MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE,
ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS
KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
(AUTONOMOUS)

(Reaccredited 'A' Grade by NAAC)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
(ISO 9001:2015)

CERTIFICATE

Name: Mr. **Suyash Gaikwad**

Roll No: **140** **Programme**: BSc IT **Semester**: II

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **IT platforms, Tools and Practices** (Course Code: **2026UISTP**) for the partial fulfillment of Second Semester of BSc IT/CS during the academic year 2020-2021.

The journal work is the original study work that has been duly approved in the year 2020-2021 by the undersigned.

External Examiner

Subject-In-Charge
(Ms.Sweety Garg)

Date of Examination: (College Stamp)

Name: **<u>Mehul Vinod Gohil</u>** **Roll No:** **<u>14</u>**

Sr. No.	**DATE**	**TITLE**	**SIGN**
1.	2/2/21	INTRODUCTION and CONTRIBUTING TO WIKIPEDIA a) What is Wikipedia? b) Steps to Create Account on Wikipedia c) Creating Page on Wikipedia d) Edit your page	
2.	9/2/21	Creating account, repository on GitHub and Cloning repository in GitHub Page	
3.	16/2/21	BASIC UNDERSTANDING ON FREE AND OPEN-SOURCE SOFTWARE a) Describe Open-Source Software with Example. b) Describe Free Software with Example c) Difference between Free and Open-Source Software.	
4.	23/2/21	WRITING EMAIL	
5.	25/2/21	Using practical examples, describe green computing. List and explain the steps that you take to contribute to green computing	
6.	2/3/21	WRITING BLOGS	
7.	9/3/21	Implementing coding practices in Python using PEP8.	
8.	9/3/21	PRESENTATION: Unified communication	

Name:- Mehul Vinod Gohil

Roll No:-14 fyiit

PRACTICAL :-1

Introduction and contribution to Wikipedia

Introduction:-

Since its debut in 2001, Wikipedia has grown to be a very popular and well-known online resource. All of the content on the site I created and edited by and offered under a creative commons attribution /sharelaike 3.0 license.it has:

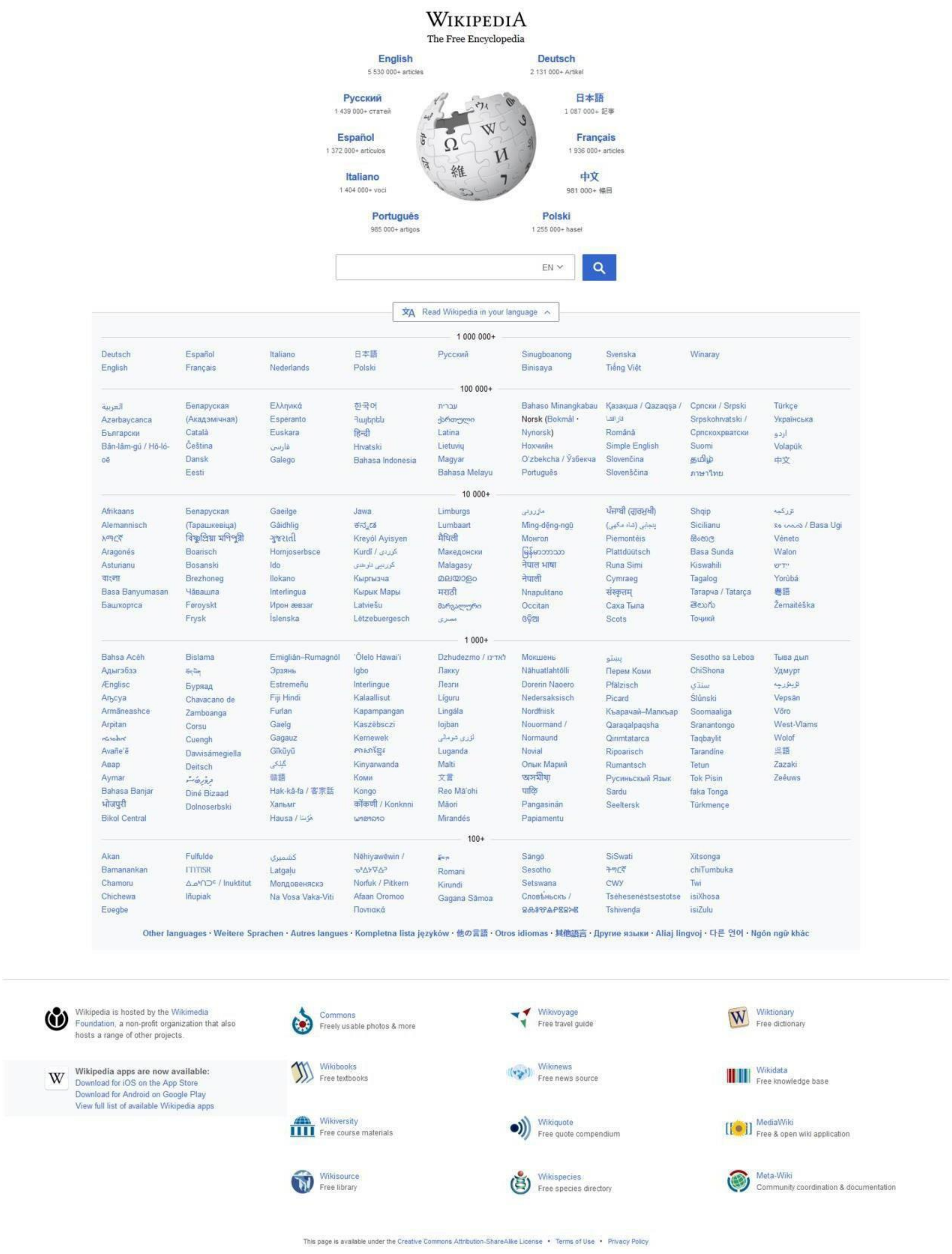
Over 4.6 million articles;

Over 33.9 million pages;

Over 22.7 million user account with 73,251 active user a of May 2014.

Contributing:-

Wikipedia is the six most popular website in the world and we almost all use it. But, do you know how to contribute to it? This guide will help you to get started making edit and additions to Wikipedia and uploading content to Wikimedia commons.



Wikipedia
The Free Encyclopedia

English
5 530 000+ articles

Deutsch
2 131 000+ Artikel

Русский
1 439 000+ статей

日本語
1 087 000+ 記事

Español
1 372 000+ artículos

Français
1 936 000+ articles

Italiano
1 404 000+ voci

中文
981 000+ 條目

Português
985 000+ artigos

Polski
1 255 000+ haseł

EN

Read Wikipedia in your language

1 000 000+

Deutsch
English
Español
Français
Italiano
Nederlands
日本語
Polski
Русский
Sinugboanong Binisaya
Svenska
Tiếng Việt
Winaray

100 000+

العربية
Azərbaycanca
Български
Bân-lâm-gú / Hō-ló-oē
Беларуская (Акадэмічная)
Català
Čeština
Dansk
Eesti
Ελληνικά
Esperanto
Euskara
فارسی
Galego
한국어
Հայերեն
हिन्दी
Hrvatski
Bahasa Indonesia
עברית
ქართული
Latina
Lietuvių
Magyar
Bahasa Melayu
Bahaso Minangkabau
Norsk (Bokmål · Nynorsk)
Нохчийн
O'zbekcha / Ўзбекча
Português
Қазақша / Qazaqşa / قازاقشا
Română
Simple English
Slovenčina
Slovenščina
Српски / Srpski
Srpskohrvatski / Српскохрватски
Suomi
தமிழ்
ภาษาไทย
Türkçe
Українська
اردو
Volapük
中文

10 000+

Afrikaans
Alemannisch
አማርኛ
Aragonés
Asturianu
বাংলা
Basa Banyumasan
Башҡортса
Беларуская (Тарашкевіца)
বিষ্ণুপ্রিয়া মণিপুরী
Boarisch
Bosanski
Brezhoneg
Чӑвашла
Føroyskt
Frysk
Gaeilge
Gàidhlig
ગુજરાતી
Hornjoserbsce
Ido
Ilokano
Interlingua
Ирон æвзаг
Íslenska
Jawa
ಕನ್ನಡ
Kreyòl Ayisyen
Kurdî / كوردی
کوردیی ناوەندی
Кыргызча
Кырык Мары
Latviešu
Lëtzebuergesch
Limburgs
Lumbaart
मैथिली
Македонски
Malagasy
മലയാളം
मराठी
მარგალური
مصرى
مازِرونی
Ming-dĕng-ngṳ̄
Монгол
မြန်မာဘာသာ
नेपाल भाषा
नेपाली
Nnapulitano
Occitan
ଓଡ଼ିଆ
ਪੰਜਾਬੀ (ਗੁਰਮੁਖੀ)
پنجابی (شاہ مکھی)
Piemontèis
Plattdüütsch
Runa Simi
Cymraeg
संस्कृतम्
Саха Тыла
Scots
Shqip
Sicilianu
සිංහල
Basa Sunda
Kiswahili
Tagalog
Татарча / Tatarça
తెలుగు
Тоҷикӣ
تۆرکجه
ᨅᨔ ᨕᨘᨁᨗ / Basa Ugi
Vèneto
Walon
ייִדיש
Yorùbá
粵語
Žemaitėška

1 000+

Bahsa Acèh
Адыгэбзэ
Ænglisc
Аҧсуа
Armãneashce
Arpitan
ܐܪܡܝܐ
Avañe'ẽ
Авар
Aymar
Bahasa Banjar
भोजपुरी
Bikol Central
Bislama
བོད་ཡིག
Буряад
Chavacano de Zamboanga
Corsu
Cuengh
Dawisámegiella
Deitsch
ދިވެހިބަސް
Diné Bizaad
Dolnoserbski
Emiliàn–Rumagnòl
Эрзянь
Estremeñu
Fiji Hindi
Furlan
Gaelg
Gagauz
Gĩkũyũ
گیلکی
贛語
Hak-kâ-fa / 客家話
Хальмг
Hausa / هَوُسَا
ʻŌlelo Hawaiʻi
Igbo
Interlingue
Kalaallisut
Kapampangan
Kaszëbsczi
Kernewek
ភាសាខ្មែរ
Kinyarwanda
Коми
Kongo
कोंकणी / Konknni
ພາສາລາວ
Dzhudezmo / לאדינו
Лакку
Лезги
Líguru
Lingála
lojban
لۊری شومالی
Luganda
Malti
文言
Reo Mā'ohi
Māori
Mirandés
Мокшень
Nāhuatlahtōlli
Dorerin Naoero
Nedersaksisch
Nordfriisk
Nouormand / Normaund
Novial
Олык Марий
অসমীয়া
पालि
Pangasinán
Papiamentu
پښتو
Перем Коми
Pfälzisch
Picard
Къарачай–Малкъар
Qaraqalpaqsha
Qırımtatarca
Ripoarisch
Rumantsch
Русиньскый Язык
Sardu
Seeltersk
Sesotho sa Leboa
ChiShona
سنڌي
Ślůnski
Soomaaliga
Sranantongo
Taqbaylit
Tarandíne
Tetun
Tok Pisin
faka Tonga
Türkmençe
Тыва дыл
Удмурт
ئۇيغۇرچە
Vepsän
Võro
West-Vlams
Wolof
吳語
Zazaki
Zeêuws

100+

Akan
Bamanankan
Chamoru
Chichewa
Eʋegbe
Fulfulde
𐌲𐌿𐍄𐌹𐍃𐌺
ᐃᓄᒃᑎᑐᑦ / Inuktitut
Iñupiak
کٲشُر
Latgaļu
Молдовеняскэ
Na Vosa Vaka-Viti
Nēhiyawēwin / ᓀᐦᐃᔭᐍᐏᐣ
Norfuk / Pitkern
Afaan Oromoo
Ποντιακά
རྫོང་ཁ
Romani
Kirundi
Gagana Sāmoa
Sängö
Sesotho
Setswana
Словѣньскъ / ⰔⰎⰑⰂⰡⰐⰠⰔⰍⰟ
SiSwati
ትግርኛ
ᏣᎳᎩ
Tsėhesenėstsestotse
Tshivenḓa
Xitsonga
chiTumbuka
Twi
isiXhosa
isiZulu

Other languages · Weitere Sprachen · Autres langues · Kompletna lista języków · 他の言語 · Otros idiomas · 其他語言 · Другие языки · Aliaj lingvoj · 다른 언어 · Ngôn ngữ khác

Wikipedia is hosted by the Wikimedia Foundation, a non-profit organization that also hosts a range of other projects.

Wikipedia apps are now available:
Download for iOS on the App Store
Download for Android on Google Play
View full list of available Wikipedia apps

Commons
Freely usable photos & more

Wikivoyage
Free travel guide

Wiktionary
Free dictionary

Wikibooks
Free textbooks

Wikinews
Free news source

Wikidata
Free knowledge base

Wikiversity
Free course materials

Wikiquote
Free quote compendium

MediaWiki
Free & open wiki application

Wikisource
Free library

Wikispecies
Free species directory

Meta-Wiki
Community coordination & documentation

This page is available under the Creative Commons Attribution-ShareAlike License • Terms of Use • Privacy Policy

a) Description about Wikipedia and it's features

Wikipedia I a free, open content online encyclopedia created through the collaborative effort of a community of users known as Wikipedians. Anyone registered on the site can create an article for publication; registration is not reuired to edit article .Wikipedia only non commercial site of the top ten.

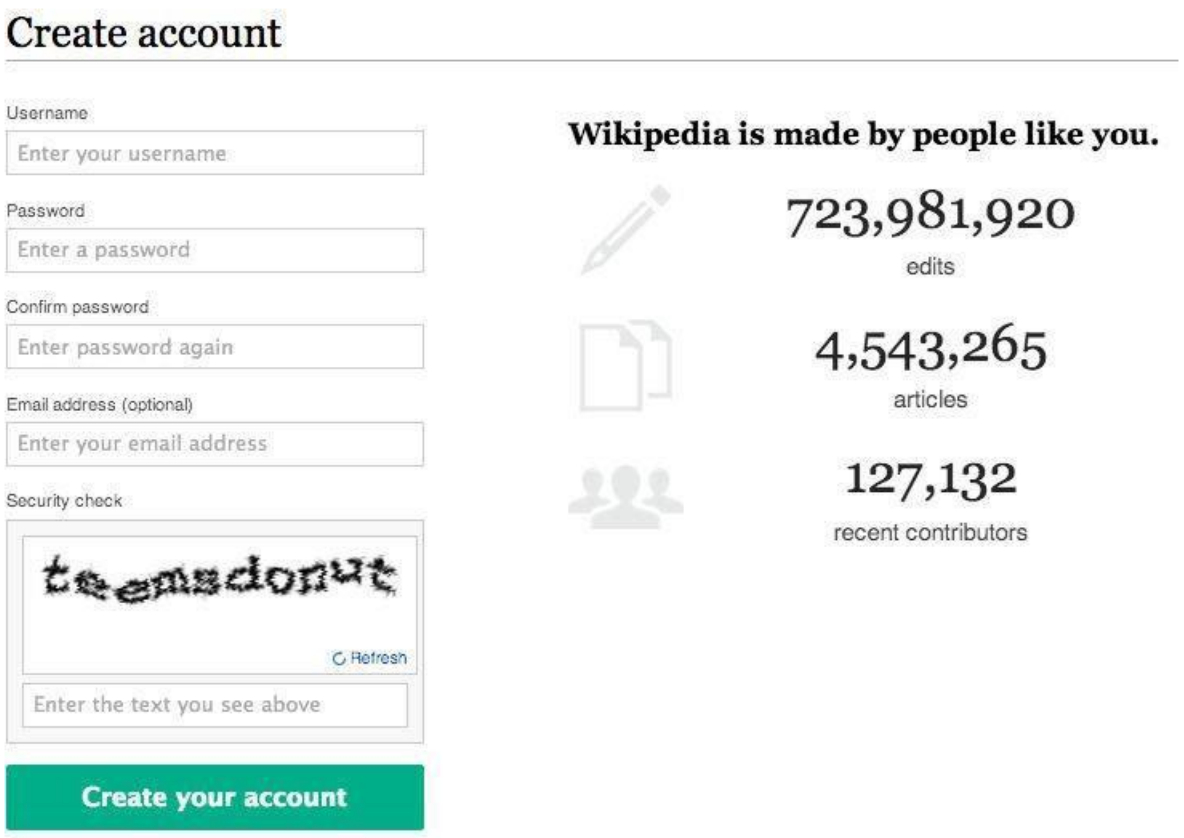
Basic features of a Wikipedia

1) Create a page.
2) Edit a page .
3) Link between pages.

b) Creating Account on Wikipedia.

Steps are given below:-

If you decide to create account, the process is very streamlined. You will need to select a username and a password. But that is only information that is required. You have the option to supply an email address, which I needed to receive email notification or rest your password if you lose it. You can see the “create Account” below.


Create account

Username

Enter your username

Password

Enter a password

Confirm password

Enter password again

Email address (optional)

Enter your email address

Security check

Enter the text you see above

Create your account

Wikipedia is made by people like you.

723,981,920 edits

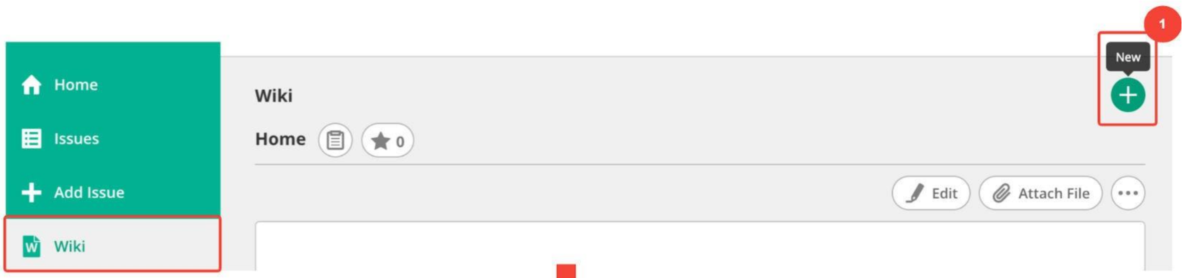
4,543,265 articles

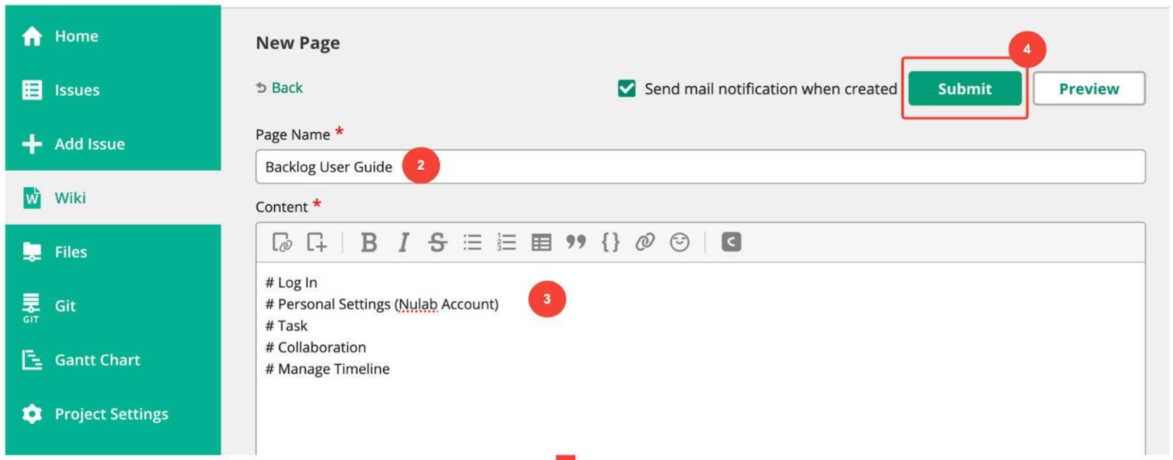
127,132 recent contributors

c) Create a page

To contribute to a Wiki, you'll start by creating a page. All pages have a title and text. Unlike pages of a normal website, you won't need to know any code to contribute content here. You can write and format text on a Wiki page like you would an email.

When you're done, all you have to do is click “Submit” to save the page and it can immediately be read by anyone.



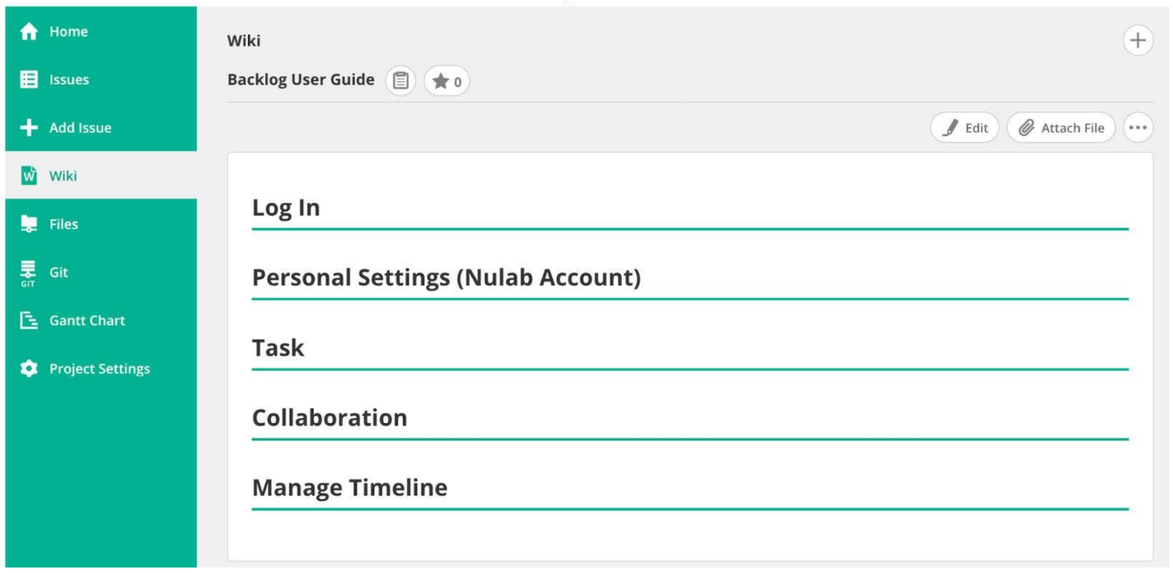


Page Name *

Backlog User Guide

Content *

```
# Log In
# Personal Settings (Nulab Account)
# Task
# Collaboration
# Manage Timeline
```



Backlog User Guide

Log In

Personal Settings (Nulab Account)

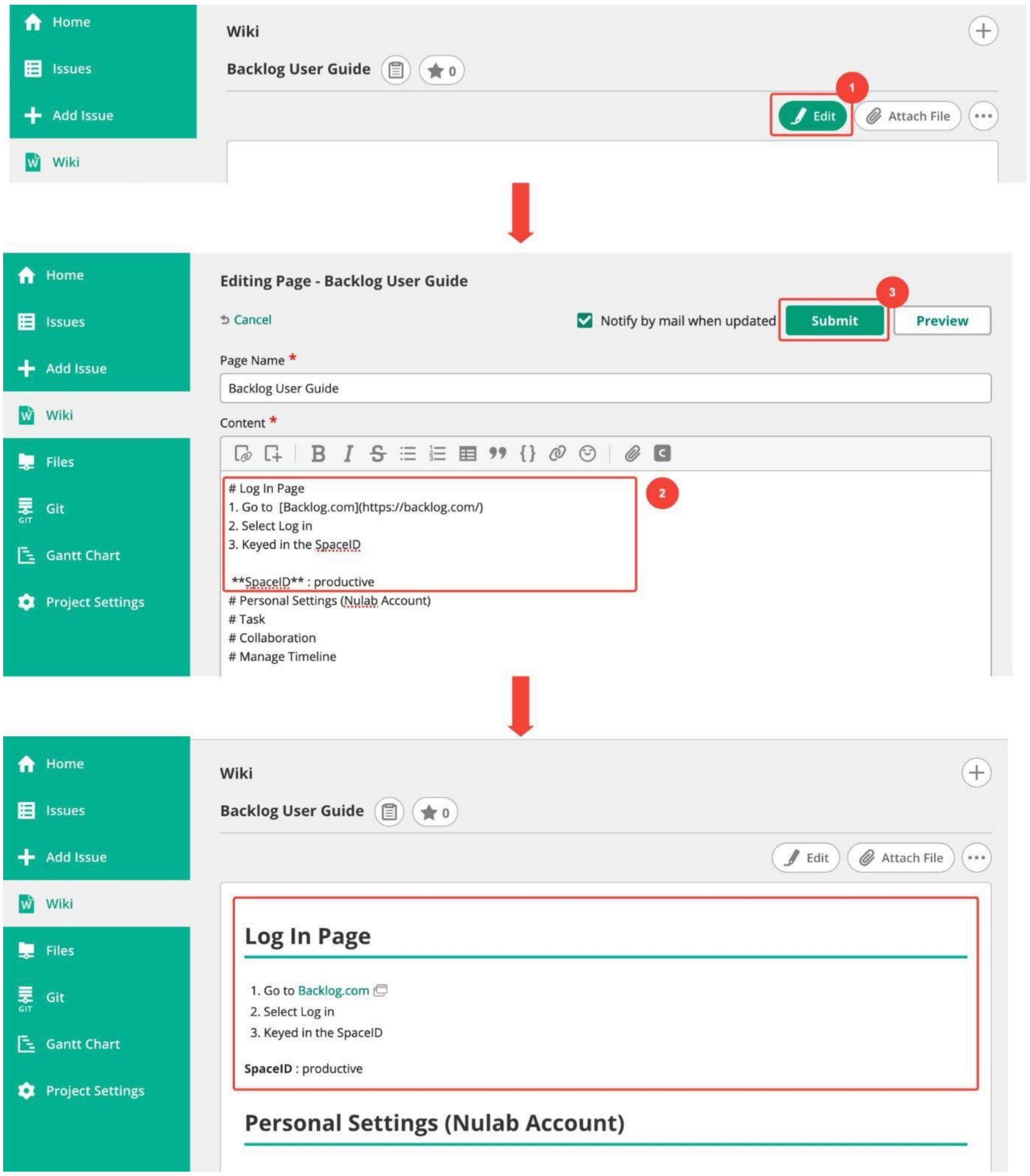
Task

Collaboration

Manage Timeline

d) Edit a page

Published pages can be edited by anyone that reads them. If, for example, you notice that a page created by someone else contains inaccurate/outdated information, you can open the editing screen and correct that information directly. Just “Submit”, and the page's contents will update immediately.



```
# Log In Page
1. Go to  [Backlog.com](https://backlog.com/)
2. Select Log in
3. Keyed in the SpaceID

 **SpaceID** : productive
# Personal Settings (Nulab Account)
# Task
# Collaboration
# Manage Timeline
```

Log In Page

1. Go to Backlog.com
2. Select Log in
3. Keyed in the SpaceID

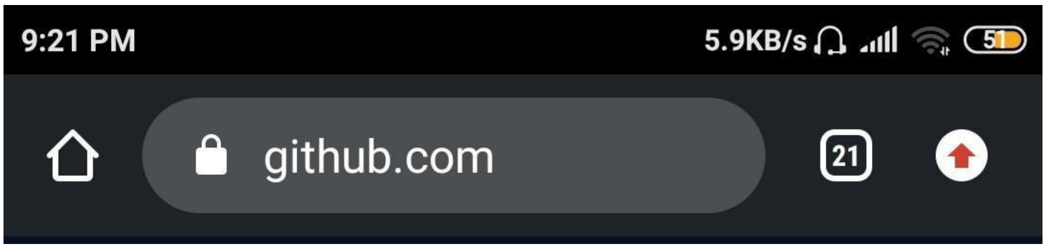
SpaceID : productive

Personal Settings (Nulab Account)

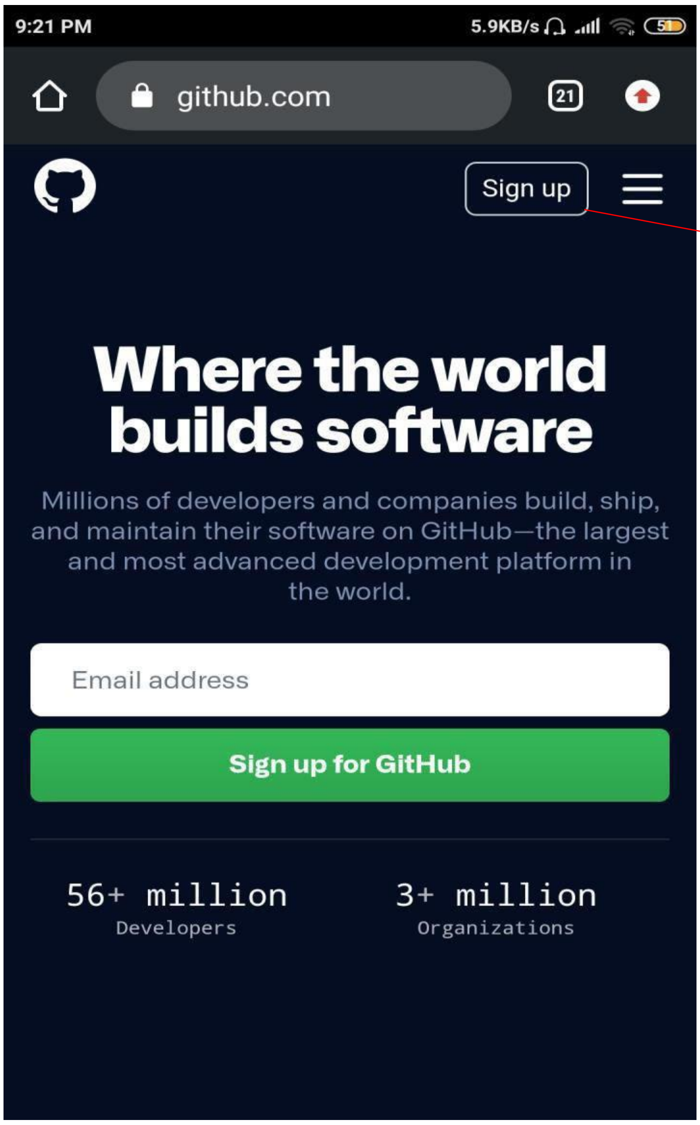
PRACTICAL 2: Creating account, repository on Github s and cloning repository in Github

a) Creating account: -

Step1: - Search Github on Google and click on the official link of Github.



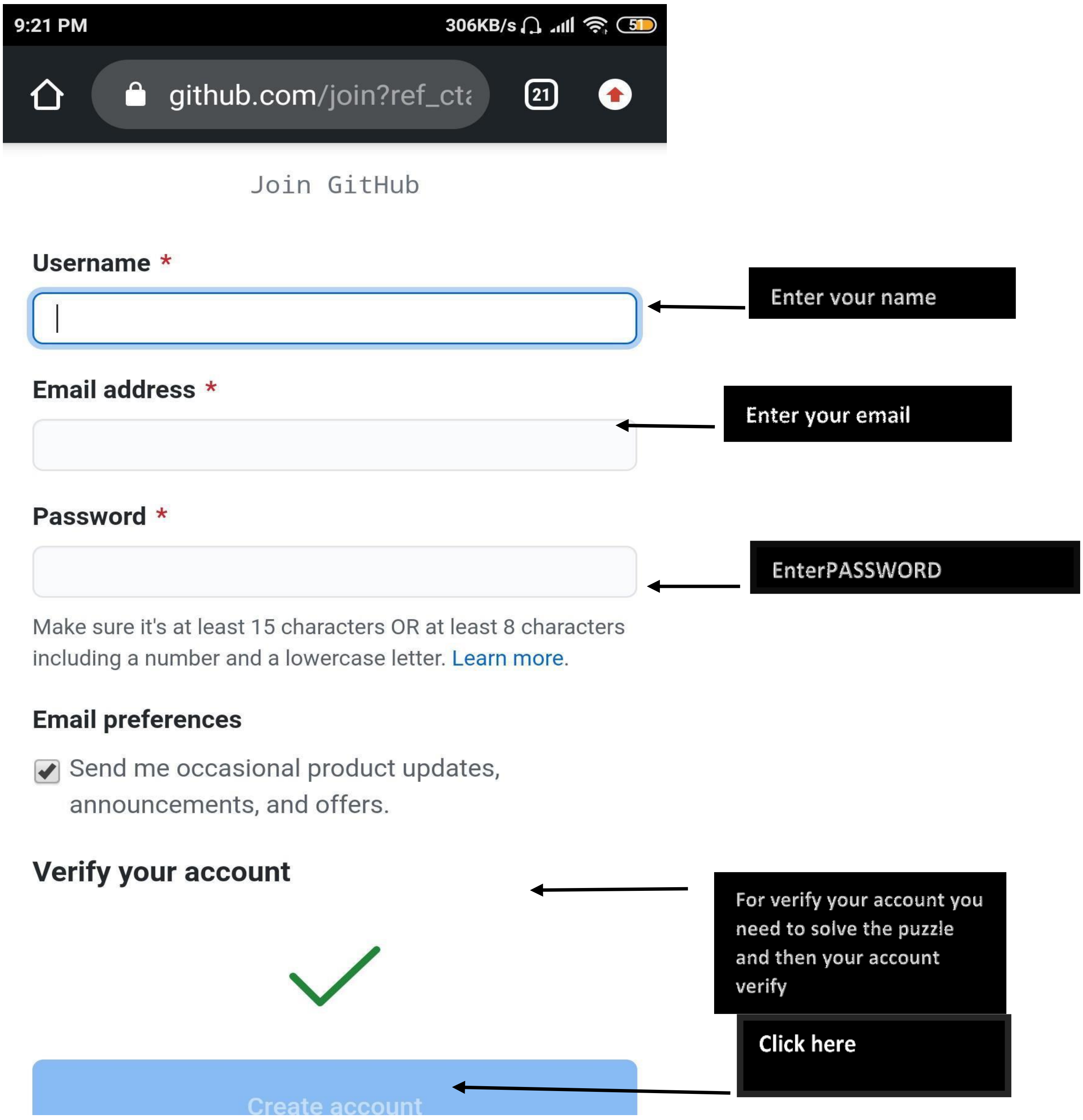
Step 2: Click on the sign up button to create an account on Github.



Where the world builds software

Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

Step 3: When you click on the signup button you will reach on this page here you have to fill all necessary information which is star marked.



9:21 PM 306KB/s

github.com/join?ref_cta

Join GitHub

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Email preferences

✓ Send me occasional product updates, announcements, and offers.

Verify your account

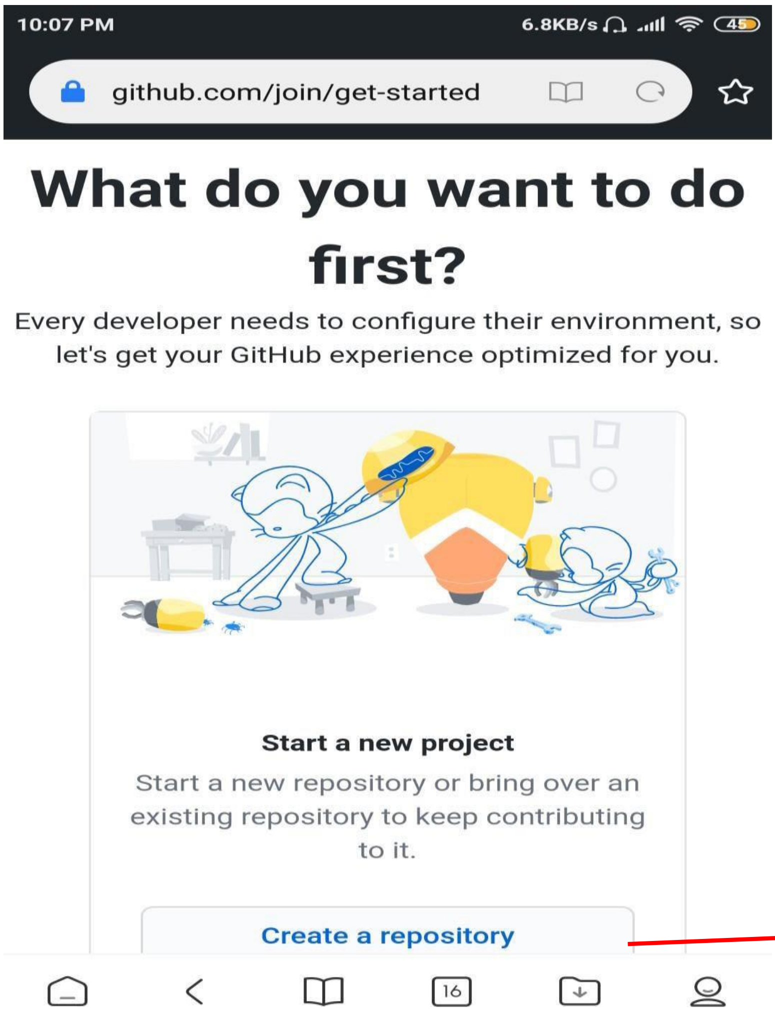
Create account

Enter vour name
Enter your email
EnterPASSWORD
For verify your account you need to solve the puzzle and then your account verify
Click here

Step 4: After fill all the information verify your account and click on the Create Account button. Now your account is created.

b) Creating repository:

Step 1: To create repository click create repository


10:07 PM 6.8KB/s

github.com/join/get-started

What do you want to do first?

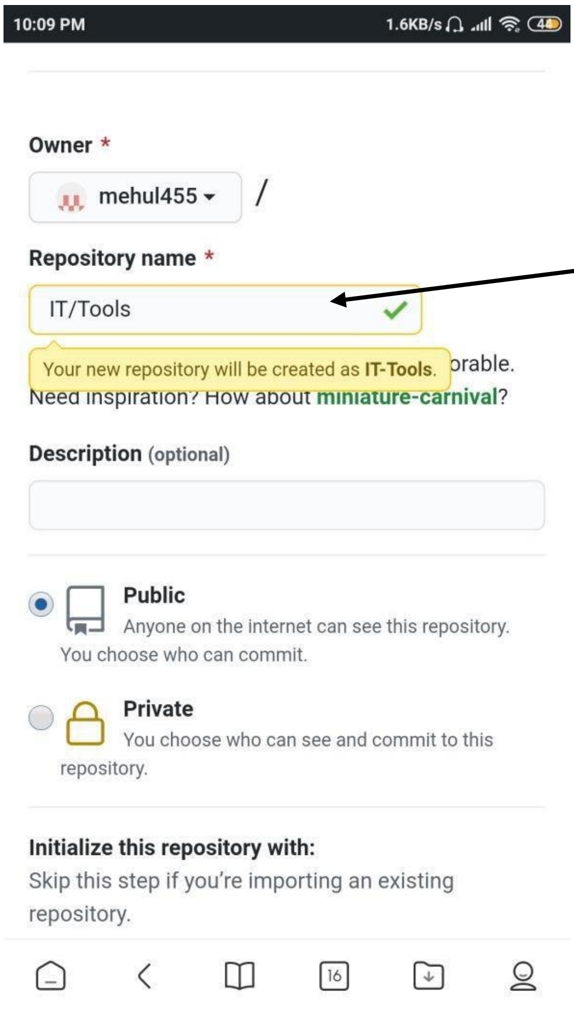
Every developer needs to configure their environment, so let's get your GitHub experience optimized for you.

Start a new project

Start a new repository or bring over an existing repository to keep contributing to it.

Step : After clicking on the create a repository you reach on this page here you have to give your Repository name, click on the private or public button.

And then last create repository



10:09 PM 1.6KB/s

Owner *

mehul455 /

Repository name *

IT/Tools

Your new repository will be created as **IT-Tools**. orable.
Need inspiration? How about **miniature-carnival**?

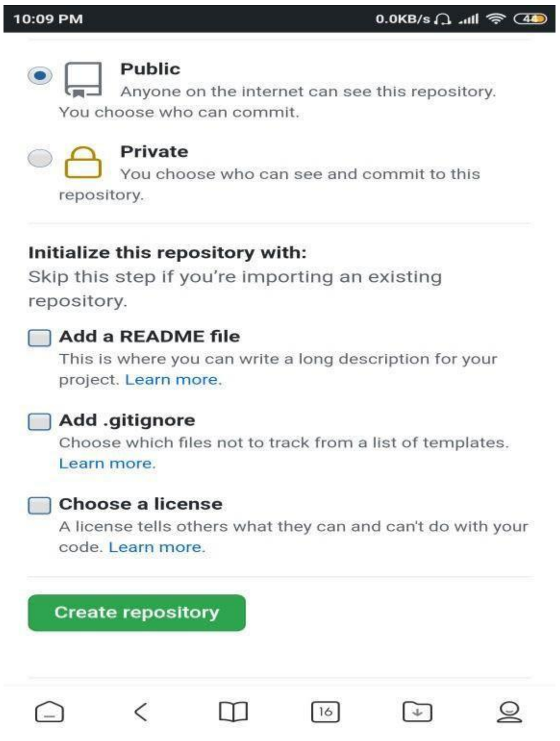
Description (optional)

(●) **Public**
Anyone on the internet can see this repository. You choose who can commit.

( ) **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

NAME OF YOUR REPOSITORY



10:09 PM 0.0KB/s

(●) **Public**
Anyone on the internet can see this repository. You choose who can commit.

( ) **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

☐ **Add .gitignore**
Choose which files not to track from a list of templates. Learn more.

☐ **Choose a license**
A license tells others what they can and can't do with your code. Learn more.

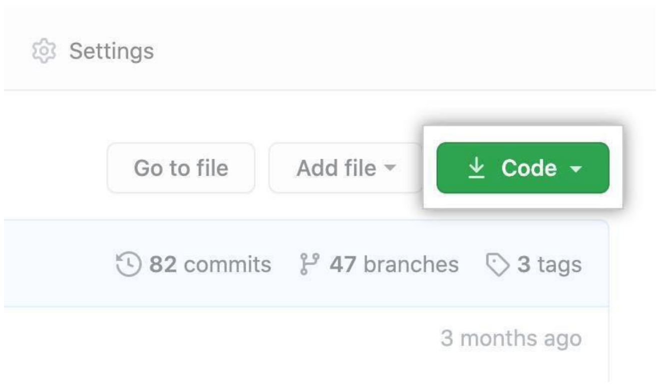
Create repository

c) Cloning repository:

After clicking on the create repository button you will reach on this page

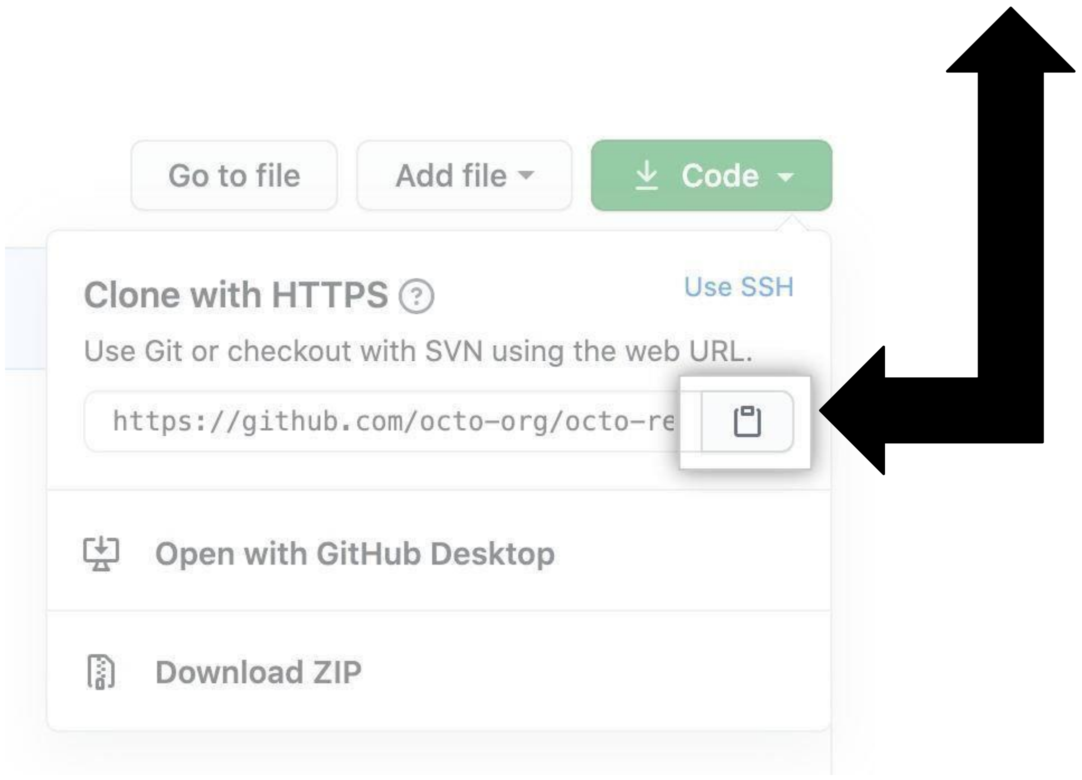
On GitHub, navigate to the main page of the repository.

Above the list of files, click "Code" button



To clone the repository using HTTPS, under "Clone with HTTPS", click . To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click Use SSH, then click . To clone a repository using GitHub CLI, click Use GitHub CLI, then click

.


Go to file
Add file
Code
Clone with HTTPS
Use SSH
Use Git or checkout with SVN using the web URL.
https://github.com/octo-org/octo-re
Open with GitHub Desktop
Download ZIP

Name: MEHUL VINOD GOHIL

Roll No: 14FYIT

Practical 3: UNDERSTANDING ON FREE AND OPEN-SOURCE SOFTWARE

a) ***<u>Describe open Source software with Example</u>***.

Ans:-

Open source software is software with source code that anyone can inspect, modify, and enhance.
"Source code" is the part of software that most computer users don't ever see; it's the code computer programmers can manipulate to change how a piece of software—a "program" or "application"—works. Programmers who have access to a computer program's source code can improve that program by adding features to it or fixing parts that don't always work correctly. Example:-

Firefox—a Web browser that competes with Internet Explorer

OpenOffice—a competitor to Microsoft Office Gimp—a

graphic tool with features found in Photoshop

Alfresco—collaboration software that competes with Microsoft Sharepoint and EMC's Documentum

Marketcetera—an enterprise trading platform for hedge fund managers that competes with FlexTrade and Portware

Zimbra—open source e-mail software that competes with Outlook server

MySQL, Ingres, and EnterpriseDB—open source database software packages that each go head-to-head with commercial products from Oracle, Microsoft, Sybase, and IBM

SugarCRM—customer relationship management software that competes with Salesforce.com and Siebel

Asterix—an open source implementation for running a PBX corporate telephony system that competes with offerings from Nortel and Cisco, among others

Free BSD and Sun's OpenSolaris—open source versions of the Unix operating system

b) Describe Free software with example.

Ans:-

"Free software" means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, "free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer". We sometimes call it "libre software," borrowing the French or Spanish word for "free" as in freedom, to show we do not mean the software is gratis.

We campaign for these freedoms because everyone deserves them. With these freedoms, the users (both individually and collectively) control the program and what it does for them. When users don't control the program, we call it a "nonfree" or "proprietary" program. The nonfree program controls the users, and the developer controls the program; Example:-

free software covers just about every field of computer applications. Because of their high quality and openness, several free software programs have become leaders in their field or comprise the core of of an entire industry. Here is a short list of free software headline acts :

The Linux kernel, of course! The Linux kernel is protected by the GPL, and is used daily by millions of people throughout the world. As the kernel, it is one of the most important components of the GNU system ;

Apache, the most widely used web server in the world. More than 56% of the web servers on this planet use Apache; far more than its fierce competitors, Microsoft and Netscape ;

The Gimp is a powerful bitmap mode digital creation program. In spite of being relatively new, The Gimp has rapidly become serious competition for Photoshop ;

PostgreSQL is an object-relational database. It is currently the most sophisticated free software database available.

c) Difference between free and open source software.

Ans:-

Difference between Free Software and Open Source Software Free Software:

"Free software" means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software.

The term "free software" is sometimes misunderstood—it has nothing to do with price. It is about freedom.

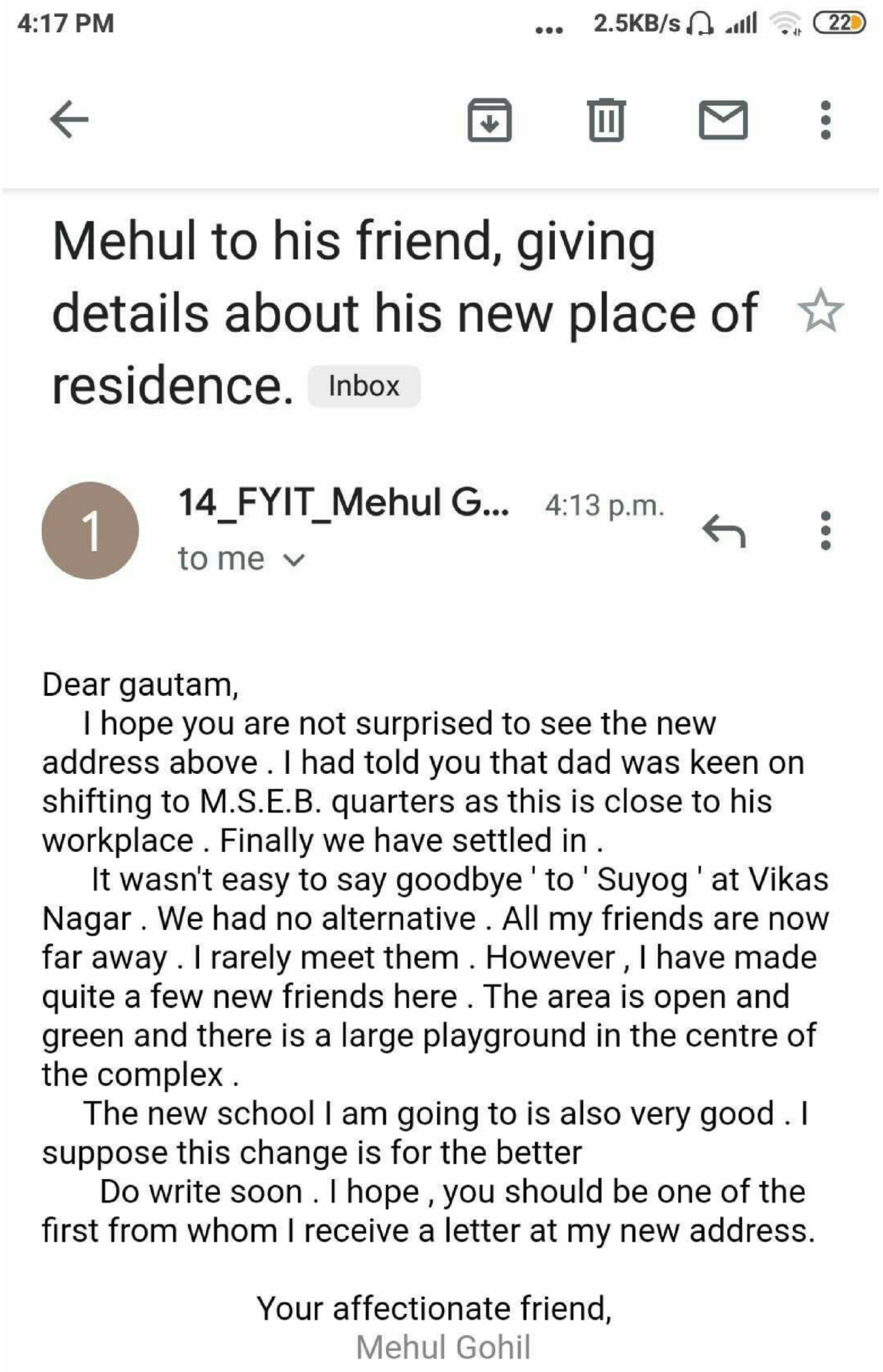
Open Source Software :

Open Source Software is something which you can modify as per your needs, share with others without any licensing violation burden. When we say Open Source, source code of software is available publicly with Open Source licenses like GNU (GPL) which allows you to edit source code and distribute it. Read these licenses and you will realize that these licenses are created to help us.

Coined by the development environments around software produced by open collaboration of software developers on the internet.
Later specified by the Open Source Initiative (OSI).
It does not explicitly state ethical values, besides those directly associated to software development.
Difference between Free Software and Open Source Software:

Free software ***= Free software usually refers open source under GNU GPL license. At least the original term by Richard Stallman did. He meant free as in freedom . Because the word free in English mean without cost the team open source was created. To hint toward the collaborative development effort ,not the price to acquire something.***

Open source = ***your source code is accessible to anyone to read and modify and redistribute depending on license condition.***


4:17 PM ... 2.5KB/s 22

Mehul to his friend, giving details about his new place of residence. Inbox

14_FYIT_Mehul G... 4:13 p.m.
to me

Dear gautam,
I hope you are not surprised to see the new address above . I had told you that dad was keen on shifting to M.S.E.B. quarters as this is close to his workplace . Finally we have settled in .
It wasn't easy to say goodbye ' to ' Suyog ' at Vikas Nagar . We had no alternative . All my friends are now far away . I rarely meet them . However , I have made quite a few new friends here . The area is open and green and there is a large playground in the centre of the complex .
The new school I am going to is also very good . I suppose this change is for the better
Do write soon . I hope , you should be one of the first from whom I receive a letter at my new address.

Your affectionate friend,
Mehul Gohil

Name: Mehul Vinod Gohil

Roll No: 14fyit

PRACTICAL:-5

Using practical examples, describe green computing. List and explain the steps that you take to contribute to green computing

Ans:-

INTRODUCTION.

Green computing is the study and practice of minimizing the environmental impact of computer system and related resources effectively and eco-friendly. It is an emerging concept towards reducing the hazardous material and save our environment from the harmful impacts of the computer, CPU, servers and other electronic devices.

Green computing is an application of environmental science which offers economically possible solutions that conserve natural environment and its resources. Green computing is designing, manufacturing, using and disposing of computers and its resources efficiently with minimal or no impact on environment. The goals of Green computing is to manage the power and energy efficiency, choice of eco friendly hardware and software, and recycling the material to increase the product's life. Go for Green computer reduced your electricity bill and give a full rest to your mind. Now in these days, we use the star management strategies and technologies that reduce energy consumption waste.

1. Energy Savings Apart from computers, there are different kinds of electrical appliances that consume significant amount of energy. This creates a demand for the energy production. Therefore, it is necessary to

decrease this energy crisis as much as possible for making a more eco-friendly environment.

Green computing makes sure that very less amount of energy is consumed by the IT processes. Thus, this can save plenty amount of energy overtime.

1. Cost Savings Green computing is highly cost effective that helps people save money. Since lots of energies are saved when using a green computing solution, it also substantially leads to financial gains. Even though green computing is with high upfront costs, still it is cost effective in the long run.
2. Recycling Process Green computing encourages recycling process by reusing and recycling electronic wastes. Most parts of the computer are constructed using eco-friendly materials instead of plastic so that it can have less environmental impacts. This makes all the electronic wastes to get separated efficiently. Hence by implementing green computing strategies, companies overall can improve their recycling process.
3. Brand Strengthen Some customers are so well concerned about the environment that they are solely preferring to go with companies that support green computing. Green computing is capable of creating public images so that they can strengthen their brand and market position all around the world.
4. Less Pollution Through conventional computing, lots of pollution issues take place in the environment. For an example, if not properly recycled all the electronic wastes from the computer may end up circulating on land. Thus, leading to soil as well as water pollution. By using green computing, the users can minimize the impact created by pollution at least to some extent.

5. GHG Emission During the production of IT hardware, tremendous amount of green house gases are released to the atmosphere. Especially, since harmful gases such as carbon dioxide are emitted, it could lead to global warming. Hence, for lowering the amount of green house gases emitted, the production of hardware components must be reduced as well. This is how green computing works effectively.
6. Chemical Exposure In most of the electronic devices, harmful chemicals such as mercury is used. If a human happens to get contacted with those substances, he/she will probably suffer from health risks. Some of the known health risks are triggering of immune responses, nerve damage or even cancer. The companies which practice green computing potentially avoid the use of non-toxic substances during the production of computer hardware.

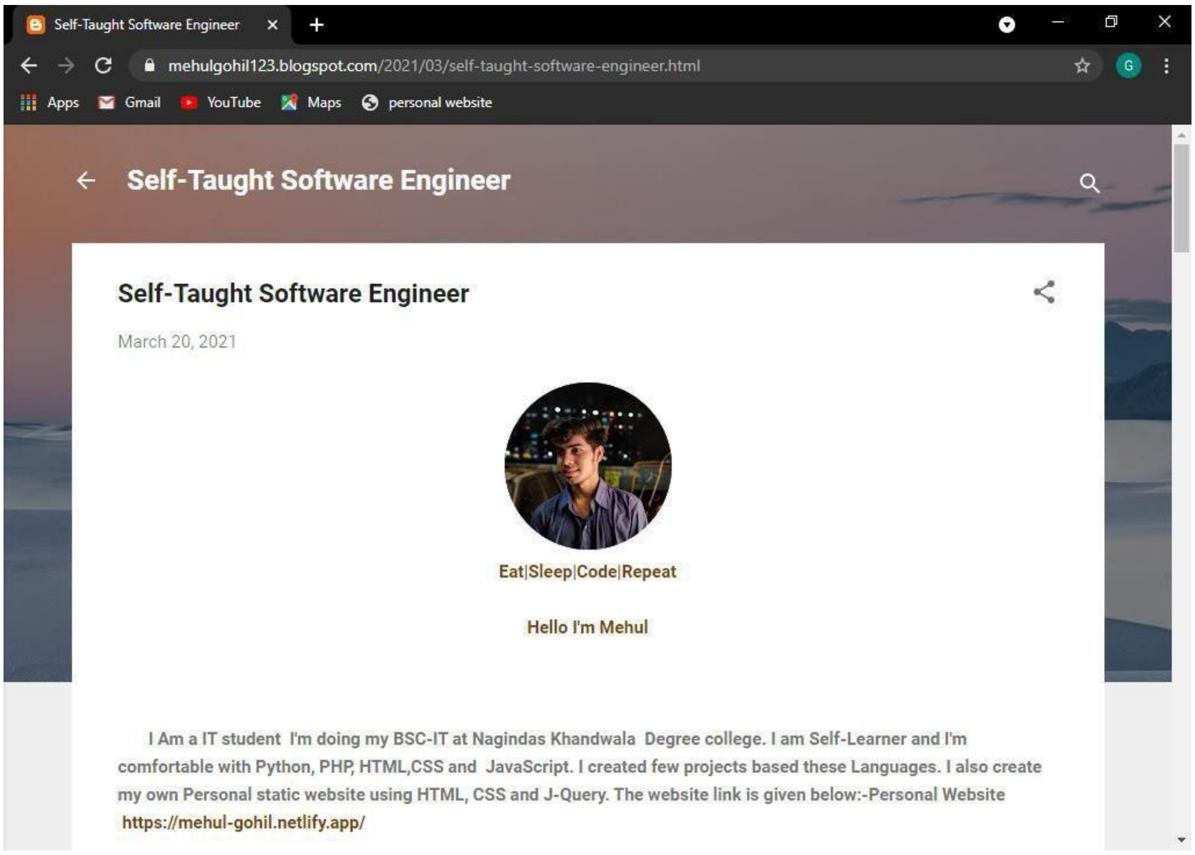
Name:- Mehul Vinod

Gohil Roll no :- 14fyit

PRACTICAL:-6

Blog Presentation

Topic:-Self-Taught Software Engineer


Self-Taught Software Engineer
mehulgohil123.blogspot.com/2021/03/self-taught-software-engineer.html
Apps
Gmail
YouTube
Maps
personal website

Self-Taught Software Engineer

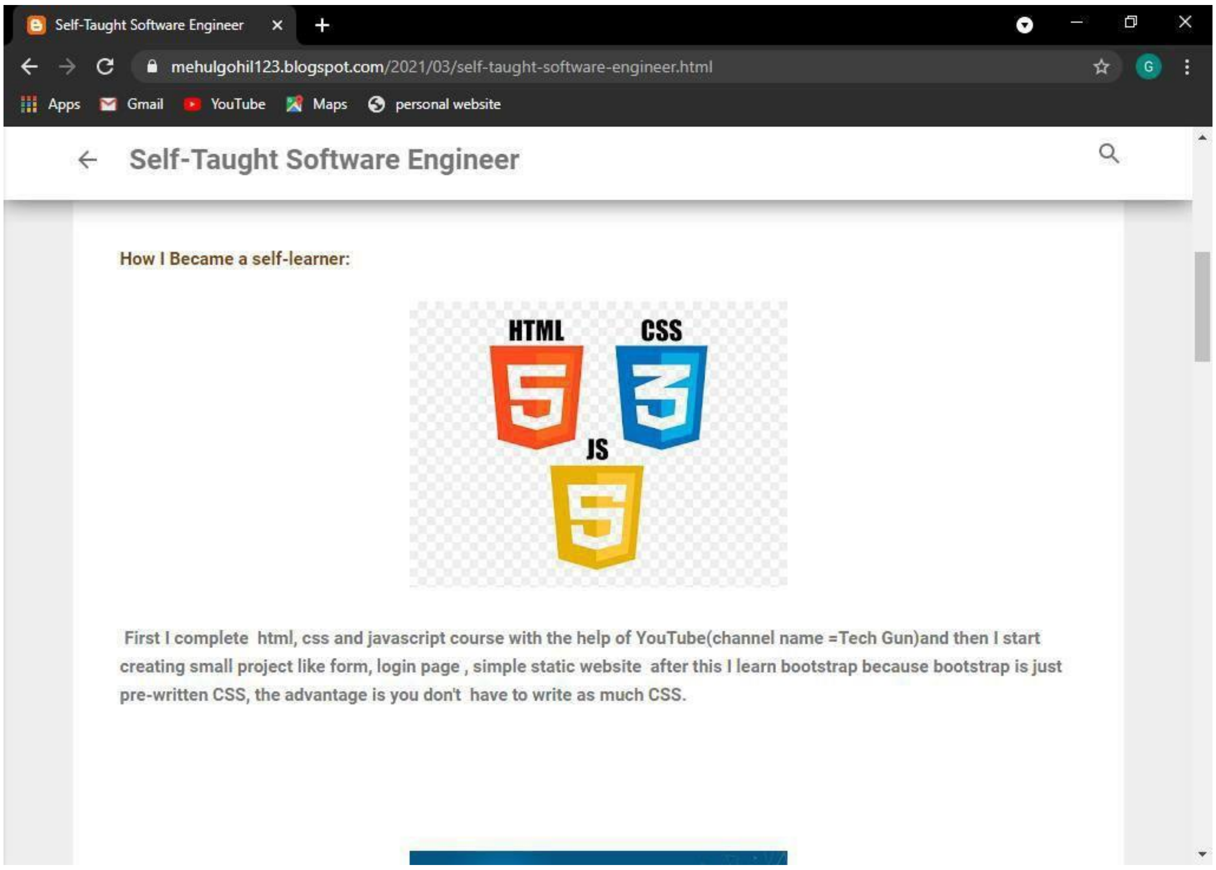
Self-Taught Software Engineer

March 20, 2021

Eat|Sleep|Code|Repeat

Hello I'm Mehul

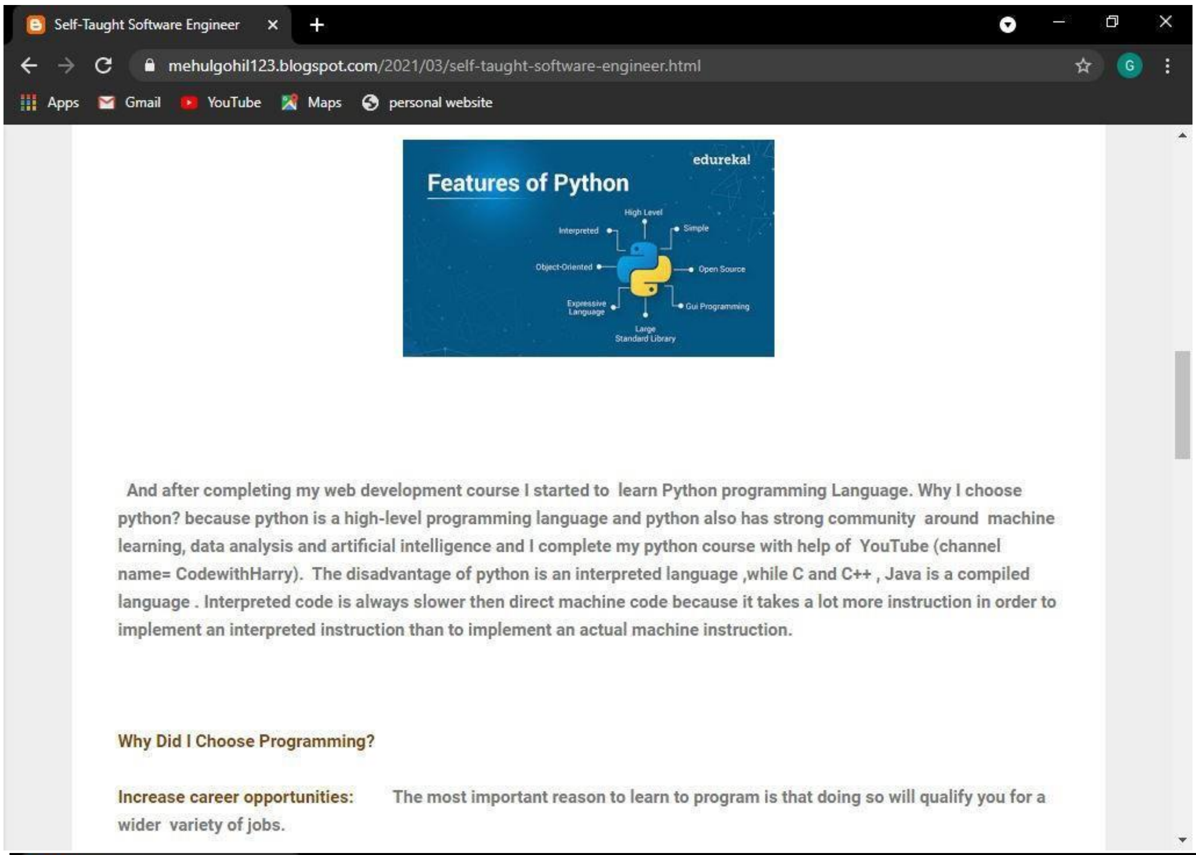
I Am a IT student I'm doing my BSC-IT at Nagindas Khandwala Degree college. I am Self-Learner and I'm comfortable with Python, PHP, HTML,CSS and JavaScript. I created few projects based these Languages. I also create my own Personal static website using HTML, CSS and J-Query. The website link is given below:-Personal Website https://mehul-gohil.netlify.app/


Self-Taught Software Engineer
mehulgohil123.blogspot.com/2021/03/self-taught-software-engineer.html
Apps
Gmail
YouTube
Maps
personal website
HTML
CSS
JS

Self-Taught Software Engineer

How I Became a self-learner:

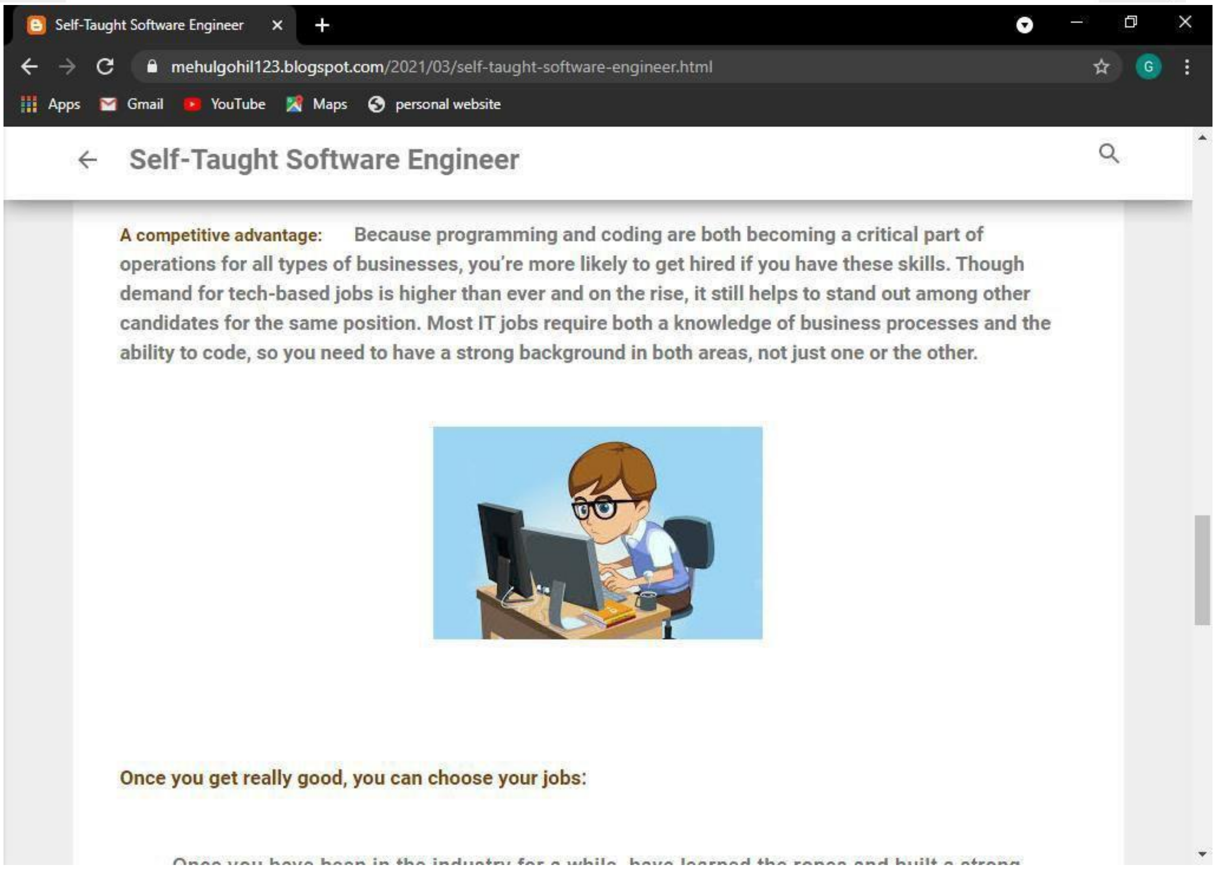
First I complete html, css and javascript course with the help of YouTube(channel name =Tech Gun)and then I start creating small project like form, login page , simple static website after this I learn bootstrap because bootstrap is just pre-written CSS, the advantage is you don't have to write as much CSS.



And after completing my web development course I started to learn Python programming Language. Why I choose python? because python is a high-level programming language and python also has strong community around machine learning, data analysis and artificial intelligence and I complete my python course with help of YouTube (channel name= CodewithHarry). The disadvantage of python is an interpreted language ,while C and C++ , Java is a compiled language . Interpreted code is always slower then direct machine code because it takes a lot more instruction in order to implement an interpreted instruction than to implement an actual machine instruction.

Why Did I Choose Programming?

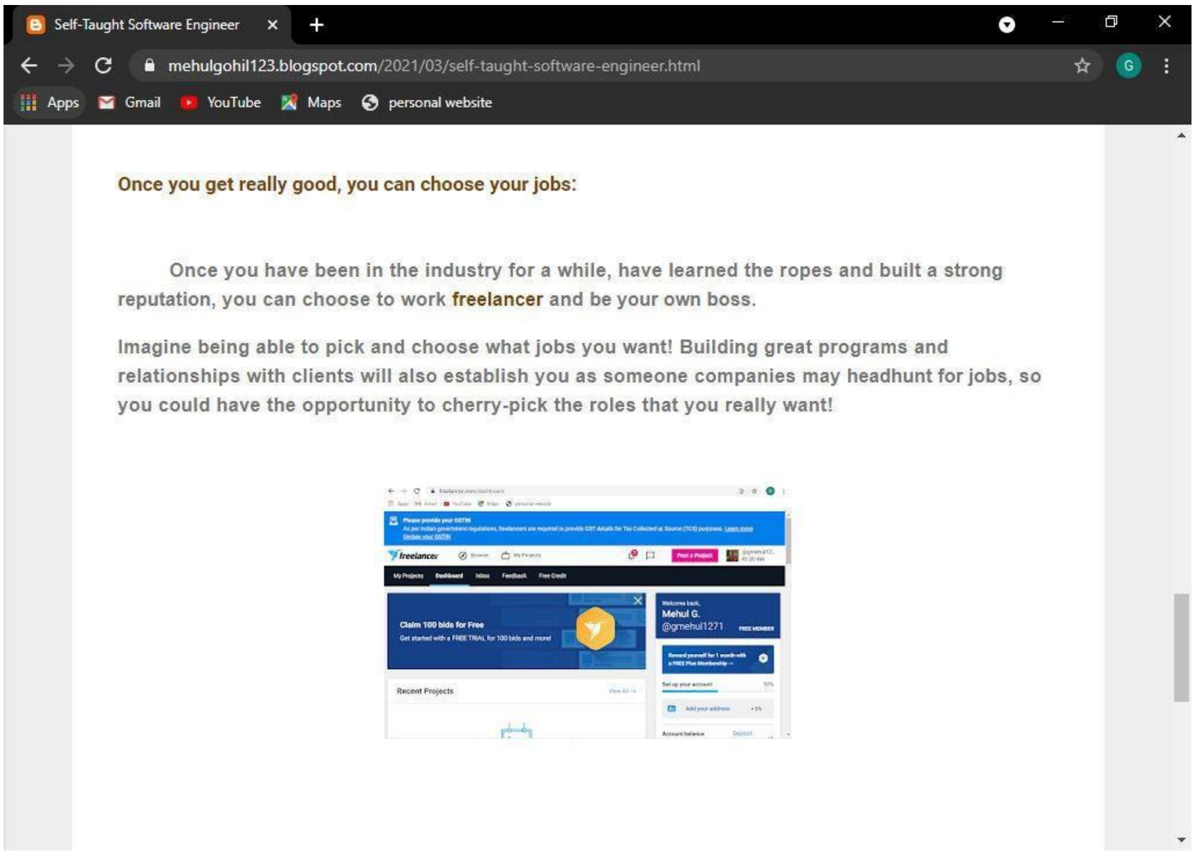
Increase career opportunities: The most important reason to learn to program is that doing so will qualify you for a wider variety of jobs.


Self-Taught Software Engineer
mehulgohil123.blogspot.com/2021/03/self-taught-software-engineer.html
Apps
Gmail
YouTube
Maps
personal website
Self-Taught Software Engineer

A competitive advantage: Because programming and coding are both becoming a critical part of operations for all types of businesses, you're more likely to get hired if you have these skills. Though demand for tech-based jobs is higher than ever and on the rise, it still helps to stand out among other candidates for the same position. Most IT jobs require both a knowledge of business processes and the ability to code, so you need to have a strong background in both areas, not just one or the other.

Once you get really good, you can choose your jobs:

Once you have been in the industry for a while, have learned the ropes and built a strong


Self-Taught Software Engineer
mehulgohil123.blogspot.com/2021/03/self-taught-software-engineer.html
Apps
Gmail
YouTube
Maps
personal website
freelancer
Claim 100 bids for Free
Get started with a FREE TRIAL for 100 bids and more!
Welcome back,
Mehul G.
@gmehul1271
Recent Projects

Once you get really good, you can choose your jobs:

Once you have been in the industry for a while, have learned the ropes and built a strong reputation, you can choose to work **freelancer** and be your own boss.

Imagine being able to pick and choose what jobs you want! Building great programs and relationships with clients will also establish you as someone companies may headhunt for jobs, so you could have the opportunity to cherry-pick the roles that you really want!

PRACTICAL:-7

Implementing coding practice in python using PEP8.

Ans:-

It gets difficult to understand a messed up handwriting, similarly an unreadable and unstructured code is not accepted by all. However, you can benefit as a programmer only when you can express better with your code. This is where PEP comes to the rescue.

Python Enhancement Proposal or PEP is a design document which provides information to the Python community and also describes new features and document aspects, such as style and design for Python.

Python is a multi-paradigm programming language which is easy to learn and has gained popularity in the fields of Data Science and Web Development over a few years and **PEP 8** is called the style code of Python. It was written by Guido van Rossum, Barry Warsaw, and Nick Coghlan in the year 2001. It focuses on enhancing Python’s code readability and consistency. Join the certification course on Python Programming and gain skills and knowledge about various features of Python along with tips and tricks.

A Foolish Consistency is the Hobgoblin of Little Minds

‘A great person does not have to think consistently from one day to the next’ — this is what the statement means.

Consistency is what matters. It is considered as the style guide. You should maintain consistency within a project and mostly within a function or module.

However, there will be situations where you need to make use of your own judgement, where consistency isn’t considered an option. You must know when you need to be inconsistent like for example when applying the guideline would make the code less readable or when the code needs to comply with the earlier versions of Python which the style guide doesn’t recommend.

In simple terms, you cannot break the backward compatibility to follow with PEP.

The Zen of Python

It is a collection of 19 ‘guiding principles’ which was originally written by Tim Peters in the year 1999. It guides the design of the Python Programming Language.

Python was developed with some goals in mind. You can see those when you type the following code and run it:

```
>>> import this

The Zen of Python, by Tim Peters


Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.
```

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

The Need for PEP 8

Readability is the key to good code. Writing good code is like an art form which acts as a subjective topic for different developers.

Readability is important in the sense that once you write a code, you need to remember what the code does and why you have written it. You might never write that code again, but you'll have to read that piece of code again and again while working in a project.

PEP 8 adds a logical meaning to your code by making sure your variables are named well, sufficient whitespaces are there or not and also by commenting well. If you're a beginner to the language, PEP 8 would make your coding experience more pleasant.

Following PEP 8 would also make your task easier if you’re working as a professional developer. People who are unknown to you and have never seen how you style your code will be able to easily read and understand your code only if you follow and recognize a particular guideline where readability is your de facto.

And as Guido van Rossum said— “Code is read much more than it is often written”.

The Code Layout

Your code layout has a huge impact on the readability of your code.

Indentation

The indentation level of line is computed by the leading spaces and tabs at the beginning of a line of logic. It influences the grouping of statements.

The rules of PEP 8 says to use 4 spaces per indentation level and also spaces should be preferred over tabs.

An example of code to show indentation:

```
x = 5

if x < 10:

  print('x is less than 10')
```

Tabs or Spaces?

Here the **print** statement is indented which informs Python to execute the statement only if the **if** statement is true. Indentation also helps Python to know what code it will execute during function calls and also when using classes.

PEP 8 recommends using 4 spaces to show indentation and tabs should only be used to maintain consistency in the code.

Python 3 forbids the mixing of spaces and tabs for indentation. You can either use tabs or spaces and you should maintain consistency while using Python 3. The errors are automatically displayed:

```
python hello.py
```

```
  File "hello.py", line 3 print(i,
     j)
              ^
```

TabError: inconsistent use of tabs and spaces in indentation

However, if you're working in Python 2, you can check the consistency by using a **-t** flag in your code which will display the warnings of inconsistencies with the use of spaces and tabs. You can also use the **-tt** flag which will show the errors instead of warnings and also the location of inconsistencies in your code.

Maximum Line Length and Line Breaking

The Python Library is conservative and **79 characters** are the maximum required line limit as suggested by PEP 8. This helps to avoid line wrapping.

Since maintaining the limit to 79 characters isn't always possible, so PEP 8 allows wrapping lines using Python's implied line continuation with parentheses, brackets, and braces:

```
def function(argument_1, argument_2, argument_3,
          argument_4):
   return argument_1
```

Or by using backslashes to break lines:

```
with open('/path/to/some/file/you/want/to/read') as example_1, \
   open('/path/to/some/file/being/written', 'w') as example_2:
   file_2.write(file_1.read())
```

When it comes to binary operators, PEP 8 encourages to break lines before the binary operators. This accounts for more readable code.

Let us understand this by comparing two examples:

```
# Example 1 # Do total = ( variable_1 +
variable_2 - variable_3 )
```

```
# Example 2

# Don't

total = ( variable_1 + variable_2 - variable_3 )
```

In the first example, it is easily understood which variable is added or subtracted, since the operator is just next to the variable to which it is operated. However, in the second example, it is a little difficult to understand which variable is added or subtracted.

Indentation with Line Breaks

Indentation allows a user to differentiate between multiple lines of code and a single line of code that spans multiple lines. It enhances readability too.

The first style of indentation is to adjust the indented block with the delimiter:

```
def function(argument_one, argument_two,

        argument_three,         argument_four):

        return argument_one
```

You can also improve readability by adding comments:

```
x = 10

if (x > 5 and x

    < 20):

    # If Both conditions are satisfied print(x)
```

Or by adding extra indentation:

```
x = 10

if (x > 5 and

      x < 20):

    print(x)
```

Another type of indentation is the **hanging indentation** by which you can symbolize a continuation of a line of code visually:

```
foo = long_function_name(
    variable_one, variable_two,
    variable_three, variable_four)
```

You can choose any of the methods of indentation, following line breaks, in situations where the 79 character line limit forces you to add line breaks in your code, which will ultimately improve the readability.

Closing Braces in Line Continuations

Closing the braces after line breaks can be easily forgotten, so it is important to put it somewhere where it makes good sense or it can be confusing for a reader.

One way provided by PEP 8 is to put the closing braces with the first white-space character of the last line:

```
my_list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
```

Or lining up under the first character of line that initiates the multi-line construct:

```
my_list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
```

Remember, you can use any of the two options keeping in mind the consistency of the code.

Blank Lines

Blank lines are also called vertical whitespaces. It is a logical line consisting of spaces, tabs, formfeeds or comments that are basically ignored.

Using blank lines in top-level-functions and classes:

```
class my_first_class:

pass class

my_second_class: pass

def

top_level_function():

    return None
```

Adding two blank lines between the top-level-functions and classes will have a clear separation and will add more sensibility to the code.

Using blank lines in defining methods inside classes:

```
class my_class:

    def method_1(self):

        return None


    def method_2(self):

        return None
```

Here, a single vertical space is enough for a readable code.

You can also use blank spaces inside multi-step functions. It helps the reader to gather the logic of your function and understand it efficiently. A single blank line will work in such case.
An example to illustrate such:

```
def calculate_average(number_list):

    sum_list = 0 for number in

    number_list:

        sum_list = sum_list + number
    average = 0

    average = sum_list / len(number_list)


   return average
```

Above is a function to calculate the **average**. There is a blank line between each step and also before the **return** statement.

The use of blank lines can greatly improve the readability of your code and it also allows the reader to understand the separation of the sections of code and the relation between them.

Naming Conventions

Choosing names which are sensible and can be easily understandable, while coding in Python, is very crucial. This will save time and energy of a reader. Inappropriate names might lead to difficulties when debugging.

Naming Styles

Naming variables, functions, classes or methods must be done very carefully. Here's a list of the type, naming conventions and examples on how to use them:

Type	**Naming Conventions**	**Examples**
Variable	Using short names with CapWords.	T, AnyString, My_First_Variable
Function	Using a lowercase word or words with underscores to improve readability.	function, my_first_function
Class	Using CapWords and do not use underscores between words.	Student, MyFirstClass

Method	Using lowercase words separated by underscores.	Student_method, method
Constants	Using all capital letters with underscores separating words	TOTAL, MY_CONSTANT, MAX_FLOW
Exceptions	Using CapWords without underscores.	IndexError, NameError
Module	Using short lower-case letters using underscores.	module.py, my_first_module.py

Type	**Naming Conventions**	**Examples**
Package	Using short lowercase words and underscores are discouraged.	package, my_first_package

Choosing names

To have readability in your code, choose names which are descriptive and give a clearer sense of what the object represents. A more real-life approach to naming is necessary for a reader to understand the code.

Consider a situation where you want to store the name of a person as a string:

```
>>> name = 'John William'

>>> first_name, last_name = name.split()
>>> print(first_name, last_name, sep='/ ')
John/ William
```

Here, you can see, we have chosen variable names like **first_name** and **last_name** which are clearer to understand and can be easily remembered. We could have used short names like **x**, **y** or **z** but it is not recommended by PEP 8 since it is difficult to keep track of such short names.

Consider another situation where you want to double a single argument. We can choose an abbreviation like **db** for the function name:

```
# Don't def

db(x):

    return x * 2
```

However, abbreviations might be difficult in situations where you want to return back to the same code after a couple of days and still be able to read and understand. In such cases, it’s better to use a concise name like **double_a_variable**:

```
# Do

def double_a_value(x):

    return x * 2
```

Ultimately, what matters is the readability of your code.

Comments

A comment is a piece of code written in simple English which improves the readability of code without changing the outcome of a program. You can understand the aim of the code much faster just by reading the comments instead of the actual code. It is important in analyzing codes, debugging or making a change in logic.

Block Comments

Block comments are used while importing data from files or changing a database entry where multiples lines of code are written to focus on a single action. They help in interpreting the aim and functionality of a given block of code.

They start with a **hash(#)** and a single space and always indent to the same level as the code:

```
for i in range(0, 10):

    # Loop iterates 10 times and then prints i

    # Newline character

    print(i, '\n')
```

You can also use multiple paragraphs in a block comment while working on a more technical program.

Block comments are the most suitable type of comments and you can use it anywhere you like.

Inline Comments

Inline comments are the comments which are placed on the same line as the statement. They are helpful in explaining why a certain line of code is essential.

Example of inline comments:

```
x = 10 # An inline comment

y = 'JK Rowling' # Author Name
```

Inline comments are more specific in nature and can easily be used which might lead to clutter. So, PEP 8 basically recommends using block comments for general-purpose coding.

Document Strings

Document strings or docstrings start at the first line of any function, class, file, module or method. These type of comments are enclosed between single quotations (''') or double quotations (""").

An example of docstring:

```
def quadratic_formula(x, y, z, t):

    """Using the quadratic formula""" t_1

    = (- b+(b**2-4*a*c)**(1/2)) / (2*a)

    t_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)
    return t_1, t_2
```

Whitespaces in Expressions and Statements

In computing, whitespace is any character or sequence of characters which are used for spacing and have an ‘empty’ representation. It is helpful in improving the readability of expressions and statements if used properly.

Whitespace around Binary Operators

When you’re using assignment operators (**=, +=, -=,and so forth**) or comparisons (**==, !=, >, <. >=, <=**) or booleans (**and, not, or**), it is suggested to use a single whitespace on the either side.

Example of adding whitespace when there is more than one operator in a statement:

```
# Don't

b = a ** 2 + 10

c = (a + b) * (a - b)



# Do

 b      = a**2 + 10

 c      = (a+b) * (a-b)
```

In such mathematical computations, you should add whitespace around the operators with the least priority since adding spaces around each operator might be confusing for a reader.

Example of adding whitespaces in an **if** statement with many conditions:

```
# Don't

if a < 10 and a % 5 == 0:

    print('a is smaller than 10 and is divisible by 5!')
# Do

if a<10 and a%5==0:
```

    print('a is smaller than 10 and is divisible by 5!')

Here, the **and** operator has the least priority, so whitespaces have been added around it.

Colons act as binary operators in slices:

ham[3:4] ham[x+1 :

x+2] ham[3:4:5]

ham[x+1 : x+2 : x+3]

ham[x+1 : x+2 :]

Since colons act as a binary operator, whitespaces are added on either side of the operator with the lowest priority. Colons must have the same amount of spacing in case of an extended slice. An exception is when the slice parameter is omitted, space is also omitted.

Avoiding Whitespaces

Trailing whitespaces are whitespaces placed at the end of a line. These are the most important to avoid.

You should avoid whitespaces in the following cases— Inside

a parentheses, brackets, or braces:

```
# Do

list = [1, 2, 3]


# Don't

list = [ 1, 2, 3, ]
```

Before a comma, a semicolon, or a colon:

```
x = 2

y = 3
```

```
# Do
print(x, y)


# Don't
print(x , y)
```

Before open parenthesis that initiates the argument list of a function call:

```
def multiply_by_2(a):
    return a * 2


# Do multiply_by_2(3)


# Don't multiply_by_2
(3)
```

Before an open bracket that begins an index or a slice:

```
# Do ham[5]
# Don't ham
[5]
```

Between a trailing comma and a closing parenthesis:

```
# Do
spam = (1,)


# Don't spam
= (1, )
```

To adjust assignment operators:

```
# Do variable_1

= 5

variable_2 = 6

my_long_var = 7


# Don't variable_1

      = 5

variable_2     = 6

my_long_var = 7
```

Programming Recommendations

PEP 8 guidelines suggest different ways to maintain consistency among multiple implementations of Python like **PyPy**, **Jython** or **Cython.**

An example of comparing **boolean** values:

```
# Don't bool_value

= 5 > 4

if bool_value == True:

return '4 is smaller than 5'


# Do

if bool_value:

return '4 is smaller than 5'
```

Since **bool** can only accept values **True** or **False**, it is useless to use the equivalence operator **==** in these type of **if** executions. PEP 8 recommends the second example which will require lesser and simpler coding.

An example to check whether a list is empty or not:

```
#  Don't  list_value  =

[]            if            not

len(list_value):

    print('LIST IS EMPTY')



# Do

list_value = [] if

not list_value:

    print('LIST IS EMPTY')
```

Any **empty list** or string in Python is falsy. So you can write a code to check an empty string without checking the length of the list. The second example is more simple, so PEP encourages to write an **if** statement in this way.

The expression **is not** and **not ... is** are identical in functionality. But the former is more preferable due to its nature of readability:

```
# Do

if x is not None:

    return 'x has a value'



# Don't

if not x is None:

    return 'x has a value'
```

String slicing is a type of indexing syntax that extracts substrings from a string. Whenever you want to check if a string is prefixed or suffixed, PEP recommends using **.startswith()** and **.endswith()** instead of list slicing. This is because they are cleaner and have lesser chances of error:

```
# Do
```

```
if foo.startswith('cat'):



# Don't

if foo[:3] == 'cat':
```

An example using **.endswith()**:

```
# Don't

if file_jpg[-3:] == 'jpg':
    print('It is a JPEG image file')



# Do

if file_jpg.endswith('jpg'):

    print('It is a JPEG image file')
```

Though there exists multiple ways to execute a particular action, the main agenda of the guidelines laid by PEP 8 is simplicity and readability.

When to Ignore PEP 8

You should never ignore PEP 8. If the guidelines related to PEP8 are followed, you can be confident of writing readable and professional codes. This will also make the lives of your colleagues and other members working on the same project much easier.

There are some exclusive instances when you may ignore a particular guideline:

- After following the guidelines, the code becomes less readable, even for a programmer who is comfortable with reading codes that follow PEP 8.
- If the surrounding code is inconsistent with PEP.
- Compatible of code with older version of Python is the priority.

Checking PEP 8 Compliant Code

You can check whether your code actually complies with the rules and regulations of PEP 8 or not. **Linters** and **Autoformatters** are two classes of tools used to implement and check PEP 8 compliance.

Linters

It is a program that analyzes your code and detects program errors, syntax errors, bugs and structural problems. They also provide suggestions to correct the errors.

Some of the best linters used for Python code:

- pycodestyle is a tool to verify the PEP 8 style conventions in your Python code.

You can run the following from the command line to install **pycodestyle** using **pip**: pip

install pycodestyle

To display the errors of a program, run **pycodestyle** in this manner:

```
pycodestyle my_code.py

my_code.py:1:11: E231 missing whitespace after '{'

my_code.py:3:19: E231 missing whitespace after ')' my_code.py:4:31:

E302 expected 2 blank lines, found 1
```

- flake8 is a Python wrapper that verifies **PEP 8**, **pyflakes**, and circular complexity.

Type the command to install **flake8** using **pip**:

```
pip install flake8
```

Run **flake8** from the terminal using the command:

```
flake8 calc.py
calc.py:24:3: E111 indentation is not a multiple of two

calc.py:25:3: E111 indentation is not a multiple of two calc.py:45:9:

E225 missing whitespace around operator
```

You can also use some other good linters like pylint, pyflakes, pychecker and mypy.

Autoformatters

An autoformatter is a tool which will format your code to adapt with PEP 8 automatically.One of the most commonly used autoformatter is black.

To install **black** using **pip**, type:

```
pip install black
```

Remember, you need to have Python 3.6 or above to install **black**.

An example of code that doesn’t follow PEP 8:

```
def add(a, b): return a+b



def multiply(a, b):

     return \

       a * b
```

Now run black following the filename from the terminal:

```
black my_code.py

reformatted my_code.py All

done!
```

The reformatted code will look like:

```
def add(a, b): return

    a + b



def multiply(a, b):

    return a * b
```

Some other autoformatters include autopep8 and yapf. Their work is similar to **black**.

PRACTICAL:-8

UNIFIED COMMUNICATION

GROUP NO :4

DevangiParmar -133

Kiran Kidecha- 130

Komal Menaria- 42

Karishma Murkar- 49

Mehul Gohil – 14

Dharmit Shah - 75

VanshNagda- 127

DharmikKothari -36

Dev Sanghavi - 73

UjjvalJain - 25

Today's Agenda

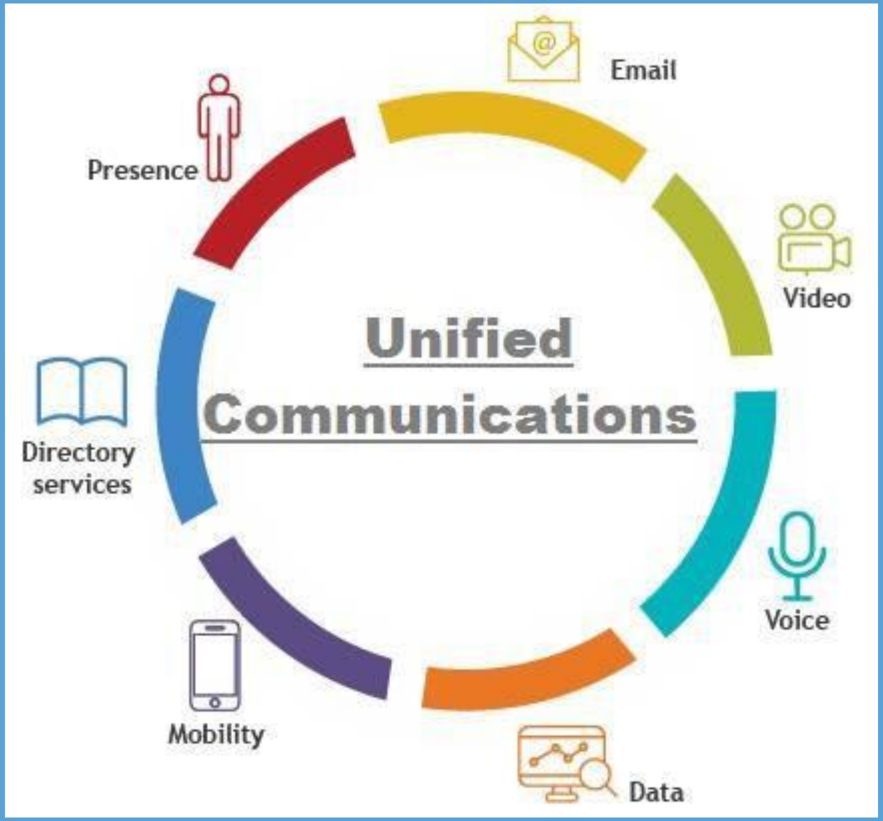
- UC Overview & Definition
- Technologies
- Benefits
- Disadvantages
- ROI(Return on Investment)
- Challenges
- Security Issues
- UC Platforms
- UC Military Drivers
- Features of UC

What is Unified Communications?

Unified communications(UC) is a business and marketing concept describing the integration of enterprisecommunicationservices such as instantmessaging(chat), presence information, voice (including IP telephony), mobility features (including extension mobility and single number reach), audio, web & video conferencing, and many more.

Overview & Definition

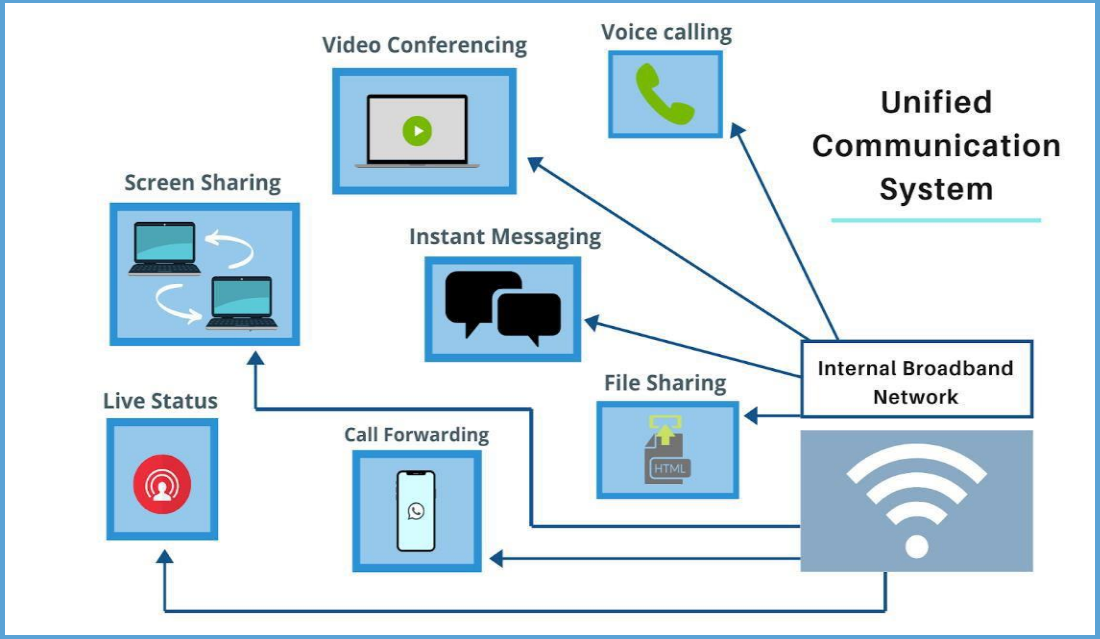
- Unified Communications (UC) intelligently integrates many communications applications
- UC is an evolving set of technologies
- Many businesses are turning to UC to enhance business processes
- Service providers are increasingly providing Unified Communication
- UC can drastically change the bottom line of business



TECHNOLOGY

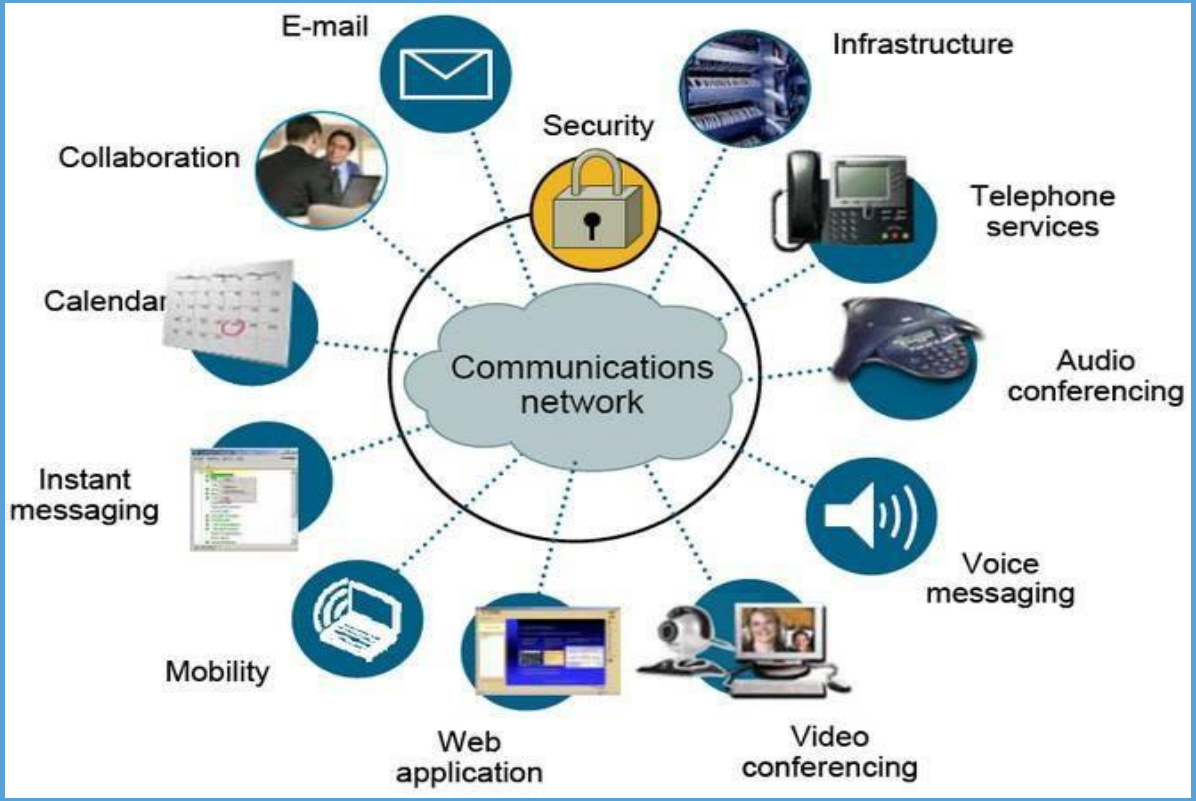
UC is integration of real time communication such as:

- Presence Information
- Telephony (especially VoIP)
- Data Sharing
- Call Control



Benefits

- Collaborations
- Communications
- Access
- Business process integration
- Presence



DISADVANTAGES

- Learning curve
- Environmental factors
- Poorly worded messages
- interoperability issues

ROI (Return on Investment)

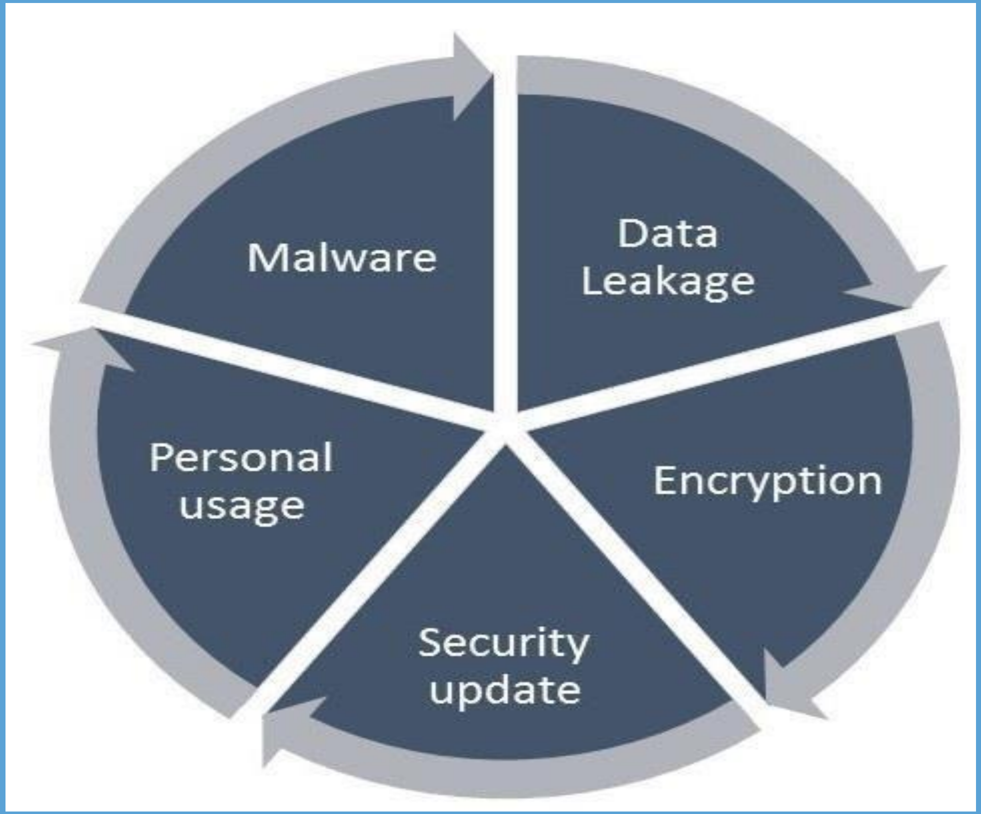
- UC investment improve productivity and availability
- Typical UC rollout
- Many UC products are complex to deploy
- A UC system can cut traditional telephony costs
- Intangible UC benefits are harder to define

Challenges

- User adoption and adaption
- Technical implementation
- Optimisingthe cost of unified communication
- Challenges in IT staff.

Security Issues

- Theft of service
- Denial of services
- Hacking tools
- Fraud prevention
- Security controls

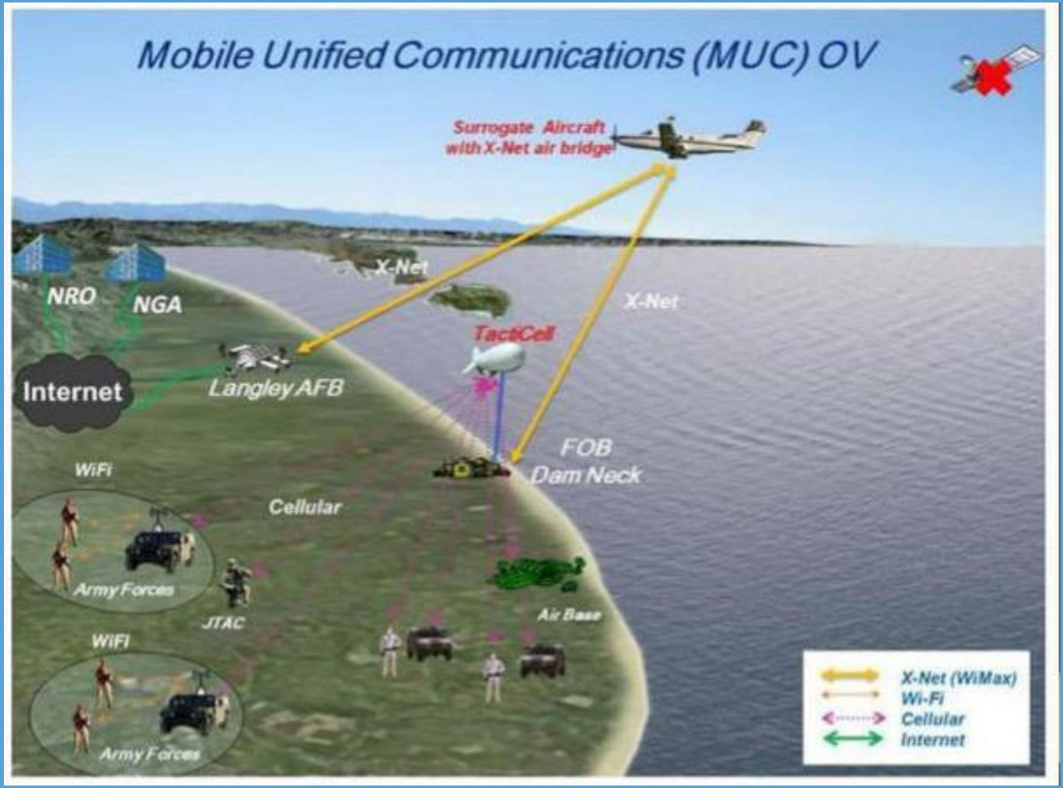


UC PLATFORMS

- Ring Central –Best Overall on quality and value
- 8x8 : Good value for the price
- Dialpad : Easy to use
- Broadvoice: Affordable upfront
- 3CX : super cheap
- Mitel : Reliable products

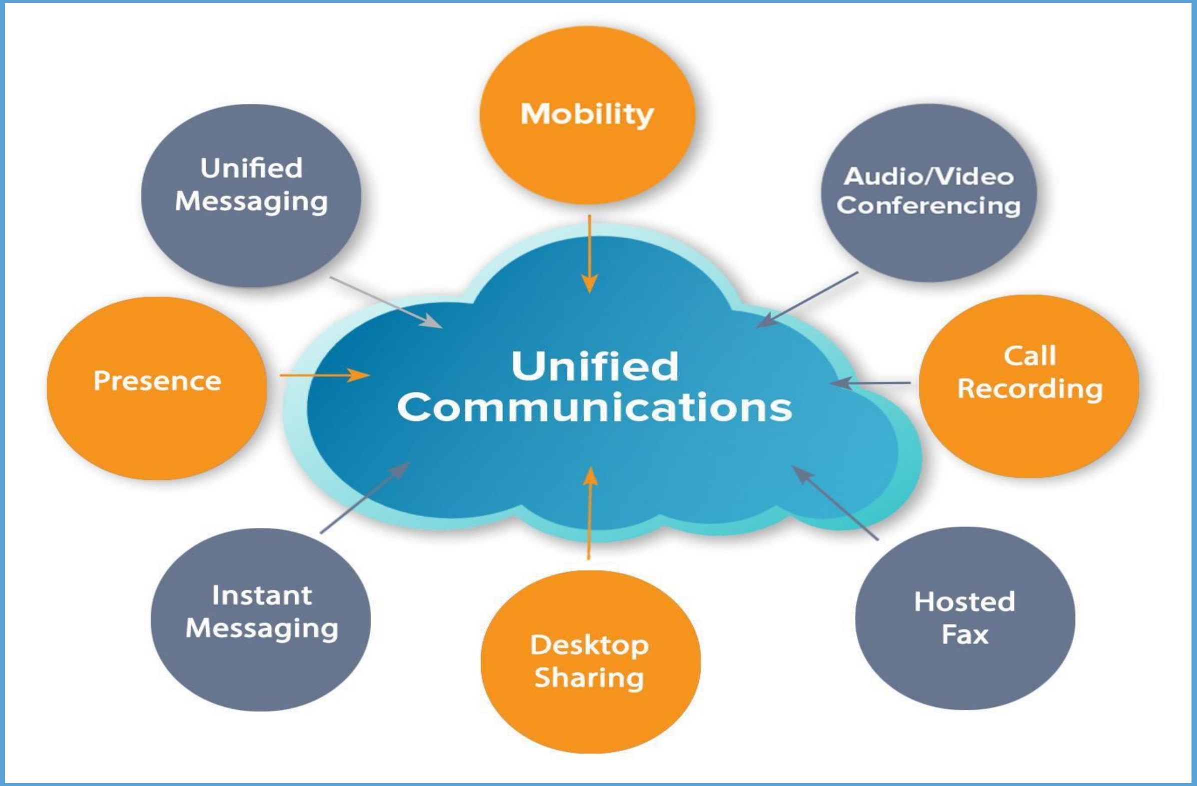
MILITARY DRIVERS

- Familiar to the UC technology
- Rapid access to information for better decision making
- Enhanced mobility
- Ability to reach each and everyone
- Reduce total cost of ownership with easy maintenance and technical training



FEATURES

- Unified Messaging
- Direct Call Routing & Forwarding
- Integration
- Dynamic Lines
- Mobile Apps



THANK YOU