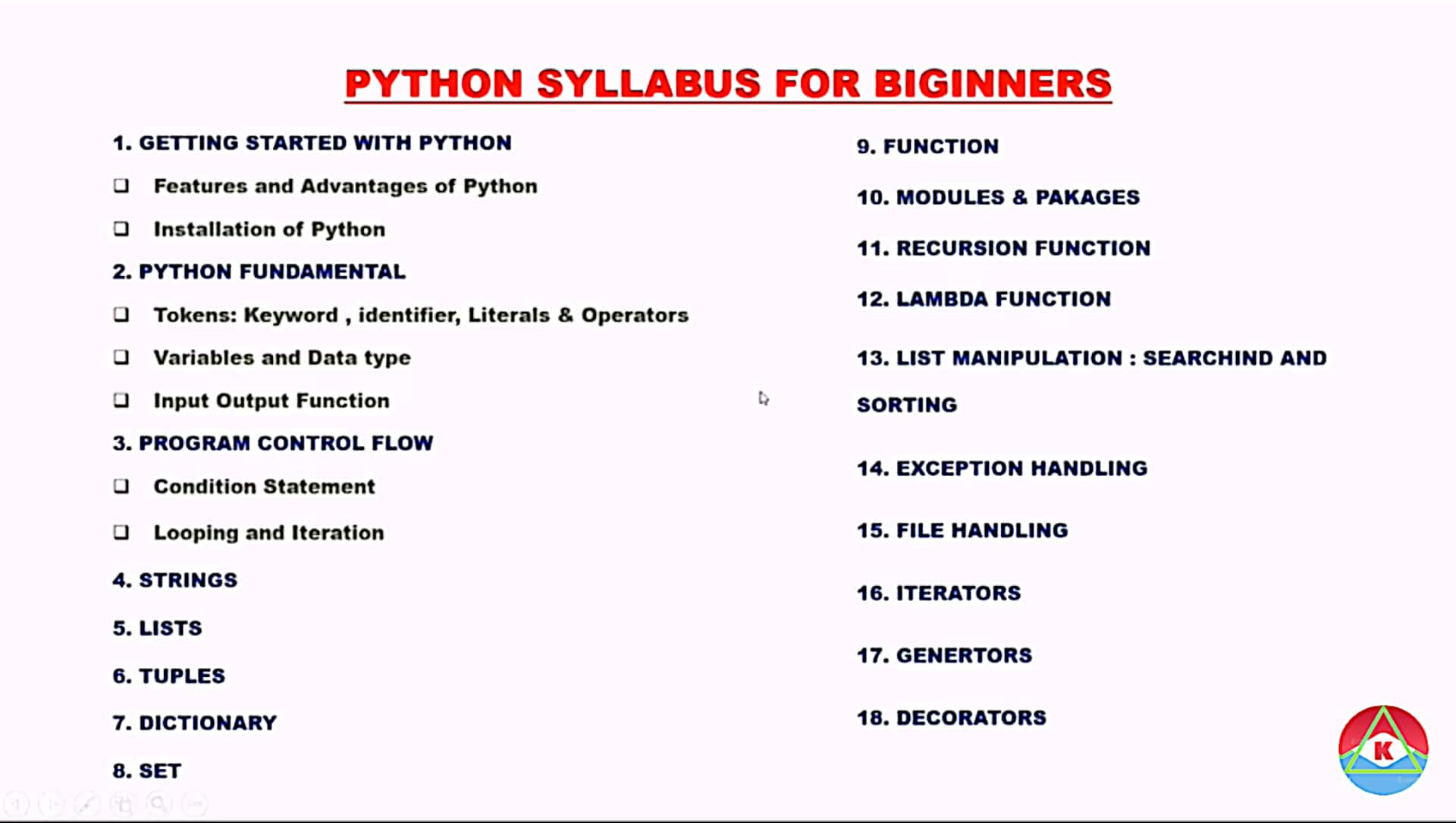
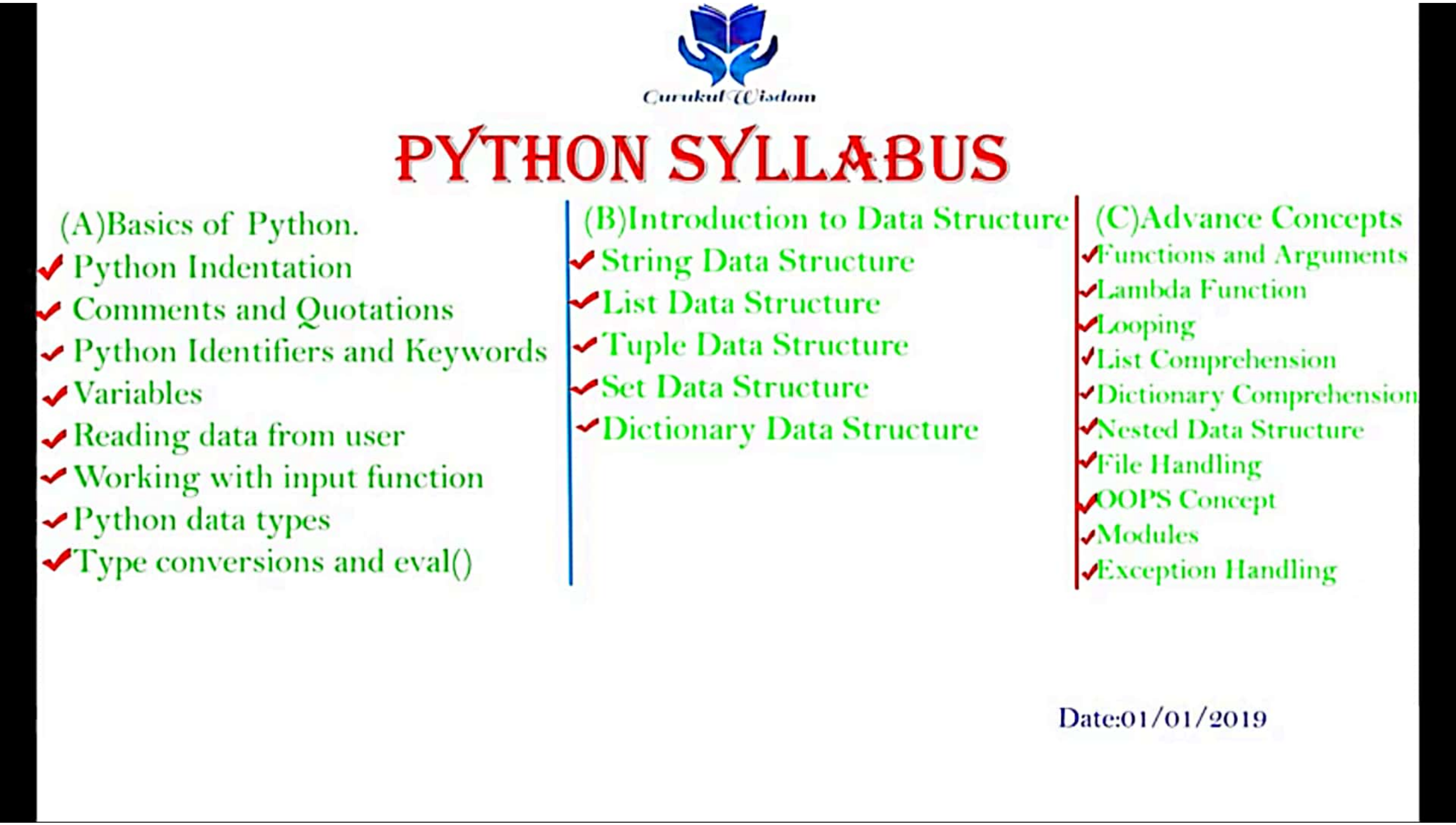


PYTHON SYLLABUS FOR BIGINNERS

1. **GETTING STARTED WITH PYTHON**
   - ❑ Features and Advantages of Python
   - ❑ Installation of Python
2. **PYTHON FUNDAMENTAL**
   - ❑ Tokens: Keyword , identifier, Literals & Operators
   - ❑ Variables and Data type
   - ❑ Input Output Function
3. **PROGRAM CONTROL FLOW**
   - ❑ Condition Statement
   - ❑ Looping and Iteration
4. **STRINGS**
5. **LISTS**
6. **TUPLES**
7. **DICTIONARY**
8. **SET**
9. **FUNCTION**
10. **MODULES & PAKAGES**
11. **RECURSION FUNCTION**
12. **LAMBDA FUNCTION**
13. **LIST MANIPULATION : SEARCHIND AND SORTING**
14. **EXCEPTION HANDLING**
15. **FILE HANDLING**
16. **ITERATORS**
17. **GENERTORS**
18. **DECORATORS**



Gurukul Wisdom

PYTHON SYLLABUS

(A)Basics of Python.

- ✓ Python Indentation
- ✓ Comments and Quotations
- ✓ Python Identifiers and Keywords
- ✓ Variables
- ✓ Reading data from user
- ✓ Working with input function
- ✓ Python data types
- ✓ Type conversions and eval()

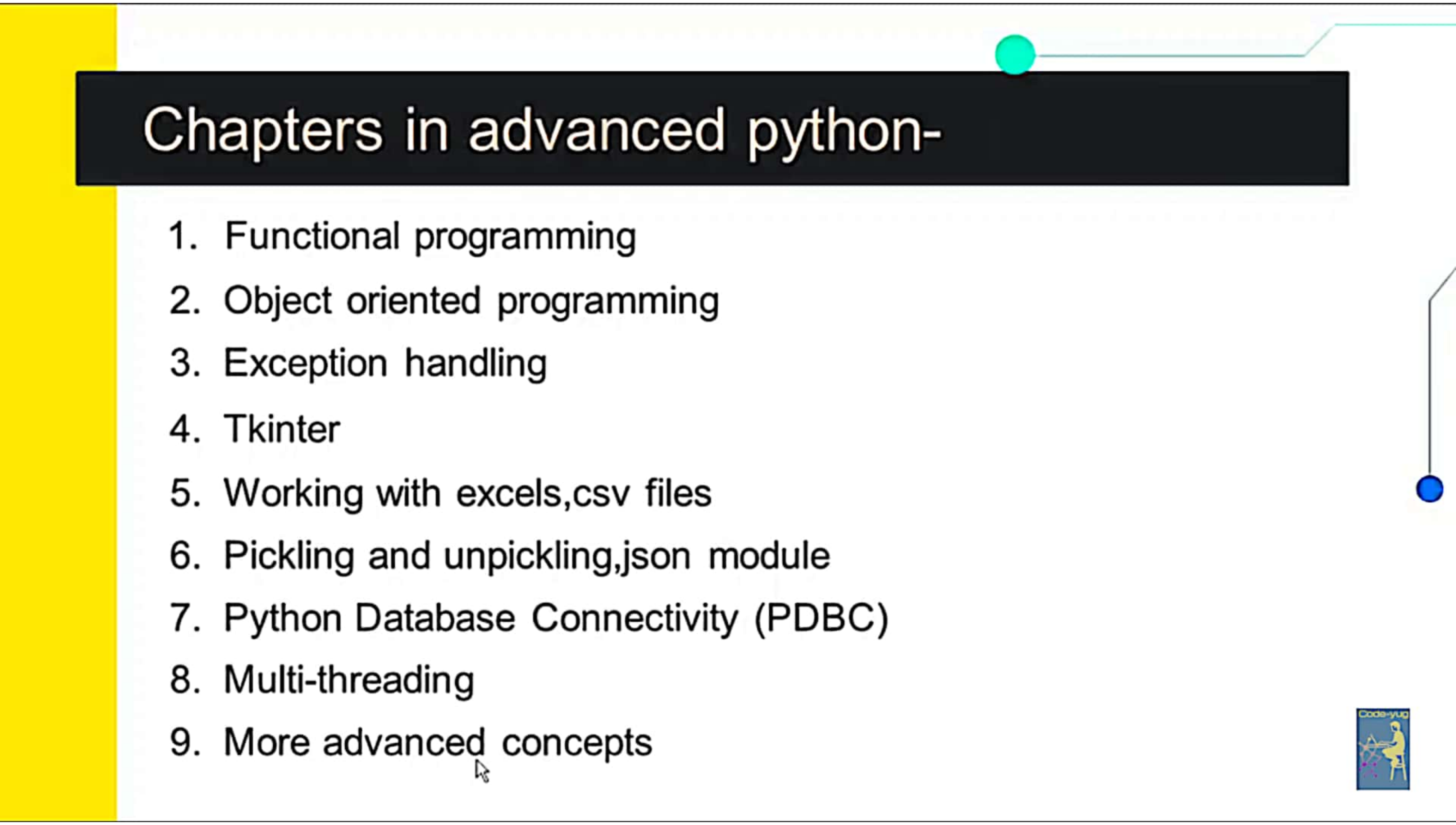
(B)Introduction to Data Structure

- ✓ String Data Structure
- ✓ List Data Structure
- ✓ Tuple Data Structure
- ✓ Set Data Structure
- ✓ Dictionary Data Structure

(C)Advance Concepts

- ✓ Functions and Arguments
- ✓ Lambda Function
- ✓ Looping
- ✓ List Comprehension
- ✓ Dictionary Comprehension
- ✓ Nested Data Structure
- ✓ File Handling
- ✓ OOPS Concept
- ✓ Modules
- ✓ Exception Handling

Date:01/01/2019



Chapters in advanced python-

1. Functional programming
2. Object oriented programming
3. Exception handling
4. Tkinter
5. Working with excels,csv files
6. Pickling and unpickling,json module
7. Python Database Connectivity (PDBC)
8. Multi-threading
9. More advanced concepts

Object Oriented programming vs Procedural oriented Programming

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of *Access modifiers* 'public', private', protected'	Doesn't use *Access modifiers*
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

What are Python OOPs Concepts?

OOP (object-oriented programming)

1. Class
2. Object
3. Method
4. Inheritance
5. Polymorphism
6. Data Abstraction
7. Encapsulation.

What are Classes and Objects?

A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behavior.

Class Employees

Objects

Emp name
Emp age
Emp salary
Emp id

What are Classes and Objects?

A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behavior.

```
class class1(): // class 1 is the name of the
                          class
```

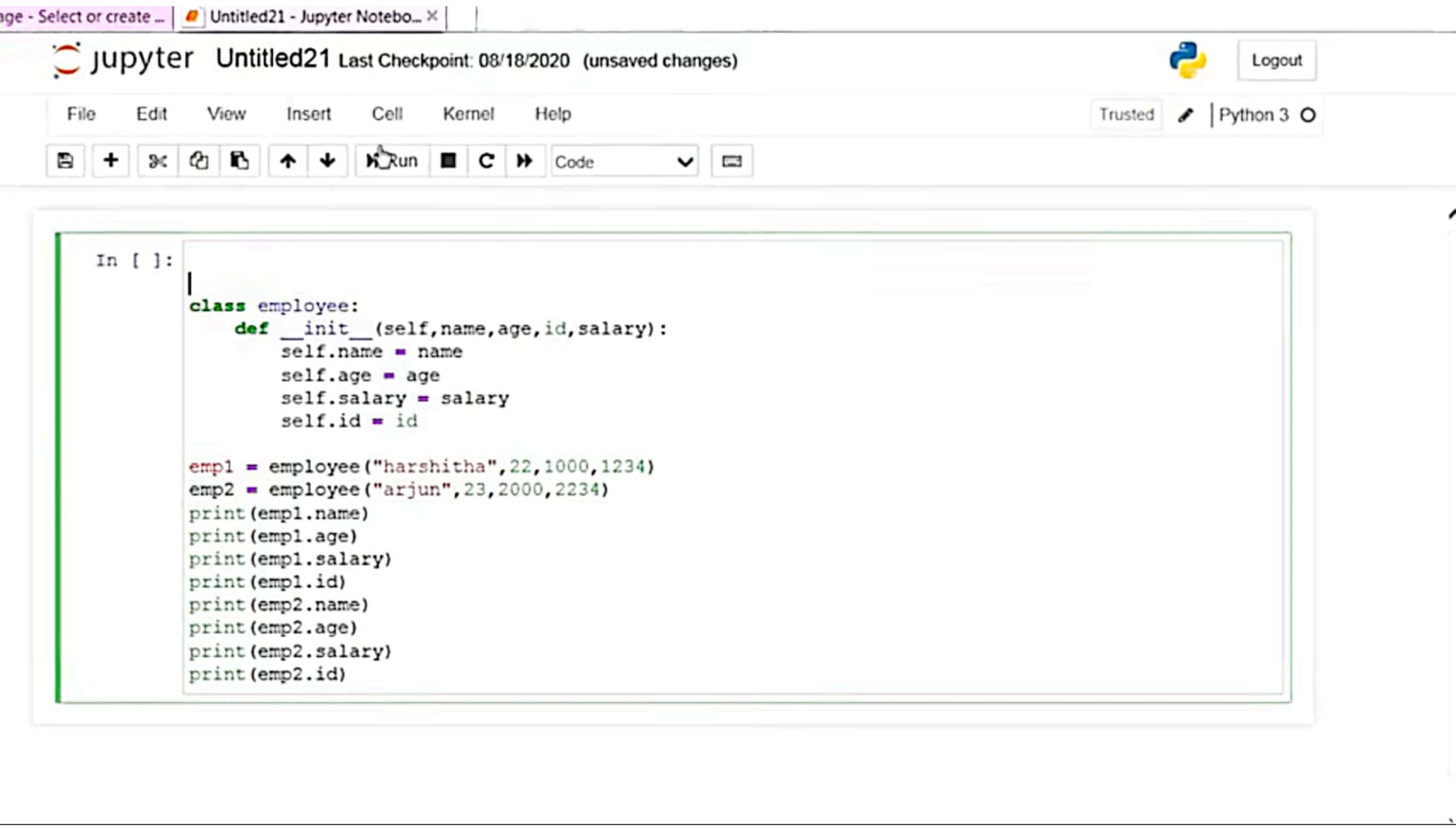
What are Objects?

Objects:

Objects are an instance of a class. It is an entity that has state and behavior. In a nutshell, it is an instance of a class that can access the data.

Syntax: obj = class1()

obj is the “object “ of class1.


age - Select or create ...
Untitled21 - Jupyter Notebo...
jupyter Untitled21 Last Checkpoint: 08/18/2020 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code

```
In [ ]:

class employee:
    def __init__(self,name,age,id,salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id

emp1 = employee("harshitha",22,1000,1234)
emp2 = employee("arjun",23,2000,2234)
print(emp1.name)
print(emp1.age)
print(emp1.salary)
print(emp1.id)
print(emp2.name)
print(emp2.age)
print(emp2.salary)
print(emp2.id)
```

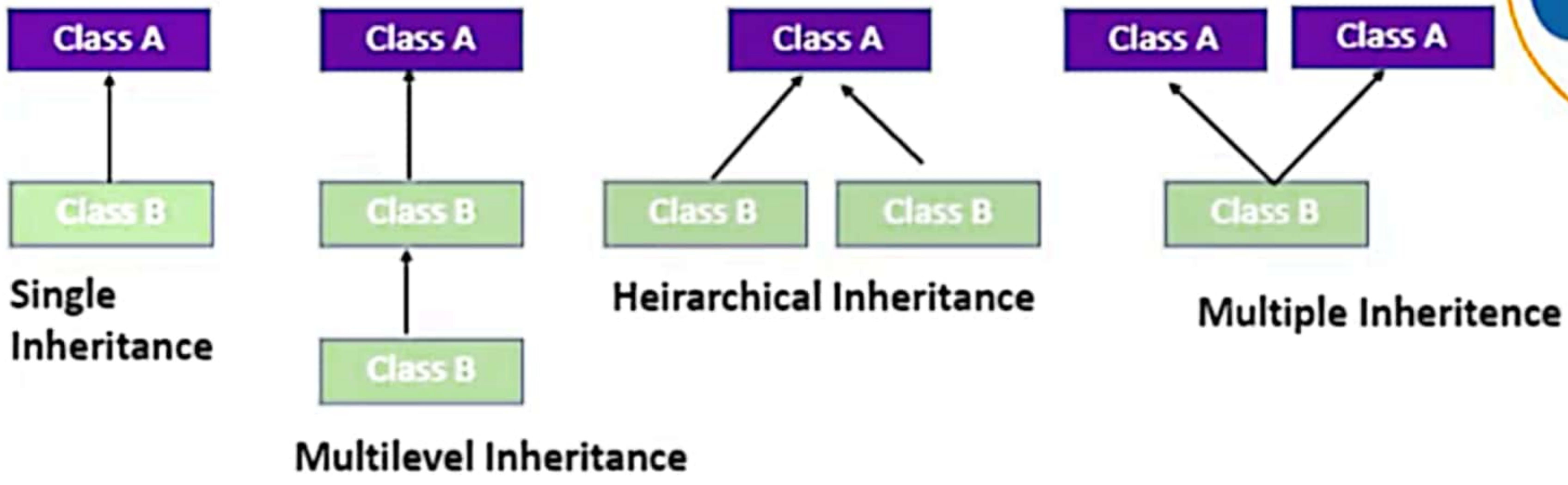
Object-Oriented Programming methodologies:

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

Inheritance

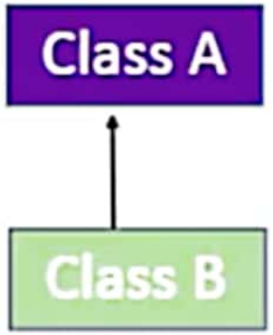
Inheriting or transfer of characteristics from parent to child class without any modification”. The new class is called the **derived/child** class and the one from which it is derived is called a **parent/base** class.

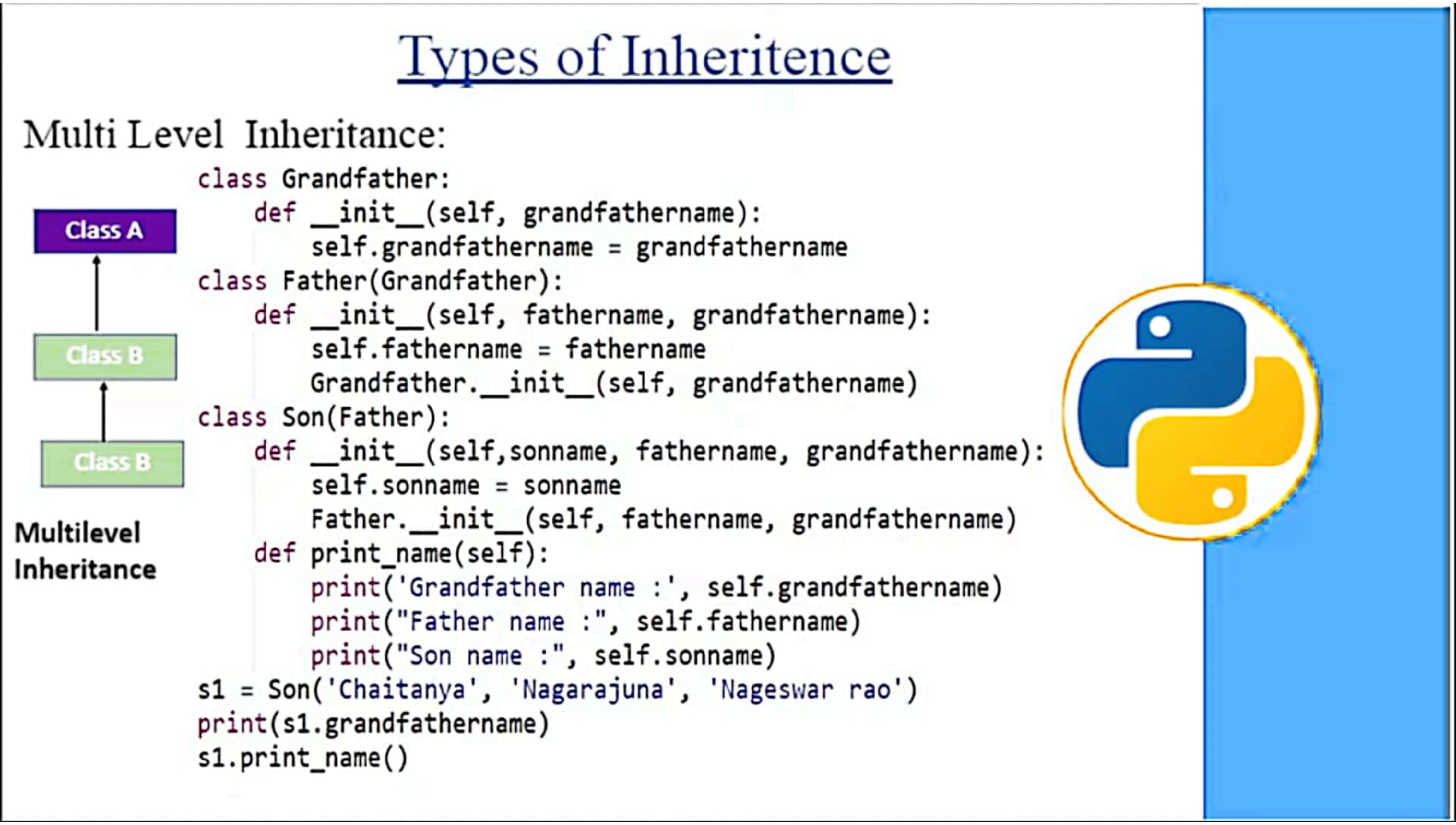
Types of Inheritence



Single Inheritance

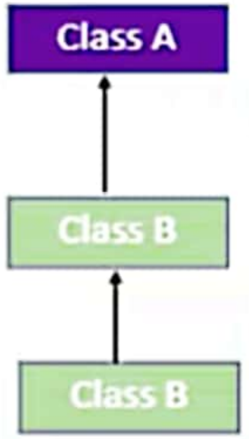
```
class Person(object):
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
    def isEmployee(self):
        return False
class Employee(Person):
    def isEmployee(self):
        return True
emp = Person("Employee1")
print(emp.getName(), emp.isEmployee())

emp = Employee("Employee2")
print(emp.getName(), emp.isEmployee())
```





Types of Inheritence

Multi Level Inheritance:

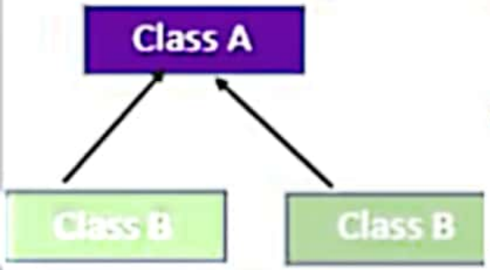


Multilevel Inheritance

```
class Grandfather:
    def __init__(self, grandfathername):
        self.grandfathername = grandfathername
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        Grandfather.__init__(self, grandfathername)
class Son(Father):
    def __init__(self,sonname, fathername, grandfathername):
        self.sonname = sonname
        Father.__init__(self, fathername, grandfathername)
    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)
s1 = Son('Chaitanya', 'Nagarajuna', 'Nageswar rao')
print(s1.grandfathername)
s1.print_name()
```

Types of Inheritence

Heirarchical Inheritance:



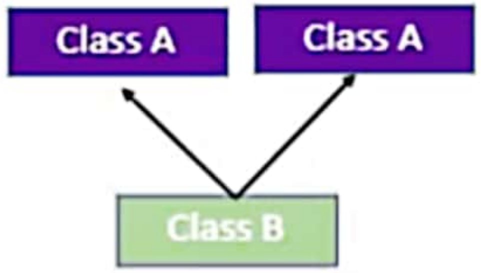
Heirarchical Inheritanc

```
class Parent:
      def func1(self):
          print("This function is in parent class.")
class Child1(Parent):
      def func2(self):
          print("This function is in child 1.")
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Types of Inheritence

Multiple Inheritance:

```
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```



Multiple Inheritence

Polymorphism

There are two types of Polymorphism

1. Compile Time Polymorphism
2. Run Time Polymorphism

Polymorphism

1. Compile Time Polymorphism

Best Example for compile time polymorphism is Method Overloading

Polymorphism

1. Compile Time Polymorphism

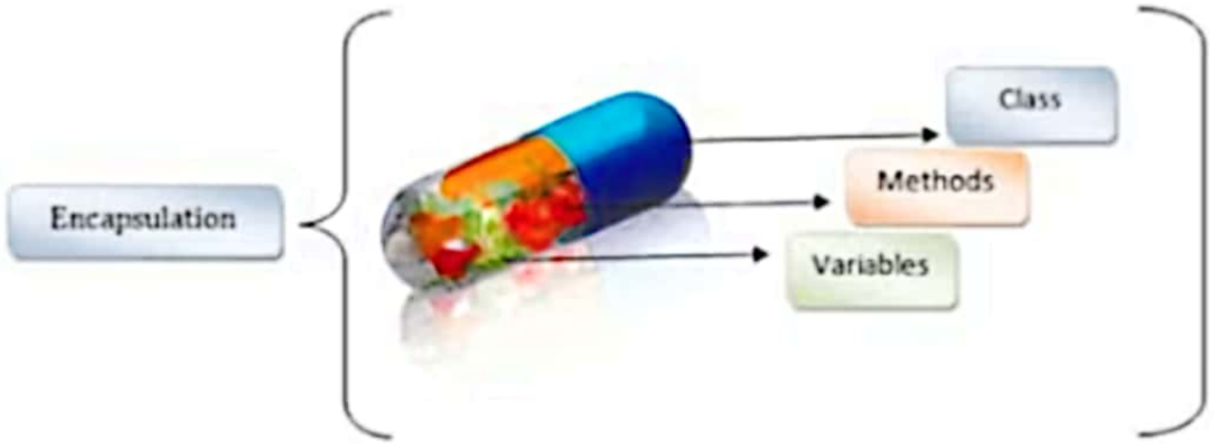
```
def product(a, b):
    p = a * b
    print(p)
def product(a, b, c):
    p = a * b*c
    print(p)
product(4, 5, 5)
```

Polymorphism

2. Run Time Polymorphism

```
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()
```

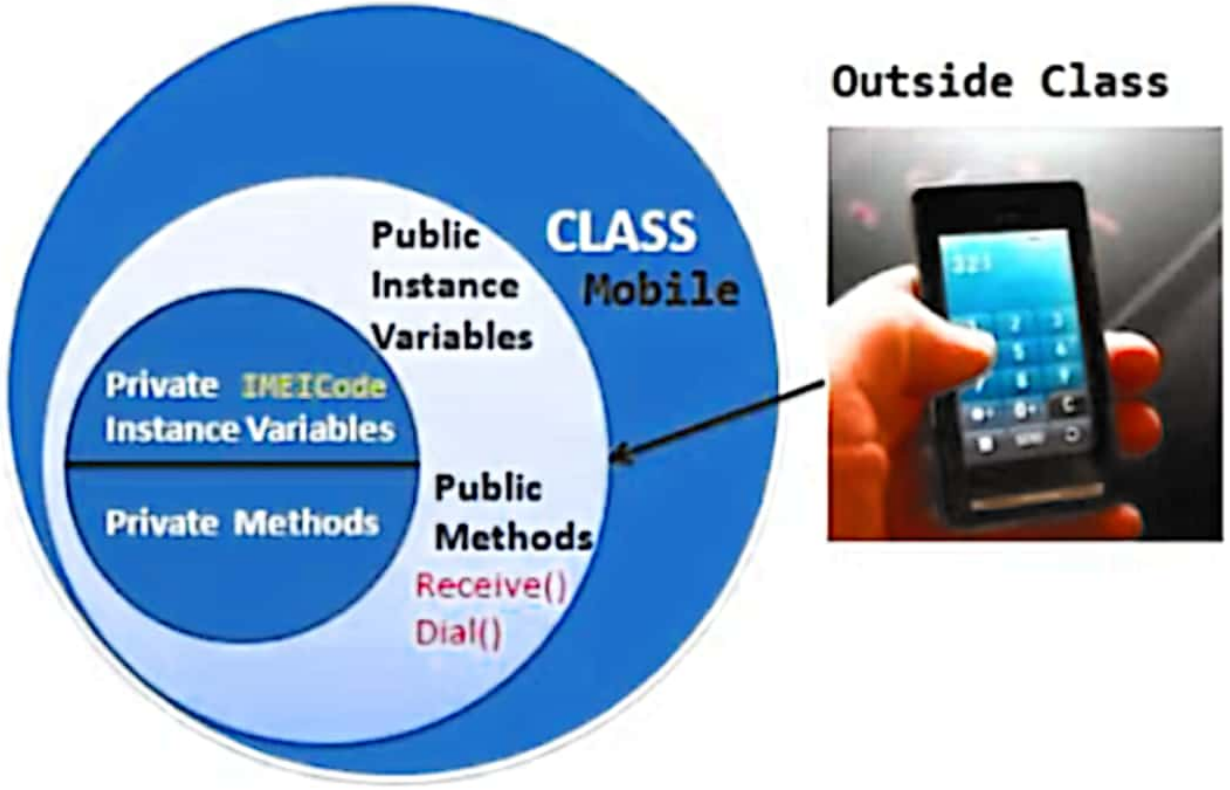
Encapsulation



Encapsulation

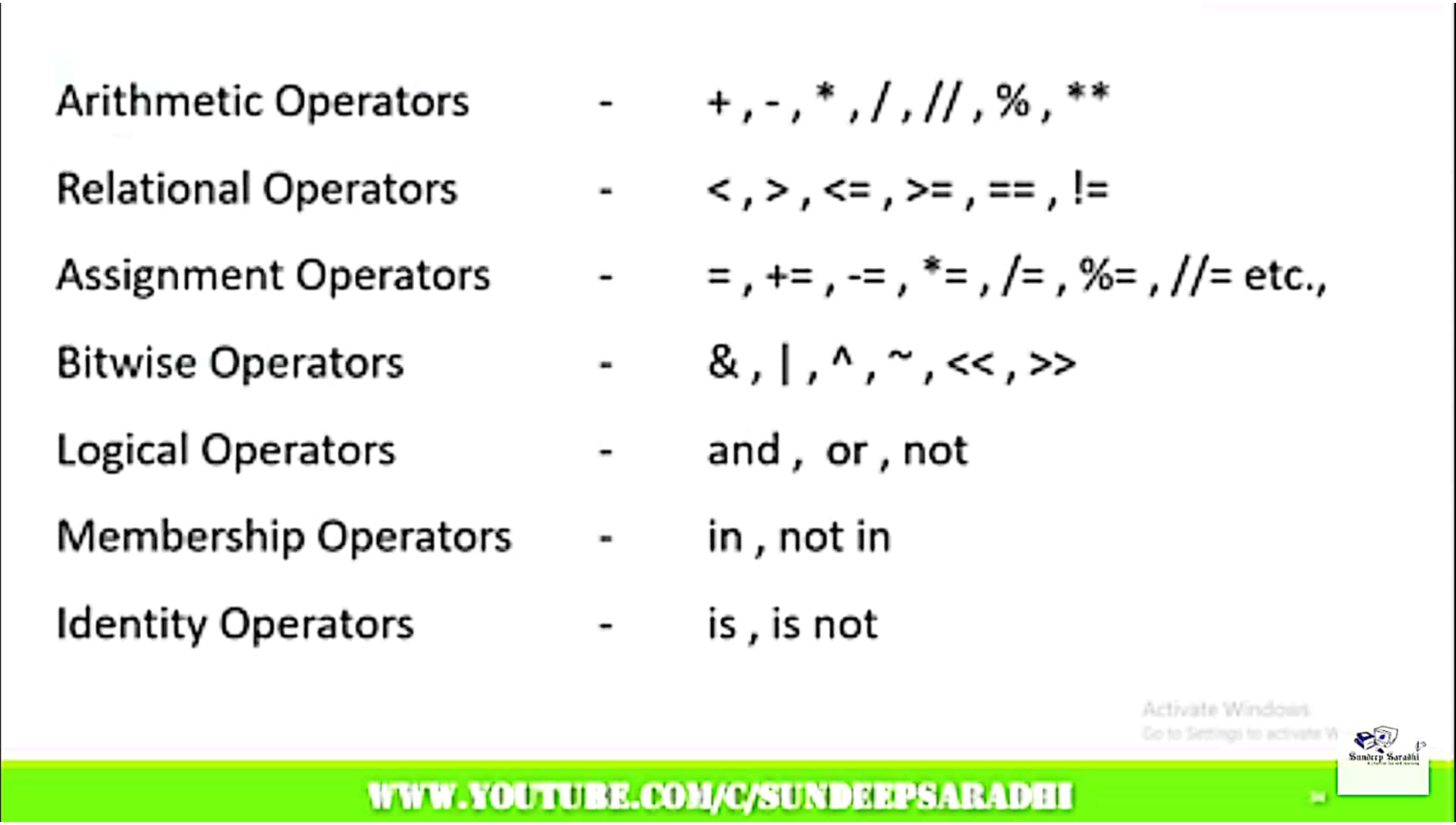
```
class Base:
    def __init__(self):
        # Protected member
        self._a = 2
# Creating a derived class
class Derived(Base):
    def __init__(self):
        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling protected member of base class: ")
        print(self._a)
obj1 = Derived()
obj2 = Base()
print(obj2.a)
```

Abstraction

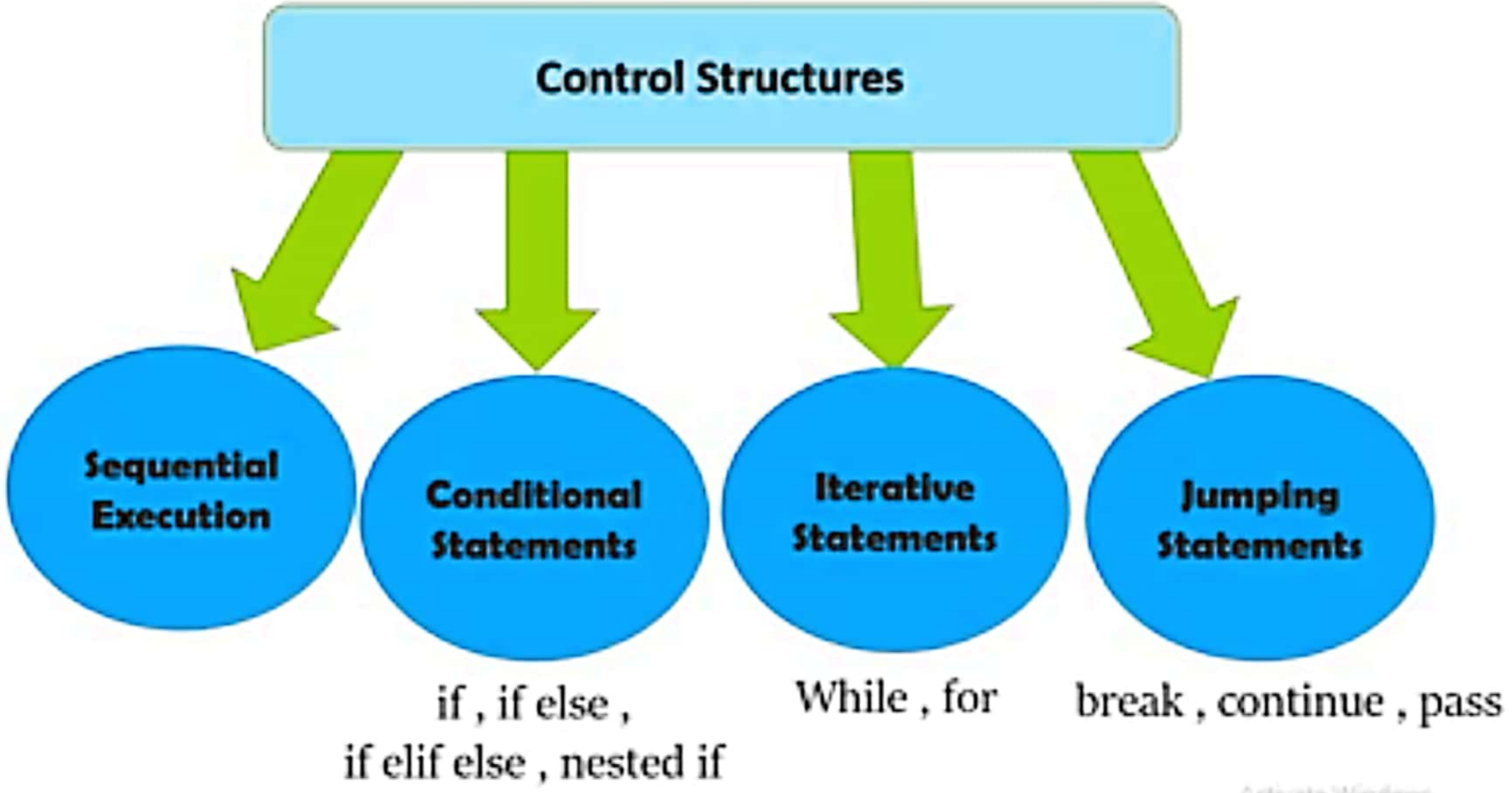


Abstraction

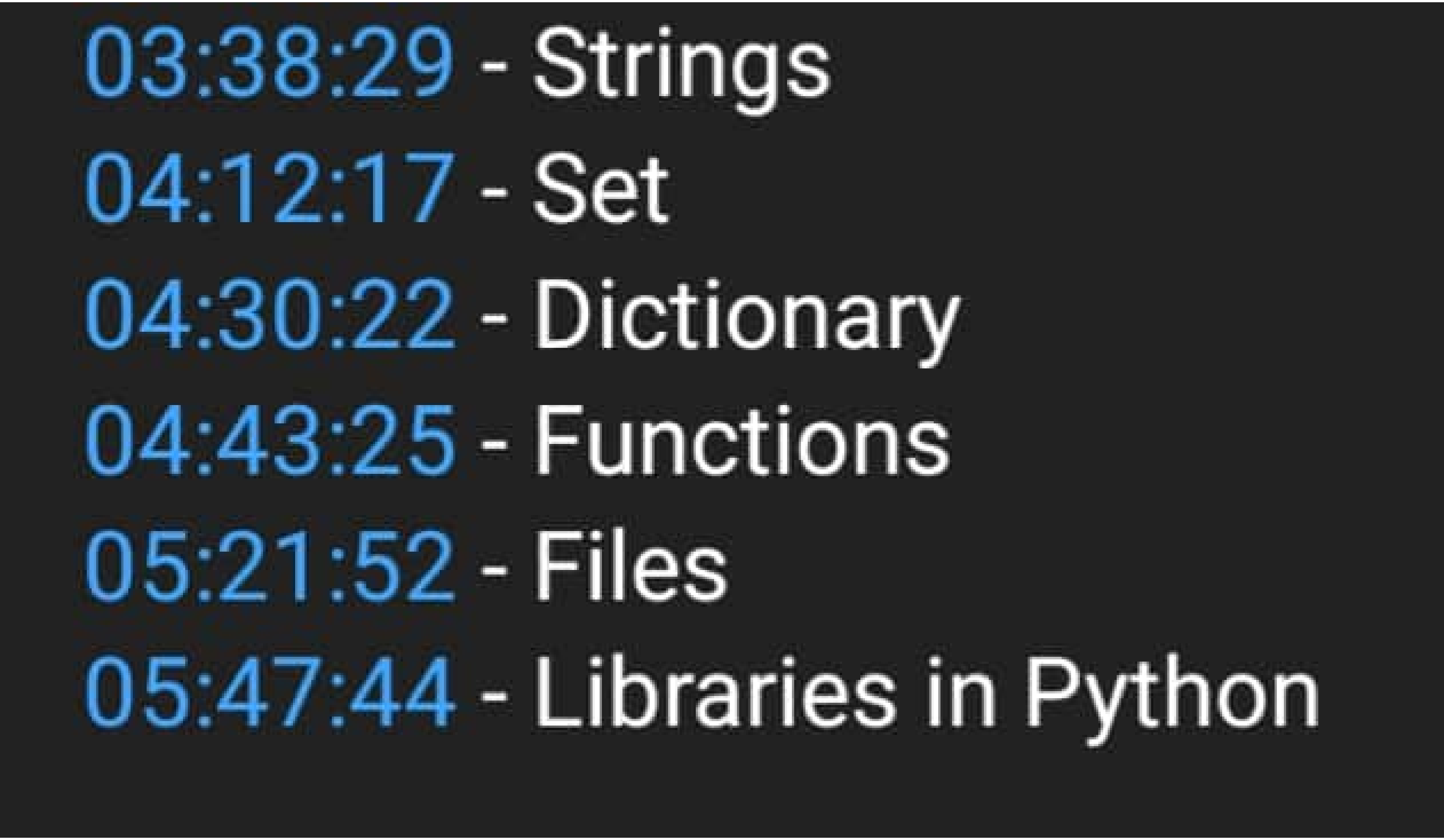
```
from abc import ABC, abstractmethod
class Absclass(ABC):
    def print(self,x):
        print("Passed value: ", x)
    @abstractmethod
    def task(self):
        print("We are inside Absclass task")
class test_class(Absclass):
    def task(self):
        print("We are inside test_class task")
class example_class(Absclass):
    def task(self):
        print("We are inside example_class task")
test_obj = test_class()
test_obj.task()
test_obj.print(100)
example_obj = example_class()
example_obj.task()
example_obj.print(200)
print("test_obj is instance of Absclass? ", isinstance(test_obj, Absclass))
print("example_obj is instance of Absclass? ", isinstance(example_obj, Absclass))
```



Arithmetic Operators	-	+ , - , * , / , // , % , **
Relational Operators	-	< , > , <= , >= , == , !=
Assignment Operators	-	= , += , -= , *= , /= , %= , //= etc.,
Bitwise Operators	-	& , | , ^ , ~ , << , >>
Logical Operators	-	and , or , not
Membership Operators	-	in , not in
Identity Operators	-	is , is not


Control Structures
Sequential Execution
Conditional Statements
Iterative Statements
Jumping Statements
if , if else ,
if elif else , nested if
While , for
break , continue , pass

00:00:00 - Introduction to Python
00:25:00 - Applications of Python
00:27:00 - IDEs
00:28:00 - Introduction to Python Interpreter
00:31:47 - Indentation & Comments
00:43:35 - Keywords
00:44:58 - Variables - Identifiers
00:52:24 - Built-In types
00:57:18 - Assigning Values to Variables
01:06:21 - Input and Output Statements
01:25:25 - Operators
01:46:40 - Control Structures
02:29:49 - Math & Random Modules
02:43:28 - List
03:24:03 - Tuple
03:38:29 - Strings



03:38:29 - Strings
04:12:17 - Set
04:30:22 - Dictionary
04:43:25 - Functions
05:21:52 - Files
05:47:44 - Libraries in Python