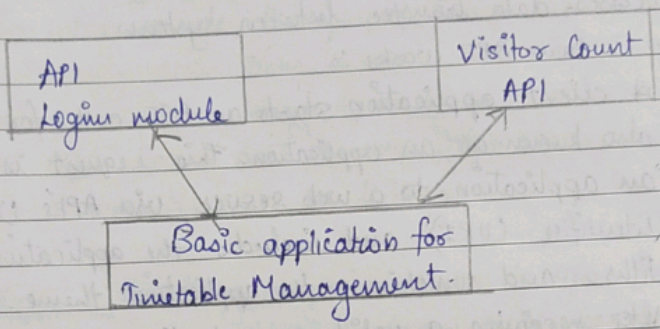
SDS Page No 1
Date

CLOUD COMPUTING API

20 Jan 2022 :-



★ What does cloud Application Programming Interface (API) mean?

- Cloud API is a software communications that allow developers to integrate cloud computing services together. Application programming interfaces (APIs) allow a single computer program to makes its data and applications available for other applications. Developers use APIs to connect software components across the network.
- Cloud APIs are often classified as those that apply to a different vendor or forum. Merchant - specific cloud APIs are written to support single - provider cloud services, while different platforms APIs allows developers to connect services from two or more cloud providers.
- Cloud APIs are usually categorized by type :-
  - PaaS APIs : Platform as a service APIs provide access to background services such as a website.
  - SaaS APIs : Software as a Service APIs facilitates communication between clouds services in the application layer.
  - IaaS APIs : Infrastructure such as service APIs enables cloud-based computer resources and storage to be delivered & deployed very quickly.

2.

✱ How an API works?

- API is a set of defined rules that describe how computers or applications interact. APIs reside between an application and a web server, which serve as an intermediate layer that processes data transfer between systems.
- Here is how API works :-
  - A client application starts an API call for information - also known as an application. This request is processed from an application to a web server via API's Uniform Resource Identifier (URI) and includes the application action, files, and sometimes, the application theme.
  - After receiving a valid request, the API makes a call to an external application or web server.
  - The server sends the API response with the request information
  - API transfers data to the first application you request.

✱ Why we need APIs?

1. Improved Collaborations :- The medium business uses approximately 1,200 cloud applications (the link stays outside IBM), many of which are disconnected. APIs enable integration so that these platforms and applications can communicate seemlessly. With this integration, companies can streamline workflows and improve workplace collaboration. Without APIs, many businesses would not be able to communicate and would suffer from information silos that compromise productivity and performance.

2. Easier Innovation :- APIs offer flexibility allowing companies to make connections with new business partners, offer new services to their existing market, and ultimately, access new markets that can bring greater returns and drive digital transformation. For example, the company Stripe started as an API with only seven lines of ~~good~~ code. The company has since partnered with many of the world's largest businesses, diversified to offer loans and ~~co~~ corporated cards, and recently had a value of USD 36 billion.

3. Data Monetization :- Many companies choose to offer free APIs, at least initially, to build a developer audience closer to their products and to build relationships with potential business partners. However, if the API provides access to important digital assets, you can make money by selling access (this is called API economy). While Accuweather launched its self-help developer website to ~~make~~ market a wide variety of API packages, it took just 10 months to attract 24,000 developers, sell 11,000 API keys and build a thriving community process.

4. Added Security :- As noted above, API creates an additional layer of security between your data and your server. Engineers can continue to strengthen API security by using Transport Layer Security (TLS) tokens, signatures, and encryption; using API gates to control and verify traffic; and practice managing the API effectively.

A Common API Examples :-

Because APIs allow companies to open access to their services while maintaining security and control, they have become an integral part of modern business. Here are some popular examples of application programming :-

1. Standard Login :- A popular example of an API is the function that allows people to access websites using their Facebook, Twitter, or Google login credentials. Their ideal feature allows any website to use an API from one of the most popular services to quickly verify the user, saving time and difficulty setting up a new profile for each website service or new membership.

2. Third Party Processing : For example, the existing "Paypal payment" function that you see on ecommerce websites works with an API. This allows people to pay for products online without disclosing any sensitive data or providing access to unauthorized person.

3. Travel Booking Comparisons :- Travel reservation sites cover thousands of flights, showing the ~~carpet~~ cheapest daily options and destinations. This service is made possible by APIs that gives users of the app access to the latest information about the availability of hotels and airlines companies. With the automatic exchange of data and applications, APIs dramatically reduce the time and effort involved in testing available flights or accomodations.

4. Google Maps :- One of the most common API examples is the Google Maps service. In addition to important APIs that display static or interactive maps, the app uses other APIs and features to provide users with their favourite directions or points. By finding location and multiple layers of data, you can connect to the Maps API when planning transit routes or tracking items on the go, such as delivery vehicle.

5. Twitter :- Each tweet contains key descriptive attributes, including author, unique ID, message, timestamp of posting, and geolocation metadata. Twitter makes public Tweets and responses available to developers and allows developers to post Tweets via the company API.

★ Types of API's :-

Today, most web-based programming links are web APIs that display application data and performance online. Here are four main types of web APIs :-

1. Open APIs are open source editing links for applications that you can access via the HTTP protocol. Also known as public APIs, they define API archive archives and application formats and responses.

2. Partner APIs are the interface for editing applications developed by strategic business partners. Generally, developers can access them APIs in self help mode through the developer API platform publicly. However, they will need to complete the boarding process and get login details to access the Partner APIs.

3. Internal APIs are visible application connectors that are always hidden from external users. These confidential APIs are not available to users outside the company and are intended to improve productivity and communication across all internal development teams.

4. Composite APIs include multiple data or service APIs. These services allow developers to reach multiple conclusions on a single phone. Composite APIs are useful in the construction of microservices where ~~programming~~ performing a single task may require information from a few sources.

★ Types of API protocols :-

- As the use of web APIs has increased, some basic principles are designed to provides users with a specific set of rules that specify acceptable data types and commands. In fact, these API agreements faciliate standard information exchange :

1. SOAP (Simple object Access Protocol) is an XML-based API protocol, which allows users to send and receive data ~~via~~ via SMTP and HTTP. With SOAP APIs, it is easy to share information between applications of software components operating in different locations or written in different languages.

2. XML-RPC is a protocol that relies on a specific XML format to transfer data, while SOAP uses a complaint

XML format. XML-RPC is older than SOAP, but it is much simpler, and lighter compared to using less bandwidth.

3. JSON-RPC protocol is similar to XML-RPC, as both are remote process calls (RPCs), but this one uses JSON instead of the XML formats to ~~to~~ transfer data. Both protocols are simple. Although phones can contain many parameters, they expect only one result.

4. REST (Representational State Transfer) is a set of web API building codes, which means that there are no official standards (unlike those with a protocol). In order to be a RESTful API (also known as the RESTful API), the visual interface must comply with certain structural parameters. It is possible to build RESTful APIs ~~with~~ with SOAP protocols, but these two standards are generally considered competing specifications.

24th January 2022

REST (Representational State Transfer).

- It is a set of web API building codes, which means that there are no official standards (unlike those with a protocol). In order to be a RESTful API, the visual.....

★ Hypertext Transfer Protocol (HTTP) :-

- A communications protocols.
- Allows retrieving inter-linked text documents
  - World Wide Web.
- HTTP Verbs
  - HEAD
  - GET
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT

★ REST (Representational State Transfer)

- Software design software for distributed hypermedia programs such as the World Wide Web.
- Presented in Roy fielding's doctoral dissertation
  - One of the main authors of the HTTP definition.
- A set of network infrastructure principles that define how resources are defined and managed.

★ REST and HTTP :-

- The motivation for REST was to capture the characteristics of the web which made the Web successful.
  - URL Addressable Resources
  - HTTP Protocol
  - Make a request - Receive Response - Display Response

- Exploits the use of the HTTP protocol beyond HTTP POST and HTTP GET
  - HTTP PUT, HTTP DELETE

★ REST – Not Standard

- REST is not a standard
  - JSR 311: JAX-RS: The Java™ API for RESTful Web Services.

- But it uses several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - ~~text~~ text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)

★ Resources:-
- The key abstraction of information ~~in~~ in REST is a resource.
- A resource is a conceptual mapping to a set of entities.
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Thane"), a collection of other resources, a non-virtual object (e.g. a person), and so on
- Represented with a global identifier (URL in HTTP)
  - http://www.google.com/

★ Naming Resources :-

- REST uses URI to identify resources
  - http://localhost/books/
  - http://localhost/books/ISBN-0011
  - http://localhost/books/ISBN-0011/authors

  - http://localhost/classes
  - http://localhost/classes/cs2650
  - http://localhost/classes/cs2650/students

- As soon we traverse the path ~~formmore~~ from more generic to more specific, for navigating the data.

★ Verbs :-

- Represents the actions to be ~~pro~~ performed on resources :-
  - HTTP GET
  - HTTP POST
  - HTTP PUT
  - HTTP DELETE

- HTTP GET :-

Q. How clients ask for the information they seek?
→ By issuing a GET Request transfers the data from the server to the client in some representation.

- GET http://localhost/books
  - Retrieve all books

- GET http://localhost/books/ISBN-0011021
  - Retrieve book identifier with ISBN-0011021

- GET http://localhost/books/ISBN-0011021/authors
  - Retrieve authors for book identified with ISBN-001102)

- HTTP PUT, HTTP POST
  - HTTP POST creates a resource
  - HTTP PUT updates a resource

- POST http://localhost/books
  - Content: {title, authors[], ...}
  - Creates a new book with given properties

- PUT http://localhost/books/isbn-111
  - Content: {isbn, title, authors[], ...}
  - Updates book identified by isbn-111 with submitted properties.

- HTTP DELETE
  - Remove the resource identified by the URI

- DELETE http://localhost/books/ISBN-0011
  - Delete book identified by ISBN-0011

★ Representation :-

Q How data is represented or returned to the client for presentation?

→ There are two main formats :
1. JavaScript Object Notation (JSON)
2. XML

→ It is common to have multiple representations for the same data.

- XML

```
<course>
    <ID> CS2650 </ID>
    <Name> Distributed Multimedia Software </Name>
</course>
```

- JSON

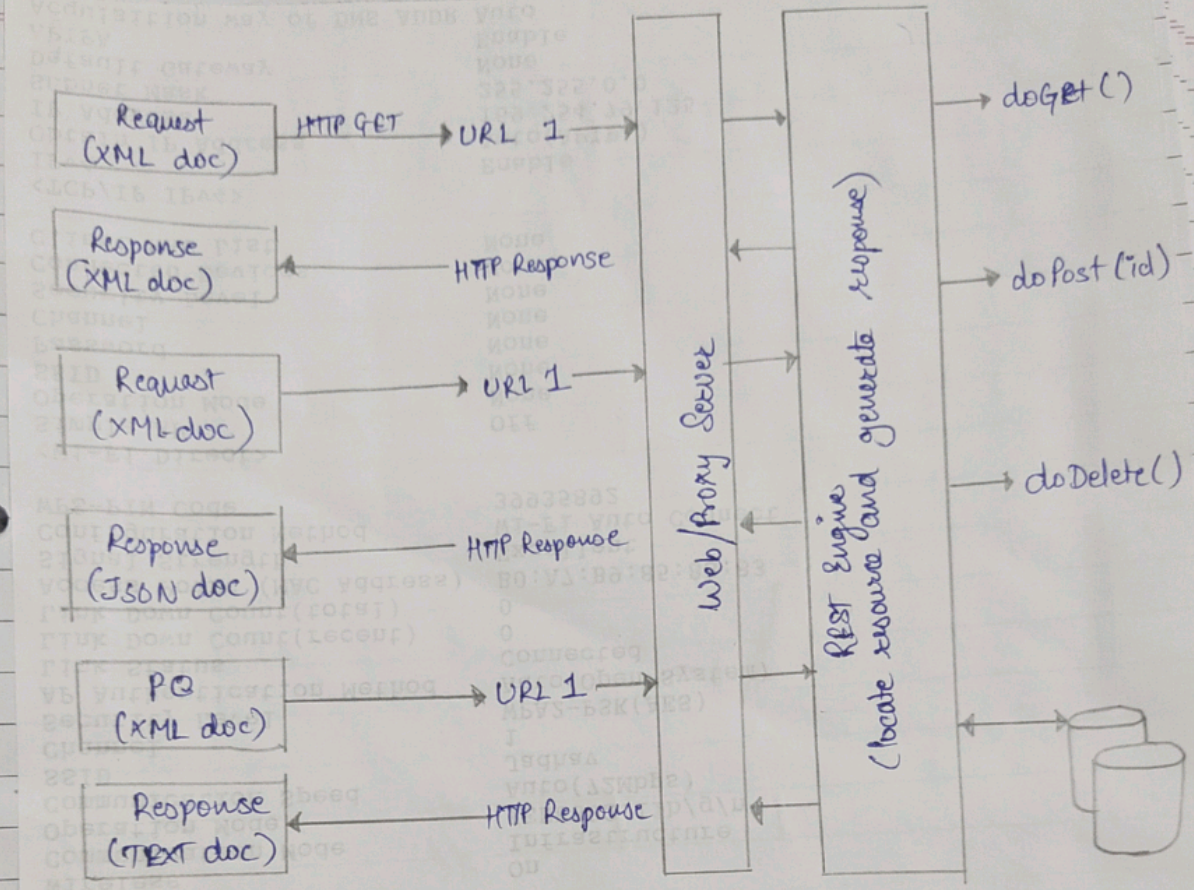
```
{ course
    { id : CS2650 }
    { name : Distributed Multimedia Software }
}
```

★ Why it is called "Representational State Transfer"?

- This term is intended to evoke an image of how a well-designed web system behaves: a network of resources (a virtual machine) where the user develops an application by selecting links (e.g. http://www.example.com/articles/21), leading to the representation of the following resource (next request status) transferred to the client and provided to the user.

★ Real life Examples :-

- Google Maps
- Google AJAX Search API
- Yahoo Search API
- Amazon Web Services



Architectural Style →

★ REST and the Web :-

- The web is an example of the REST system!
- All of those Web services that you have been using for years - book order services, search services, online dictionary services, etc - are REST-based Web services.
- Alas, you were using REST, building REST resources and you didn't even known it.

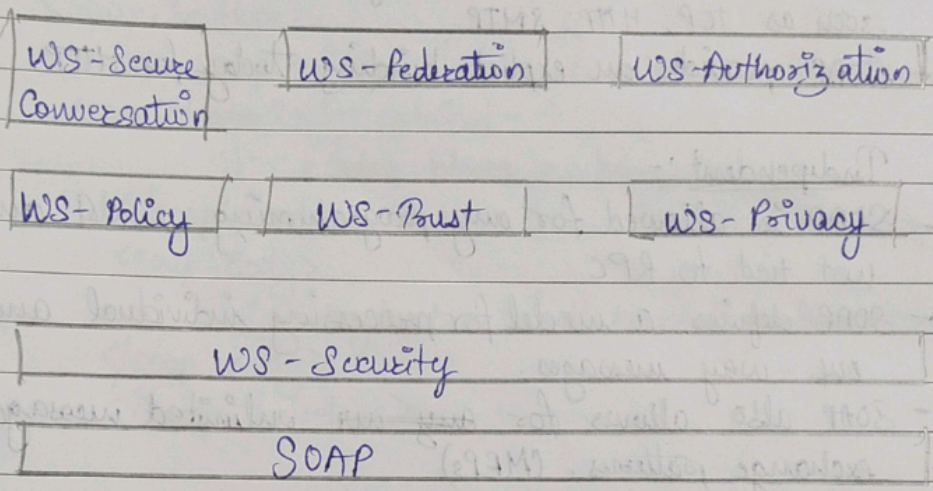
- REST Implementations :-
- Restlet :
  http://www.restlet.org/
- Project Zero
  http://www.projectzero.org/
- Glassfish Jersey
  http://jersey.dev.java.net/
- JBoss RESTeasy
  http://www.jboss.org/resteasy/

SOAP
Simple Object Access Protocol

Q What is SOAP?

→ SOAP is lightweight protocol intended for the exchange of structured information in a separate, distributed environment. SOAP uses XML technology to define expendable message framework, which provides message structure that can be customized with a variety of basic principles. The framework is designed to be independent of any specific planning model and other implementation semantics.

☆ SOAP is the foundation :-



☆ Simply Put.....
SOAP is a single operating system to connect to the same or different operating system, using HTTP (or other transport protocol) and XML.

☆ SOAP Messaging framework :-
- XML-based messaging framework that is
  1. Extensible
  2. Interoperable
  3. Independent

1. Extensible :-
- Simplicity remains one of the main design objectives of SOAP.
- SOAP defines a communication framework that allows features such as security, routing, and reliability to be added later as ~~horizontal~~ layered extensions.

2. Interoperable
- SOAP can be used over any transport protocol such as TCP, HTTP, SMTP
- SOAP provides an explict binding today for HTTP.

3. Independent :-
- SOAP is allowed for any programming model and is not tied to RPC
- SOAP defines a model for processing individual and one-way messages.
- SOAP also allows for ~~any~~ unlimited message exchange patterns. (MEPs)

★ SOAP Message format :-
- SOAP Message consists of three parts :
  - SOAP Envelope
  - SOAP Header (optional)
  - SOAP Body

1. SOAP Envelop :-
- It defines an overall framework for expressing what is in a message and who should deal with it.
- The Envelope is the top element of the XML document representing the message.
  - ~~It~~ this element is always the ~~root~~ root element of a SOAP message.
  - It has ~~a~~ a mandatory Body element. after an optional Header element.

code for SOAP Elevelope :-

```
<soap:Envelope
  xmlns:soap = "https://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header><!--optional-->
        <!-- header blocks goes here-->
  </soap:Header>
  <soap:Body>
        <!-- payload or fault element goes here -->
  </soap:Body>
  </soap:Envelope>
```

2. SOAP Header :-
- This header element is a generic container for controlling information
- It may contain any number of elements for any namespace.
- Header blocks should contain information that influences payload processing
- Header is optional element.

Code

```
<soap:Header>
      <!-- security credentials -->
      <s:credentials xmlns:s = "urn:examples-org:security">
      <username> dave </username>
      <password> exad </password>
  </s:credentials>
  </soap:Header>
```

3. SOAP Body :-
- This element represents to message payload.

Code :-

```
<soap:Body>
<x:Transferfunds xmlns:x="urn:examples-org:banking">
  <from> 22-342439 </from>
  <to> 98-283843 </to>
  <amount>100.00 </amount>
</x:Transferfunds>
</soap:Body>
```

★ SOAP Security :-
- The SOAP specifications does not define any kind of encryption for XML Web Services.
- This is left up to the implementor of the SOAP protocol.

★ Issues about Security :-
- Encryption puts an reliance on transfer protocol.
- Does the transport protocol support secure communications
- What is the cost of ~~does~~ encrypting all the data versus part of the data?

★ SOAP Code with Encryption :-

```
<%@ WebService Language="C#" Class="CreditCardService" %>

using System.Web.Services;
public class CreditCardService {
    [WebMethod]
    [EncryptionExtension(Encrypt=EncryptMode.Response]
```

namespace

```
public String GetCreditCardNumber() {
    return "MC: 4111-1111-1111-1111";
  }
}
```

★ Request Encrypted :-

[code in ppt Cloud computing API - slide - 54]

★ Response Encrypted :-

[11 - slide 55]

SOAP	REST
- An XML-based message protocol.	- An architectural style protocol.
- Uses WSDL for communication between consumer and provider	- Uses XML or JSON to send and receive data.
- Invokes services by calling RPC method	- Simply calls services via URL path.
- Does not return human readable result.	- Results are readable which is in plain XML or JSON file.
- Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	- Transfer is over HTTP only.
- Javascript can call SOAP, but it is difficult to implement	- Easy to call from JavaScript
- Performance is not great compared to REST	- Performance is better than SOAP - with less CPU intensive, leaner code, etc.

25th Jan 2022

* Types of cloud APIs :-

- Cloud APIs are of three types

1. Control APIs :-
   - These API deals with infrastructure management of the cloud. Developers program these APIs mainly for integrating additional program.

2. Data APIs :-
   - A data API deals with the flow of data from the external element to the cloud service itself.

3. APIs related to the Application functionality
   - These APIs deal with the flow of data from the external element to the cloud service itself.
   - These APIs deals with the data present in the cloud and manage its flow for the end-users.

27th Jan 2022

Cloud - PPT

Flow chart of

(Service Providess of PaaS, Saas, IaaS)

28th Jan 2022 :-

APEX

Q What is Apex?

- Apex is a proprietary language developed by Salesforce.com. According to the official definition, Apex is a ~~sbt~~ strongly typed, object-oriented language that allows developers to use flow control statements created on the Force.com platform servers in conjunction with calls to the force.com API.

- It has a Java like-syntax and works like processes stored on a website. Allows developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web Service requests and from triggers on objects. Apex ~~is code~~ is included in Performance Edition, Unlimited Edition, Enterprise Edition, and Developer Edition.

☆ Features of Apex Language :-

- Integrated
- Apex has built in support for DML operations like INSERT, UPDATE, DELETE and also DML Exception Handling. It has support for inline ~~SO~~ SOQL and SOSL query handling which returns the set of Object records. ~~We will study the sObject~~.
- Java like syntax and easy to use.

- Apex is easy to use as it uses the syntax like Java. For eg. variable declaration, loop syntax and conditional statements.
- Strongly integrated with Data.
- Apex is data focused and designed to execute multiple queries and DML statements together. It issues multiple transaction statements on Database.
- Strongly Typed
- It is a strongly typed language. It uses direct reference to schema objects like sObjects and any invalid reference quickly fails if it is deleted or if is of wrong data type.
- Multitenant Environment
- Apex runs in a multitenant environment. Consequently, the Apex runtime engine is designed to guard closely against runaway code, preventing it from monopolizing shared resources. Any code that violates limits fails with easy to understand error messages.
- Upgrades automatically
- Apex is upgraded as part of Salesforce release. No need to upgrade it manually.
- Easy Testing
- Apex provides built-in support for unit test creation and execution, including test results that indicate how much code is covered, and which parts of your code can be more efficient.

☆ When should a Developer choose Apex?

- Apex should be used when we are not be able to implement the complex business functionality using the pre-built and

existing out of the box functionalities.

- When can we use Apex, when we want to :-
  - Create Web services with integrating other systems.
  - Create email services for email blast or email setup.
  - Perform complex validations over multiple objects at the same time and custom validations implementation.
  - Create complex business ~~protocols~~ processes that are not supported by existing workflow functionality or flows.
  - Create custom transactional logic like using the Database methods for updating the records.
  - Perform some logic when a record is modified or modify the related objects record when there is some event which has caused the trigger the cause.

★ Understanding the Apex Syntax :-

- Apex code typically contains many things that we might be familiar with from other programming languages.
- Variable Declaration.
- As strongly typed language, you must declare every variable with data type in Apex. ~~As seen~~
- SOQL Query (Salesforce Object query Language).
  - This will be used to fetch the data from Salesforce database.
- Loop Statement
  - Used for iterating over a list or iterating over a piece of code for a specified number of time.
- Flow Control Statement :-
  - The If statement is used for flow control in this code. Based on certain condition, it is decided whether to go

for execution or to stop the execution of the particular piece of code.

- DML statement :-
  - Performs the records insert, update, insert, delete operation on the records in database.

★ Sandbox or Developer :-

- A Sandbox ~~is an~~ organization is the copy of your organization in which you can write code and test it without taking any risk of data modification or disturbing normal functionality.

★ Apex Code Development Tools :-

- In all the editions, we can use any of the following three tools to develop the code :-
  - force.com Developer Console
  - force.com IDE
  - Code Editor in the Salesforce User Interface

- force.com Developer Console :- It is an integrated development environment with a collection of tools that can be used to create, debug and test applications in Salesforce Organization.

★ Following are a few operations that can be performed using the Developer Console :-

- Writing and Compiling Code - You can write the code using the source code editor. When you save a trigger ~~or~~ class, the code is automatically compiled. Any compilations errors will be reported immediately.

- Debugging - You can view debug logs and set checkpoints that aid in debugging.
- Testing - You can execute tests of specific test classes or all classes in your organization, and you can view test results.
- Checking Performance - You can inspect debug logs to locate performance bottlenecks.
- Color coding and autocomplete - The source code editor uses a color schema for easier readability of code element and provides auto completion for class and method names.

31st Jan 2022.

DATA CENTERS :-

- DataCenters (DC) is a portable center used by businesses to build computer infrastructure and store in a variety of networked formats.
- The main task is to deliver the equipment needed for personnel :-
  - Power
  - Cooling
  - Shelter
  - Security
- Standard data center size :
  500 - 5000 sqm buildings
  1 MW upto $\neq$ 10 - 20 MW power (Average 5 MW)

★ Modern DC for the Cloud Architecture :-

1. Geography:
   - Two or more regions
   - Meets data residency requirements
   - Fault-tolerant from complete region failures.

2. Regions:
   - Set of datacenters with a metropolitan area
   - Network latency perimeter < 2ms

3. Availability Zones:
   - Unique physical locations within a region
   - Each zone made up of one or more DC's
   - Independent power, cooling, networking
   - Inter-AZ network latency < 2ms
   - Fault tolerance from DC failure

★ Data Centers :-

- Traditional data centers:
  - It hosts a large number of applications with small or medium size, each operating on a separate hardware infrastructure and protected by other systems in the same area.
  - Usually in many organizational units or companies.

- **Modern data centers**
  - It is usually owned by one company to process a small number of large applications
    - Google, facebook, Microsoft, Amazon, etc.
  - Use the same hardware and system software
  - Share a common systems management system
  - Sizes may vary according to needs.

☆ Scale-up vs Scale-out
- Scaleup - high cost powerful CPUs, more cores, more memory
- Scaleout - adding more low cost, commodity servers.

☆ Supercomputer vs data center :-
- Scale
  - Blue waters = 40K 8-core "servers"
  - Microsoft Chicago Data Centers = 50 containers = 100K & more servers
- Network Architectures
  - Supercomputers: InfiniBand, low-latency, high bandwidth protocols
  - Data Centers: Ethernet based networks
- Storage:
  - Supercomputers: separate data from
  - Data Centers: use disk on node + memory cache.

☆ Main Components of a datacenters

DC - Slide 8.

01 Feb 2022:-

- Todays ~~data~~ DC use shipping containers packed with 1000s servers each.
- For repairs, whole containers are replaced.

★ Costs for operating a data center:-

- DC's consume 3% of global electricity supply (416.2 TWh > UK's 300 TWh)
- DC's produce 2% of total greenhouse gas emissions
- DC's produce as much CO_2 as the Netherlands or Argentina.

Servers – 57%
Networking Equipment – 8%
Power Distribution and cooling – 18%
Power – 13%
Other Infrastructure – 4%

Monthly Cost:
$3'530'920

★ Power Usage Effectiveness (PUE):-

- PUE is the ratio of
  - The total amount of energy used by a DC facility
  - To the energy delivered to the computing equipment.

- PUE is the inverse of data center infrastructure efficiency.
- Total facility power = covers IT systems + other equipments.

★ Achieving PUE :-

- ~~Do~~ DC Location - cooling and power loading feature
- Raise temperature of aisles :-
  - Usually 18-20C; Google 27C°
  - Upto 35°C (trading failure compared to cooling costs)
- Reduce power conversion
  - Eg, Google motherboards, that operates at 12V instead of 3.3/5V
- Go to extreme environments
  - ~~Arctic~~ ~~Artic Ci~~ Arctic circle (facebook)
  - Floating boats (Google)
  - Underwater DC (Microsoft)
- ~~Reduc~~ Reuse the disipated heat.

★ Evolution of datacenter design :-

- Gen 1 : Colocation (1989 - 2005)
  - 2.0+ PUE
  - Server
  - Capacity
  - 20 year Technology
- Gen 2: Density (2007)
  - 1.4-1.6 PUE
  - Rack
  - Density & Deployment
  - Minimized Resource Impact

Gen 3: Containment (2009)
→ 1.2-1.5 PUE
→ Containers, PODs
→ Scalability & Sustainability
→ Air & Water
→ Economization
→

Gen 4: Modular (2014)
→ 1.12-1.20 PUE
→ ITPAC & Colocations
→ Reduced Carbon
→ Right-Sized
→ Faster-Time-to-Market
→ Outside Air Cooled

- Gen 5 : SW Defined (2015)
  - 1.7 - 1.19 PUE
  - Fully integrated
  - Resident Software
  - Common Infrastructure
  - Operational Simplicity
  - Flexible & Scalable

Gen 6 :- scalable formfactor (2017)
- 1.17 - 1.19 PUE
- Reduced infrastructure, scale to demand

Gen 7 :- Ballard (2018)
- 1.15 - 1.18 PUE
- Design execution efficiency
- Flex capacity enabled

- Gen 8 : Rapid deploy datacenter (2020)
  - Modular construction and delivery
  - Equipment skidding and preassembly
  - Faster speed to market

- Project Natick (future) - 1.07 PUE or less

★ Challenge 1 : Cooling data centers :
Cooling planet at a Google DC in Oregon

★ Challenge 2 : Energy Proportional Computing
- Average real world DC and servers are too inefficient.
  - waste 2/3 rd of their energy.

- Energy consumption is not proportional to the load
  - CPUs are not so bad but the other components are.
  - CPU is the dominant energy consumer in servers - using 2/3 of energy when active/idle.
  - Type to optimize workloads.

★ Challenge 3: Servers are idle most of the time

- For non-virtualized servers 6-15% utilization
- Server virtualization can boost to an average 30% utilization
- Need for resource ~~pending~~ pooling and applications & server consolidation.
- Need for resource virtualization.

★ Challenge 4: Efficient monitoring :-
- Even with virtualization and software defined DC, resource utilization can be poor.
- Need for efficient monitoring (measurement) and cluster management.
- Goal to meet SLOs and SLIs.
- Job's tail latency matters!

★ Improving resource utilization :-
- Hyper-scale system management software
  - Treat the datacenter as a warehouse scale computer.
  - Software defined datacenters
  - System software that allows DC operations to manage the entire DC infrastructure
  - Compose a system using pooled resources of compute, network, and storage based on workload requirement.

- Dynamic resource allocation
  - Virtualization is not enough to improve efficiency.
  - Need the ability to dynamically allocate CPU resources across servers and racks, allowing admins to quickly migrate resources to address the shifting demand.

- Drive 100-300% better Utilization for virtualized WLs, 200-600% for bare metal WLs.

☆ Software Defined DC :-

- Traditional rack architecture	- Disaggregated rack architecture
• GPU Server	• CPU RACK
- Intel/AMD CPU	- Intel/AMD CPU
- DRAM	- Intel/AMD CPU
- NVIDIA GPU	- Intel/AMD CPU
• GPU Server	- "
- Intel/AMD CPU	• Memory Rack
- DRAM	- DRAM x4
- AMD GPU	" " "
• General Server	• GPU Rack
- Intel/AMD CPU	- NVIDIA GPU
- DRAM	- NVIDIA GPU
• SCUTI-O server	- AMD GPU
- Intel/AMD CPU	• SCUTI-O rack
- DRAM	- Scuti-O Local Storage x4
- SCUTI-O Local storage	" " "

☆ Challenge 5: Managing Scale and growth :-

- In 2016, Gartner estimated that Google has 2.5 million servers.

☆ Size and Growth of DC (2016 - 2020)

- The scale and complexity of DC operations grows constantly.
- By 2020, Cisco estimated that we would have 600 million GB of new data saved each day (200 million GB big data).
- So the volume of Bigdata by 2020 was estimated to be as much as all of the stored data in 2016.

★ Challenge 6 : networking at scale (cont.)

- Building the right abstractions to work for a range of workload at hyper-scale.

↓

(SDN) Software Defined Networking

- Within DC, 32 billion GBs will be transported in 2020
  - src: Cisco's report 2016-2020

- "Machine to Machine" traffic is magnitude larger than what goes out on the internet.

★ Cloud & Cloud Computing :-

- Datacenter hardware and software that the vendors use to offer the computing resources and services
- The cloud has a large pod of easily usable virtualized computing resources, development platforms, and various services and applications
- Cloud computing is the delivery of computing as a service.
- The shared resources, software, and data are provided by a provider as a metered service over a network.

★ Application Users :-

- Cloud users :-
  - Software / websites that serve real users
    - Netflix, Pinterest, Instagram, Spotify, AirBnb, Lyft, Slack, Expedia
  - Data analytics, machine learning and other data services.
    - Databricks, Snowflake, GE Healthcare.

- Mobile and IoT backends
  - Snapchat, Zynga (AWS → zCloud → AWS)
- Datacenters own software
  - Google Drive/One Drive, search, etc.

- Cloud Providers
  - Companies with large DCs
    - Amazon AWS, Microsoft Azure, Google Cloud Platform, Alibaba Cloud.

★ Types of Cloud Computing :-

- Public vs Private :-
  - Public : resources owned and operated by the one organization aka the cloud vendor.
  - Private :- Resources used exclusively by a single business or organization.

- On-premise vs Hosted :-
  - On-premise (on-prem) : resources located locally (at a datacenter that the organization operates)
  - Hosted : resources hosted and managed by a third-party provider.

Private cloud can be both on-prem and hosted (virtual private cloud).

- Hybrid cloud :-
  - Combines public and private clouds, allows data and applications to be shared between them.
  - Better control over sensitive data and functionalities.
  - Cost effective, scale well and is more flexible.

- Multi-Cloud
  - Use multiple clouds for an application / service.
  - Avoids data lock-in
  - Avoids single point of failure.
  - But, need to deal with API differences and handle migration across clouds.

☆ Cloud Service models (XaaS) :-

- Infrastructure as a Service (IaaS) :-
  - Rent IT Infrastructure - servers and virtual machines (VMs), storage, network, firewall, and security.
- Platform as a Service (PaaS) :-
  - Get on-demand environment for development, testing and management of software applications : servers, storage, network, OS, databases, etc.
- Serverless, function as a Service (FaaS)
  - Overlapping with PaaS, serverless focuses on building app functionality without managing the servers and infrastructure required to do so.
  - Cloud vendors providers set-up, capacity planning, and server management.
- Software as a Service (SaaS)
  - Deliver software applications over the Internet, on demand.
  - Cloud vendor handles software application and underlying infrastructure

★ Infrastructure as a Service (IaaS):-

- Immediately available computing infrastructure, provisioned and managed by a cloud provider.
- Computing resources pooled together to server multiple users/tenants.
- Computing resources include:
  Storage, processing, memory, network bandwidths, etc.

★ Platform as a Service (PaaS)

- Complete development and deployment environment.
- Includes systems software (OS, middleware), platforms, DBMSs, BI services, and libraries to assist in development and deployment of cloud-based applications.
- Examples:- Google app Engine, Windows Azure, Oracle cloud platform, Amazon web services.

★ Software as a Service (SaaS)

Examples:- Google Apps, Dropbox, salesforce, ~~Exc~~ Microsoft, Slack.

★ Cloud pros and cons:-

- <u>User's benefit:-</u>
  - Elimination of up-front commitment
  - Speed - services are provided on demand
  - Global scale and elasticity
  - Productivity
  - Performance and security
  - Customizability
  - Ability to pay for use of computing resources on a short-term basis (as needed)

- Users' concerns :-
  - Dependability on network and internet connectivity.
  - Security and privacy
  - Cost of migration
  - Cost and risk of vendor lock-in

#HTML 5

★ What is HTML 5 ?

→ HTML5 is the new standard for HTML.

→ The previous version of HTML was - HTML 4.01, came in 1999.

→ HTML5 is designed to deliver almost everything you want to do online without requiring additional plugins. It does everything from animation to apps, music to movies, and can also be used to build complicated applications that run in your browser.

→ HTML5 is also cross-platform (it does not care whether you are using a tablet or a smartphone, a notebook, ~~a booth~~ or a smart TV).

★ Difference between HTML 4 and HTML 5 ?

- HTML5 is a work in progress.
- Simplified Syntax
- The new <canvas> Element for 2D drawings.
- New content-specific elements, like <article>, <header>, <footer>, <nav>, <section>.
- New <menu> and <figure> Elements
- New <audio> and <video> Elements.
- New form controls, like calander, date, time, email, url, search.
- No more <frame>, , <big>, and <b>, <font>
- Support for local storage.

★ Browser Support for HTML5 :-

- HTML5 is not yet an official standard, and no browsers have full HTML5 support.
- But all major browsers (Safari, Chrome, Firefox, Opera, Internet Explorer) continue to add new HTML5 features to their latest versions.

★ The HTML5 <!DOCTYPE>

- In HTML5 there is only one <!doctype> declaration, and it is very simple :

```
<!DOCTYPE html>
```

★ Minimum HTML5 Document :-

- Here is a small HTML5 document, with the minimum of required tags :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> Title of the document </title>
</head>
<body>
        Content of the document
</body>
</html>
```

☆ HTML5 New Elements :-

1. The New <canvas> Element :-
   - The <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

2. New Media Elements :-
   - <audio> - Defines sound content
   - <video> - Defines a ~~audio~~ video or movie
   - <source> - Defines multiple media resources for <video> & <audio>.
   - <embed> - Defines a container for an external application or interactive content (a plug-in)
   - <track> - Defines text tracks for <video> and <audio>.

3. New Form Elements :-
   - <datalist> - Specifies a list of pre-defined option for input controls.
   - <keygen> - Defines a key-pair generator field (for forms)
   - <output> - Defines the result of a calculation.

4. New Semantic / Structural Element :-
   - <article> - Defines an article
   - <aside> - Defines content aside from the page content.
   - <bdi> - Isolates a part of text that might be formatted in a different direction from other text outside it.
   - <command> - Defines a command button that a user can invoke
   - <details> - Defines additional details that the user can view or hide
   - <dialog> - Defines a dialog box or window
   - <summary> - Defines a visible heading for a <details> element.

- <figure> - Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
- <figcaption> - Defines a caption for a <figure> element.
- <footer> - Defines a footer for a document or section.
- <header> - Defines a header for a document or section.
- <mark> - Defines marked/highlighted text.
- <meter> - Defines a scalar measurement within a known range.
- <nav> - Defines navigation links.
- <progress> - Represents the progress of a task.
- <ruby> - Defines a ruby annotation
- <rt> - Defines an explanation/pronounciation of characters.
- <rp> - Defines what to show in browsers that do not support ruby annotations
- <section> - Defines a section in a document.
- <time> - Defines a date/time
- <wbr> - Defines a possible line-break.

★ Removed Elements :-

<acronym>, <applet>, <basefont>, <bgsound>, , <dir>, <font>, <frame>, <frameset>, <noframes>, <strike>, <tt>

☆ New features :-

- New Semantic Elements : These are like <header>, <footer> and <section>.
- Forms 2.0 - Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- Persistent Local Storage - To achieve without resorting to third-party plugins.
- WebSocket - A next-generation bidirectional communication technology for web applications.

- Server - Sent Events - HTML5 introduces events which flows from web server to the web browsers and they are called Server - Sent Events (SSE).
- Canvas - This supports a two-dimensional drawing surface that you can program with JavaScript.
- Audio & Video - You can embed audio or video on your webpages without resorting to third - party plugins.
- Geolocation - New visitors can choose to share their physical location with your web applications.
- Microdata - This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- Drag and drop - Drag and drop the items from one location to another location on the same webpage.

08 Feb 2022 - Practicals
10 Feb 2022 - Practicals
11 Feb 2022 - Practicals
15 Feb 2022 - Practicals

21st Feb 2022 :-

microsoft SILVERLIGHT

- A cross-browser, cross-platform plug-in for delivering the next generation of media experiences & rich interactive applications (RIAs) for the web.

* Silverlight overview :-
- Enable richer interactive web experiences.
  - Vector Graphics, Media, Animation.

- Integrate clearly within existing sites
  - XML markup with AJAX JavaScript
  - Easily incorporated within HTML pages.
- Cross Browser and Cross Platform
  - Enabled via small, one-time, 1.1 Mb download
  - IE, Firefox, Safari support on Windows and Macintosh systems (both Intel and PPC)

★ A few core Silverlight scenarios :-
- Media
- Interactive Content Experiences
- Rich Internet Applications

★ Media Support :-
- Built in Audio and Video Codec Support
  - MP3 and WMA Audio
  - WMV video
- Supports downloading media via standard HTTP requests (works with any web server)
- Will also support ~~bandwidth~~ broadcast/live streaming later this spring for webcasts, events, etc.
- Supports 720 HD video, full screen projection, and best compression in industry.

★ Interactive Content Experiences :-
- Powerful Vector graphics engine
  - Device independent resolution scaling
- Flexible animation system
  - Enable declarative animation of any element.
- Declarative markup approach enables great tool integration and designer/dev workflow.
- Easy AJAX scripting with JavaScript.

★ Rich Internet Applications :-
- Future: not all features enabled in 1.1 Silverlight Alpha
- Rich control encapsulation model
  - Databinding
  - Layout managers
  - Built-in common ~~consoles~~ controls
- .NET framework programming model for these scenarios.

★ Media Interactive Content Demo :- (New topic)

★ .NET for Silverlight & Desktop :-
- .NET for Silverlight is a factored subset of the full .NET
  - Desktop ~50MB (Windows Only)
  - Silverlight + .NET Alpha ~4MB (Cross platform)
  - Additional pieces of .NET available in a pay for play model.
- Same core development framework.
  - The shared technologies and APIs are the same.
  - The tools are the same.
- Highly compatible
  - Minimal changes needed to move from Silverlight to Desktop
  - However, not binary compatible by default.

★ The Sandbox :-

- All apps run in Sandbox
  - Conceptually similar to the HTML DOM sandbox.
- Apps

[Not Continuing this topic, Since this is not in the Portion] But Slidename - Silverlight]

22nd Feb ⟶ Slide No. 18

24th Feb

25th Feb ~~#~~ RUBY

★ MVC :-

— Model-View-Controller (MVC) is a structural pattern that divides an application into three logical main components : model, view, and control. Each of these components is designed to handle specific application development features. MVC is one of the most widely used web development frameworks in the industry to build so ~~mor~~ ~~our~~ many projects and expand.

★ Model :-

Part of the model is compatible with the whole mind related to the data that the user is working on. This may represent data that is transferred between the viewing and control components or any other data related to business thinking. for example, the Customer item will retrieve customer information on the site, trick it and update the data back to the website or use it to provide data.

☆ View :-
The viewing section is used for the entire UI logic of the application. For eg, Customer view will include all UI components such as text boxes, drop-downs, etc. that the final user interacts with.

☆ Controller :-
- Administrators act as a visual link between the Model and View components to process the entire business mindset and incoming applications, manage data using the model section and work with views to deliver final output. For example, the customer controller will manage all the interaction and input from Customer View and update the site using the Customer Model. The same controller will be used to view customer data.

φ What is Ruby?
- Progeamming language
- Object-oriented
- Interpreted

☆ Interpreted Languages :-
- Not compiled like Java.
- Code is written and then directly executed by an Interpreter
- Type commands into interpreter and see immediate results.

Java :- Code → Compiler → Runtime Env → Computer
Ruby :- Code → Interpreter → Computer

☆ What is Ruby on Rails (RoR)
- Development framework for web applications written in Ruby.
- ~~Used by~~

☆ Advantages of framework:-
- Standard features/functionality are built-in
- Predictable application organization
  - Easier to maintain
  - Easier to get things going.

☆ Installation :-

Windows -
- Navigate to : http://www.ruby-lang.org/en/downloads/
- Scroll down to "Ruby on Windows"
- Download the "One-click Installer"
- Follow the install instruction

☆ hello_world.rb
```
puts "hello world!"
```

☆ puts vs print :-
- "puts" adds a new line after it is done
  - analogous System.out.println()
- "prints" does not add a new line
  - analogous to System.out.print()

★ Running Ruby Programs :-

- Use the Ruby interpreter
  ruby hello_world.rb
  - "ruby" tells the computer to use the Ruby interpreter

- Interactive Ruby (irb) console
  irb
  - Get immediate feedback
  - Test Ruby features

★ Comments :-

~~#~~ this is a single line comment

=begin
This is a multiline comment
nothing in here will be part of the code
=end

★ Variables

- Declaration - No need to declare a "type"
- Assignment - same as in Java
- Example :-

```
x = "hello world"     #String
y = 3                 #Fixnum
z = 4.5               # Float
r = 1....10           # Range
```

SDS Page No 28 Date

★ Objects :-

- Everything is an object
  - Common type (classes): Numbers, Strings, Ranges
  - nil, Ruby's equivalent of null is also an object.
- Uses "dot-notation" like Java objects
- You can find the class of any variable
  ```
  x = "hello"
  x.class  → String
  ```
- You can find the methods of any variables of class
  ```
  x = "hello"
  x.methods
  String.methods
  ```
- There are many methods that all Objects have
- Include the "?" in the methods names, its is a Ruby naming convention for boolean methods
  - → nil?
  - → eql? / equal?
  - → ==, !=, ===
  - → instance_of?
  - → is_a?
  - → to_s?

★ Numbers :

- Numbers are Object
- Different Classes of Numbers
  - Fix Num, Float

```
> 3.eql? 2      → false
> -42.abs       → 42
> 3.4round      → 3
> 3.6.round     → 4
> 3.2.ceil      → 4
> 3.8.floor     → 3
> 3.zero?       → false
```

★ String Methods

```
"hello world".length  → 11
"hello world".nil?    → false
"".nil?               → false
"ryan" > "kelly"      → true
"hello-world".instance_of? String
                      → true
"hello" * 3           → "hellohellohello"
"hello" + "world"     → "helloworld"
"hello world".index("w") → 6
```

☆ Operations and Logic

- Same as Java
  - Multiplication, division, addition, subtraction, etc.
- Also same as Java AND Python
  - "and" and "or" as well as "&&" and "||"
- Strange things happen with String
  - String concatenation (+)
  - String multiplication (*)
- Case and Point : There are many ways to solve a problem in Ruby

☆ If/elseif/else/end

- Must use "elseif" instead of "else if"
- Notice use of "end". It replaces closing curly braces in Java.
- Example

```
if (age < 35)
    puts "young whipper-snapper"
elseif (age < 105)
    puts "80 is the new 30"!
else
    puts "wow.... grats"
end
```

★ Inline "if" statement

- Original if statement

```
if age <50
    puts "don't worry, you are still young"
end
```

- Inline if-statement

```
puts "dont worry, you are still young" if age <105
```

★ for-loops

- for-loops can use ranges
- Example 1:-

```
for i in 1...10
    puts i
end
```

- Can also use blocks ~~(covered not)~~

```
3.times do
    puts "Ryan!"
end
```

★ for loops and Ranges

- You may need a more advanced range for your for-loop
- Bounds of a range can be expressions
- Example

```
for i in 1..(2*5)
    puts i
end
```

★ while loops :-
- Can also use blocks
- cannot use "i++"
- Example

```
i = 0
while i<5
  puts i
  i = i+1
end
```

★ unless
- unless is the logical opp. of "if"
- Example :-

```
unless (age >=105)   # if (age <105)
puts "young"
else
puts "old"
end
```

★ until
- Similarly "until" is the logical opposite of "while"
- Can specify a condition to have the loop stop (instead of continuing)
- Example

```
i=0
until (i>=5)   # while(i<5), parenthesis not required
  puts I
i = i + 1
end
```

★ Methods :
- Structure

```
def method-name (parameter1, parameter 2,...)
    Statements
end
```

- Simple Example :-

```
def print_ryan
  puts "Ryan"
end
```

★ Parameters
- No class/type required, just name them!
- Example:

```
def cumulative_sum(num1, num2)
sum=0
for i in num1 .... num2
    sum = sum + i
end
return sum
end
# call the method and print the result
puts (cumulative_sum(1,5))
```

★ Return
- Ruby methods return the value of the last statement in the method, so...

```
def add(num1, num2)
  sum = num1 + num2
  return sum
end
```

- Can become

```
def add(num1, num2)
  num1 + num2
end.
```

★ User Input
- "gets" method ~~objects~~ obtains input from a user
- Example

```
name = gets
puts "hello" + name + "!"
```

- Use chomp to get rid of extra line
  puts "hello " + name.chomp + "!"
- chomp removes trailing new lines.

★ Changing types
- You may want to treat a string a number or a number as a string
  - to_i - converts to an integer (FixNum)
  - to_f - converts a string to a float
  - to_s - converts a number to a string.

Examples
"3.5".to_i ⟶ 3
"3.5".to_f ⟶ 3.5
3.to_s ⟶ "3"

★ Constants :-
- In Ruby, constants begin with an Uppercase
- They should be assigned a value at most once
- This is why local variables begin with a lowercase
- Example :-

```
width = 5
def square
  puts ("*" * width + "\n") * width
end
```

01 March 2022 :- Ruby Example Word File
03 March 2022 :- Ruby program to demonstrate word.
04 March 2022 :- Ruby Example.
07 March 2022 - Graphics in Ruby (last slide game in Ruby).

08 March 2022 - ~~Go~~ RubyEd.com (Programs)
10 March 2022 - Ruby Example word file
11 March 2022 - Ruby Mine IDE
15 March 2022 - No class
24 March 2022 - cloud.ibm.com ← Online site
25 March 2022 - No class
28 March 2022 - Fog Computing and Cloud Computing

Cloud Computing Limitations

1. Not Always Connected:
   - Connected to the Cloud is a pre-requisite of cloud computing
   - Some IoT systems need to work even if connection is temporarily unavailable.

2. Not Always Enough Bandwidth:-
   - Cloud computing assumes that there is enough bandwidth to collect the data.
   - That can become an overly strong assumptions for industrial IoT applications

3. Cloud Computing centralized analytics:-
   - Thus defining the lower bound reaction time of the systems.
   - Some IoT applications won't be able to wait for the data to get to the cloud, be analyzed and for insights to get back.

4. Security Shortcomings:-
   - Existing data protection mechanisms in Cloud Computing such as encryption failed in securing the data from the attackers.

★ Fog Computing :-

- Whereas the cloud is "up there" in the sky somewhere, distant and remote and deliberately abstracted. The "fog" is close to the ground, right where things are getting done.
- Fog computing or fog networking, also known as fogging, is an architecture that uses edge devices to carry out a substantial amount of computing, storage and communication locally and routed over the ~~cable~~ internet backbone.
- Fog computing (aka. Edge computing) is a paradigm that extends Cloud Computing and services to the edge of the network.
- Fog computing places processes and resources at the edge of the cloud, often on network devices, while data remains stored in the cloud.

★ Fog Computing Characteristics :-

- Geographical distribution :- The services and applications objective of the fog is widely distributed
- Support for mobility :- Fog devices provide mobility techniques like decouple host ~~&~~ identity to locate identity.
- Real time interactions :- Fog computing requires real time interactions for speedy service.
- Heterogeneity : Fog nodes can be deployed in a wide variety of environments.
- Interperability : Fog components must be able to interoperate in order to give wide range of services like streaming.

★ Fog vs Cloud

- Reduction in data movement across the network resulting in reduced congestion.
- Elimination of bottlenecks resulting from centralized computing systems.
- Improved security of encrypted data as it stays closer to the end user.

29 March 2022 - TP
01 April 2022 - DC
04 April 2022 - DC
05 April 2022 - TP
07 April 2022 - DC
08 April 2022 - Go Lang.

Q What is Go Lang?

- It is open source language developed on the year 2007 at Google by Robert Griesemer, Rob Pike, Ken Thompson.
- ~~It is compiled~~
- The code is compiled, statically typed and has very fast compilation
- C-like syntax.
- garbage collection
- built-in concurrency
- no classes or type inheritance or overloading or generics
  - unusual interface mechanism instead of Inheritance.

★ features of Go Programming :-

→ the most important features of Go programming are listed below:

- Support for environment adopting patterns similar to dynamic languages.
- for examples, type inference (x := 0 is valid declaration of a variable x of type int)
- ~~Compi~~ Compilation time is fast.
- Inbuilt concurrency support: lightweight processes (via goo routines), channels, select statement.
- Go programs are simple, concise, and safe.
- Support for interface and type embedding.
- Production of statically linked native binaries without external dependencies.

★ A Go program basically consists of the following parts :-

- Package Declaration
- Import Packages
- Functions
- Variables
- Statements and Expressions
- Comments.

★ How Go Lang Program works? printing the words "Hello World!" :-

```
package main
import "fmt"
func main() {
    /* This is my first sample program. */
    fmt.Println("Hello, World!")
}
```

☆ Lets us take a look at the various parts of the above program:

1. The first line of the program package main defines the package name in which this program should lie. It is a mandatory statement, as Go programs run in packages, The main package is the starting point to run the program. Each package has a path and name ~~should~~ associated with it.

2. The next line import "fmt" is a preprocessor command which tells the Go compiler to include the files lying in the package fmt.

3. The next line func main() is the main function where the program execution begins.

4. The next line /*....*/ is ignored by the compiler and it is there to add comments in the program. Comments are also represented using // similar to Java or C++ comments.

5. The next line fmt.Println(...) is another function available in Go which causes the message "Hello, World!" to be displayed on the screen. Here fmt packages has exported Println method which is used to display the message on the screen 6. Notice the capital P of Println method. For Go language, a name is exported if starts with capital letter. Exported means the function or variable/constant is accesible to the importer of the respective package.

★ Executing a Go Program :-

1. Open a text editor and add the program.
2. Save the file as hello.go
3. Open the command prompt.
4. Go to the directory where you saved the file.
5. Type go run hello.go and press enter to run your code.
6. If there are no errors in your code, then you will see "Hello World!" printed on your screen.

★ Tokens in Go

- A Go program consists of various tokens. A token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following Go statement consists of six tokens :

```
fmt.Println("Hello, World!")
```

The individual tokens are

```
fmt
.
Println
(
"Hello World!"
)
```

★ Line Separator :-

In a go program, the line separator, key is a statement terminator, that is, individual statements don't need a special seperator like ";" in C. The Go compiler internally places ";" as the statement terminator to indicate the end of one logical entity.

For eg, take a look at the following statements :-

```
fmt.Println("Hello, World!")
fmt.Println("I am in Go Programming World!")
```

★ Comments :-

- Comments are like helping texts in your Go program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below :

```
/* my first program in Go */
```

★ Identifiers :-

A Go identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore... followed by zero or more letters, underscores and digits (0 to 9).

identifier = letter { letter | unicode_digit }.

Go does not allow punctuation characters such as @, $, and % within identifiers. Go is a case-sensitive programming language. Thus, manpower and Manpower are two different identifiers in Go. Here are some eg :-

mahesh kumar abc move_name a_123
myname50 _temp j a2869 retVal

✱ Data Type

- In the Go progeamming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupied in storage & how the bit pattern stored is interpreted.

Types :-

1. Boolean types :-
- They are boolean types and consists of the two predefined constants : (a) true (b) false

2. Numeric Types :-
- They are again arithmetic types and they represents a) Integer types or b) floating point values throughout the progeam.

3. String Types :-
- A string type represents the set of string values. Its value is a sequence of bytes. Strings are immutable types. That is once they are created, it is not possible to change the contents of a string. The predeclard string type is string

4. Derived types :-
They include (a) Pointer Type, (b) Arsay Types (c) Structure types (d) Union types and (e) function types (f) Slice types (g) function types (h) Interface types (i) Map types (j) Channel Types.

☆ Variables :-

1. byte - Typically a single octet (one byte). This is an byte type.
2. int - The most natural size of integer for the machine.
3. float32 - A single-precision floating point value.

☆ Variable Definition in Go

- A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

```
var variable_list optional_data_type;
```

- Here optional_data_type is a valid Go data type including byte, int, float32, complex64, boolean or any user-defined object etc., and variable_list may consist of one or more identifier names separated by commas. Some valid declarations :-

```
var     i, j, k int;
var     c, ch byte;
var     f, salary float32;
d = 42;
```

☆ Operators :-

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Miscellaneous Operators.