Task

1.Http and Https   
HTTP and HTTPS are both protocols used for transferring data over the internet, but HTTPS is a secure version of HTTP. HTTPS encrypts the data transmitted between the client (e.g., web browser) and the server, while HTTP transmits data in plain text, making it less secure.Https encrypt data using SSL(Secure soket layer) Or TLC(Transport Layer Security) protocols.

Use **HTTP** when security is not a concern.

Use **HTTPS** for secure communication (login pages, banking, shopping, etc.).

|  |  |  |
| --- | --- | --- |
| Feature | HTTP (HyperText Transfer Protocol) | HTTPS (HyperText Transfer Protocol Secure) |
| Full Form | HyperText Transfer Protocol | HyperText Transfer Protocol Secure |
| Security | Not secure | Secure (uses SSL/TLS) |
| Data Encryption | Data is sent in plain text | Data is encrypted |
| Port | Uses port 80 | Uses port 443 |
| URL Prefix | Starts with http:// | Starts with https:// |
| Used For | General websites without sensitive data | Websites with login, payments, sensitive info |
| SEO Ranking | Lower | Preferred by Google for better ranking |
| Padlock Icon | ❌ Not shown in browser | ✅ Padlock shown in browser |

Other Protocol

1. FTP (File Transfer Protocol)

Use: Transfers files between a client and a server.

Port: 21

Secure Version: FTPS or SFTP

2. SMTP (Simple Mail Transfer Protocol)

Use: Sends emails from client to server or between servers.

Port: 25 (or 587 for secure)

3. POP3 (Post Office Protocol version 3)

Use: Downloads emails from the server to the client (then deletes from server).

Port: 110

4. IMAP (Internet Message Access Protocol)

Use: Accesses and manages emails directly on the mail server.

Port: 143

5. DNS (Domain Name System)

Use: Translates domain names (like google.com) into IP addresses.

Port: 53

6. SSH (Secure Shell)

Use: Secure remote login to another computer (command-line access).

Port: 22

7. Telnet

Use: Remote command-line access (not secure).

Port: 23

8. SNMP (Simple Network Management Protocol)

Use: Manages and monitors network devices like routers and switches.

Port: 161

9. DHCP (Dynamic Host Configuration Protocol)

Use: Assigns IP addresses to devices in a network.

Port: 67 (server), 68 (client)

10. TCP/IP (Transmission Control Protocol / Internet Protocol)

Use: Base protocol for all internet communication.

MRO

MRO stands for Method Resolution Order. It is the order in which Python looks for a method in a hierarchy of classes. MRO follows a specific sequence to determine which method to invoke when it encounters multiple classes with the same method name.. For Example :

# Python program showing

# how MRO works

class A:

def rk(self):

print("In class A")

class B(A):

def rk(self):

print("In class B")

r = B()

r.rk()

In the above example the methods that are invoked is from class B but not from class A, and this is due to Method Resolution Order(MRO).   
The order that follows in the above code is- class B – > class A .

In multiple inheritances, the methods are executed based on the order specified while inheriting the classes. For the languages that support single inheritance, method resolution order is not interesting, but the languages that support multiple inheritance method resolution order plays a very crucial role. Let’s look over another example to deeply understand the method resolution order:

class A:

def rk(self):

print("In class A")

class B(A):

def rk(self):

print("In class B")

class C(A):

def rk(self):

print("In class C")

# classes ordering

class D(B, C):

pass

r = D()

r.rk()

In the above example we use multiple inheritances and it is also called **Diamond inheritance** and it looks as follows:   
 Python follows a depth-first lookup order and hence ends up calling the method from class A. By following the method resolution order, the lookup order as follows.   
Class D -> Class B -> Class C -> Class A   
Python follows depth-first order to resolve the methods and attributes. So in the above example, it executes the method in class B.

### Old Style vs New Style Classes in Python

|  |  |  |
| --- | --- | --- |
| Feature | **Old-Style Class** | **New-Style Class** |
| **Introduced In** | Python 2.x (before 2.2) | Python 2.2+ and default in Python 3.x |
| **Syntax** | class MyClass: | class MyClass(object): |
| **Inheritance Base** | Does **not** inherit from object | Inherits from built-in object class |
| **Method Resolution Order** | **Depth-first, left-to-right** (DLR) | **C3 Linearization (MRO)** |
| **Support for super()** | ❌ Not supported | ✅ Fully supported |
| **Properties, Descriptors** | ❌ Not available | ✅ Available |
| **Recommended?** | ❌ No (Obsolete) | ✅ Yes |

## DLR Algorithm

**What is DLR?**

* **DLR = Depth → Left → Right**
* This algorithm was used in **old-style classes** (until Python 2.1).
* When **multiple inheritance** is used, it helps Python decide **from which class the method should be called**.

**Rules of DLR:**

1. **Depth first**: Go deep into the class tree (parents of parents) first.
2. **Left to right**: If multiple parents exist, check the **left one first**, then right.
3. **Avoid duplicates**: If a class is already checked once, skip it the next time.

**Example:**

class A: pass

class B: pass

class C(A, B): pass

class D(B, A): pass

class E(C, D): pass

**Class Tree:**

E

├── C

│ ├── A

│ └── B

└── D

├── B

└── A

**DLR Resolution Steps:**

Start checking from E:

* E
* → C
* → → A
* → → B
* → D
* → → B ❌ (Already checked)
* → → A ❌ (Already checked)

✅ Final DLR Method Resolution Order (MRO):

E → C → A → B → D

**❗ Problems with DLR:**

* **Inconsistent behavior**: If you change the order of parent classes (e.g., D(B, A) to D(A, B)), the resolution order also changes.
* This causes **confusion and bugs** in large programs.

**The Solution: C3 Linearization**

* Introduced in **Python 2.3 and used in Python 3.x**
* It provides:
  + **Consistent**
  + **Predictable**
  + **Reliable** method resolution order
* C3 works only with **new-style classes**:  
  class ClassName(object):

## What is C3 Linearization Algorithm?

**C3 Linearization** is an algorithm used to determine **Method Resolution Order (MRO)** in **new-style Python classes** (Python 2.3+ and Python 3.x).

It's a **solution to the inconsistency** created by **DLR (Depth-Left-Right)** algorithm in old-style classes.

## C3 Algorithm Works on 3 Main Rules:

1. **Children before Parents**  
   → Child class methods are always looked at **before** parent class methods.
2. **Respect the Order of Base Classes**  
   → If you write class C(A, B), Python will always look into A before B.
3. **Monotonicity**  
   → The MRO must be consistent and follow a straight path. Changing one class should not change another’s behavior **unpredictably**.

## Why was C3 Needed? (Problem in DLR)

DLR gives **different results** when the order of base classes is changed.  
That creates **bugs and confusion** in large programs.

Example:

class A: pass

class B: pass

class C(A, B): pass

class D(B, A): pass

class E(C, D): pass

In DLR, the MRO depends on inheritance order and sometimes repeats classes or skips things unpredictably.  
That’s where C3 gives a **clear, consistent, and monotonic** result.

## How to Check MRO in Python?

You can check a class’s MRO using:

ClassName.\_\_mro\_\_ # returns a tuple

ClassName.mro() # returns a list

## Example:

class A:

def rk(self):

print("In class A")

class B:

def rk(self):

print("In class B")

class C(A, B): # C inherits A first, then B

def \_\_init\_\_(self):

print("Constructor of C")

obj = C()

print(C.\_\_mro\_\_)

print(C.mro())

### Output:

Constructor of C

(<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class 'object'>)

[<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class 'object'>]

✅ It clearly shows:  
C → A → B → object  
Even if methods are same (rk()), Python will always use method from A first because it is listed first.

## DLR vs C3 Comparison

|  |  |  |
| --- | --- | --- |
| Feature | DLR (Old Style) | C3 Linearization (New Style) |
| Order Type | Depth-first, Left to Right | Linear, consistent (C3 algorithm) |
| Duplicate Classes | May appear more than once | Avoids duplicates |
| Behavior | Unpredictable and inconsistent | Predictable and monotonic |
| Used In | Python 2.1 and earlier | Python 2.3+, Python 3.x (New-style) |
| Best for Projects? | ❌ Not reliable | ✅ Yes, stable and modern |

## Conclusion:

**C3 Linearization is better** and always preferred in Python projects.  
 It gives **clear, consistent MRO** and avoids confusion from DLR.  
 Always use **new-style classes** by inheriting from object:

class MyClass(object): # new-style class

pass

# Cross Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) is a security mechanism implemented by web browsers to restrict web pages from making requests to a different domain than the one that served the web page. This prevents malicious websites from accessing sensitive data from other websites.

In Python, CORS can be enabled using libraries like Flask-CORS for Flask applications or django-cors-headers for Django applications. These libraries add the necessary HTTP headers to the server's responses, allowing the browser to permit cross-origin requests.