Basic Python OOP, Computer Networking, and Intro to Linux for Data Engineering

By Big Three:

Ilham Ahsanudin

Ilyas Rizky Ibrahim

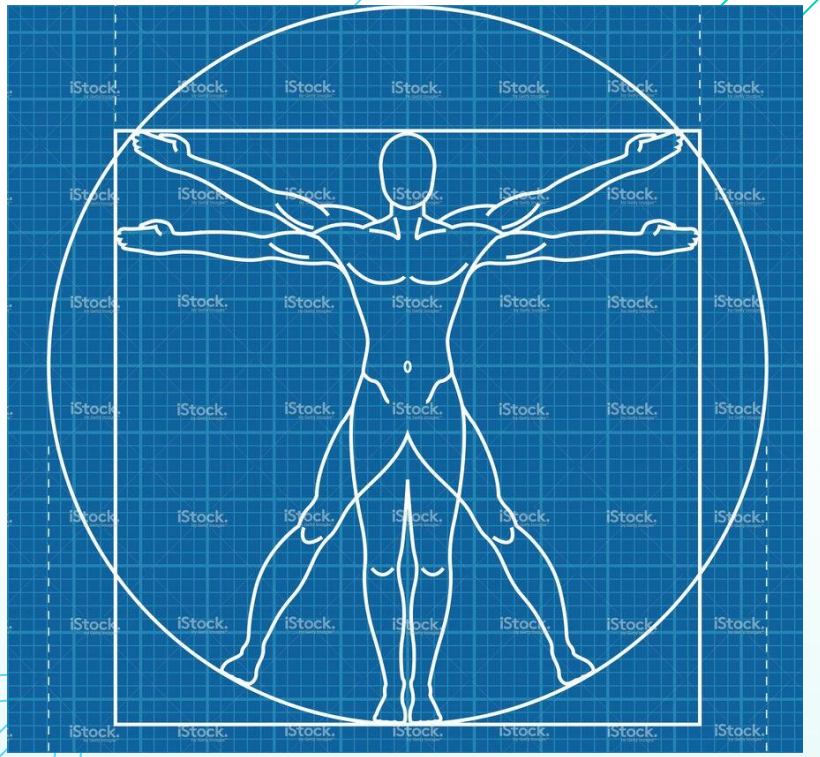
Harryyanto Ishaq Agasi

Table of contents

1

About OOP:

www.dorrisassociates.com



<< Blueprint to make object >>

What is OOP?

- OOP (**Object Oriented Programming**) is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into an objects.
- Another programming paradigm is **Procedural Oriented Programming** (POP), which structures a program like a recipe in that it provides a set of steps.

Here is the difference between OOP and POP

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

realpython.com/python3-object-oriented-programming

https://www.edureka.co/blog/object-oriented-programming-python/

Why do we need OOP?

<< Pros >>

- Object-oriented programming fosters **reusability**. A computer program is written in the form of objects and classes, which can be reused in other projects as well.
- The modular approach used in object-oriented programming results in **highly maintainable code**.
- In object-oriented programming, every class has a specific task. If an error occurs in one part of the code, you can rectify it locally without having to affect other parts of the code. **Helpful for debugging**.

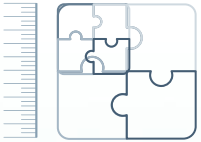
<< Cons >>

- Detailed domain knowledge of the software being developed is needed in order to create objects.
- Not every entity in software is a candidate for being implemented as an object. It can be hard for newbies to identify this fine line.
- As you add more and more classes to the code, the size and complexity of the program grows exponentially.

stackabuse.com/object-oriented-programming-in-python/

Class and Instance

<< Class >>



(Like blueprint)

instantiation

<< Instance >>

(Like object)

A class is a blueprint from which you can create the instance, i.e., objects.	An object is the instance of the class, which helps programmers to use variables and methods from inside the class.
A class is used to bind data as well as methods together as a single unit.	Object acts like a variable of the class.
Classes have logical existence.	Objects have a physical existence.
A class doesn't take any memory spaces when a programmer creates one.	An object takes memory when a programmer creates one.
The class has to be declared only once.	Objects can be declared several times depending on the requirement.

https://www.w3schools.in/java-questions-answers/difference-between-classes-objects/

Class and Instance

<< Class >>

```python
## Pembuatan class Hero ##
class Hero:
    ## class attributes ##
    identitas = 'Hero'
    tempat_tinggal = "land of dawn"


    ## instance attributes: nama dan tipe_serangan ##
    def __init__(self, nama, tipe_serangan):
        self.nama = nama
        self.tipe_serangan = tipe_serangan
```

<< Instance >>

```python
zilong = Hero("Zilong", "physical")
cecilion = Hero("Cecilion", "magical")
```

- This process is called “instantiation” to make object
- Now, zilong and cecilion are object

https://www.w3schools.in/java-questions-answers/difference-between-classes-objects/

Attribute and Method of Class and Instance

<< Class Attribute >>

- A class attribute is a variable that belongs to a certain class, and not a particular object. Every instance of this class shares the same variable. These attributes are usually defined outside the __init__ constructor.

<< Instance Attribute >>

- An instance/object attribute is a variable that belongs to one (and only one) object. Every instance of a class points to its own attributes variables. These attributes are defined within the __init__ constructor.

https://medium.com/swlh/class-and-object-attributes-python-8191dcd1f4cf#:~:text=A%20class%20attribute%20is%20a,(and%20only%20one)%20object.

Attribute and Method of Class and Instance

```python
## Pembuatan class Hero ##
class Hero:
    ## class attributes ##
    identitas = 'Hero'
    tempat_tinggal = "land of dawn"


    ## instance attributes: nama dan tipe_serangan ##
    def __init__(self, nama, tipe_serangan):
        self.nama = nama
        self.tipe_serangan = tipe_serangan


    ## instance attributes: hit ##
    ## instance method ##
    def basic_attack(self, hit):
        print(self.nama, "melakukan basic attack", hit)

    ## class method: battle_spell ##
    @classmethod
    def battle_spell(cls, nama_battle_spell):
        print(cls.identitas,"menggunakan battle spell",nama_battle_spell)

    ## static method: skill ##
    @staticmethod
    def spawn():
        print(Hero.identitas,"telah respawn")
```

<< Class Attribute >>

<< Instance Attribute >>

```python
zilong = Hero("Zilong", "physical")
cecilion = Hero("Cecilion", "magical")


## print class attribute ##
print(Hero.tempat_tinggal)

## print instance attribute ##
print(zilong.nama)
```

instantiation

```
(base) ishaq@ishaq:~/
land of dawn
Zilong
```

output

Attribute and Method of Class and Instance

<< Class Method >>

- A class method is one that belongs to the class as a whole. It doesn't require an instance. Instead, the class will automatically be sent as the first argument. A class method is declared with the @classmethod decorator.

<< Instance Method >>

- On the other hand, an instance method requires an instance in order to call it, and requires no decorator. This is by far the most common type of method.

https://softwareengineering.stackexchange.com/questions/306092/what-are-class-methods-and-instance-methods-in-python

Attribute and Method of Class and Instance

```python
## Pembuatan class Hero ##
class Hero:
    ## class attributes ##
    identitas = 'Hero'
    tempat_tinggal = "land of dawn"


    ## instance attributes: nama dan tipe_serangan ##
    def __init__(self, nama, tipe_serangan):
        self.nama = nama
        self.tipe_serangan = tipe_serangan


    ## instance attributes: hit ##
    ## instance method ##
    def basic_attack(self, hit):
        print(self.nama, "melakukan basic attack", hit)

    ## class method: battle_spell ##
    @classmethod
    def battle_spell(cls, nama_battle_spell):
        print(cls.identitas,"menggunakan battle spell",nama_battle_spell)

    ## static method: skill ##
    @staticmethod
    def spawn():
        print(Hero.identitas,"telah respawn")
```

<< Instance method >>

<< Class method >>

Attribute and Method of Class and Instance

```
# instantiation object from parent class #
zilong = Hero("Zilong", "physical")
cecilion = Hero("Cecilion", "magical")

# calling instance method #
zilong.basic_attack("serangan tombak")

# child class #
class Hero_mage(Hero):
    identitas = 'Hero Mage'

# instantiation object from child class #
kagura = Hero_mage("Kagura","magical")

# calling class method #
kagura.battle_spell("flameshot")
```

output

```
(base) ishaq@ishaq:~/Documents/DATA ENGINEER -
Zilong melakukan basic attack serangan tombak
Hero Mage menggunakan battle spell flameshot
```

Static Method

```python
## Pembuatan class Hero ##
class Hero:
    ## class attributes ##
    identitas = 'Hero'
    tempat_tinggal = "land of dawn"


    ## instance attributes: nama dan tipe_serangan ##
    def __init__(self, nama, tipe_serangan):
        self.nama = nama
        self.tipe_serangan = tipe_serangan


    ## instance attributes: hit ##
    ## instance method ##
    def basic_attack(self, hit):
        print(self.nama, "melakukan basic attack", hit)

    ## class method: battle_spell ##
    @classmethod
    def battle_spell(cls, nama_battle_spell):
        print(cls.identitas,"menggunakan battle spell",nama_battle_spell)

    ## static method: skill ##
    @staticmethod
    def spawn():
        print(Hero.identitas,"telah respawn")

# Calling static method #
zilong = Hero("Zilong", "physical")
zilong.spawn()
Hero.spawn()
```

Static method

output

```
(base) ishaq@ishaq:~/Docum
Hero telah respawn
Hero telah respawn
```

Inheritance, Overriding, and Overloading

<< Inheritance >>

- capability of childclass/subclass to inherit attributes and methods of parent class

<< Overriding >>

- capability of a childclass / subclass that has a different behavior from the parent class, but with the same method

<< Overloading >>

- capability of a childclass / subclass that has the same method but different parameters

Inheritance, Overriding, and Overloading

```python
## Inheritance ##
class Hero_mage(Hero):
    identitas = 'Hero Mage'


## Overriding ##
class Hero_fighter(Hero):
    identitas = 'Hero Fighter'
    def basic_attack(self,hit):
        print(f"{self.nama} memberikan damage dengan {hit}")

## Overloading ##
class Hero_tank(Hero):
    identitas = 'Hero Tank'
    def basic_attack(self, hit=None):
        if (hit == None):
            print(f"{self.nama} gagal melakukan basic attack")
        else:
            print(f"{self.nama} memberikan damage dengan {hit}")

class Hero_assassin(Hero):
    identitas = 'Hero Assassin'
```

Inheritance, Overriding, and Overloading

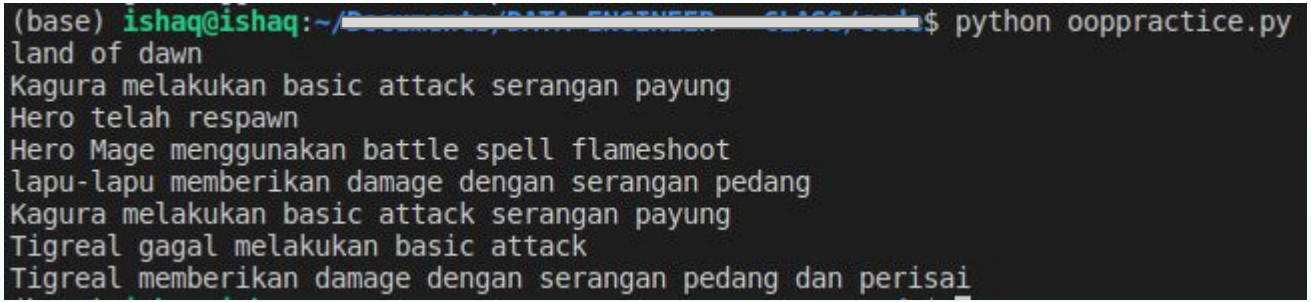
```
## Inheritance ##
kagura = Hero_mage("Kagura","magical")
print(kagura.tempat_tinggal) # class attribute inheritance
kagura.basic_attack("serangan payung") # instance method inheritance
kagura.spawn() # static method inheritance
kagura.battle_spell("flameshoot") #class method inheritance

## Overriding ##
lapu_lapu = Hero_fighter("lapu-lapu","physical")
lapu_lapu.basic_attack("serangan pedang") # with overriding method
kagura.basic_attack("serangan payung") # without overriding method

## Overloading ##
tigreal =  Hero_tank("Tigreal","physical")
tigreal.basic_attack() # hit = None
tigreal.basic_attack("serangan pedang dan perisai") # hit != None
```

Inheritance, Overriding, and Overloading

Output:



2

Networking:

You could enter a subtitle here if you need it

Why Network?

Why a Data Engineer should study computer networks?

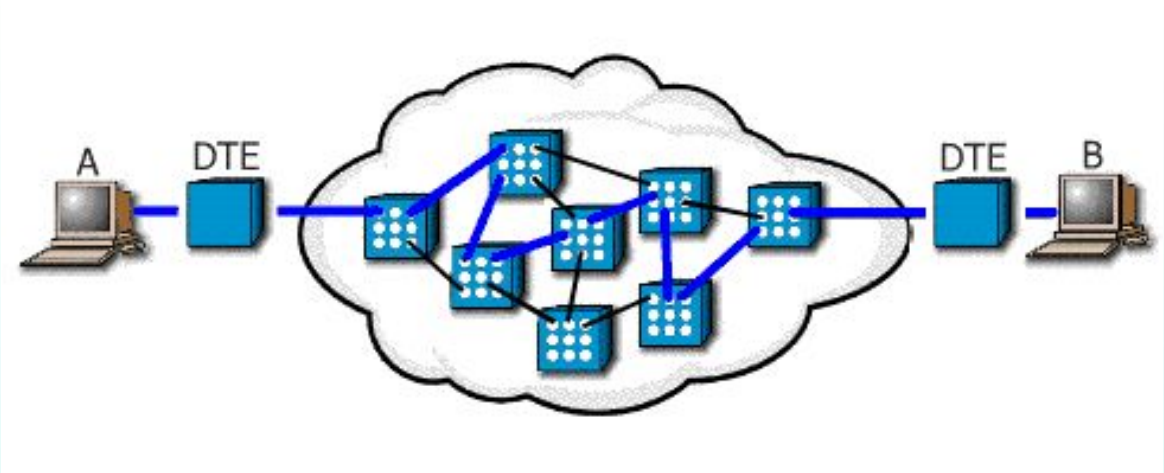
Transfer Data

Multiprocessing Data

Cloud

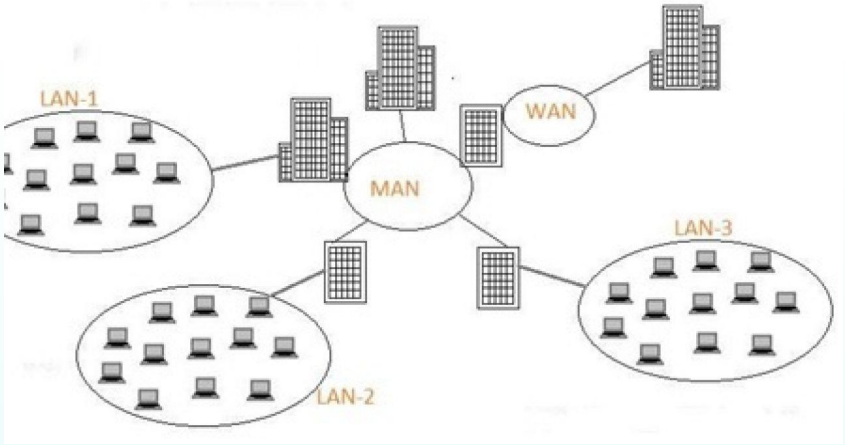
How to Connect?

With 2 computers that connected to each other, there's a process behind it that occurs by passing through many networks
Look at the following simulation:

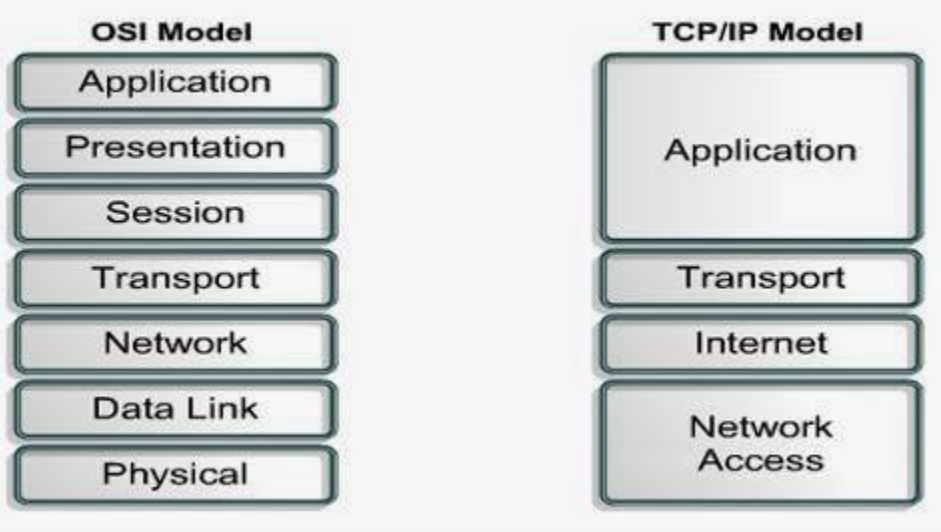


Computer Network

LAN: Local Area Network, a network in one area
MAN: Metropolitan Area Network, a network in one city that consist of several LANs
WAN: Wide Area Network, a collection of several MAN



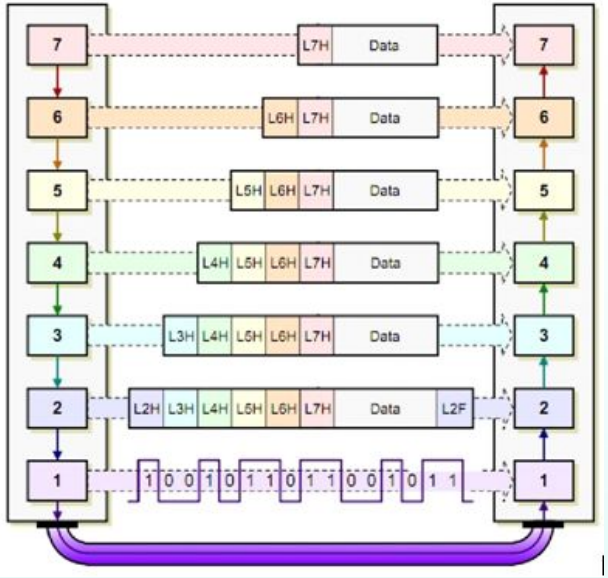
Network Standard



A network that is currently being implemented for the internet that is TCP / IP

Network Standard

Every data that passes through each layer, will be encapsulated with HEADER and FOOTER from the data



3

Linux :

Why Linux?

- Free
- Open source
- Ease of maintenance
- Runs on any hardware
- Common OS used by Data Engineer

Linux Basic Commands

Command	Is used to...
ls	Listing the content of directory
echo	Printing word to monitor
mkdir	Creating a folder
cd	Moving to another directory
pwd	Knowing the directory location
touch	Creating a file
cat	Reading a file

Linux Basic Commands(2)

Command	Is used to...
less	Reading a file in less view
head	Reading a file at top of content
tail	Reading a file at bottom of content
vim	Opening a text editor
nano	Opening a text editor
cp	Copying a file
mv	Moving a file

Linux Basic Commands(3)

Command	Is used to...
rm	Removing a file
rmdir	Removing a folder
grep	Searching a variable in a file
find	Searching a file in a folder

Linux Basic Commands(4)

File Modes and Permission

Linux, like other Unix-like operating systems, allows multiple users to work on the same server simultaneously without disrupting each other.

drwxrwxrwx

d = Directory
r = Read
w = Write
x = Execute

7	rwx	111
6	rw-	110
5	r-x	101
4	r--	100
3	-wx	011
2	-w-	010
1	--x	001
0	---	000

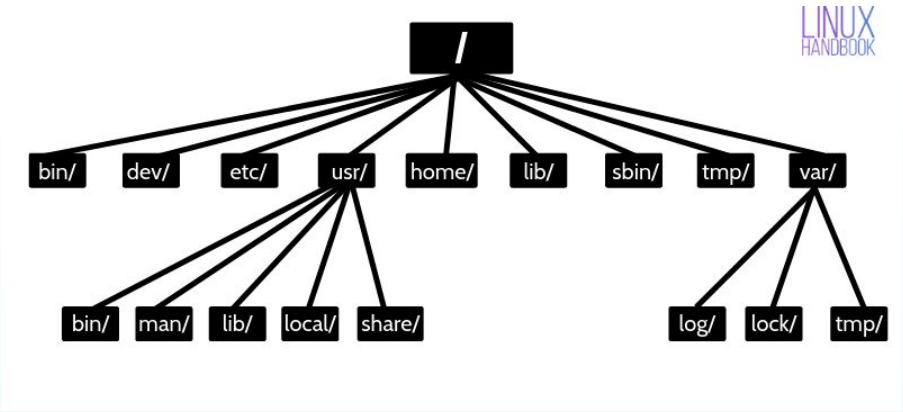
chmod 777

rwx|rwx|rwx
Owner|Group|Others

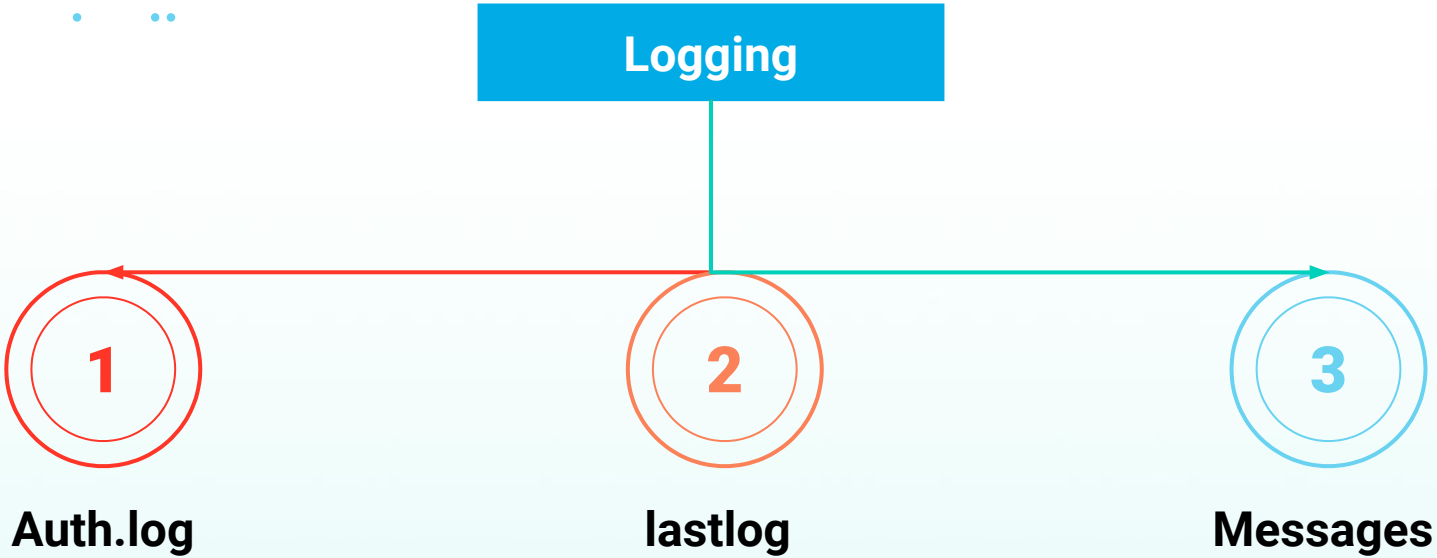
https://phoenixnap.com/kb/linux-file-permissions

Linux Directory Hierarchy

Linux is a bit different from windows.
You can see the directory hierarchy of linux on the side.



Logging, System Time and Users



Logging, System Time and Users(2)

System Time

To display date and time under Linux operating system using command prompt use the date command.

Command	Used to
date	Display current date and time
time	Display past time
uptime	Display uptime server

Logging, System Time and Users(3)

Users

- users

Users is a reguler user without any privilege.

- root user

The root is the user name or account that by default has access to all commands and files on a Linux or other Unix-like operating system. It is also referred to as the root account, root user, and the superuser.