Done--------------- 7,8,9 June ------------------------revision--------------------01 April to iOS start ----

<https://developer.apple.com/documentation/swift>

<https://www.youtube.com/c/LetCreateAnApp>

<https://manasaprema04.medium.com/application-life-cycle-in-ios-f7365d8c1636>

<https://www.javatpoint.com/ios-app-lifecycle>

<https://iosapptemplates.com/blog/ios-development/data-persistence-ios-swift>

<https://github.com/ivanvorobei/SwiftUI#animatable-cards>

<https://www.raywenderlich.com/28684964-swiftui-layout-interfaces/lessons/1>

<https://codewithchris.com/swift-tutorial-complete/#variables-and-constants>

<https://www.ioscreator.com/tutorials/prototype-cells-table-view-ios-tutorial>

<https://www.programiz.com/swift-programming/basic-input-output>

variable : - variable is an abstract storage location paired with symbolic name which stored some known or unknown quantity of information…or it is a container of set of bits or data.

For Loop : - for loop is based on counting. You can decide how many time loop statement will execute .

for \_ in CountStart...CountEnd

reverse loop from high to low

ex - for i in (12...23).reversed()

create loop range in steps other than 1

ex - for i in stride(from: 1, to: 10, by: 2)

Nested Loop : - Loop inside a Loop is called nested loop.

Terminator : - terminator is used to print next value on same line.

print("Hello ", terminator:" ")

--------------------------------------------------------------------------------------------------------------------------

While Loop : - while loop - a loop that repeats the code until condition will be or is true

Infinite Loop -----while(i==0) or while(i=true)

--------------------------------------------------------------------------------------------------------------------------

Optional Types : - if we want to declare any var or data type blank(nothing in the place of value) or nil /null/nothing to solve this problem we declare data or variable type ( ? or optional ) after declaring it we will not get nil Error ..

And to get ride of Optional type we need to unwrap it after use with ( ! ) .

\*\*\* we cannot use optional in ant arithmetic operations .

Jab jab apko optional ko use krna hoga to use unwrap krke use krna pdega ya use kise dusre variable main unwrap krke rakhna padega.

\*\*\* Cheking nil (if n =! (Equal not) nil) { Message or code} else { code ya error} \*\*\*

Jab ham optional ko unwrap krenge to bo us type ka ban jayega jis type ka data Type defined h …

--------------------------------------------------------------------------------------------------------------------------

// Switch\_case/default : - to select one option in multiples options available.

fallthrough : - it is used to jump from last operation/or selected option to next remaining operations/conditions ex 1…..7 condition and we call for 5th condition if fallthrough there remaining 2 condition will also work

--------------------------------------------------------------------------------------------------------------------------

Function / method : - A group of statements with a given name, which can be used again and again. func findSquare() { Code }

* - We call it by its name whenever required - -

Method : - When we make function inside the Class we call it method

Variable that’s is out of function or class is called global variable.

Parameterized Functions : - function that has its variables or parameters on the its own .

Parameter very useful when we need to change value oftenly .

func name(a:Int,b:Int,c:Int) {code}

FunctionWithReturn : - kise function ko chalane k baad uska jo result aya usko use krna ho kahi pr to ham return type function banate h . func name(a:Int,b:Int,c:Int) -> Int {Code}

(function that return the value to its own object or variable that is created while calling Function)

Ex . func name () {operation return m}

Var a = name() – Isme jo variable (a) create kiya h uske pass result ki value (m) aa jati h ..

If we don’t want to return any data type then we declare - > Void or nothing.

Recursion : - when function call itself is called as Recursion ….function used inside its own function .

Closure or Nested Function : - function inside another function .

func name(Code) {name1(Code)}name2(Code)}

Function Types -> Data type associated with function declaration (declaration of data type)

var name : ()->Void = { print(Code) }

Closure expression : - a function without name. These functions are stored in function variables and are passed in argument to another function .(isme func nhi likhte balki Variable create karke use { } m dalke run krte h or variable name se call krte h .

Parameterized hota h ye .

//closures can be pass in arguments in place of functions

//closure can be created whenever required

//alternate - if closure or function is last argument then write it direct after call to your function

//pass by reference - memory address or ref of a var is pass to function so that original var can be modified from within function

--------------------------------------------------------------------------------------------------------------------------

Tuple : - is a complex or composite type to store different types of values in a single variable. var name:(Int,Float,String) = (1,5.6,"Hello")

Access /modify part of Tuple : - (tuple Name.IndexValue)

--------------------------------------------------------------------------------------------------------------------------

Any type : - is used to create generic type variables where we can store any type of values any time

var name : Any

Arrays : - is a collection of similar types of data items. Any numbers of data values can be stored in an array. items in array are accessed using their position which start from zero.

var name : [Int] = [0,8,6,9,7,5,3,2,1,9]

Swift array are dynamic in nature i.e. items can be added or removed from any position any time

1.remove all elements from array

myArray.removeAll()

2.replace/modify/access from array

let n = myArray.remove(at: 1)

3. add new element in array at the end

myArray.append(8)

4. append multiple elements at end

myArray.append(contentsOf: [56,23]

5. replace any value in array

myArray[0] = 66

6. find size array

print(myArray.count)

7. insert anywhere in array

myArray.insert(23, at: 4) //23 inserted at position 4 and elements already at 4 and above move to next positions

8. insert multiple elements at a position

myArray.insert(contentsOf: [14,28], at: 3)

9. check if element exist in array

let exist = myArray.contains(8) //true

10. eplace multiple elements at once with another elements

myArray.replaceSubrange(2...4, with: [12,9,11])

11. declare empty array

var marksArray:[Int] = []

12.reversed Array

let reversed1 = Array(numbers.sorted().reversed())

--------------------------------------------------------------------------------------------------------------------------

2-D Array -

--------------------------------------------------------------------------------------------------------------------------

Dictionary : - Dictionary is a collection of Key and Value’s Pair ….var name : [Int:String]

Keys and values can be of any types …

We cannot repeat key (same) …keys are always uniqe

Pairs Coma lagakr banate h ex .(key:value,Key:value,……..,…….,..)

Key ki help se ham value ko pta karte h ..

Keys are always Optional .

?? – nil-coalenscing operator (double question mark) is used to provide fefalut value to an optional value .. ?? k bad koi bhi message deliver kr skte h ya (default value)

Add new pair = name[key] = value

dictionaries do not maintain order of storage

update value of key in dictionary = name[key] = value

remove a pair from dictionary = var city = name.removeValue(forKey: Value)

empty Dictionary = var name :[String:Float] = [:]

get all keys = var name = name.keys

get all values = var name = name.values

array of dictionary = var name :[ [String:Any] ] = [name1, name2,name3,name4,name5]

dictionary in argument to function = func name(name:[String:Any]) -> Void { Code }

--------------------------------------------------------------------------------------------------------------------------

-------------------------------OOP Concepts --------------------------------

**Object oriented programming -** an approach to organize real world data with the code that works on data. It is common concept for all programming languages.

1. **Object -** any real world thing is an object like Employee, Car, Book, Product, Seat, Food etc.

------------------

Common objects in food delivery app.

Ex - FoodItem, Delivery Boy, Payment, Customer.

//**Attributes/properties of object –** (Information about the object)represented by variables)

Ex - FoodItem - price, ingredient, delivery time, cosines type, name - pizza, 350, 12:50 pm, veg

------------------

Delivery Boy - name, id, temp, contact

e.g - Atul, 101, 98, 9345611223

//**Behviour/actions related with object. –** (Represent using function)

Delivery Boy -> pickup food item, drop item, collect payment, collect feedback

------------------

Customer -> order food, rating, track delivery, compare food price

2. **Class -** definition of object description which includes name of attributes/properties (description of behaviour of object )is known as class.. class name { Code }

Class - definition of some kind of objects. Object properties are declared using variables and behaviour or action are declared using functions.

Class defines an object. It is a blueprint before objects are created.

\*\*\*class ko memory dene k liye object create krte h let name:classname=classname()\*\*\*

name is a reference of memory type class(name)

reference kitni bhi create kr skte h class ki memory k.

1 memroy k kitne bhi reference ho sakte h..

Object Variable / instance variable/Data member : - variables that used in class to store data are called as object variable

Method : - When we make function inside the Class we call it method.

Local variable : - function k ander or initializer m ate h unhe local variable hote h jinki memory lapse ho jati h function k use khtm hote hi.

Memory dene k bad us memory ko access krte h memory name(name).property = value.

-------------------

Passion pro - Hero Honda Model is designed by Her company and after that multiple objects of this type of bike are manufactured.

3. **Inheritance -** old design is used to create new design to create new type of objects.

Passion Pro Basic -> Passion pro Premium -> BS6

Vehicle - a tool of transportation

- TwoWheeler

-cycle

— simple, mountain , all

-bike –

simple, moped, scooter,

- FourWheeler

- car - suv, sedan van, jeep

-bus

-truck

4. **Abstraction -** hiding unwanted things and exposing only required thing.

Like we use car gear but don’t know how gear box work

5. **Encapsulation –** (data declaration + code) or ((variable+function)=Encapsulation) is bind together to form a single unit of code. Class helps to create encapsulation.

6. **Polymorphism -** multiple forms of object and it’s behaviour .

bike lock behaviour - key, electronic lock, padlock, wheel lock

Bike start - self, kick, push,

Ex – class Employee //Discription

{

//Properties

Var name:String = “ ”

Var id:Int=0

Var salary:Float=0

// Behaviour

func display() -> Void

{

Print(name,”-“,ID”-“,salary)

}

--------------------------------------------------------------------------------------------------------------------------

**Initializer Initiazer : -** Is like a constructure or a function which execute when we create object of a class . init() { Code } for setting default value .

Kise bhi class ka sabse sabse pahle default initializer call hota h .

**Parameterized initializer : -** initializer in which we can pass data with data Type init(var:Int,var:Int) { Code }

**Self : -** Self confirms the object variable is the main variable on which we need to work not on local variable.it can also used to distinguish which is object variable if var have same names .reference of current object

**Static Variable /Class variable : -** thevariable shared between all objects of class or data at class level .

Or aise variable ko ham static se declare krte h jisse usko bar bar memory declare nhi krni padti or use ham jarurat k hisab se edit kr skte h

Because it is whole class variable so it can only be accessed by classname. if you modify it it will modify for all objects.

**Static method /Class method or function : -**  the method/function which do not use object variable or data of current object can be made static .there method only accessed only through class name **.**

Static method can only used by static object or variable .

--------------------------------------------------------------------------------------------------------------------------

**Property Type : -** - properties are object variables

1. Stored property - gets memory as soon as object is created

2. Computed property - never gets memory they are always dependent on other properties and they behave like closures return values.

3. Lazy property - gets memory when used first time not when object is created. Isko banana k liye lazy prefix use krte h varname k age ex. – lazy varname

--------------------------------------------------------------------------------------------------------------------------

**Property Observer : -** the code that observe value of any object variable.

Is a closure it help to take a decision before or after change in value of object variable

**Will set :** - key word is used to decide code before change in variables value.

It cannot change the upcoming change that you want to make ….if you change it it will automatically made memory and save it even you mistype anything

**Did set : -** Keyword is used to decide code before change in variable . you can only make changes in did set .it is decision maker in property observer.

\*\*\* will-set and did-set by default are optional\*\*\*

**Has a Relationship : -** ownership of objects .

Isme ARC hota h means automatic reference counting .

--------------------------------------------------------------------------------------------------------------------------

**Inheritance : -** Inheritance is a way to define new class based of defination of existing class. New class includes all variable and methods defined in previous/exisiting class. New class is known as child/sub class and existing class is known as parent/super class. Inheritance allows re-usability of code.

In real world new models of existing products are released time to time by making changes in existing product design.

Single inheritancing : - Class B , class A ko inherit kre

Multilevel inheritance inheritancing : - Class C , Class B OR Class A dono ko inherit kre .

Hierarchical inheritancing : - (Jab Class B ko inherit kre ) OR ( Class C Classs A ko inherit kre) alag alag

Super Class / Parent : - bo class jisko inherit krke nai class m use kiya h .

Child Class / Sub : - bo class jise banaya gya h kise or class ka reference leke .

Kise bhi class ko ham inherit krke uske object ya methods ka use krte h .

--------------------------------------------------------------------------------------------------------------------------

//**method overriding** - when child class modify code of inherited methods with same name defined in super class then it is known as method overriding. It allows dynamic polymorphism.

--------------------------------------------------------------------------------------------------------------------------

**//protoco**l or **abstract Class**- is an abstract class . incomplete class. template class. protocol only declare variable and functions but do not provide any default value to variables and body to functions.

//let s = TwoDShape()

//protocol reference

//multiple inheritance

//------protocol can inherit another protocol

//Concept of {get/set}

( aise information jiski apko detail nhi pta h pr declare krna jaruri h to protocol use krte h ya fir future m ane wale code k liye ye guidline/khaka /template ka kam krga)

…..Isme na code hota h na koi func hota bss protocol ko inherit kiya jaa skta h…..

Ye bas ane wali classes ki superclass ki tarah hoti h taki jab nai class aye to use ham protocol ko inherit krke use kr sake….

Isko ham chahe to use kar skte h ya omit kr skte h mtlb jrurat pade to use kro brna pda rehne do ..future k liye

Isme ham sirf refrence le skte h protocol na object nhi ban skta iska kyunki usme kuch h hi nhi

[Jab bhi ptocol ko inherit krte h to uske ander jo bhi data hota h usko initialize krna hota h (sare k sare)]

\*\*\*Super class main child class ka object aa skta h \*\*\*

--------------------------------------------------------------------------------------------------------------------------

**// RUN time Polymorphism –**

**\*\*\*we can override any function any times \*\*\*\*jitni bar chahe**

**\*\*\*or upper wali jo superclass h use ham super.(dot)krke usko bhi override kr skte h \*\*\*\***

**\*\*\*OverRide krte time hame return Type decide krte h**

[ RunTimePolimorphism = ek jaise methods ko ham polymorphism bolte h or konsa method chalega ya ham jab chalyenge bo run time pr decide hoga (code k dwara(based on data passing) ..or ism ham 1 object ko 2 refrence deke ek method se dusri methods/value ko change kr skte h )]

--------------------------------------------------------------------------------------------------------------------------

Structure : - structure are same as classes in many cases .Like classes we can declare structures .A structure contain variables , methods. initializers ,computed property ,stored property , property Observer etc . But there are some differences in class and Structure .

1. Class are pass by reference and structures are pass by value
2. Structure cannot be inherited to create another Structure
3. Structure can only inherit Protocol
4. Structure do not have initializer
5. Parameterised initializer created by default in Structures

\*\* Structure se hi Int, Float ,String ,Character ,Array , Dictionary Bane h \*\*\*

( iseliye ye sab structure ke jaise pass by Value pr kam krte h )

De -Initializer --- it execute just before object is deleted from the Memory

( iska use ham kise tarh ka message dene k liye use kr skte h jab memory 0 ho jay to )

Structure Main De – Initializer nhi hota

--------------------------------------------------------------------------------------------------------------------------

**// Generics : -**  Generics syntax allows you to create code that can work on Multiple Data Types .It is Used to create common algorithms like Searching , Copying , Merging , Shorting etc .

**In Out :**- Because functions initializer main jo bhi variable default declare hote h bo Let type k hote h to agar hame value change krni h unki to ham Inout (prefix) ka use krte h.jisse bo pass by reference type ho jate h .

**\*\*\*** jab bhi address pass krte h to (&)var lagakar pass krte h kyunki max data types structure se milke bane h to bo pahle se Pass by Value type honge \*\*\*

----Generics main aap apne hisab se kuch bhi data type bana skte h ex ………………… var :m(datatype decided-ye kise bhi tarah k data ko accept kr lega)---

Yaha pr ab hame operator ko bhi change krna pdega kyunki ye har tarah k data type pr nhi chalte ..uske liye ham Numeric Protocol ko inherit kar denge function m .

// Extensions : - (partial classes) extension ka use class m bina badlao kiye bina inherit kiye usme kuch naya jodne k liye use hota h ….

Syntex : -

Isme ham extension likhkar Class ka Naam dal dete h jisse extension us class m jake jud jata h.

\*\*\* kise value m INT m kitne number h use count krne k tareeke –

1. 10 se devide krte jao while loop se jab tk 0 na a jaye or count lga do .
2. Value to string m convert kro or .count laga do .
3. .digitCount() lagakar bhi Check kr skte h \*\*\*

------------Apple Site Data -------------

Control Flow

Use if and switch to make conditionals, and use for-in, while, and repeat-while to make loops. Parentheses around the condition or loop variable are optional. Braces around the body are required.

## Functions and Closures

Use func to declare a function. Call a function by following its name with a list of arguments in parentheses. Use -> to separate the parameter names and types from the function’s return type.

## Objects and Classes

Use class followed by the class’s name to create a class. A property declaration in a class is written the same way as a constant or variable declaration, except that it’s in the context of a class. Likewise, method and function declarations are written the same way.

## Enumerations and Structures

Use enum to create an enumeration. Like classes and all other named types, enumerations can have methods associated with them.

Storage Types in swift : -

* UserDefaults
* Keychain
* Saving files to disk
* Core Data
* SQLite
* Property Lists

**Basic Shortcuts**

* Build: ⌘ + B.
* Run: ⌘ + R.
* Test: ⌘ + U.
* Stop: ⌘ + .
* Clean: ⌘ + ⇧ + K.
* Clean the build folder: ⌘ + ⇧ + ⌥ + K.
* Open quickly: ⇧ + ⌘ + O.
* Code completion: ⌃ + Space.

# Fix All Errors In-Scope

# Shortcut: ⌃+ ⌥ + ⌘ + F

( Control + Option + Command + F)

Multiple Cursors on Multiple Selections(select all same words)

(Option + Command + E)

Jump to Definition

( Control + Command + J)

## **Search & Replace in the whole project: ⇧ +⌥ + ⌘ + F**

(Shift + Command + Option + F)

## UIKit

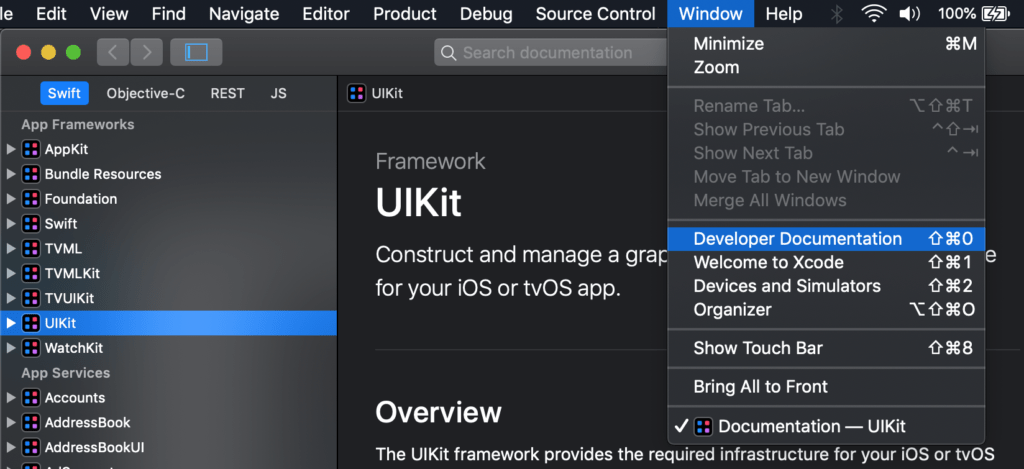
UIKit is an Apple framework that contains many classes for common elements that most iOS apps share, i.e. for labels, buttons, text fields, and images.

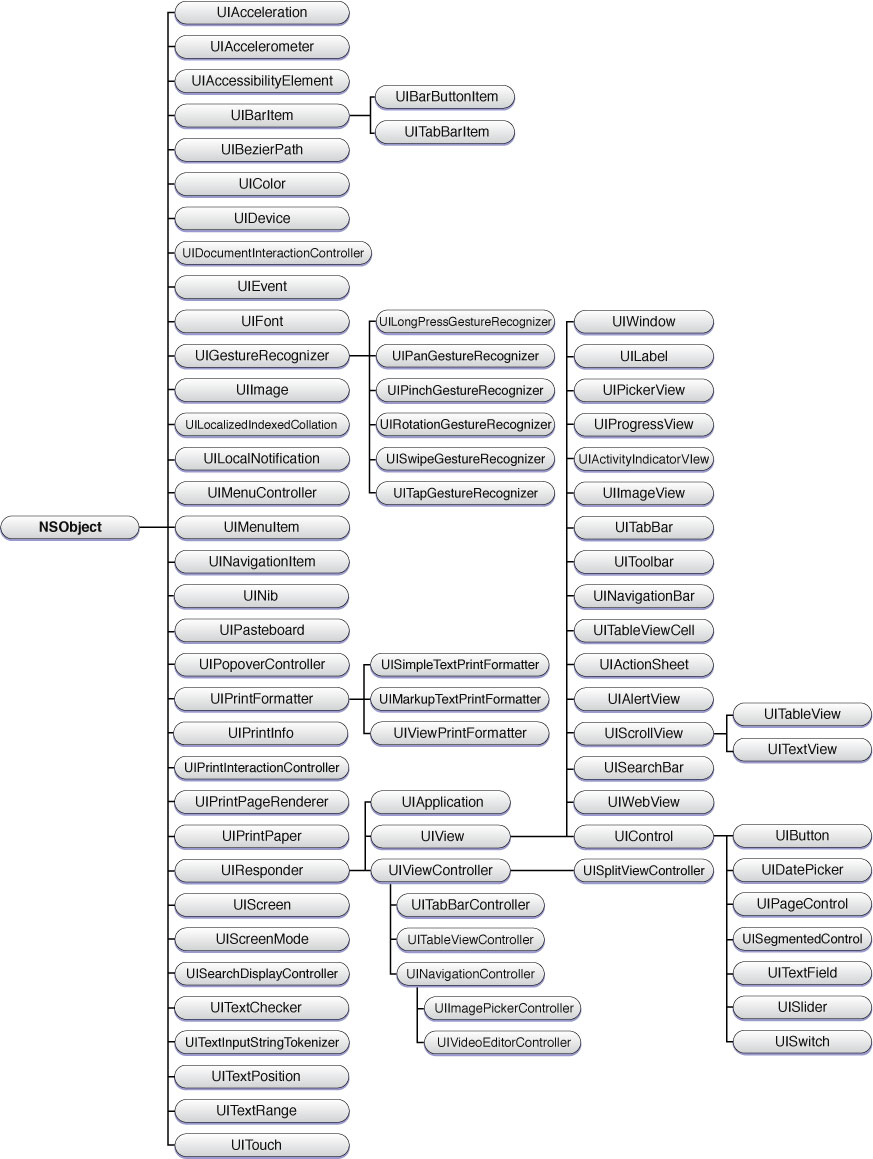
**Important**

Use UIKit classes only from your app’s main thread or main dispatch queue, unless otherwise indicated. This restriction particularly applies to classes derived from UIResponder or that involve manipulating your app’s user interface in any way.

To find a page on any UIKit element, search for “UIKit” + the element name on Google

Another really handy way to explore the UIKit, which I recommend even more, is through Xcode. You can access it from “Window” > “Developer Documentation.” Under “Swift” in the menu to the left, you will find UIKit near the top.





In the rightmost column, you can see UIButton, the classic or standard iOS button you would put in your view. By following the lines along the UIKit, you’ll see that UIButton is a subclass of UIControl, a subclass of UIView, a subclass of UIResponder, which is ultimately a subclass of NSObject:

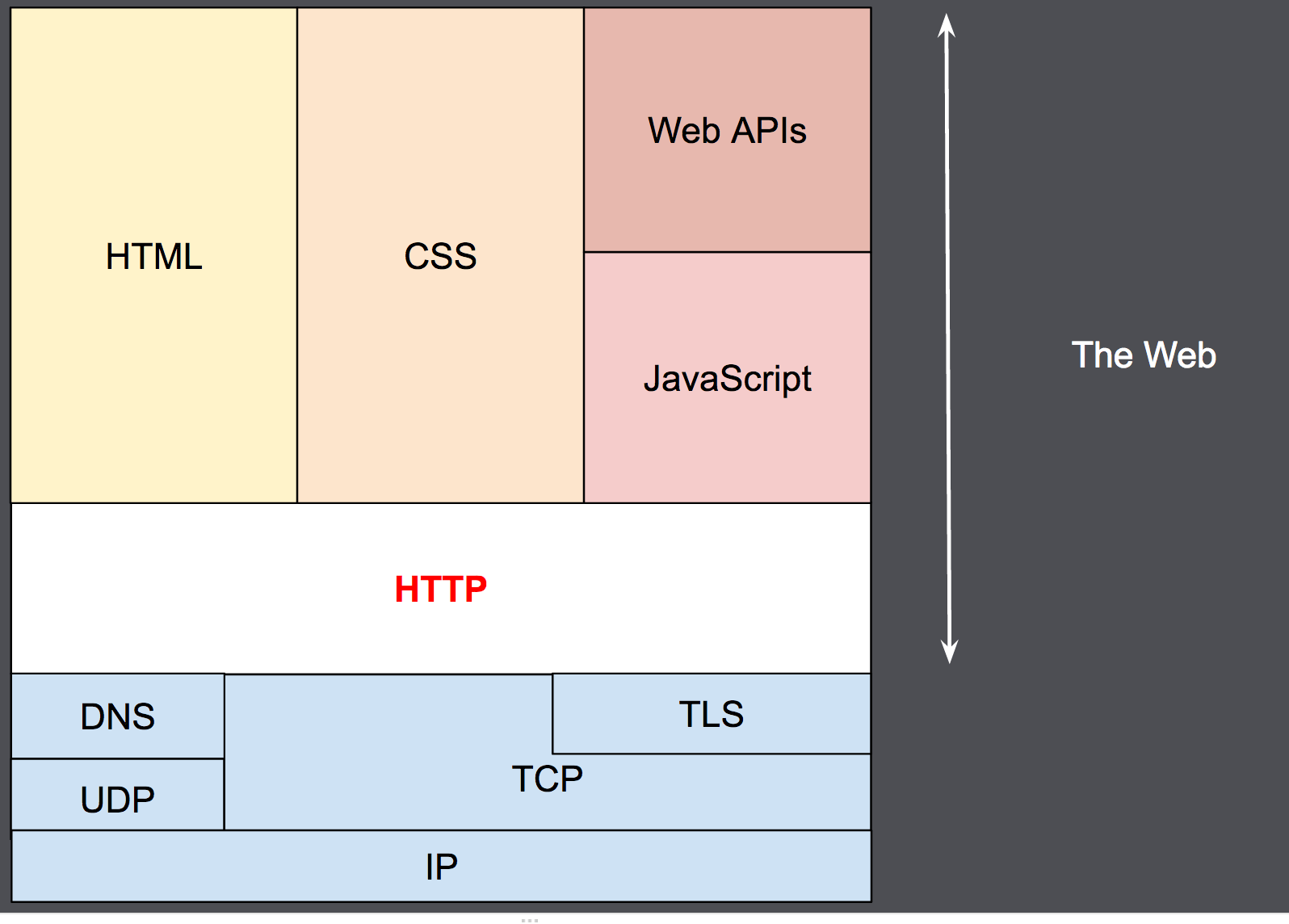
**An object is called an instance of a class**. For example, suppose Bike is a class then we can create objects like bike1 , bike2 , etc from the class. Here's the syntax to create an object.

In Swift, we can use the argument labels **to define a function in an expressive and sentence-like manner**. For example, func sum(of a: Int, and b: Int) { ... } Here, the sum() function has argument labels: of and and . While calling a function, we can use the argument label instead of parameter names.

**HTTP** is a [protocol](https://developer.mozilla.org/en-US/docs/Glossary/Protocol) for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.



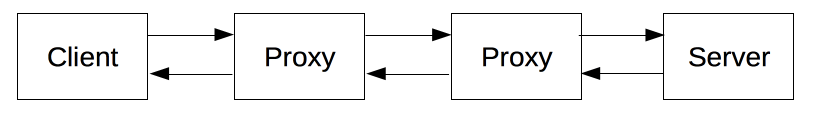
Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses.

Designed in the early 1990s, HTTP is an extensible protocol which has evolved over time. It is an application layer protocol that is sent over [TCP](https://developer.mozilla.org/en-US/docs/Glossary/TCP), or over a [TLS](https://developer.mozilla.org/en-US/docs/Glossary/TLS)-encrypted TCP connection, though any reliable transport protocol could theoretically be used. Due to its extensibility, it is used to not only fetch hypertext documents, but also images and videos or to post content to servers, like with HTML form results. HTTP can also be used to fetch parts of documents to update Web pages on demand.

## [Components of HTTP-based systems](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#components_of_http-based_systems)

HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example, a robot that crawls the Web to populate and maintain a search engine index.

Each individual request is sent to a server, which handles it and provides an answer called the response. Between the client and the server there are numerous entities, collectively called [proxies](https://developer.mozilla.org/en-US/docs/Glossary/Proxy_server), which perform different operations and act as gateways or [caches](https://developer.mozilla.org/en-US/docs/Glossary/Cache), for example.



In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important for diagnosing network problems, the underlying layers are mostly irrelevant to the description of HTTP.

### [Client: the user-agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#client_the_user-agent)

The user-agent is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by programs used by engineers and Web developers to debug their applications.

The browser is **always** the entity initiating the request. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

To display a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then combines these resources to present the complete document, the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.

A Web page is a hypertext document. This means some parts of the displayed content are links, which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web. The browser translates these directions into HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

### [The Web server](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#the_web_server)

On the opposite side of the communication channel is the server, which serves the document as requested by the client. A server appears as only a single machine virtually; but it may actually be a collection of servers sharing the load (load balancing), or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the [Host](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host) header, they may even share the same IP address.

### [Proxies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#proxies)

Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called **proxies**. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

* caching (the cache can be public or private, like the browser cache)
* filtering (like an antivirus scan or parental controls)
* load balancing (to allow multiple servers to serve different requests)
* authentication (to control access to different resources)
* logging (allowing the storage of historical information)

# Protocol

A **protocol** is a system of rules that define how data is exchanged within or between computers. Communications between devices require that the devices agree on the format of the data that is being exchanged. The set of rules that defines a format is called a protocol.

# TCP

**TCP (Transmission Control Protocol)** is an important network [protocol](https://developer.mozilla.org/en-US/docs/Glossary/Protocol) that lets two hosts connect and exchange data streams. TCP guarantees the delivery of data and packets in the same order as they were sent.

TCP's role is to ensure the packets are reliably delivered, error free. TCP has concurrence control, which means the initial requests start small, increasing in size to the levels of bandwidth the computers, servers, and network can support.

# Transport Layer Security (TLS)

**Transport Layer Security (TLS)**, formerly known as [Secure Sockets Layer (SSL)](https://developer.mozilla.org/en-US/docs/Glossary/SSL), is a [protocol](https://developer.mozilla.org/en-US/docs/Glossary/Protocol) used by applications to communicate securely across a network, preventing tampering with and eavesdropping on email, web browsing, messaging, and other protocols. Both SSL and TLS are client / server protocols that ensure communication privacy by using cryptographic protocols to provide security over a network. When a server and client communicate using TLS, it ensures that no third party can eavesdrop or tamper with any message.

All modern browsers support the TLS protocol, requiring the server to provide a valid [digital certificate](https://developer.mozilla.org/en-US/docs/Glossary/Digital_certificate) confirming its identity in order to establish a secure connection. It is possible for both the client and server to mutually authenticate each other, if both parties provide their own individual digital certificates.