```
Result is : 1.5
This block will always execute
Error, zero in denominator
This block will always execute
```


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

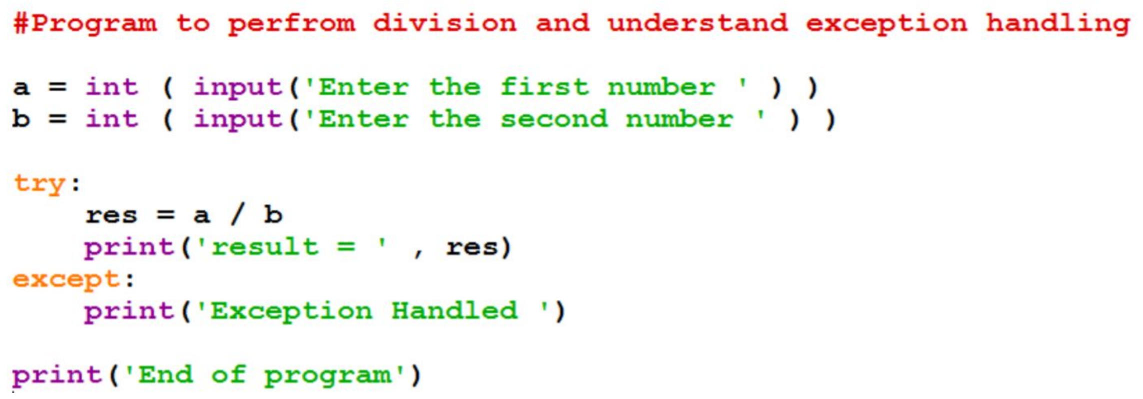
❖ Exception Handling – Examples

Exception



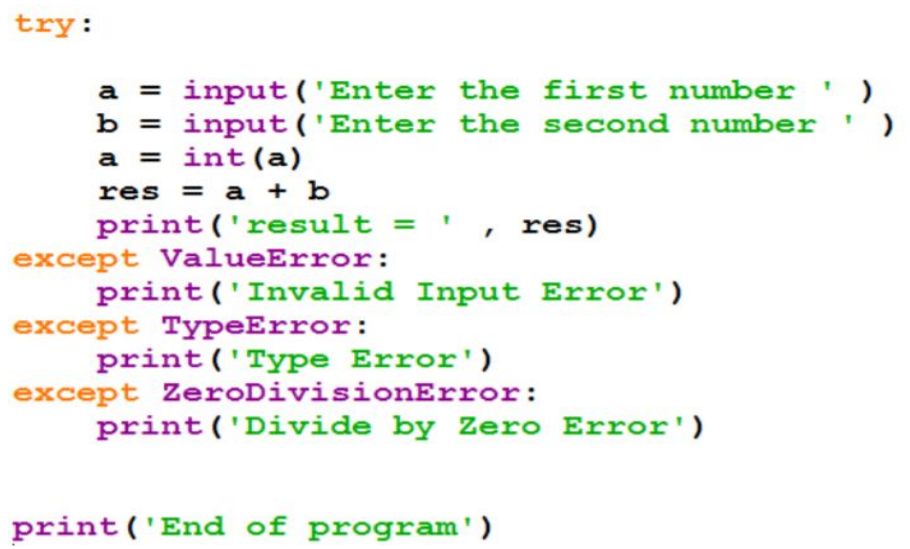
```
>>> a = 5
>>> b = 0
>>> res = a / b
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    res = a / b
ZeroDivisionError: division by zero
```

Try /Except



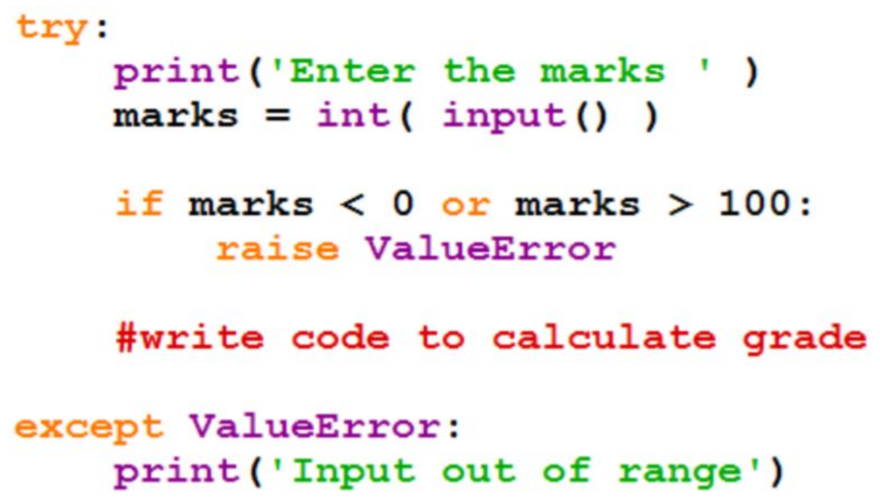
```
#Program to perfrom division and understand exception handling

a = int ( input('Enter the first number ' ) )
b = int ( input('Enter the second number ' ) )

try:
    res = a / b
    print('result = ' , res)
except:
    print('Exception Handled ')

print('End of program')
```

Nested Try



```
try:

    a = input('Enter the first number ' )
    b = input('Enter the second number ' )
    a = int(a)
    res = a + b
    print('result = ' , res)
except ValueError:
    print('Invalid Input Error')
except TypeError:
    print('Type Error')
except ZeroDivisionError:
    print('Divide by Zero Error')


print('End of program')
```

Raising Exception



```
try:
    print('Enter the marks ' )
    marks = int( input() )

    if marks < 0 or marks > 100:
        raise ValueError

    #write code to calculate grade

except ValueError:
    print('Input out of range')
```

❖ Web Scraping with Python

Imagine you have to pull a large amount of data from websites and you want to do it as quickly as possible. How would you do it without manually going to each website and getting the data?

Well, “Web Scraping” is the answer. Web Scraping just makes this job easier and faster.

Topics to Cover:

- Why is Web Scraping Used?
- What Is Web Scraping?
- Is Web Scraping Legal?
- Why is Python Good For Web Scraping?
- How Do You Scrape Data From A Website?
- Libraries used for Web Scraping
- Web Scraping Example : Scraping Amazon Website

❖ Why is Web Scraping Used?

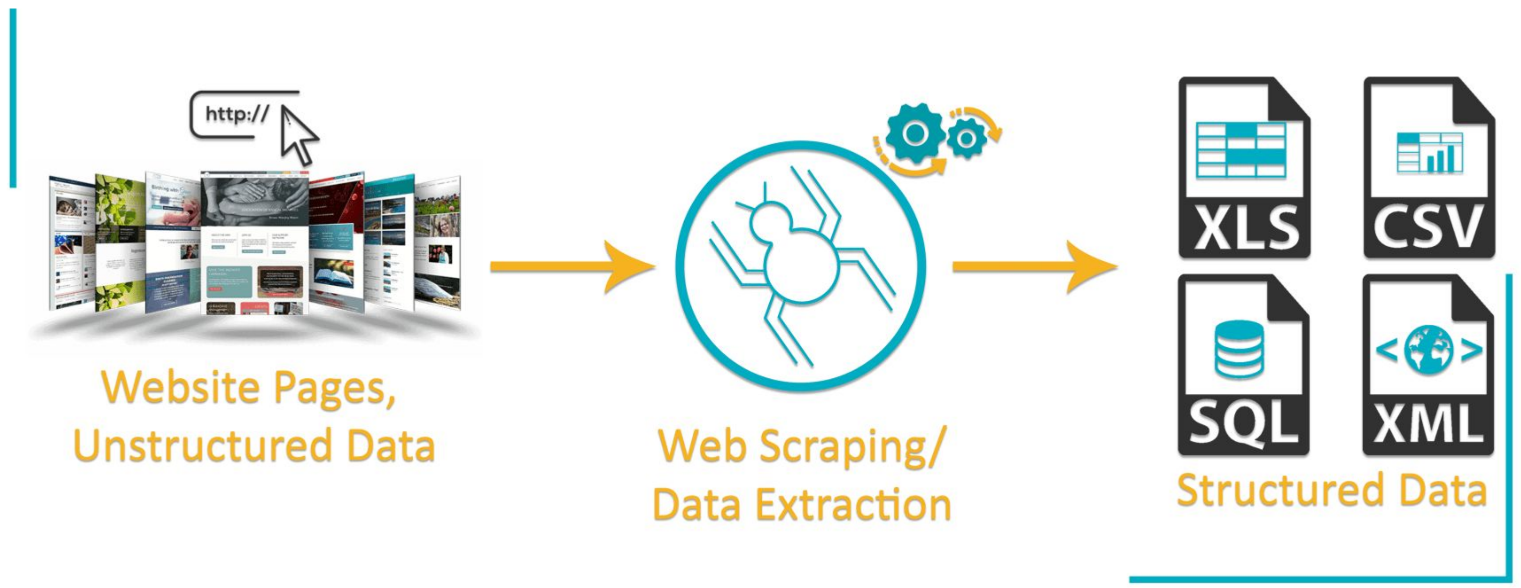
Web scraping is used to collect large information from websites. But why does someone have to collect such large data from websites? To know about this, let's look at the applications of web scraping:

- Price Comparison: Services such as ParseHub use web scraping to collect data from online shopping websites and use it to compare the prices of products.
- Email address gathering: Many companies that use email as a medium for marketing, use web scraping to collect email ID and then send bulk emails.
- Social Media Scraping: Web scraping is used to collect data from Social Media websites such as Twitter to find out what's trending.
- Research and Development: Web scraping is used to collect a large set of data (Statistics, General Information, Temperature, etc.) from websites, which are analyzed and used to carry out Surveys or for R&D.
- Job listings: Details regarding job openings, interviews are collected from different websites and then listed in one place so that it is easily accessible to the user.

❖ What is Web Scraping?

Web scraping is an automated method used to extract large amounts of data from websites. The data on the websites are unstructured. Web scraping helps collect these unstructured data and store it in a structured form. There are different ways to scrape websites such as online Services, APIs or writing your own code.

Beautiful Soup is a pure Python library for extracting structured data from a website. It allows you to parse data from HTML and XML files. It acts as a helper module and interacts with HTML in a similar and better way as to how you would interact with a web page using other available developer tools.


http://
Website Pages, Unstructured Data
Web Scraping/ Data Extraction
XLS
CSV
SQL
XML
Structured Data

❖ Is Web Scraping Legal?

Talking about whether web scraping is legal or not, some websites allow web scraping and some don't. To know whether a website allows web scraping or not, you can look at the website's "robots.txt" file. You can find this file by appending "/robots.txt" to the URL that you want to scrape.

For this example, I am scraping Amazon website.So, to see the "robots.txt" file, the URL is www.amazon.com/robots.txt

A must read article:
https://www.jdsupra.com/legalnews/supreme-court-grants-certiorari-in-web-5488247/

❖ How Do You Scrape Data From A Website?

When you run the code for web scraping, a request is sent to the URL that you have mentioned. As a response to the request, the server sends the data and allows you to read the HTML or XML page. The code then, parses the HTML or XML page, finds the data and extracts it.

To extract data using web scraping with python, you need to follow these basic steps:

- Find the URL that you want to scrape
- Inspecting the Page
- Find the data you want to extract
- Write the code
- Run the code and extract the data
- Store the data in the required format

❖ Web Scraping Demo

Let's understand it with an example:

- Basic code to understand web Scraping

Web Scrapping from wikipedia website.ipynb

- Advanced Code

Web_Scrapping-Amazon_Reviews.ipynb

❖ Application in real life - Demo

How to create a web scraping job to report the content of website with keeping daily logging activity and generating separate data file for analysis?

Solution:

- Create a class to reuse the code to append the status of application everytime it runs.
- Create a web scraping program to fetch contents from website
- Write the required content to a seperate data file everyday
- Let’s Code

ant wak

Questions