Getting Started with Mac OS X/Linux Command Terminal

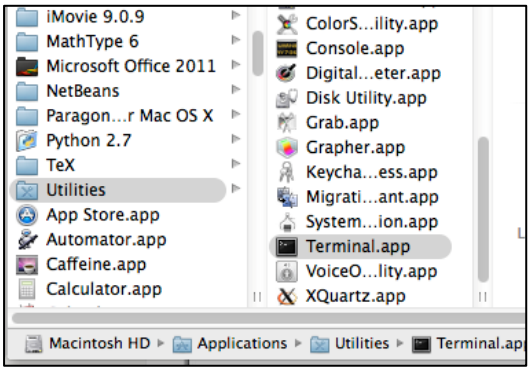
Ziheng Yang
University College London

Asif Tamuri
European Bioinformatics Institute

Updated March 2015

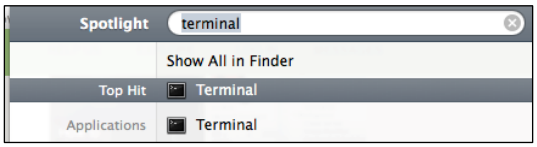
What is a Command Terminal?

In the good old days, users interacted with computers through a command window. This is a text-based window for typing commands and receiving text-based output (see screen shot above). Mouse and menu do not work here but the command line is a powerful interface and is very convenient for running certain programs.



How do I start a Command Terminal?

On Mac OS X, you can open the Terminal from Applications>Utilities>Terminal. Alternatively, you can use the Spotlight search in the top-right corner by searching for the keyword 'terminal'.



Commands for manipulating directories (cd, md)

The OS X or Unix file system consists of a number of directories and sub-directories arranged hierarchically. The root directory is /. When I start the command terminal, I should be in my home directory. This may be `/Users/ziheng/` or `/home/ziheng`, etc., depending on the system setup. The command prompt may show the current (working) directory, as follows: `potto:~ ziheng$`. Here I am user 'ziheng' on a machine called 'potto', and I am in my home directory (which is indicated by the tilde symbol ~). The dollar symbol $ is the command prompt.

Use `cd` to change directory. You can use an absolute path containing the entire directory structure. An *absolute* path starts with a backslash, which means we begin from the root of the file system. Without the leading backslash, the directory is *relative* to your working directory. The tilde character (~) represents your home directory. Thus no matter where you are,

```
cd /
```

will take you to the root directory, and

```
cd
```

or

```
cd ~
```

will take you to your home directory. Also

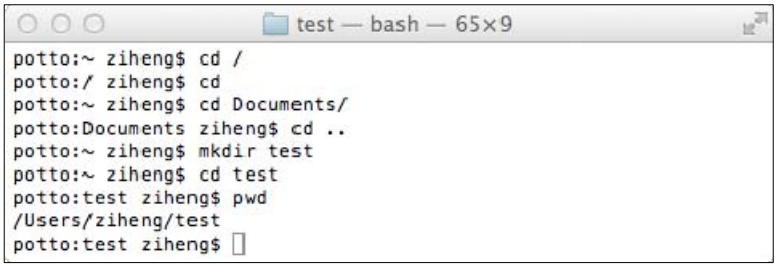
```
cd ~/test
```

will take you to the `test` directory inside your home directory.

`cd ..`
moves up a level to the parent directory.

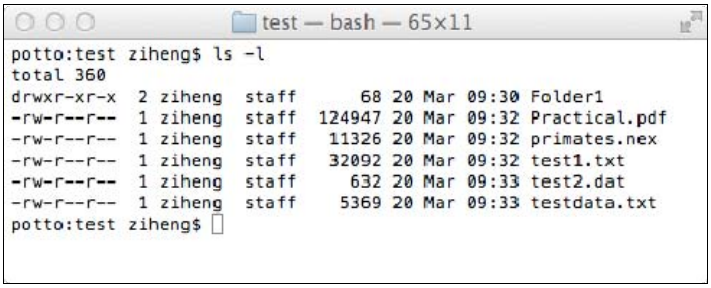
The command `pwd` prints the current (working) directory.

To make a new directory called `test` in the current directory, type
`mkdir test`



Getting directory listings (ls)

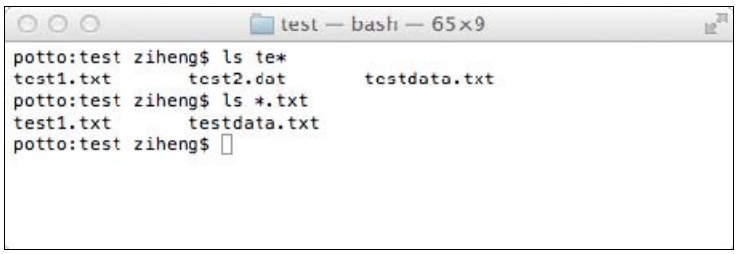
To list the contents of a directory, type `ls`
`ls -l`
The option `-l` means a long listing. The output may look like the following.



Each line provides the file's permissions (which we explain later), the owner (`ziheng`) and group (`staff`) of the file, the size of the file in bytes, the date and time the file was last modified and, finally, the filename.

Wildcards

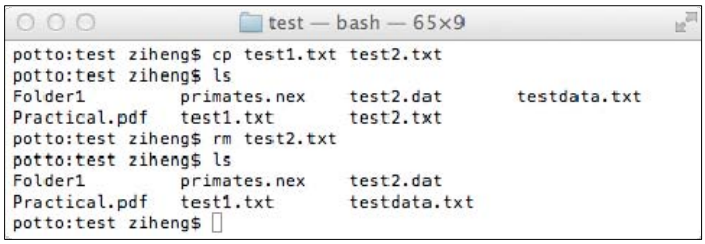
The special characters * and ? can be used as wildcards when you specify file or directory names. The asterisk * means any number of any characters while ? means one character of any kind. Thus
`ls te*`
will list all the files and directories that start with "te".
`ls *.txt`
lists all files that end with .txt (the text files).



Copying and deleting files

The commands `cp` and `rm` are for copying and removing files.

```
cp test1.txt test2.txt
ls -lF
rm test2.txt
ls -lF
```



Wildcards and relative paths can be used together. Suppose I have two directories `test` and `test2` in my home directory, and I am currently in `test`. Then the first command below will copy all files in the `test` folder that have the string `fish` in their names into the `test2` folder, and the second command will delete all files in `test2` that end with `.txt`.

```
cp *fish* ../test2/
ls –l ../test2/
rm ../test2/*.txt
ls –l ../test2/
```

Viewing files on the screen

```
cat test1.txt
less test1.txt
less ../test2/test1.txt
```

The command `cat` shows the content of the file on the screen. This works for plain text files only. If the file is binary (executables and picture files are for example binary files), rubbish and noise will pop up. The command `less` does the same as `cat` but allows forward and backward movement within the file using the arrow and page-up/down keys.

Running programs from the command line

Programs are executable files. You run the program by typing the file name at the command line. The following will run a program called BPP, which is in the `bin/` directory under my home account:

```
~/bin/bpp
```

File permissions

```
drwxr-xr-x 2 ziheng users      4096 Mar 10 14:43 b/
-rw-r--r-- 1 ziheng users     15889 Mar 10 14:43 test1.txt
-rw-r--r-- 1 ziheng users       358 Mar 10 14:43 z.bat
```

The above shows the output from the `ls -lF` command.
In the first column above, **d** means a **d**irectory while dash (-) means a file. The next 9 fields specify the file permissions, in which **r**, **w**, **x**, mean **r**eadable, **w**ritable, and e**x**ecutable while a dash (-) means no permission. The 9 fields are in three blocks, for user (owner), group and other (world), respectively. Thus for the file `test1.txt`, `rw-` means the user can read and write but not execute the file, `r--` means the group can read but not write or execute, while `r--` means that other (everyone with an account on the system) can read the file but can't write or execute it. In other words, `test1.txt` is readable by everybody (user, group and other), writable by owner only, and is not executable. Note that you need executable permission to move (`cd`) into a directory.

Sometimes the file is an executable program, but you can't run it if its permission is not set correctly. This happens often when files are transferred across platforms. In that case you use the `chmod` (change mode) command to set the permissions. The following makes program1 executable by user (owner) and group.

```
chmod ug+x program1
```

A few tips

- Use forward-slash / to specify folders on OS X or UNIX. Use back-slash \ on Windows.
- Commands and file and directory names are case-sensitive on OS X or UNIX, while they are case-insensitive on Windows (MS-DOS).
- Given that different fields on the command line are separated by spaces, it is in general a good practice to avoid using spaces or other strange symbols in file names.

Getting help

Use the command `man` to view the manual page for any particular command.

```
man cp
```

Common useful Windows/Unix commands

Windows	UNIX/OSX	Function
`cd`	`cd`	Change directory (folder)
`md or mkdir`	`md or mkdir`	make a new directory
`dir`	`ls`	List files and directories
`copy file1 file2`	`cp file1 file2`	Make a copy of file1 and name it file2
`ren file1 file2`	`mv file1 file2`	Rename file1 as file2
`move file1 file2`		
`del`	`rm`	delete (remove) files
`rd`	`rmdir`	remove an empty directory
`time`	`time`	date and time mean different things in windows and unix
`date`	`date`	
`exit`	`exit`	exit
`help`	`man`	help or manual
`more`	`more`	show file a screen a time
`type`	`cat`	show file
↑, ↓	↑, ↓	Use the Up & Down arrow keys (↑ and ↓) to
←, →	←, →	cycle through past commands. Then use ← and → or Ctrl-← and Ctrl-→ to move around to edit.
`Tab`	`Tab`	The Tab key completes file or folder names
`>`	`>`	redirection: screen output will go into file
`<`	<	redirection: keyboard input will come from file
`|`	`|`	pipe: output from one program will be input to the next program
`Esc`	`Esc`	Cancel command
`Ctrl-C`	`Ctrl-C`	terminate job
	`nice +20 mb`	run a job at low priority
	`nice +20 mb &`	& places the job at the background
	`Ctrl-Z`	pause a foreground job
	`bg`	place the paused jot at the background

Operating System

What is an operating system?

An operating system (OS) is a program that acts as an interface between the ***user and the computer hardware and controls the execution of all kinds of programs***. OS is responsible for the management and coordination of activities and the sharing of the resources of the computer.

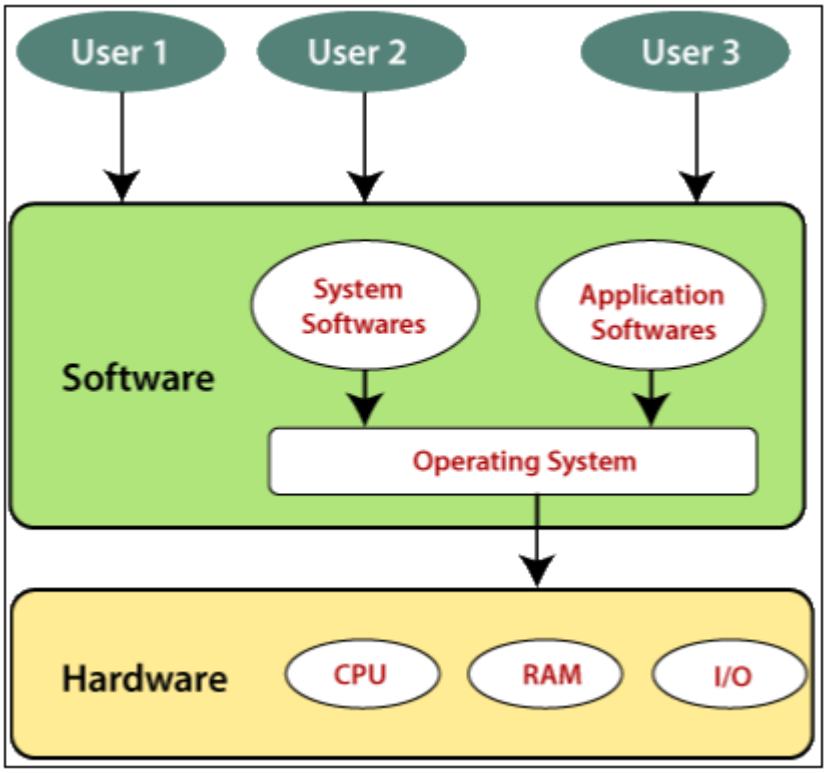


Figure 1. A typical example of an OS

Five of the most common operating systems for personal computers, smart phones and tablets are

- **Microsoft Windows**
- **Apple macOS**
- **Linux**
- **Google Android**
- **Apple iOS.**

Common command line OS commands include:

1) View the contents of a directory
2) Change from one directory to another
3) Create and delete directories
4) Create file in a directory
5) Change from one drive to another
6) Copy files
7) Rename files
8) Delete files
9) Delete a directory

The Command Prompt

Command prompt or DOS prompt look like as follows.

```
C:\>_
```

The flashing underscore next to the command prompt is called the **cursor**. The cursor shows where the command you type will appear.

Typing a Command

This section explains how to type a command at the command prompt and demonstrates the "*Bad command or file name*" message.

Commands are not case sensitive (i.e., you can type the commands either in upper or lower case letters)

Type the following command and hit **Enter**. If you make a typing mistake, press the **BACKSPACE** key to erase the mistake, and then try again.

```
C:\> h
```

The following message appears:

```
Bad command or file name
```

The "*Bad command or file name*" message appears when you type something that MSDOS does not recognize. Because ***h*** is not a valid MS-DOS command, MS-DOS displays the "Bad command or file name" message.

Now, type the following command at the command prompt:

```
C:\> ver
```

The following message appears on your screen:

```
Microsoft Windows [Version 10.0.19042.1110]
```

1) Viewing the Contents of a Directory

In this section, you will view the contents of a directory by using the **dir** command. The **dir** command stands for "**directory**." A name holder for multiple files.

```
C:\> dir
```

A list similar to the following appears:

```
Volume in drive C has no label.
Volume Serial Number is 6CC5-3E92

Directory of C:\

03/06/2021  04:30 AM    <DIR>          3DP
03/06/2021  05:07 AM    <DIR>          AMD
03/06/2021  04:29 AM    <DIR>          Dell
05/12/2021  12:03 PM    <DIR>          gurobi751
07/29/2021  01:19 AM    <DIR>          gurobi912
07/29/2021  01:27 AM    <DIR>          Intel
07/29/2021  05:03 PM    <DIR>          logs
12/07/2019  10:14 AM    <DIR>          PerfLogs
07/30/2021  06:22 PM    <DIR>          Program Files
07/29/2021  05:20 PM    <DIR>          Program Files (x86)
05/13/2021  02:24 PM    <DIR>          Python39
03/06/2021  01:57 PM    <DIR>          Users
07/22/2021  06:15 PM    <DIR>          Windows
               0 File(s)                0 bytes
              13 Dir(s)  152,262,991,872 bytes free
```

This is called a ***directory list***. A directory list is a list of all the files and subdirectories that a directory contains. In this case, you see all the files and directories in the main or ***root*** directory of your drive. All the files and directories on your drive are stored in the root directory.

2) Changing Directories

Look at the list on your screen. All the names that have < ***DIR*** > beside them are directories. You can see a list of the files in another directory by changing to that directory, and then using the **dir** command again. In this case, you will change to the **Windows** directory.

Use the following commands and see the results.

```
C:\> CD windows
C:\Windows> dir
```

If you do not see a line in the directory list indicating that you have a directory named Windows, type the following at the command prompt:

```
C:\Windows> dir /s Windows
```
S is used for search

You will see a message that includes a line such as the following:

```
Directory of C:\WINDOWS
```

3) Change from one directory to another

To change enter another directory (i.e., ABC), type the following command and hit enter.

```
C:\Windows>CD ABC
```

Note: Make sure the ABC directory exists within windows directory. You can use any other directory name that exists within the window directory.

```
C:\Windows\ABC>
```

To leave the directory, type the CD with double dots. For example, to leave the ABC directory type use the following command and press enter.

```
C:\Windows\ABC>CD..
```

Output:

```
C:\Windows>
```

If you are at ABC directory i.e, **C:\Windows\ABC>** and you want to go direct/jump to C (root) directory, use the following command.

```
C:\Windows\ABC>CD\
```

Output:

```
C:\>
```

To view the contents of a directory one screen at a time, type the following command.

```
C:\> dir /p
```

P stand for pause after each screen

One screen of information appears. At the bottom of the screen, you will see the following message:

```
Press any key to continue . . .
```

To view the next screen of information, press any key on your keyboard. Repeat this step until the command prompt appears at the bottom of your screen.

When you typed the **dir** command this time, you included the **/p** switch after the command. A ***switch*** modifies the way MS-DOS carries out a command. Generally, a switch consists of a forward slash (/) that is followed by one or more letters or numbers. When you used the /p switch with the dir command, you specified that MS-DOS should pause after it displays each screen of directory list information. The p actually stands for "page".

Another helpful switch you can use with the **dir** command is the **/w** switch. The **/w** switch indicates that MS-DOS should show a wide version of the directory list.

To view the contents of a directory in wide format, type the following command and press enter.

```
C:\> dir /w
```

The directory list appears, with the filenames listed in wide format. Note that only filenames are listed. No information about the files' size or date and time of creation appears.

If the directory contains more files than will fit on one screen, you can combine the ***/p*** and ***/w*** switches as follows:

```
C:\> dir /w /p   or  C:\> dir /p /w
```

4) Creating a Directory

Creating a directory is helpful if you want to organize related files into groups to make them easy to find. Before you begin this section, make sure the command prompt looks like the following:

`C:\User\Username>` in my case `C:\User\Shahid>.`

Go to desktop, using CD command as follows and hit enter:

```
C:\User\Shahid> CD Desktop
```

Output:

```
C:\User\Shahid\Desktop>
```

To create a directory, you will use the md command. The md command stands for "make directory."

Type the following at the command prompt:

```
C:\User\Shahid\Desktop>md Professional_Skills
```

OR

```
C:\User\Shahid\Desktop>mkdir Professional_Skills
```

Where both **md** and **mkdir** stands for make directory. To test the directory, type the following command.

```
C:\User\Shahid\Desktop>CD Professional_Skills
```

Output:

```
C:\User\Shahid\Desktop\Professional_Skills>
```

5) Deleting a Directory

If you no longer use a particular directory, you may want to delete it to simplify your directory structure. Deleting a directory is also useful if you type the wrong name when you are creating a directory and you want to delete the incorrect directory before creating a new one.

Before you begin this section, make sure the command prompt looks like the following:

```
C:\User\Shahid\Desktop>
```

To delete a directory, use the **rd** command and press enter. The **rd** command stands for "remove directory."

```
C:\User\Shahid\Desktop>rd Professional_Skills
```

To verify, use the following command and press enter.

```
C:\User\Shahid\Desktop>CD Professional_Skills
```

Output:

```
The system cannot find the path specified.
```

6) Creating a File in a Directory

Creating a file in a directory similar to creating a directory, except you have to provide the file extension, like .txt, etc. But make sure the directory where you are going to create file should be active:

For example, we first create a directory named "***Professional Skills***", made it active and then create a file named "***CS.txt***". Type the following commands and press enter each time.

```
C:\User\Shahid\Desktop>mkdir Professional_Skills

C:\User\Shahid\Desktop>CD Professional_Skills

C:\User\Shahid\Desktop\Professional_Skills>
```

To create the file the general syntax is "***fsutil file createNew <filename> <length>***", the ***fsutill*** stands for ***file system utility.*** Type the following command and press enter.

```
C:\User\Shahid\Desktop\Professional_Skills> fsutil file createnew CS.txt 1000
```

Where 1000 mean it will create 1 kb file.

You can also use copy con command. With copy con you can create file and write to the file as well.

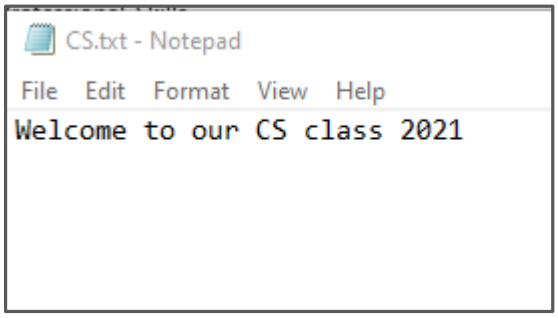
```
C:\User\Shahid\Desktop\Professional_Skills> Copy con CS1.txt
This is a test file…
```

This command will create file CS1.txt and will write the text ***This is a test file…***

Writing to a text file:
To write to the CS.txt file, type the echo command and press enter.

```
C:\User\Shahid\Desktop\Professional_Skills> echo Welcome to our CS111 class 2021> CS.txt
```

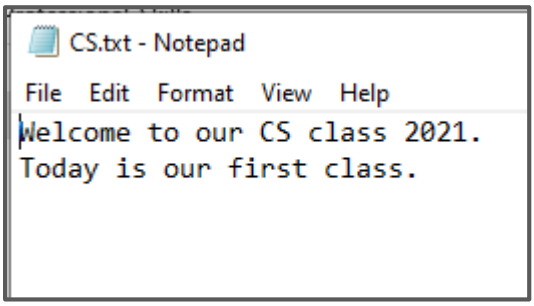
Output:



The > operator delete the old text and write the new one, if you want to keep the previous text use the append >> operator.

```
C:\User\Shahid\Desktop\Professional_Skills> echo Today is our first class>> CS.txt
```

Output:



Renaming Files:

You may want to rename a file if the information in it changes or if you decide you prefer another name.

To rename a file, you will use the **ren** command. The **ren** command stands for "***rename***."

When you use the ren command, you must include two parameters.

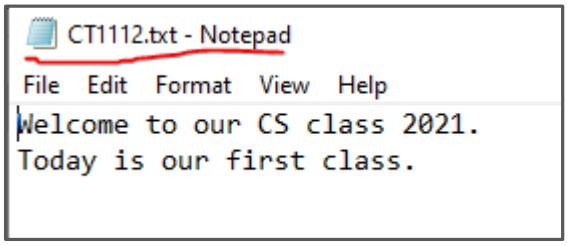
The first is the file you want to rename, and the second is the new name for the file. You separate the two names with a space. The ren command follows this pattern:

ren oldname newname

Type the following command and press enter.

```
C:\User\Shahid\Desktop\Professional_Skills>ren
CS.txt CT1112.txt
```

Output:



Copying Files:

Copying files creates a duplicate of the original file and does not remove the original file. This is useful for many reasons. For example, if you want to work on a document at home, you can copy it from your computer at work to a floppy disk and then take the floppy disk home.

USB drive

To copy a file, you will use the copy command. When you use the copy command, you must include two parameters. The first is the ***location and name*** of the file you want to copy, or the source. The second is the ***location to which you want to copy*** the file, or the destination. You separate the source and destination with a space.

copy source destination

```
copy
c:\Users\Shahid\Desktop\Professional_Skills\CT1112.
txt C:\Users\Shahid\Desktop
```

This will copy the CT112.txt file to the Desktop. You will see the following output.

Output:

```
1 file(s) copied.
```

Deleting Files:
To delete a file, you will use the ***del*** command. The general syntax for deleting file is as follows.

del file_name.extension

```
C:\Users\Shahid\Desktop> del CT1112.txt
```

This command will delete the CT1112.txt file from desktop.

Deleting multiple files:
To delete multiple files, use the following command and press enter.

```
C:\Users\Shahid\Desktop\Professional_Skills> del
*.txt
```

Deleting a directory:
To delete a directory, use the following command and press enter.

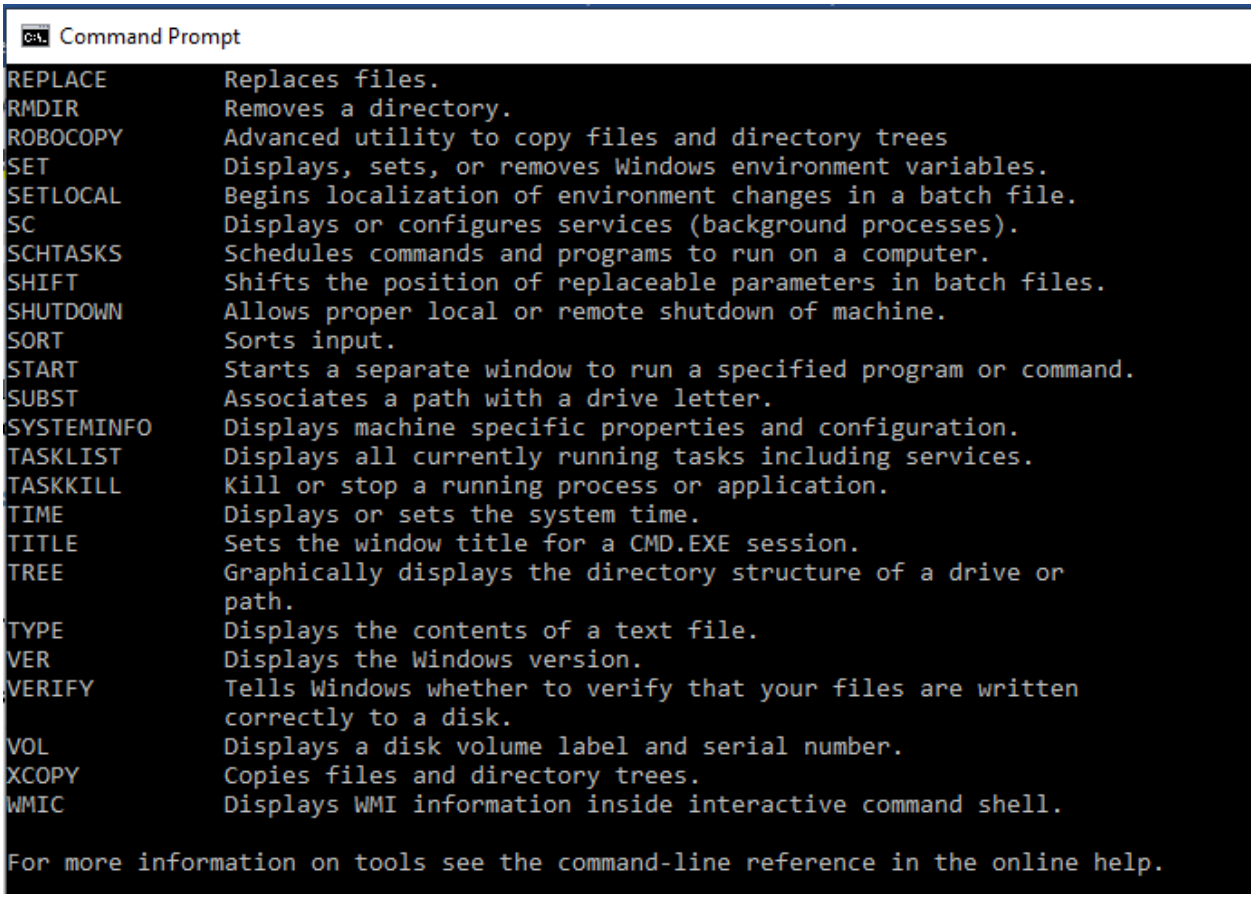
```
C:\Users\Shahid\Desktop> del Professional_Skills
```

Output:
Are you sure (Y/N)? Y

Note: In this lecture we covered basic commands, there are many other commands for learning use the ***Help*** and press enter.

```
C:\Users\Shahid\Desktop> Help
```

Output:



Exercise:

Practice all the above commands and Using your PC's terminal window, create a new directory (aka folder) on your Desktop called Professional Skills. Make this folder your active directory and make new subfolders called LaTeX, Python and Excel where you can store your exercise files

Latex Lecture-2

1) Introduction
2) Registration
3) Starting a new project
4) Components of overleaf editor
5) Different options for new project
6) Searching in latex file
7) Best practice for article management
8) Reserved characters
9) Hot/shortcut keys
10) Preamble (Introduction)
11) Body (Introduction)
12) Real-time tracking feature
13) Compile timeout error

1. Introduction:

- LaTeX (pronounced LAY-tek) is a higher layer of the TeX programming language specifically designed to create ***professionally formatted documents*** that include ***complex mathematical expressions***. Such documents include:
  - ✓ Academic Journals
  - ✓ Book
  - ✓ Formal Letters
  - ✓ Homework Assignment
  - ✓ Poster
  - ✓ Presentation
  - ✓ Project/Lab Report
  - ✓ Resume/CV
- LaTeX can be installed on your PC but most users now use cloud based applications e.g. www.overleaf.com (free).
- Overleaf gives instant access to the LaTeX programming language and has an extensive help environment.
- A LaTeX program has two parts: The ***Preamble*** containing packages or modules of Tex code and the ***Body***.
- All keyword commands begin with a ***backslash*** (\ ) and parameters or arguments are passed using ***curly brackets*** ({}).
- Comments can be added using the **%** character. The program example below contains ***two statements*** in the ***Preamble*** and ***three*** in the ***Body***:

```
\documentclass[a4paper,12pt]{article}     % Preamble section
\title{Hello world!}
```

```
\begin{document}                                % Body section
\maketitle
\end{document}
```

2. **Registration:**
    - To start using Overleaf go to www.overleaf.com.
    - If you don't have an account enter your e-mail address and set a password in the corresponding boxes below ***Get started now***, click ***Register*** and that's it, you will be redirected to the project management page where you will be guided into how to create a new project.
    - If you already have an account, click **Login** in the upper right corner, then type in your ***email*** and ***password*** and click the Login button (Figure 1).

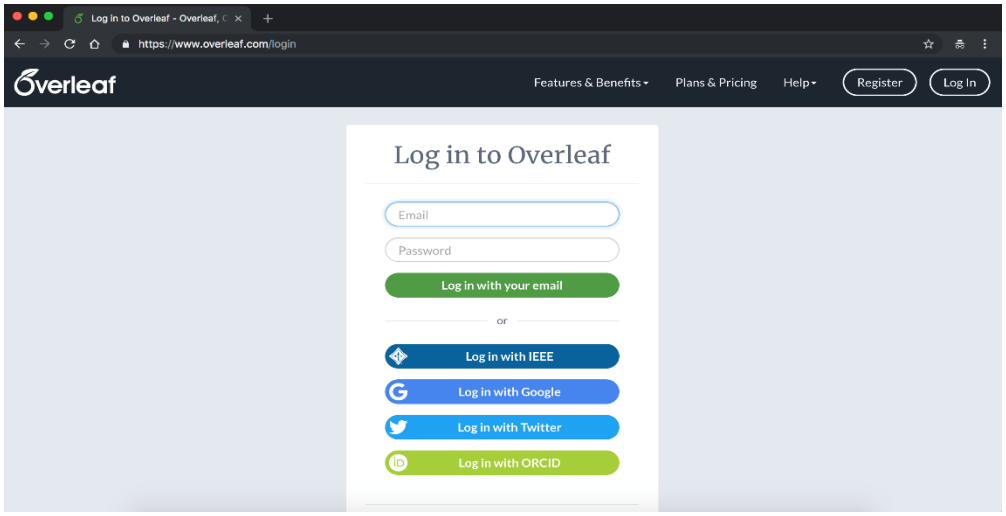


Figure 1. Registering to overleaf

- Once you are logged in, you should see the Overleaf Project Management page it will look like the Figure 2.

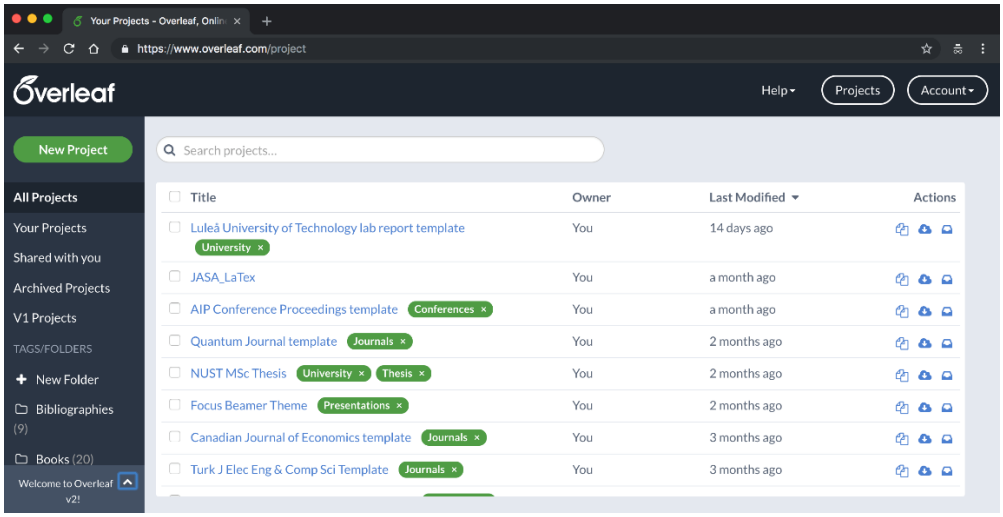


Figure 2. Overleaf Project Management

3. **Starting a new project:**
   - To start a project, click on ***New Project***. There are ***several option*** to start a new project. The ***first two options*** are for creating a project from scratch, with a basic setting.
     - ✓ Blank Project
     - ✓ Example Project
     - ✓ Upload Project
     - ✓ Import from GitHub
   - Click on the Blank Project and you will see a ***text box*** where you should enter the ***name of your new project***

4. **Components of overleaf editor:**

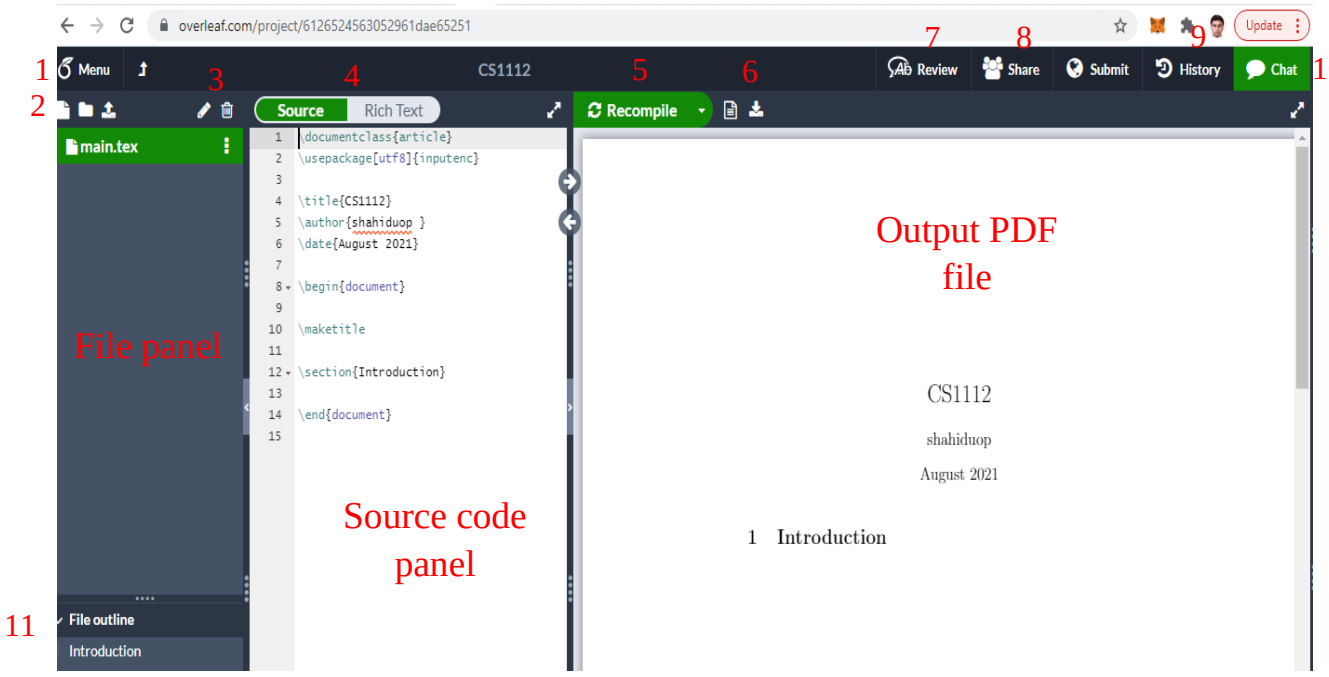


Figure 3. An overview of the overleaf editor

1. **Menu:** The menu has the following main sections.
   - ✓ Download (Source code (zip) and PDF files)
   - ✓ Action (Copy and ***word count***)
   - ✓ Syn (Dropbox, Git, GitHub)
   - ✓ Setting (Compiler, version, document name, version…., etc.)
   - ✓ Help
2. **Three icons:**
   - ✓ Create file: For creating a new file.
   - ✓ Create folder: For creating a new folder
   - ✓ Upload file: For uploading files (i.e., template, file, image, etc.)
3. **Rename and delete project or file**
4. **View options:** Latex format or rich text format
5. **Recompile:** The recompile results in an updated output
6. **Download icon:** This option allows you to download the PDF file.
7. **Review:** This option allows others (i.e., experts) to review a documents and give feedback.
8. **Share:** This option let you to share your documents with other team members for working in a common document.
9. **Submit:** This option let you to submit your article (. i.e., to the journal/conference, etc.)
10. **Chat:** When sharing with other colleagues, you can chat and discuss using the chat option.
11. **Sections/Caption:** This option shows the main sections/label/caption of your article/document, such as Introduction/Related work/Proposed work etc.

5. **Different options for new project:**
   A. **Blank project:** This option creates a project from the scratch with a basic layout as shown (Source code) in the following Figure 3.

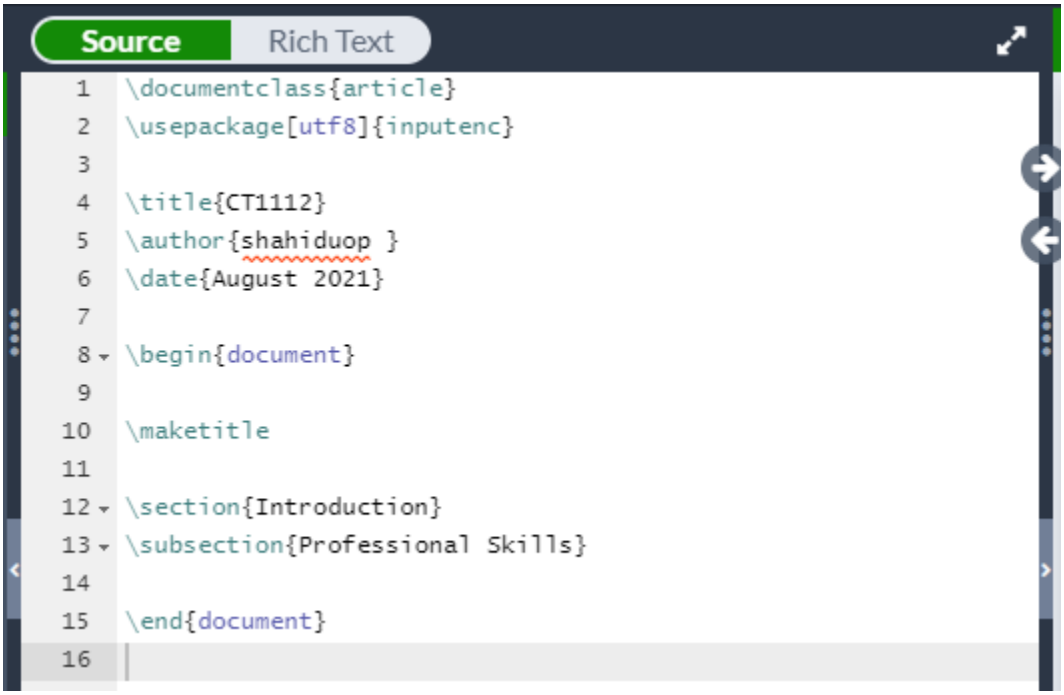


Figure 4. An overview of source code for a ***Blank Project***

   B. **Example project:** This option creates a new project with examples for different sections. You need to change/edit the sections according to your needs. The layout looks like shown in Figure 5.

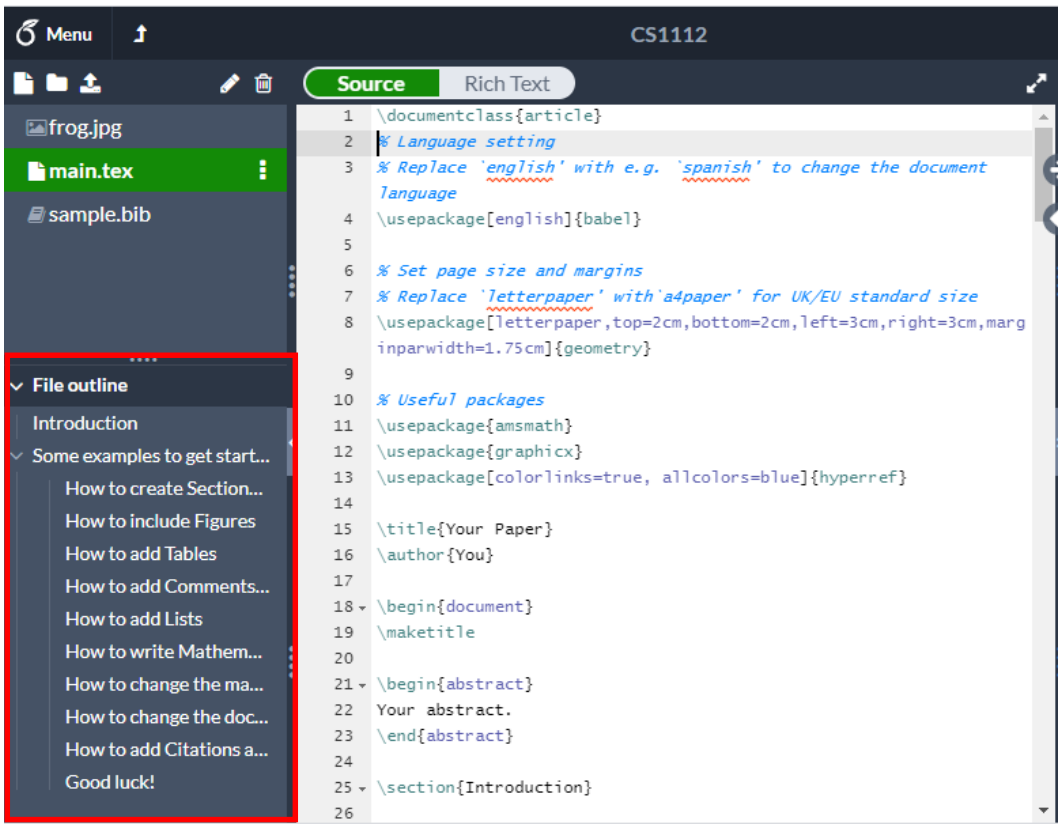


Figure 5. An overview of source code for ***Example Project***

C. **Upload project:** Most of the academic journals have specific format and they provide a latex template (set of classes etc.). This option is for uploading the project with specific layout. The following simple steps are used.

1) Search the specific template and download the latex package for the specific journal. For example, considering the IEEE ACCESS journal, you can find the latex template in their submission guideline. Click on the ***Latex*** to download the template.

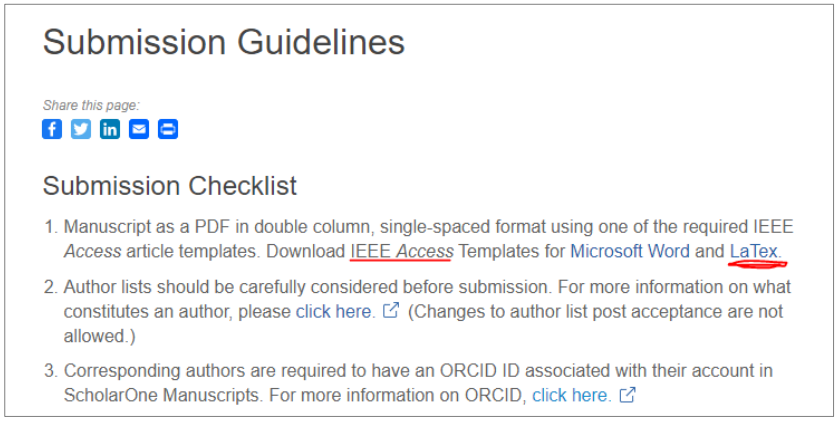


Figure 6. Latex template for IEEE ACCESS (An example)

2) Click on the ***upload project*** and upload the ***zip file,*** that you just downloaded. The uploading window will look like Figure 7.

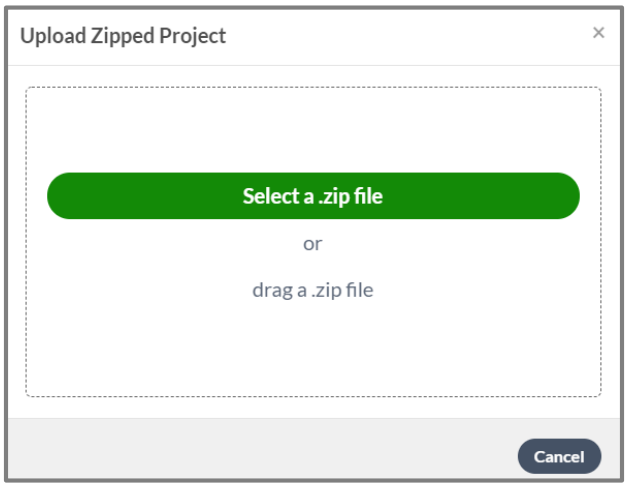


Figure 7. Upload Zip file of Latex template

Once you upload, you will see the designated template, for example in the case of IEEE ACCESS, you will see the following layout.

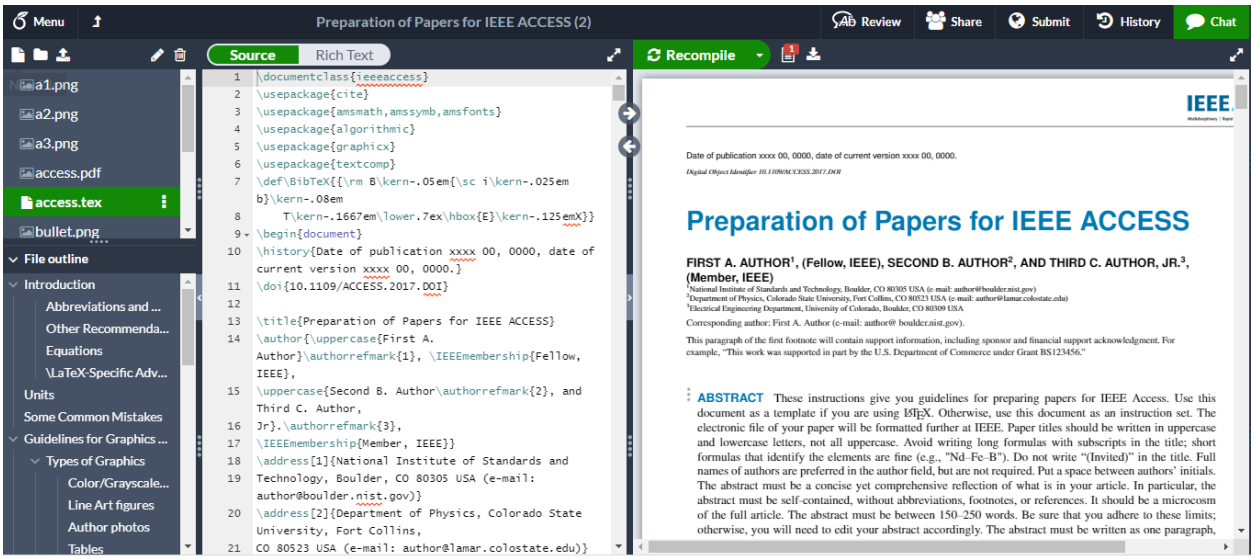


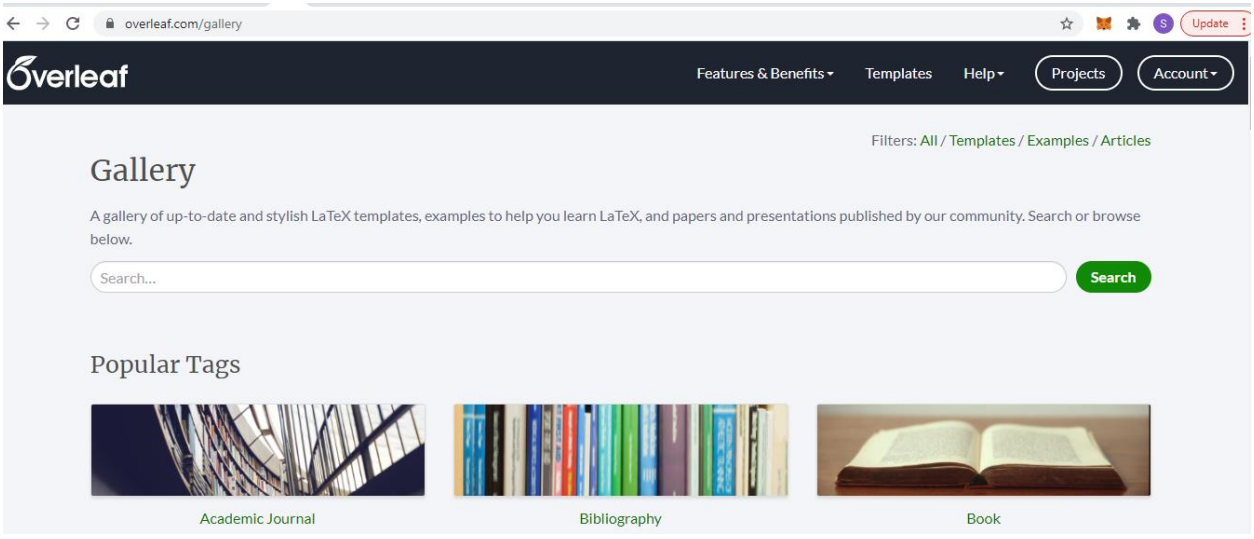
Figure 8. Example of IEEE ACCESS template in latex

D. Import from GitHub project: You can import a project from GitHub repository. But it's a premium feature and would work for a trail with limited accessibility. This option result in the following output message.

Figure 9. Importing a project from GitHub (output window)

E. **Template:** The template option let you to search online for specific templates and use them. The template includes:
   - ✓ Academic Journals
   - ✓ Book
   - ✓ Formal Letters
   - ✓ Homework Assignment
   - ✓ Poster
   - ✓ Presentation
   - ✓ Project/Lab Report
   - ✓ Resume/CV

Click on any of the template and then click on the ***Show all Gallery Items***, this will lead you to the search option.



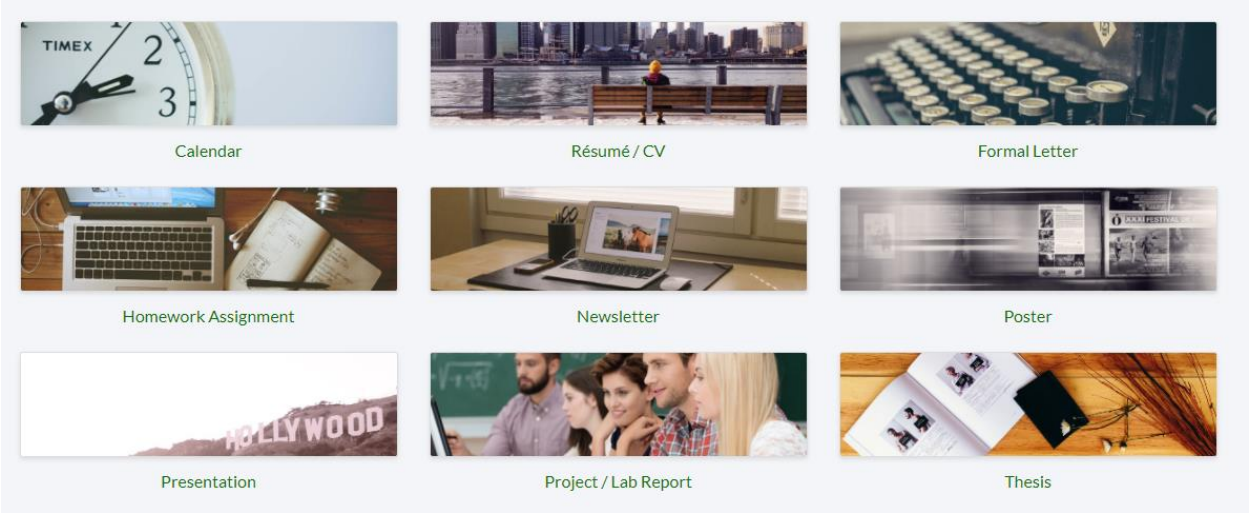
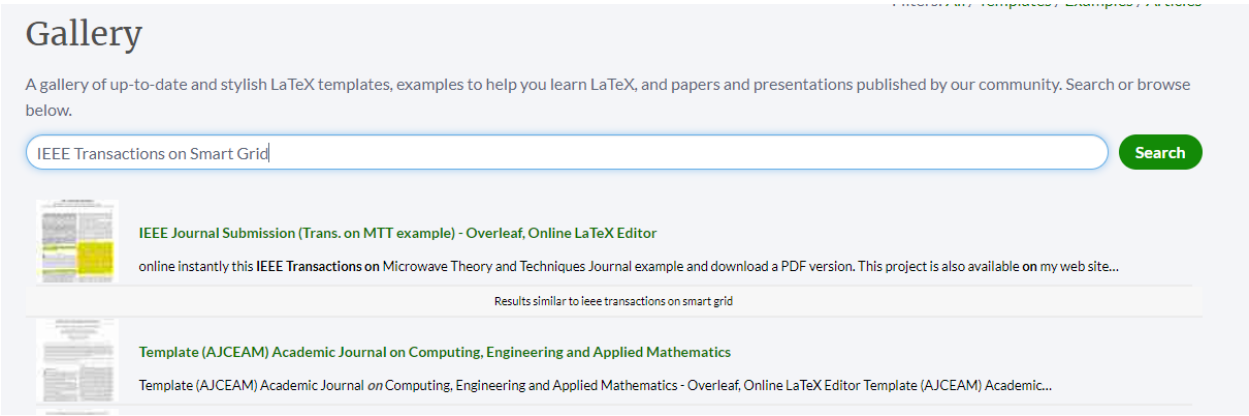
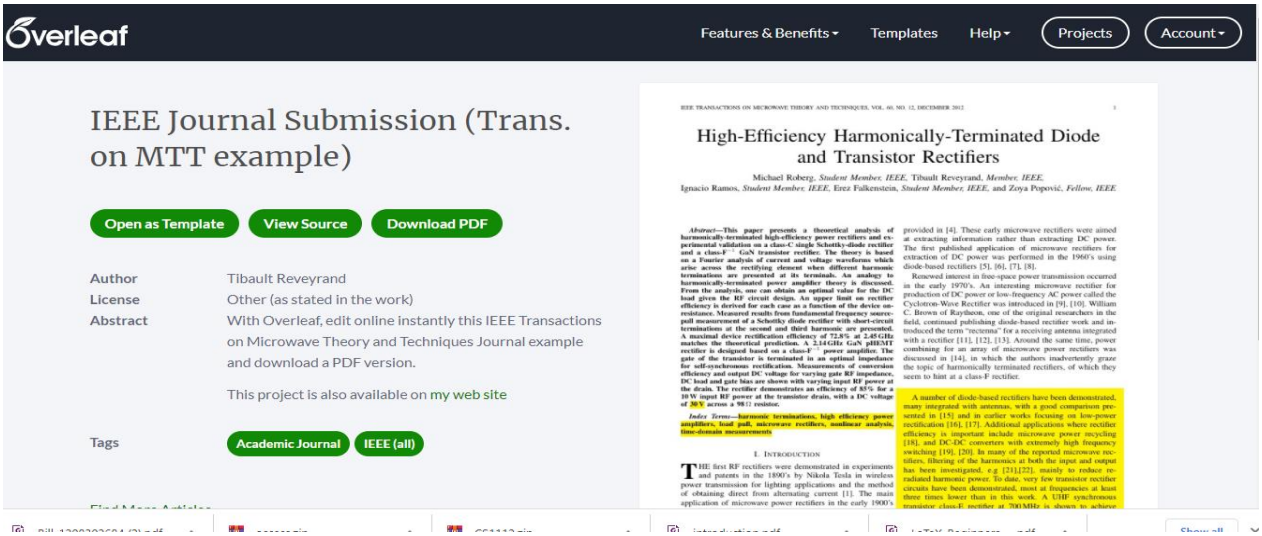


Figure 10. Illustration of searching options

For example, we want to search for **IEEE Transactions on Smart Grid.** It gives a bunch of options for different transactions templates. Scroll down to find the desired the templates.





6. **Searching in latex file:** You can search both in ***latex*** file and in the ***PDF*** file using the ***Crtl+F*** key. For example, searching the abstract result the following output.

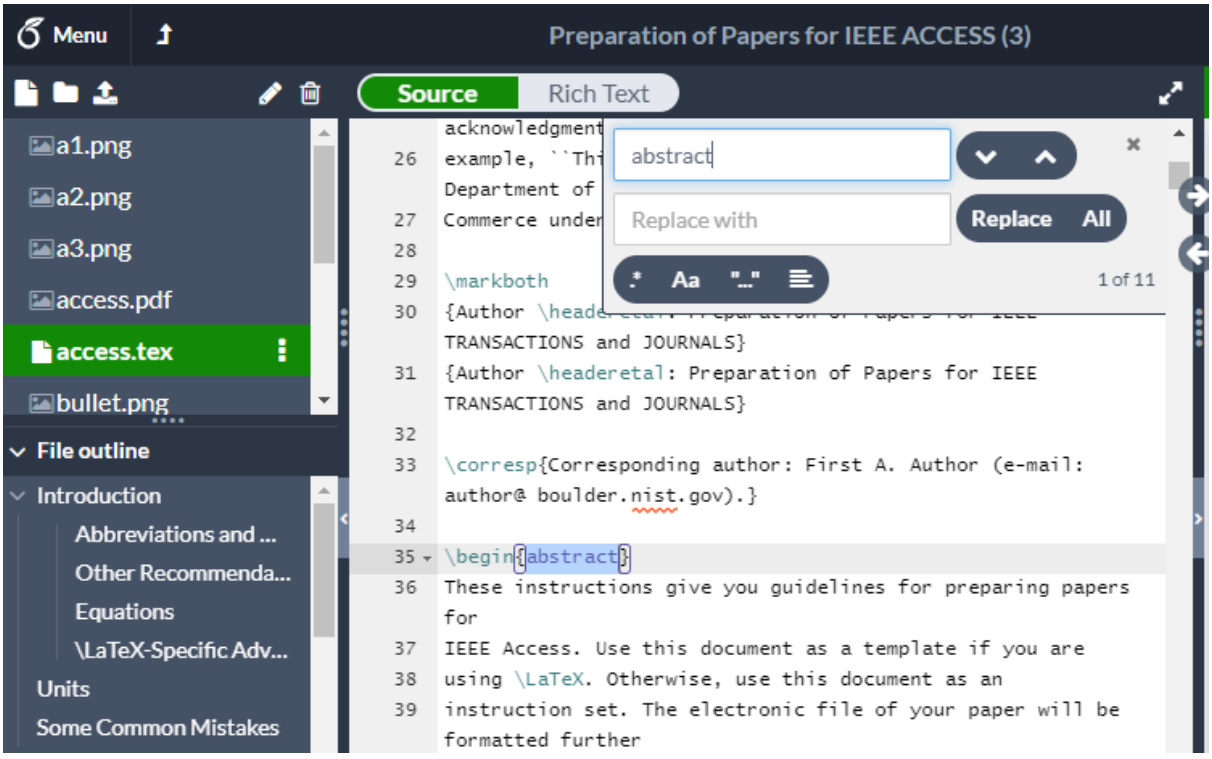


Figure 11. Searching in latex

7. **Best practice for article management:**
   - The best practice for managing article is to maintain the files (latex and bibliography) and figures separately as shown in the Figure below.

- Keep the original version by remaining the article. This will help to get the original version back; in case you need it.
- If you are using several images, a high resolution images may cause time-out error. For images PDF files are recommended instead of PNG image.

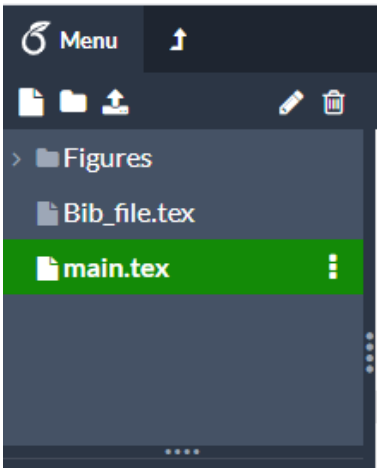


Figure 12. Managing files (example)

8. Reserved characters: The following (Figure 13) shows the list of reserved characters and how to print them if needed.

Character	Function	How to print it
#	Macro parameter	\#
$	Math mode	$
%	Comment	\%
^	Superscript (in math mode)	\^{} or \textasciicircum
&	Separate column entries in tables	\&
_	Subscript (in math mode)	_
{ }	Processing block	\{ \}
~	Unbreakable space, use it whenever you want to leave a space which is unbreakable	\textasciitilde or \~{}
\	Starting commands, which extend until the first non-alphanumerical character	\textbackslash or \

Figure 13. List of reserved words

9. Hot/shortcut keys: The hot/shortcut keys save the time, a list of such keys with detailed description is given in Figure 14.

Figure 14. List of hot/shortcut keys with detailed description

10. Preamble: The preamble defines the ***type of document*** you are writing, the ***spoken language*** you are writing in and the ***packages (modules)*** of Tex code you would like to use. Keywords are case sensitive e.g. writing "***Documentclass***" below will return a syntax error.

```
\documentclass{article}              % package for formatting articles
\usepackage[utf8]{inputenc}          % package for character encoding
\usepackage[margin=25mm]{geometry}   % package for formatting margins
\usepackage{natbib}                  % …formatting citations and
                                     bibliographies
\usepackage{graphicx}                % …formatting and numbering figures
\usepackage{amsmath}                 % …formatting equations
\usepackage{xcolor}                  % …text colours red, green, etc
```

```
\title{Report Title}
\author{Connor Adams, ID: 20379631}
\date{October 2020}
```

11. **Body:** The body of the program contains all of the printable text in addition to various formatting commands.

```
\begin{document}                % begin the printable document
\maketitle{title_CS1112}        % place title here (title, author,date,…)
\tableofcontents                % place "table of contents" here
\pagebreak                      % place a page break here
```

12. **Real-time tracking feature: K**eep an up-to-date list of all of the things you need to work through, without having to sift through out-of-date notes or dig out old emails.
It is a premium feature and need a payment; however, you can try a trail. A tracking example with ***accept/reject*** option is shown in Figure 15.

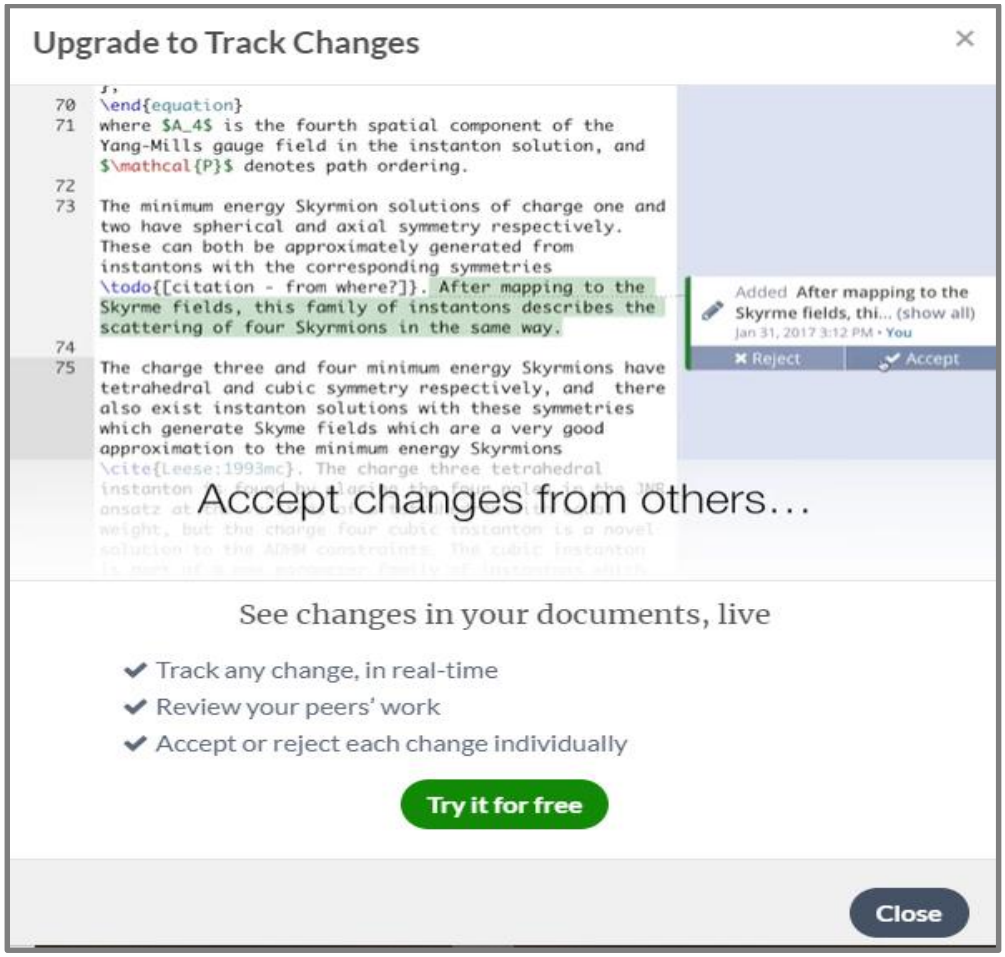


Figure 15. Real-time tracking feature

13. **Compile timeout error:** Sometime the users experience the compile timeout error message as shown in Figure 16. A default recompilation time for free version is set to 1 minute. If your document takes more than a minute, overleaf will result in ***Time out*** error.

Possible reasons: There are several possible reasons for the ***Time out*** error; however, the most common is the use of having ***many images with high resolutions***. A list of the possible reasons is as follows.

1. Large, High-resolution images: ***600dpi/1200dpi etc.***
2. Complicated TikZ or pgfplots drawings: ***TikZ*** (visualization tool) ***pgfplots*** package
3. mhchem: A ***package*** for ***chemical molecular formulas and equations***
4. biblatex: Package that ***re-implement the bibliographic***
5. Tracing/debugging calls: Records lots and lots of lines
6. Infinite loops: A package calling itself (recursion) causes the time out
7. Fatal compile errors blocking the compilation: Block the ***latexmk*** build process
8. Fair Use limits: Time out limit is 1 and 4 minutes for free and paid versions
9. Still stuck? Other system related issues or network traffic issue

Please visit the following URL to find more about the possible reasons for the Time out error.

https://www.overleaf.com/learn/how to/Why_do_I_keep_getting_the_compile_timeout_error_message%3F

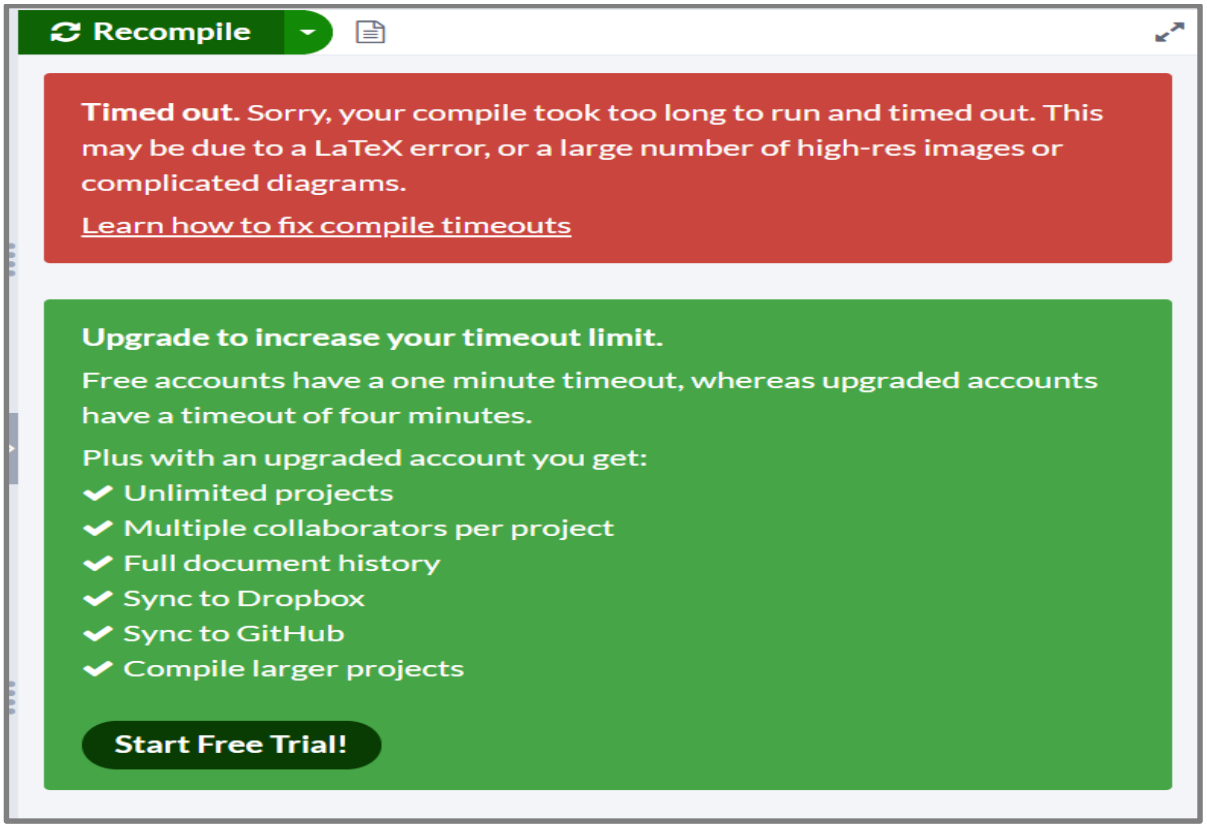


Figure 16. Example of compile time out error message

Exercise: Register to overleaf and practice the different options, discussed in the lecture.

Latex Lecture-3

1) Preamble
2) Body
3) Sections
4) Alignment text
5) Paragraphs and new lines
6) Fonts effects
7) List and its types

1. **The preamble of a document:**
   The part of your ***.tex*** file before the ***\begin{document}*** point is called the ***preamble***. In the preamble, you define the ***type of document*** you are ***writing*** and ***the language***, ***load extra packages*** you will need, and ***set several parameters***. For instance, a normal document preamble would look like this:

```
\documentclass[12pt, letterpaper]{article}
\usepackage[utf8]{inputenc}

\title{First document}
\author{Hubert Farnsworth \thanks{funded by the Overleaf team}}
\date{February 2014}
```

Figure 1. Example of preamble

Description:

`\documentclass[12pt, letterpaper]{article}`

- Type of document ***(article.),*** the other options are given in Figure 2.
- Font size (***12pt***) and the paper size (***letterpaper***).

`\usepackage[utf8]{inputenc}`

- This is the encoding for the document UTF8 stands for ***Unicode Transformation Format*** 8-bit

`\title{First document}`

- This is the title

`\author{Hubert Farnsworth}`

- Here you put the name(s) of the author(s) and, as an optional parameter

`\thanks{funded by the Overleaf team}`

- This can be added after the name of the author, inside the braces of the title command. It will add a superscript and a footnote with the text inside the braces. Useful if you need to thank an institution in your article.

```
\date{February 2014}
```

- You can enter the date manually or use the command **\today** so the date will be updated automatically at the time you compile your document.

Document type	Description
article	For short documents and journal articles. Is the most commonly used.
report	For longer documents and dissertations.
book	Useful to write books
letter	For letters
slides	For slides, rarely used

Figure 2. Available options in the documentclass

2. **The body of a document:** The body of your document you can use the next commands for the information to be printed as shown in the code.

```
\documentclass[12pt, letterpaper, twoside]{article}
\usepackage[utf8]{inputenc}

\title{First document}
\author{Hubert Farnsworth \thanks{funded by the Overleaf team}}
\date{February 2014}

\begin{document}

\begin{titlepage}
\maketitle
\end{titlepage}

In this document some extra packages and parameters
were added. There is an encoding package
and pagesize and fontsize parameters.

\end{document}
```

\begin{titlepage} \end{titlepage}

This declares an ***environment***, a block of code with a specific behavior. In this case whatever you include in this ***titlepage*** environment will appear in the ***first page of your document.***

\maketitle

This command will print the ***title, the author and the date***.

3. **Sections:** The body consists of several sections and subsections, including special sections such as the ***nomenclature, abstract***, keywords etc., and normal sections such as introduction, related work and so on. Sections and subsections are very easy to create.

```
\section{First level section}
\subsection{Second level section}
\subsubsection{Third level section}
\section*{Unnumbered section}
\paragraph{...}
\subparagraph{...}
```

3.1. Special sections:

- **Nomenclature:** The nomenclature requires the ***\usepackage{nomencl} with the following command.***

```
\makenomenclature
\nomenclature{Variable \quad {Description}}  % \quad used for space
\nomenclature[VR]{$\cup$, $\backslash$, $\odot$}{\qquad  Union, subtraction and composition operations}
\printnomenclature
```

Output:

NOMENCLATURE

Variable	Description
$\cup$, $\backslash$, $\odot$	Union, subtraction and composition operations
l'	Laxity of EVs
$\mu(x)$	Membership degree of x
$\tilde{p_i}$	Fuzzy weight control variable for the i-th EV
θ	Ratio of RST to laxity
A, B, C	Fuzzy sets

Figure 3. Example output for nomenclature output

- **Abstract:** Define the summery of your work and can be written using the following command.

```
\begin{abstract}
```

Abstract text here…

```
\end{abstract}
```

- **Keywords:** The keywords are defined in their specific section. The following command is used.

```
\begin{IEEEkeywords}
```
Keyword1, keyword2,….
```
\end{IEEEkeywords}
```

- **Footnote:** For the footnote use the `\footnote{footnotes text here}` command

- **Page header:** Use the package `\usepackage{fancyhdr}` as shown in the following code.

```
\documentclass{article}
\usepackage[english]{babel}
\usepackage[utf8]{inputenc}
\usepackage{fancyhdr}

\pagestyle{fancy}
\fancyhf{}
\rhead{Overleaf}
\lhead{Guides and tutorials}
\rfoot{Page \thepage}

\begin{document}

\section{First Section}

Hello, here is some text without a meaning. This
text should show what a printed text will look like at
this place. If you read this text, you will get no information.
Really? Is there no information? Is there a difference between
this ...

\end{document}
```

Output:

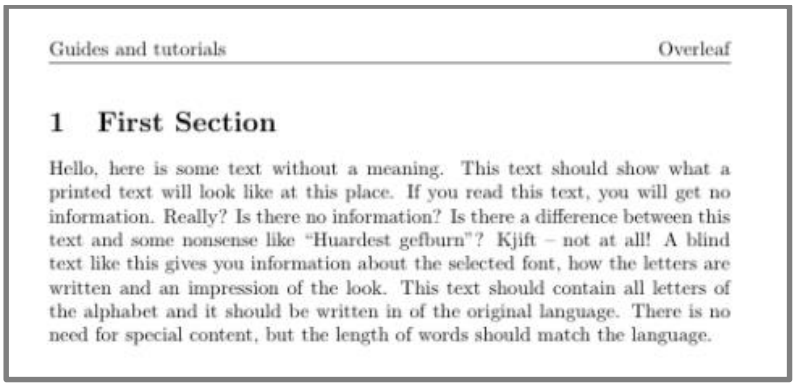


Figure 4. Example output of page header

- **Multiline commends:** The percent sign (%) is used to comment a single line; however, if multiple lines comments are required, use the package **\usepackage{comment}** in preamble and follow the following syntax.

```
\begin{comment}
This document contains a lot of comments, none of them
will appear here, only this text.
\end{comment}
```

Syntax and commands for the different level of sections.

```
\section{First level section}
\subsection{Second level section}
\subsubsection{Third level section}
\section*{Unnumbered section}
```

Output:

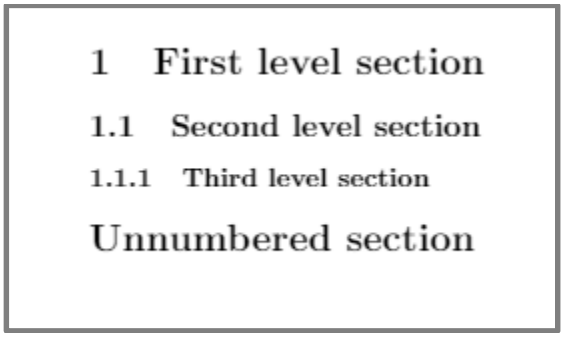


Figure 5. Output example for different level of sections

4. **Alignment text:** The following command center align the text .

```
\begin{center}
```
Example 1: The following paragraph (given in quotes) is an example of Center Alignment using the center environment.

``LaTeX is a document preparation system and document markup language. LaTeX uses the TeX typesetting program for formatting its output, and is itself written in the TeX macro language. LaTeX is not the name of a particular editing program, but refers to the encoding or tagging conventions that are used in LaTeX documents".

```
\end{center}
```

Output:

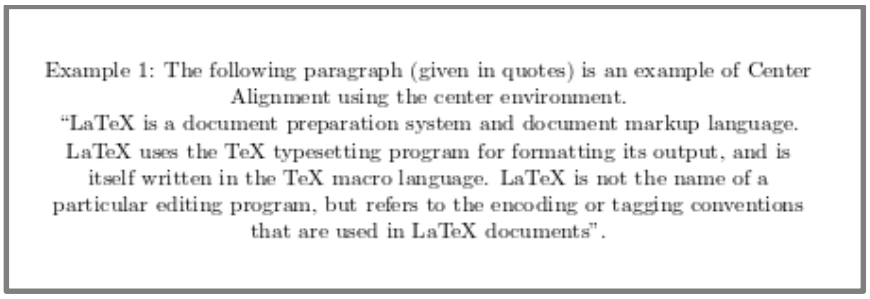


Figure 6. Example out of center aligned text

5. **Paragraphs:** To start a new paragraph, you must ***leave a blank line*** in between. There's a `\par` command that start a new paragraph, the code look like the following.

```
This is the text in first paragraph. This is the text in first
paragraph. This is the text in first paragraph. \par
This is the text in second paragraph. This is the text in second
paragraph. This is the text in second paragraph.
```

Output:

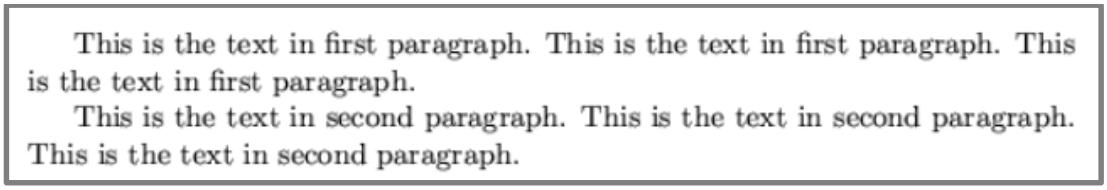


Figure 7. Example of a new paragraph through **\par** command

5.1. Paragraph Alignment (Text Justification): Paragraphs in LaTeX are fully justified using the ***flush*** command environment. For instance, center, flushleft, and flushright, for center, left, and right justifying the paragraph. The full justifying (i.e., both left and right) requires the **\usepackage{ragged2e}** package. The following two commands are used to illustrating a left and full justifying paragraph.

Left justifying:

```
\begin{flushleft}
``LaTeX is a document preparation system and document markup
language. LaTeX uses the TeX typesetting program for formatting
its output, and is itself written in the TeX macro language.
LaTeX is not the name of a particular editing program, but refers
to the encoding or tagging conventions that are used in LaTeX documents".
\end{flushleft}
```

Output

"LaTeX is a document preparation system and document markup language. LaTeX uses the TeX typesetting program for formatting its output, and is itself written in the TeX macro language. LaTeX is not the name of a particular editing program, but refers to the encoding or tagging conventions that are used in LaTeX documents".

Figure 8. Example of Left justifying paragraph

Full justifying: Note don't forget to use the package `\usepackage{ragged2e}`

```
\begin{flushleft}
\justifying ``LaTeX is a document preparation system and document markup
language. LaTeX uses the TeX typesetting program for formatting
its output, and is itself written in the TeX macro language.
LaTeX is not the name of a particular editing program, but refers
to the encoding or tagging conventions that are used in LaTeX documents".
\end{flushleft}
```

Output

LaTeX is a document preparation system and document markup language. LaTeX uses the TeX typesetting program for formatting its output, and is itself written in the TeX macro language. LaTeX is not the name of a particular editing program, but refers to the encoding or tagging conventions that are used in LaTeX documents.

Figure 9. Example of full justifying paragraph

Summary of text alignment

Summary of environments and commands for text alignment

Alignment	Environment	Switch command	ragged2e environment	ragged2e switch command
Left	`flushleft`	`\raggedright`	`FlushLeft`	`\RaggedRight`
Right	`flushright`	`\raggedleft`	`FlushRight`	`\RaggedLeft`
Centre	`center`	`\centering`	`Center`	`\Centering`
Fully justified			`justify`	`\justifying`

5.2. Paragraph Indentation: By default, LATEX does not indent the first paragraph of a section or a chapter. The paragraph indents is determined by `\parindent`. The following code illustrates the indentation.

```
\setlength{\parindent}{4em}
\begin{document}
```

This is the text in first paragraph. This is the text in first paragraph. This is the text in first paragraph. \par

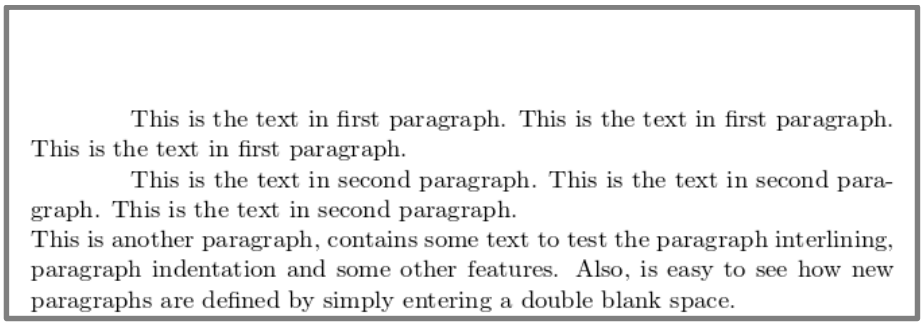
This is the text in second paragraph. This is the text in second paragraph. This is the text in second paragraph.

This is another paragraph, contains some text to test the paragraph interlining, paragraph indentation and some other features. Also, is easy to see how new paragraphs are defined by simply entering a double blank space.

...

```
\end{document}
```

Output:


This is the text in first paragraph. This is the text in first paragraph. This is the text in first paragraph.

This is the text in second paragraph. This is the text in second paragraph. This is the text in second paragraph.

This is another paragraph, contains some text to test the paragraph interlining, paragraph indentation and some other features. Also, is easy to see how new paragraphs are defined by simply entering a double blank space.

Figure 10. Example output of indentation

5.3. Paragraph spacing: The length parameter that characterizes the paragraph spacing is `\parskip`, this determines the space between a paragraph and the preceding text.

```
    \setlength{\parindent}{4em} % Define this in preamble
    \setlength{\parskip}{1em}
\begin{document}
```

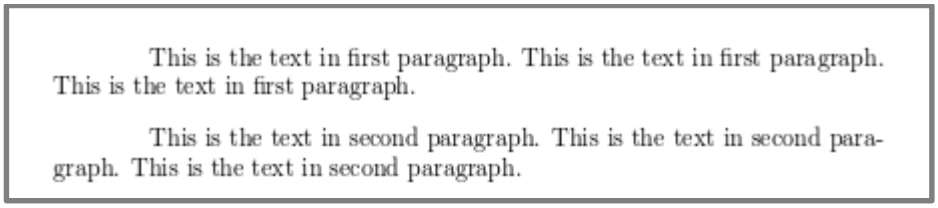
This is the text in first paragraph. This is the text in first

paragraph. This is the text in first paragraph. \par

This is the text in second paragraph...

\end{document}

Output:


This is the text in first paragraph. This is the text in first paragraph. This is the text in first paragraph.

This is the text in second paragraph. This is the text in second paragraph. This is the text in second paragraph.

Figure 11. Space in paragraphs

5.4. Line spacing: The line space is defined by three commands \baselinestretch, \setlength{\baselineskip}{value}, and \linespread{value} command. An example for the \baselinestretch is illustrated below.

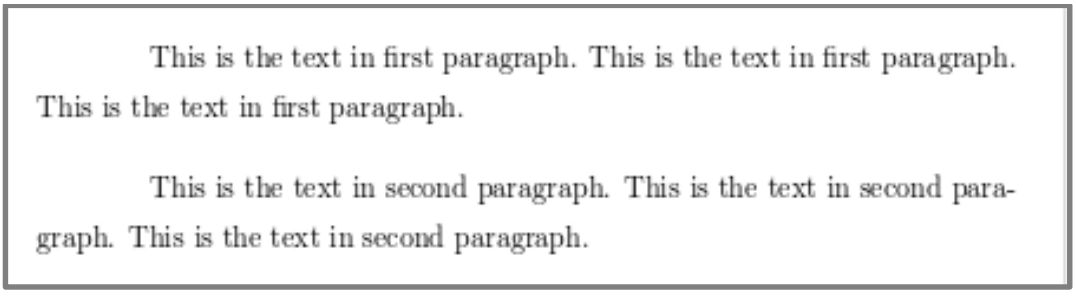
\renewcommand{\baselinestretch}{1.5} % define this in preamble

\begin{document}

This is the text in first paragraph. This is the text in first paragraph. This is the text in first paragraph. \par
This is the text in second paragraph...

\end{document}

Output:


This is the text in first paragraph. This is the text in first paragraph. This is the text in first paragraph.

This is the text in second paragraph. This is the text in second paragraph. This is the text in second paragraph.

Figure 12. Illustration the line space

For the other two command, use their specific commands in the preamble. A list of spacing value with the other two commands is given below.

Value	Line spacing
1.0	single spacing
1.3	one-and-a-half spacing
1.6	double spacing

Figure 13. Different line space for the value used in the other two commands

Table 1. Horizontal and vertical space commands

Command	Type	Description
`\hspace{1cm}`	Horizontal space	1 cm horizontal space
`\hfill`		Inserts a blank space that will stretch accordingly to fill the space available.
`\vspace{5mm}`	Vertical space	5mm vertical space
`\vfill`		fill the vertical space available

Name	Command	Example
default space		*abc* →←*abc*
thin space	`\,`	*abc* →←*abc*
thin neg. space	`\!`	*abc* →←*abc*
medium space	`\:`	*abc* →←*abc*
large space	`\;`	*abc* →←*abc*
0.5em space	`\enspace`	*abc* →←*abc*
1em space	`\quad`	*abc* →←*abc*
2em space	`\qquad`	*abc* →←*abc*
custom space	`\hspace{3em}`	*abc* →←*abc*
fill empty space	`\hfill`	*abc* →···

Created by http://texblog.org

Figure 14. Additional commands for spacing

5.5. **Line break:** The following three commands are used for the line break.

`\\` (two backslashes)

`\newline`

`\hfill \break`

Example code

```
\begin{document}
Something in this document. This paragraph contains no information
and its purposes is to provide an example on how to insert white
spaces and lines breaks.\\
When a line break is inserted, the text is not indented, there
are a couple of extra commands do line breaks. \newline
This paragraph provides no information whatsoever. We are exploring
line breaks. \hfill \break
And combining two commands

\end{document}
```

Output:

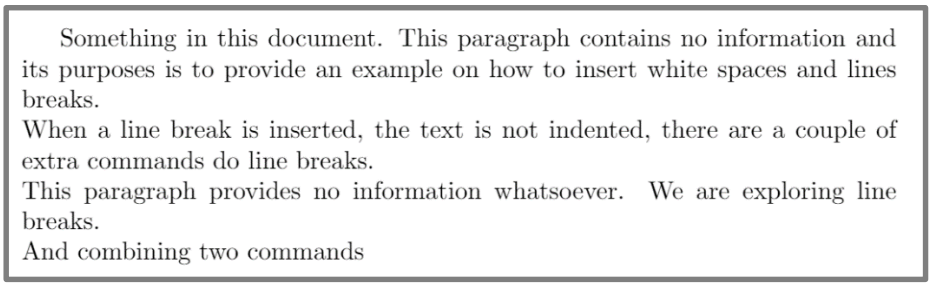

Something in this document. This paragraph contains no information and its purposes is to provide an example on how to insert white spaces and lines breaks.
When a line break is inserted, the text is not indented, there are a couple of extra commands do line breaks.
This paragraph provides no information whatsoever. We are exploring line breaks.
And combining two commands

Figure 15. Line break example using three different commands

5.6. **Page break:** The `\newpage` command is used for a page break. Use the `\newpage` command and test the result.

6. **Fonts effects:** The font effects such as ***italics***, ***bold***, ***underlined,*** or ***color*** words highlight the main concept and can change the perception of the reader. An example of such effect is given in the following.

Please note the color command requires the `\usepackage{xcolor}`

These are `\textit`{words in italics}.`\\`

These are also `\emph`{words in italics}.`\\`

These are `\textbf`{words in bold}.`\\`

```
These are \textsf{sans serif words}.\\
These are \textrm{roman words}.\\
These are \underline{underlined words}.\\
These are \textbf{\textit{words in bold and italics}}.\\
These are {\color{red}red coloured words}.\\
These are \textbf{\textcolor{blue}{blue coloured words}}.\\
These are \uppercase{words in capital letter}.\\
These are \MakeUppercase{{words in capital letter}}.\\
These are \lowercase{WORDS IN \textbf{SMALL} LETTER}.\\
These are also \MakeLowercase{WORDS IN \textbf{SMALL} LETTER}.
```

Output:

These are *words in italics*.
These are also *words in italics*.
These are **words in bold**.
These are sans serif words.
These are roman words.
These are underlined words.
These are ***words in bold and italics***.
These are red coloured words.
These are **blue coloured words**.
These are WORDS IN CAPITAL LETTER.
These are also WORDS IN CAPITAL LETTER.
These are words in **small** letter.
These are also words in **small** letter.

Figure 16. Example output of different font effects

Some more font effects:

Command	Type Size
{**tiny** size}	text
{**scriptsize** size}	text
{**footnotesize** size}	text
{**small** size}	text
{**normalsize** size}	text
{**large** size}	text
{**Large** size}	text
{**LARGE** size}	text
{**huge** size}	text

7. **Lists and their different types:** Two types of list (i.e., unordered and ordered lists) are commonly used in overleaf. The lists are defined through the `\begin{...}` command and end with the `\end{...}` command with the environment variables `itemize` and `enumerate` for the unordered and ordered lists.

   7.1. Unordered list: Following code is used for the unordered list

```
Below is an example code of the \textbf{\textit{unordered list}}
\begin{itemize}
  \item List entries start with the command.
  \item Individual entries are indicated with a black dot, a so-called bullet.
  \item The text in the entries may be of any length.
\end{itemize}
```

Output:

Below is an example code of the ***unordered list***

- List entries start with the command.
- Individual entries are indicated with a black dot, a so-called bullet.
- The text in the entries may be of any length.

Figure 17. Unordered list (example)

7.2. Ordered list: Example code for ordered list is given below.

```
Numbered (ordered) lists are easy to create:
\begin{enumerate}
  \item Items are numbered automatically.
  \item The numbers start at 1 with each use of the \texttt{enumerate}
        environment.
  \item Another entry in the list
\end{enumerate}
```

Output:

Numbered (ordered) lists are easy to create:

1. Items are numbered automatically.
2. The numbers start at 1 with each use of the `enumerate` environment.
3. Another entry in the list

Figure 18. Ordered list (example)

7.3. Nested list: The following code illustrates an example of nested ordered and unordered lists.

<table>
<tr><th>Code</th><th>Output</th></tr>
<tr><td>

```
\begin{enumerate}
    \item Level 1
    \begin{enumerate}
        \item Level 2
        \item Level 2
        \item Level 2
    \end{enumerate}
    \item Level 1
    \item Level 1
\end{enumerate}
```

</td><td>

1. Level 1
    (a) Level 2
    (b) Level 2
    (c) Level 2
2. Level 1
3. Level 1

</td></tr>
<tr><td>

```
\begin{enumerate}
    \item Level 1
    \begin{itemize}
        \item Level 2
        \item Level 2
        \item Level 2
    \end{itemize}
    \item Level 1
    \item Level 1
\end{enumerate}
```

</td><td>

1. Level 1
    - Level 2
    - Level 2
    - Level 2
2. Level 1
3. Level 1

</td></tr>
<tr><td>

```
\item Level 1
   \begin{enumerate}
       \item [*]Level 2
       \item [*]Level 2
       \item [!]Level 2
   \end{enumerate}
   \item Level 1
   \item Level 1
\end{itemize}
```

</td><td>

- Level 1
    * Level 2
    * Level 2
    ! Level 2
- Level 1
- Level 1

</td></tr>
</table>

Latex Lecture-4

1) Equations
2) Maths symbols
3) Subscript and superscript
4) Brackets and Parentheses
5) Fractions
6) Managing long equations
7) List of operators
8) Verbatim
9) Tables
10) Figures
11) Algorithms (Pseudocode)
12) Bibliography

1. **Equations:** Two writing modes for mathematical expressions: the ***inline math mode*** and ***display math mode***.
   - ***Example code:*** A basic example for both modes is given in the following code.

```
\documentclass{article}
\begin{document}

The well known Pythagorean theorem \(x^2 + y^2 = z^2\) was
proved to be invalid for other exponents.
Meaning the next equation has no integer solutions:
\[ x^n + y^n = z^n \]

\end{document}
```

Output:

The well known Pythagorean theorem $x^2 + y^2 = z^2$ was proved to be invalid for other exponents. Meaning the next equation has no integer solutions:

$$x^n + y^n = z^n$$

Figure 1. Illustration of basic equation

1.1. Inline math mode:

- Inline math mode is used to write formulas that are ***part of a paragraph.***
- You can use any of these ***"delimiters"*** to typeset your math in inline mode:

  - `\(...\)`

- `$...$`
- `\begin{math}...\end{math}.`

Example:

Command	**Use**
`\(...\)`	In physics, the mass-energy equivalence is stated by the equation `\(E=mc^2\),` discovered in 1905 by Albert Einstein.
`$...$`	In physics, the mass-energy equivalence is stated by the equation `$E=mc^2$`, discovered in 1905 by Albert Einstein.
`\begin{math}...\end{math}`	In physics, the mass-energy equivalence is stated by the equation `\begin{math}E=mc^2\end{math}`, discovered in 1905 by Albert Einstein.
Output: All the three commands results in the similar output	
In physics, the mass-energy equivalence is stated by the equation $E = mc^2$, discovered in 1905 by Albert Einstein. **Figure 2.** Example output of inline equations	

1.2. Display math mode:

- display math mode is used to write expressions that are ***not part of a paragraph***, and are therefore put on separate lines
- The displayed mode has two versions – ***numbered and un-numbered***.
- Many math mode commands require the module `\usepackage{amsmath}` in the preamble.
- Use one of these constructions to typeset math in display mode:
  - `\[...\]`
  - `\begin{displaymath}...\end{displaymath}`
  - `\begin{equation}...\end{equation}`

Example	
Command	**Use**
- `\[...\]`	The mass-energy equivalence is described by the famous equation discovered in 1905 by Albert Einstein.

	`\[E=mc^2\]  % 1st example of unnumbered equation`

`\begin{displaymath}`
 	
`...`
 	
`\end{displaymath}`
	The mass-energy equivalence is described by the famous equation discovered in 1905 by Albert Einstein.
	
	`\begin{displaymath}`
	`E=m    % 2nd Example of unnumbered equation`
	`\end{displaymath}`

`\begin{equation}`
 	
`...`
 	
`\end{equation}`
	The mass-energy equivalence is described by the famous equation (1) discovered in 1905 by Albert Einstein.
	
	`\begin{equation}`
	`E=m        % Example of numbered equation`
	`\end{equation}`

Output:

The mass-energy equivalence is described by the famous equation discovered in 1905 by Albert Einstein.

$$E = mc^2$$

The mass-energy equivalence is described by the famous equation discovered in 1905 by Albert Einstein.

$$E = m$$

The mass-energy equivalence is described by the famous equation (1) discovered in 1905 by Albert Einstein.

$$E = m \tag{1}$$

Figure 3. Example output of display math mode equations

2. **Math symbols:**

- Below is some common maths symbols shown in Figure-4.
- For detailed symbols please check the following links
  - https://www.overleaf.com/learn/latex/List_of_Greek_letters_and_math_symbols
  - https://www.math.uci.edu/~xiangwen/pdf/LaTeX-Math-Symbols.pdf

description	code	examples
Greek letters	`\alpha \beta \gamma \rho \sigma \delta \epsilon`	$\alpha\ \beta\ \gamma\ \rho\ \sigma\ \delta\ \epsilon$
Binary operators	`\times \otimes \oplus \cup \cap`	$\times\ \otimes\ \oplus\ \cup\ \cap$
Relation operators	`< > \subset \supset \subseteq \supseteq`	$<\ >\ \subset\ \supset\ \subseteq\ \supseteq$
Others	`\int \oint \sum \prod`	$\int\ \oint\ \sum\ \prod$

Figure 4. Common math symbols

3. **Subscript and superscript:** The symbols _ and ^ are used for defining the subscripts and superscripts, as given in the following example code.

```
\[ a_1^2 + a_2^2 = a_3^2 \]

\[ \sum_{i=1}^{\infty} \frac{1}{n^s}
= \prod_p \frac{1}{1 - p^{-s}} \]
```

Output:

$$a_1^2 + a_2^2 = a_3^2$$

$$\sum_{i=1}^{\infty} \frac{1}{n^s} = \prod_p \frac{1}{1 - p^{-s}}$$

Figure 5. Output for superscript and subscript

List of some subscript and superscript

LaTeX markup	Renders as
a_{n_i}	a_{n_i}
\int_{i=1}^n	$\int_{i=1}^{n}$
\sum_{i=1}^{\infty}	$\sum_{i=1}^{\infty}$
\prod_{i=1}^n	$\prod_{i=1}^{n}$
\cup_{i=1}^n	$\cup_{i=1}^{n}$
\cap_{i=1}^n	$\cap_{i=1}^{n}$
\oint_{i=1}^n	$\oint_{i=1}^{n}$
\coprod_{i=1}^n	$\coprod_{i=1}^{n}$

Figure 6. List of different subscript and superscript

4. **Brackets and Parentheses:**
The brackets and parentheses can be manually set, where the size will adjust accordingly.

```
\[
F = G \left(\frac{m_1 m_2}{r^2} \right)
\]
```

Output:

$$F = G\left(\frac{m_1 m_2}{r^2}\right)$$

Figure 7. Example output for parenthesis

A list of different parenthesis is given in the following figure

Type	LaTeX markup	Renders as
Parentheses; round brackets	`(x+y)`	$(x+y)$
Brackets; square brackets	`[x+y]`	$[x+y]$
Braces; curly brackets	`\{ x+y \}`	$\{x+y\}$
Angle brackets	`\langle x+y \rangle`	$\langle x+y \rangle$
Pipes; vertical bars	`|x+y|`	$\|x+y\|$
Double pipes	`\\|x+y\\|`	$\\|x+y\\|$

Figure 8. List of different types of parenthesis

Commands for the size of different types of parenthesis

LaTeX markup	Renders as
`\big( \Big( \bigg( \Bigg(`	$\big(\Big(\bigg(\Bigg($
`\big] \Big] \bigg] \Bigg]`	$\big] \Big] \bigg] \Bigg]$
`\big\{ \Big\{ \bigg\{ \Bigg\{`	$\big\{ \Big\{ \bigg\{ \Bigg\{$
`\big \langle \Big \langle \bigg \langle \Bigg \langle`	$\big \langle \Big \langle \bigg \langle \Bigg \langle$
`\big \rangle \Big \rangle \bigg \rangle \Bigg \rangle`	$\big \rangle \Big \rangle \bigg \rangle \Bigg \rangle$
`\big| \Big| \bigg| \Bigg|`	$\big\| \Big\| \bigg\| \Bigg\|$
`\big\\| \Big\\| \bigg\\| \Bigg\\|`	$\big\\| \Big\\| \bigg\\| \Bigg\\|$
`\big \lceil \Big \lceil \bigg \lceil \Bigg \lceil`	$\big \lceil \Big \lceil \bigg \lceil \Bigg \lceil$
`\big \rceil \Big \rceil \bigg \rceil \Bigg \rceil`	$\big \rceil \Big \rceil \bigg \rceil \Bigg \rceil$
`\big \lfloor \Big \lfloor \bigg \lfloor \Bigg \lfloor`	$\big \lfloor \Big \lfloor \bigg \lfloor \Bigg \lfloor$
`\big \rfloor \Big \rfloor \bigg \rfloor \Bigg \rfloor`	$\big \rfloor \Big \rfloor \bigg \rfloor \Bigg \rfloor$

Figure 9. Illustration of parenthesis size

5. **Fractions:** The appearance of the fraction may change depending on the context, example code is and output is illustrated.

Fractions can be used alongside the text, for example `\( \frac{1}{2} \),` and in a mathematical display style like the one below:

```
\[\frac{1}{2}\]
```

Output:

Fractions can be used alongside the text, for example $\frac{1}{2}$, and in a mathematical display style like the one below:

$$\frac{1}{2}$$

Figure 10. Example of fraction

6. **Managing long equations:** There are multiple ways to manage long equations.
   - Displaying long equations
   - Splitting and aligning

- Displaying long equations

```
\begin{multline*}

p(x) = 3x^6 + 14x^5y + 590x^4y^2 +
19x^3y^3\\

- 12x^2y^4 - 12xy^5 + 2y^6 - a^3b^3

\end{multline*}
```

Output:

$$p(x) = 3x^6 + 14x^5y + 590x^4y^2 + 19x^3y^3 - 12x^2y^4 - 12xy^5 + 2y^6 - a^3b^3$$

- Splitting and aligning

```
\begin{align*}
2x - 5y &=  8 \\
3x + 9y &=  -12
\end{align*}
```

Output

$$\begin{aligned} 2x - 5y &= 8 \\ 3x + 9y &= -12 \end{aligned}$$

7. **List of operators**

Operator	Renders as
\cos	cos
\csc	csc
\exp	exp
\ker	ker
\limsup	lim sup
\min	min
\sinh	sinh
\arcsin	arcsin
\cosh	cosh
\hom	hom
\lim	lim
\log	log
\sec	sec
\tan	tan
\arg	arg
\coth	coth
\dim	dim
\liminf	lim inf
\max	max
\sin	sin
\tanh	tanh
\deg	deg
\gcd	gcd
\lg	lg
\ln	ln
\Pr	Pr
\sup	sup
\arctan	arctan
\cot	cot
\det	det

8. **Verbatim:** The `\begin{verbatim}` command prints text in monospaced font and prints spaces, tabs, etc. verbatim. This can be used for displaying code snippets:

```
\begin{verbatim}

#include
Int main()
    {
        std::cout << "Hello, world";
        return 0;
    }

\end{verbatim}
```

Output:

```
#include
Int main()
    {
        std::cout << "Hello, world";
        return 0;
    }
```

Figure 11. Example output for illustrating verbatim

9. **Tables:** The tabular command is used to typeset tables. This is an example of a table and referral in the text to the table.

```
\begin{table}[h!]
\centering                          % centre the table on the page
\begin{tabular}{|c r l|}            % c = cjt column, l = ljt column, etc.
\hline                              % insert a horizontal line
Col1 & Col2 & Col2 \\
\hline
A1 & B1 & C1 \\                     % A1, A2, etc. are data
A2 & B2 & C2 \\
A3 & B3 & C3 \\
\hline
\end{tabular}
\caption{Table Caption}
\label{table:example}
\end{table}

Table \ref{table:example} is an example of referred table.
```

Output:

Col1	Col2	Col2
A1	B1	C1
A2	B2	C2
A3	B3	C3

Table 1: Table Caption

Table 1 is an example of referred table.

Figure 12. Illustration of Table

Alternative option: An easy way is to manage the table using the following online tool. https://www.tablesgenerator.com/

10. **Figures:** Figures can be included and automatically ***numbered sequentially*** and ***centre justified***. Figures are stored ***outside the main.tex file***. Images should be PDF, PNG, JPEG or GIF files. The `\usepackage{graphicx}` needs to be called within the preamble:

```
\usepackage{graphicx}

%%%%%%%%%

Insert the figure here:

\begin{figure}[h]                    % Insert the figure about here
\centering                           % This centres the image
\includegraphics[scale=2]{myimage}
\caption{My Image}
\label{fig:myimage}                  % Figures to be referred to in the text
\end{figure}
```

Figures are cited within the body text as follows:

Refer to figure `\ref{fig:myimage}` above and on the following page `\pageref{fig:myimage}`

If figures are stored in a folder named "figures", then to refer to the figure using the following command.

```
\ref(figures/fig:myimage)
```

`\includegraphics[scale=2]{myimage}` has several options such as:

```
\includegraphics[width=\linetwidth] myimage} % adjust according to linewidth

\includegraphics[width=\textwidth] myimage} % adjust according to textwidth

\includegraphics[width=0.5\textwidth] myimage} % 50% of textwidth
```

11. **Algorithms (Pseudocode):**
    - The algorithm/pseudocode is an abstract level of representation for the computer program.
    - Usually, the algorithm/pseudocode defines the main concept of the computer program.
    - Latex uses the `\usepackage{algorithm}` for writing the algorithm/pseudocode

```
\documentclass{article}
\usepackage{algorithm}
\usepackage{algpseudocode}
\begin{document}
\begin{algorithm}
\caption{An algorithm with caption}\label{alg:cap}
\begin{algorithmic}
\Require $n \geq 0$
\Ensure $y = x^n$
\State $y \gets 1$
\State $X \gets x$
\State $N \gets n$
\While{$N \neq 0$}
\If{$N$ is even}
    \State $X \gets X \times X$
    \State $N \gets \frac{N}{2}$  \Comment{This is a comment}
\ElsIf{$N$ is odd}
    \State $y \gets y \times X$
    \State $N \gets N - 1$
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

\end{document}
```

Output:

Algorithm 1 An algorithm with caption

Require: $n \geq 0$

Ensure: $y = x^n$

$y \leftarrow 1$
$X \leftarrow x$
$N \leftarrow n$
while $N \neq 0$ **do**
  if N is even **then**
    $X \leftarrow X \times X$
    $N \leftarrow \frac{N}{2}$ ▷ This is a comment
  else if N is odd **then**
    $y \leftarrow y \times X$
    $N \leftarrow N - 1$
  end if
end while

A detailed about different styles of algorithms is available at:

https://www.overleaf.com/learn/latex/Algorithms

12. **Bibliography:**

- References are includes using the major bibliography management programs (packages): ***bibtex, biblatex***, and ***natbib***.
- You will also need the **\usepacke{cite}**
- Bibliography is a list of references cited throughout the text with a References list placed at the end of the report.
- Bibliographies can be embedding at the end of your document as follows:

```
\begin{thebibliography}{9}

\bibitem{latexcompanion}
Michel Goossens, Frank Mittelbach, and Alexander Samarin.
\textit{The \LaTeX\ Companion}.
Addison-Wesley, Reading, Massachusetts, 1993.

\bibitem{knuthwebsite}
Knuth: Computers and Typesetting,
\\\texttt{http://www-cs-faculty.stanford.edu.html}

\bibitem{b1} G. O. Young, ``Synthetic structure of industrial
plastics,'' in \emph{Plastics,} 2\textsuperscript{nd} ed., vol. 3, J.
Peters, Ed. New York, NY, USA: McGraw-Hill, 1964, pp. 15--64.

\bibitem{b2} W.-K. Chen, \emph{Linear Networks and Systems.} Belmont,
CA, USA: Wadsworth, 1993, pp. 123--135.

\bibitem{b3} J. U. Duncombe, ``Infrared navigation---Part I: An
assessment of feasibility,'' \emph{IEEE Trans. Electron Devices},
vol. ED-11, no. 1, pp. 34--39, Jan. 1959, 10.1109/TED.2016.2628402.

\end{thebibliography}
```

References are cited in text as follows:

```
This is a reference \cite{knuthwebsite} cited in the text
```

12.1. Biblatex package:

- Use package `\usepackage{biblatex}`
- You will also need the `\usepacke{cite}`
- Add the bib resource file `\addbibresource{file-name.bib}`
- Before end of the `\end{document}` command use the `\printbiblography`

Example:

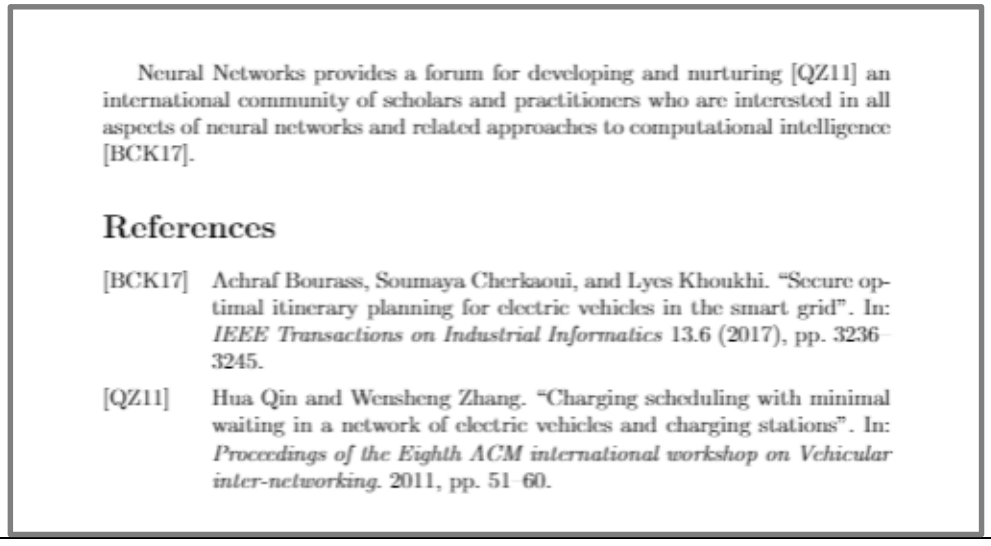
Main.tex

```
\documentclass{article}
\usepackage{cite}
\usepackage[style=alphabetic]
{biblatex}
\addbibresource{CT1112bib.bib}

%----------preamble------------

\begin{document}

Neural Networks provides a forum for developing and nurturing \cite{
qin2011charging} an international community of scholars and practitioners who are interested in all aspects of neural networks and related approaches to computational intelligence \cite{bourass2017secure}.

\printbibliography
\end{document}
```

CT1112bib.bib

```
@inproceedings{qin2011charging,
  title={Charging scheduling with minimal waiting in a network of electric vehicles and charging stations},
  author={Qin, Hua and Zhang, Wensheng},
  booktitle={Proceedings of the Eighth ACM international workshop on Vehicular inter-networking},
  pages={51--60},
  year={2011}
}

@article{bourass2017secure,
  title={Secure optimal itinerary planning for electric vehicles in the smart grid},
  author={Bourass, Achraf and Cherkaoui, Soumaya and Khoukhi, Lyes},
  journal={IEEE Transactions on Industrial Informatics},
  volume={13},
  number={6},
  pages={3236--3245},
  year={2017},
  publisher={IEEE}
}
```

Output:



Neural Networks provides a forum for developing and nurturing [QZ11] an international community of scholars and practitioners who are interested in all aspects of neural networks and related approaches to computational intelligence [BCK17].

References

[BCK17] Achraf Bourass, Soumaya Cherkaoui, and Lyes Khoukhi. "Secure optimal itinerary planning for electric vehicles in the smart grid". In: *IEEE Transactions on Industrial Informatics* 13.6 (2017), pp. 3236–3245.

[QZ11] Hua Qin and Wensheng Zhang. "Charging scheduling with minimal waiting in a network of electric vehicles and charging stations". In: *Proceedings of the Eighth ACM international workshop on Vehicular inter-networking*. 2011, pp. 51–60.

12.2. natbib package:

- Use package `\usepackage{natbib}`

- You will also need the `\usepacke{cite}`
- Use the style `\bibliographystyle{style name}`
- Before end of the `\end{document}` command use the `\bibliography{file-name.bib}`

Example:

main.tex

```
\documentclass{article}
\usepackage{cite}
\usepackage{natbib}
\bibliographystyle{alpha} % style
\begin{document}
Neural Networks provides a forum for developing and nurturing \cite{bourass2017secure} an international community of scholars and practitioners who are interested in all aspects of neural networks and related approaches to computational intelligence \cite{qin2011charging}.
\bibliography{CT1112bib.bib}
\end{document}
```

CT1112.bib

```
@inproceedings{qin2011charging,
  title={Charging scheduling with minimal waiting in a network of electric vehicles and charging stations},
  author={Qin, Hua and Zhang, Wensheng},
  booktitle={Proceedings of the Eighth ACM international workshop on Vehicular inter-networking},
  pages={51--60},
  year={2011}
}

@article{bourass2017secure,
  title={Secure optimal itinerary planning for electric vehicles in the smart grid},
  author={Bourass, Achraf and Cherkaoui, Soumaya and Khoukhi, Lyes},
  journal={IEEE Transactions on Industrial Informatics},
  volume={13},
  number={6},
  pages={3236--3245},
  year={2017},
  publisher={IEEE}
}
```

Output:

Neural Networks provides a forum for developing and nurturing [BCK17] an international community of scholars and practitioners who are interested in all aspects of neural networks and related approaches to computational intelligence [QZ11].

References

[BCK17] Achraf Bourass, Soumaya Cherkaoui, and Lyes Khoukhi. Secure optimal itinerary planning for electric vehicles in the smart grid. *IEEE Transactions on Industrial Informatics*, 13(6):3236–3245, 2017.

[QZ11] Hua Qin and Wensheng Zhang. Charging scheduling with minimal waiting in a network of electric vehicles and charging stations. In *Proceedings of the Eighth ACM international workshop on Vehicular inter-networking*, pages 51–60, 2011.

Available styles: `dinat, plainnat, abbrvnat, unsrtnat, rusnat, ksfh_nat`

Python Lecture-5

1) Introduction to programming
2) Installation
3) Python IDE (Anaconda and Juypter Notbook)
4) Python Could based IDE (Google Colab)
5) Writing program
6) Basic keywords
7) Print function
8) Comments
9) Variables and datatypes
10) Concatenation and addition
11) String manipulation
12) Input function
13) Format function

1. Introduction

- When we type a letter written in plain English, into say ***MS Word*** on the Personal Computer, the application is designed to show us ***exactly what we type on the computer screen***.
- To achieve this, MS Word was written in the ***programming language, C++*** and designed to accept our keyboard characters and display them on the screen verbatim.
- In doing this, the C++ programme converts our ***keyboard characters into machine code*** that is then manipulated by the CPU.
- There are hundreds of high level programming languages, like C++.
- They have all been designed around ***human-readable syntax***.
- Listed below are some of the more common programming languages.

Language	Description
C/C++	General-purpose language
C#	Microsoft .NET platform
Java	Object-oriented language
JavaScript	Scripting language for web browsers and unrelated to Java
Python	General purpose language with readable code
PHP	Server side scripting and web app development
SQL	Relational database requests
Ruby	Web apps that use the Rails framework
NODE.js	JavaScript runtime environment for outside a browser
R	Like mathlab and used for queries
LaTeX	High level language for typesetting documents
Swift	Language for Apple applications
Excel	Spreadsheet application for data analytics

- Computer programmers typically learn a number of programming languages e.g. Java, JavaScript, Python, C++, LaTeX and SQL.
- In addition, Programmers who specialise in web applications will also learn HTML and CSS.
- ***Python and JavaScript have fewer syntax requirements (fewer lines of code) and are good starting points for learning the logic of programming before progressing to other languages.***

- LaTeX is a programming language used for formatting and layout of reports, papers and mathematical expressions.
- Examples of programming language syntax for four common languages are given below for the function 'print the characters Hello World! onto the computer screen'.

- **Python**

```
print('Hello, world!')
```

- **LaTeX**

```
\title{Hello world!}
```

- **C++**

```
#include
Int main()
{
std::cout << "Hello, world";
return 0;
}
```

- **JavaScript**

```
document.write ('Hello, world!');
```

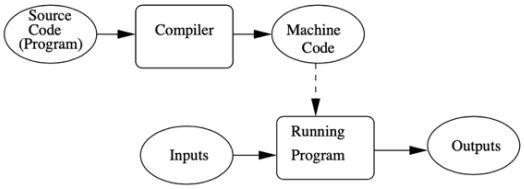
1.1. Writing Programs

- Simple text editors can be used to create programs or source code e.g. ***TextEdit (Mac)*** and ***Notepad (PC)***. Applications like ***Word are unsuitable*** because of ***hidden formatting*** that may later cause syntax errors.
- ***Python*** text files are saved with the ***.py*** file extension. ***Javascript***, for example, uses ***.js*** and so on.
- ***Integrated Development Environments (IDE)*** are preferred by programmers over simple text files for creating programs since they provide special features such as ***syntax highlighting (e.g. print('Hello, world!')*** and access to libraries of reusable source code called modules or packages. Common IDEs include:

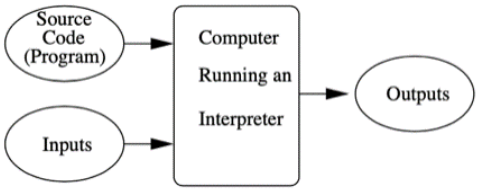
  - XCode (Apple Mac and iPhone languages)
  - Visual Studio Code (Most languages)
  - Android Studio (Android applications)
  - RubyMine (Ruby language)
  - PyCharm (Python language)

- Running a program is achieved by either ***compiling the program into an executable (.exe) file or interpreting the source code*** in a text file (e.g. game.py) during computer run-time.
- Increasingly, programs utilise both compiled and interpreted elements.
- ***Common compiler languages*** include C/C++. Common interpreter languages include ***Python and JavaScript***.
- Languages such as ***Java, C# and Python*** can use a ***combination of both***.
- Interpreted languages can save time during program development because no compilation and linking is necessary.

- **Compiler process**



- **Interpreter process**

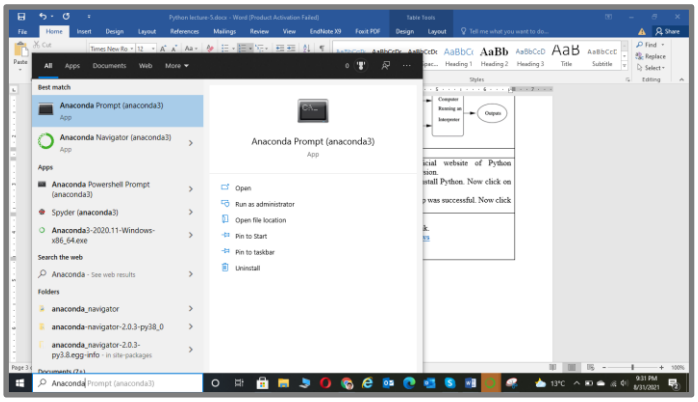


2. **Installation**

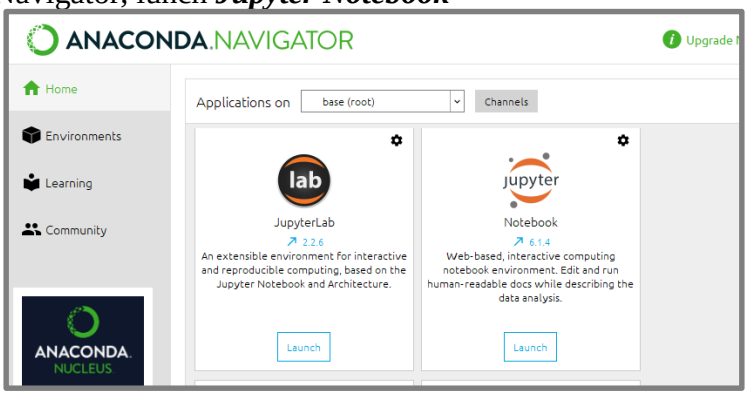
1. To download and install Python, visit the official website of Python https://www.python.org/downloads/ and choose your version.
2. Once the download is completed, run the .exe file to install Python. Now click on Install Now.
3. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

3. **IDE Installation:**

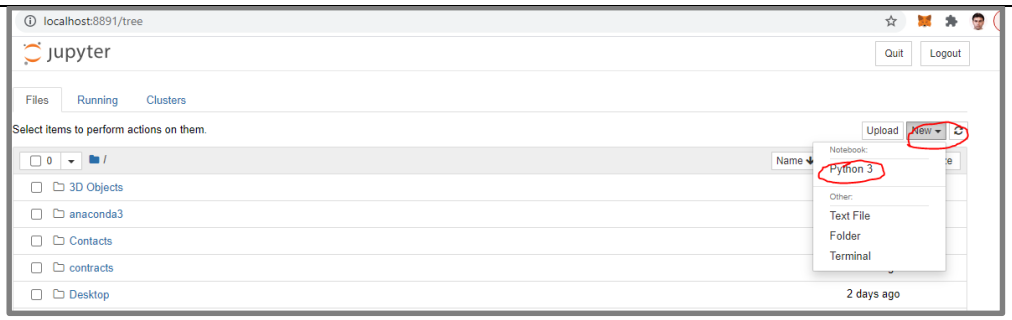
1. Download ***Anaconda*** and install using the following link. https://www.anaconda.com/products/individual#windows
2. Open/run Anaconda



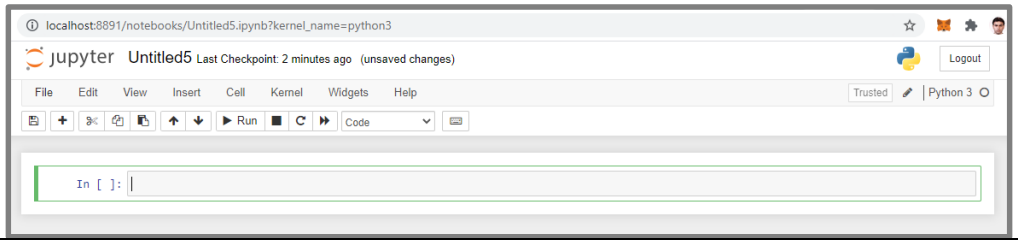
3. From Anaconda Navigator, lunch ***Jupyter Notebook***



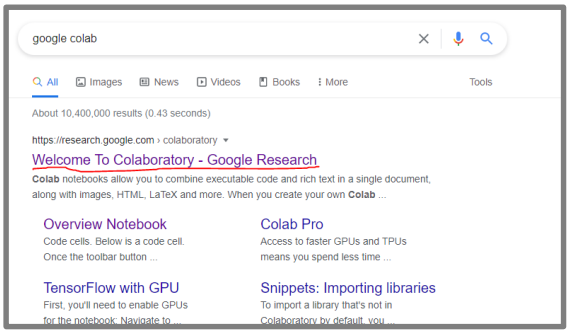
4. Once Jupyter Notebook lunched, click on ***new and python 3***



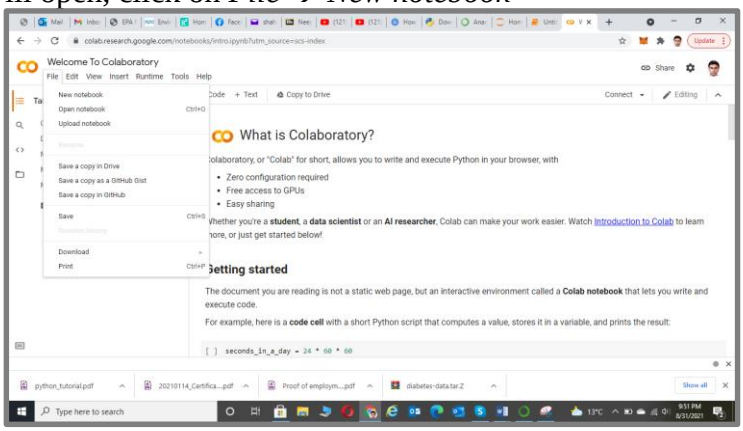
5. Python is now ready



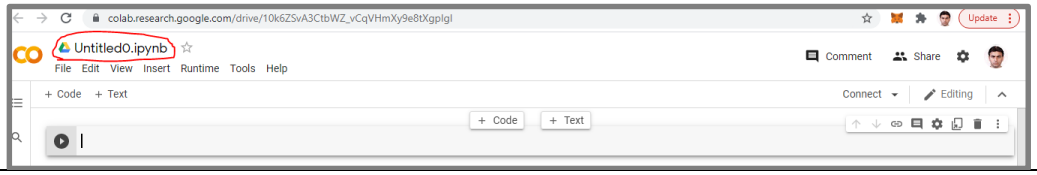
4. **Could base IDE**
   - Google colab provides a cloud base IDE for python
   - Search the ***Google Colab,*** make sure you are logged in with yours google account



   - The google colab will open, click on *File* → *New notebook*

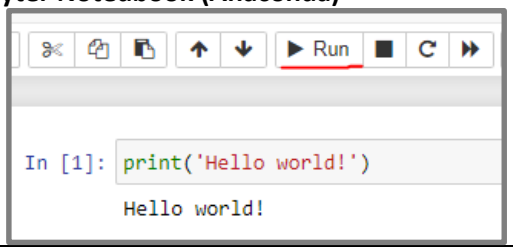


   - A cloud base Jupyter will open

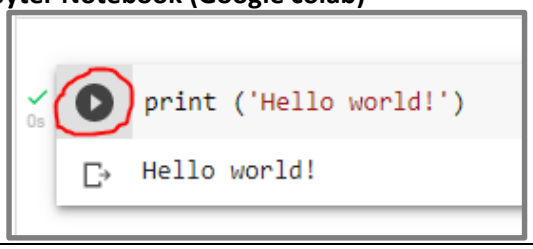


5. **Writing program**
   - Write the following code and test the output.

Jupyter Notedbook (Anaconda)

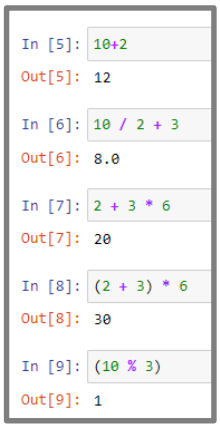


Jupyter Notebook (Google colab)




```
>>> 10 + 2
>>> 10 / 2 + 3
>>> 2 + 3 * 6
>>> (2 + 3) * 6
>>> 10 % 3
```

Jupyter Notedbook (Anaconda)



Jupyter Notebook (Google colab)

What will happen if

```
In [4]: 10+2
        10 / 2 + 3
        2 + 3 * 6
        (2 + 3) * 6
        (10 % 3)
```

?

```
In [5]: print(10+2)
        print(10 / 2 + 3)
        print(2 + 3 * 6)
        print((2 + 3) * 6)
        print(10 % 3)
```

?

6. **Python Keywords**

- Keywords are python recovered words for designated tasks and thus can't be used for variable or functions. Following is a list of python keywords

  Value: True, False, None
  Operator: and, or, not, in, is
  Control Flow: if, elif, else
  Iteration: for, while, break, continue, else
  Structure: def, class, with, as, pass, lambda
  Returning: return, yield
  Import: import, from, as
  Exception-Handling: try, except, raise, finally, else, assert
  Asynchronous Programming: async, await
  Variable Handling: del, global, nonlocal

7. **Print function**
   - The ***print()*** function outputs the ***specified message*** to the screen or other output device.
   - The message can be a ***string, or other data type (e.g. integer, float or Boolean).***
   - Functions contain ***parenthesis ()*** to ***store parameters*** and ***arguments*** (e.g. strings).
   - Below, the argument ***"Hello World!"*** is passed into the print function for display on the terminal screen.

```
print("Hello world!")

print(" _________________ ")
print("|                 |")
print("| Hello World!  |")
print("|                 |")
print("|_________________|")

print('Hello World!')

print('Hello world "again"')

print("Hello \nWorld") # \n moves the word World to a new line
```

Note:

- Anything included in ***single or double quotes*** is considered as ***string*** (i.e., the data type is string). Here in this example the arguments ***Hello world*** is of string data type.
- The hash sing (#) represents the comments.

Output:

```
In [14]: print("Hello world!")

         print(" _____________ ")
         print("|             |")
         print("| Hello World! |")
         print("|             |")
         print("|_____________|")

         print('Hello World!')

         print('Hello world "again"')

         print("Hello \nWorld")  # \n moves the word World to a new line
```

```
Hello world!
 _____________
|             |
| Hello World! |
|             |
|_____________|
Hello World!
Hello world "again"
Hello
World
```

```
if 5 > 2:                              # if is a python keyword
  print("Five is greater than two!")

print(10 > 9)  # Conditional statement or expression (True/False)
print(10 == 9)
print(10 < 9)
```

Output:

```
In [15]: if 5 > 2:                    # if is a python keyword
           print("Five is greater than two!")

         print(10 > 9)
         print(10 == 9)
         print(10 < 9)
```

```
Five is greater than two!
True
False
False
```

8. **Comments**
   - White space (spaces and empty lines) can be used liberally in most areas throughout the code to provide clarity and emphasis.
   - Comments can be added by using the ***# (hash) or single/double qoutes*** character .
   - The ***compiler ignores*** all characters after the # character.
   - The # character can be used in front of lines of code to temporarily stop these lines of code from executing.
   - Docstrings use triple single or double quotes (''' or """) at the start and end of a block of comments.

Example: Single and multiline comments

```
In [22]: """
         Multiline commends anything between
         these triples double lines will be ignored
         """

         '''
         Another way of multiline commends anything between
         these triples single lines will be ignored
         '''

         print('Hellow world!')   # This single line comment

         Hellow world!
```

9. **Variables and datatypes**
    - A variable is a ***container with a name*** that contains a value. The equal sign below means 'take the value on the right and store it in the variable on the left'. It does not mean that both sides are equal.
    - first_name = **"David"**
      age = 20
      is_male = **True**
    - **"David"** is a type of data called a **string**, 20 is called an integer and **True** is a **Boolean data type**.
    - Other common data types include Floats (e.g. 20.5), Lists, Dictionaries and Classes (objects).
    - Adding variables is called ***concatenation***. Integers and Floats need to be converted to strings, using the str() function, before they can be concatenated.

List of different data types

Example	Data Type	Try it
x = "Hello World"	str	Try it »
x = 20	int	Try it »
x = 20.5	float	Try it »
x = 1j	complex	Try it »
x = ["apple", "banana", "cherry"]	list	Try it »
x = ("apple", "banana", "cherry")	tuple	Try it »
x = range(6)	range	Try it »
x = {"name" : "John", "age" : 36}	dict	Try it »
x = {"apple", "banana", "cherry"}	set	Try it »
x = frozenset({"apple", "banana", "cherry"})	frozenset	Try it »
x = True	bool	Try it »
x = b"Hello"	bytes	Try it »
x = bytearray(5)	bytearray	Try it »
x = memoryview(bytes(5))	memoryview	Try it »

Finding datatypes: Python provides the ***type (arguments)*** *for finding the datatypes*

```
In [27]: x = "Hello World"   # Datatype: string
         print (type(x))

         x = ["apple", "banana", "cherry"] # Datatype: list
         print (type(x))

         x = ("apple", "banana", "cherry") # Datatype: tuple
         print (type(x))

         x = {"apple", "banana", "cherry"} # Datatype: set
         print (type(x))

         <class 'str'>
         <class 'list'>
         <class 'tuple'>
         <class 'set'>
```

10. **Concatenation and addition:** By default, adding variables results in concatenation for ***string types*** and sum for ***integer and float data types***.

```
In [30]: x = "Hello world!."
         y = "This is our first python program."
         print(x+y)  # concatenation

         x1 = 20
         y1= 10
         print(x1+y1) # addition

         print (str(x1)+str(y1)) # Explicit type converstion from int to str and concatination

         Hello world!.This is our first python program.
         30
         2010
```

11. **String manipulation:** To manipulate strings, we can use some of Pythons built-in methods.
    - **Accessing: Use [ ] to access characters in a string**

      Example:

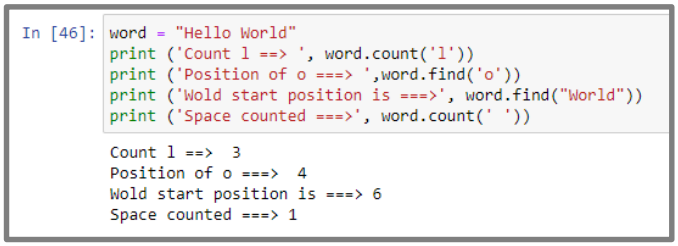
	Output

Accessing: Use ***[ ]*** to access characters in a string
word = "Hello World"
letter=word[0]
print (letter)
	`In [32]: word = "Hello World"`
	`letter=word[0]`
	`print (letter)`
	`H`

Length: Use ***len()*** to find the length (including space) of a string
word = "Hello World"
print (len (word))
	`In [33]: word = "Hello World"`
	`print (len(word))`
	`11`

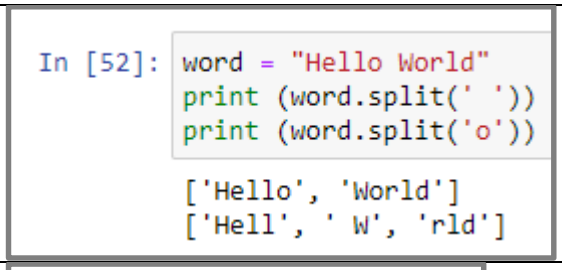
Finding: Use ***count (), find(), and index()*** to find the number of characters & spaces, the character position, and the substring in a given string.

```
word = "Hello World"
print ('Count l ==> ', word.count('l'))
print ('Position of o ===> ',word.find('o'))
print ('Wold start position is ===>',
word.find("World"))
print ('Space counted ===>', word.count(' '))
```



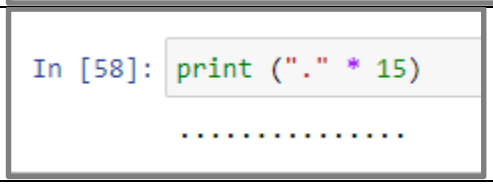
Split Strings: Use the ***split()*** function to split the string accordingly.

```
word = "Hello World"
print (word.split(' '))
print (word.split('o'))
```



Strings: Repeat the number of string.

```
print ("." * 15)
```

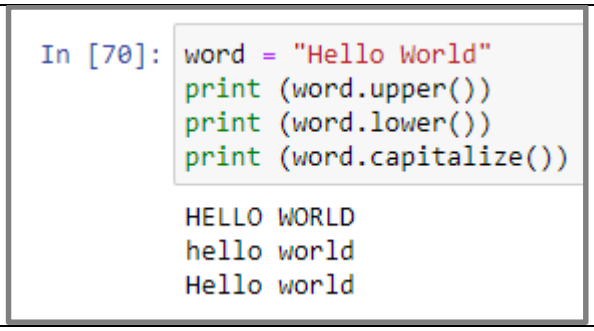


Replacing: Use the ***Replac***e function to replace a word.

```
word = "Hello World"
word.replace("Hello", "Welcome to the")
```

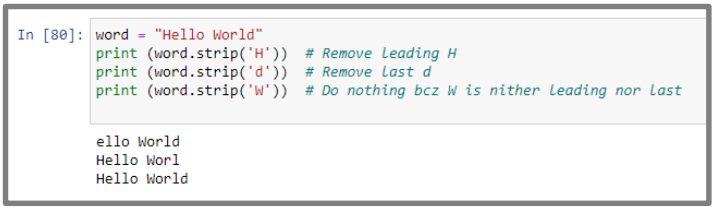
Changing Upper and Lower Case Strings: Use ***upper(), lower(), and capitalize()*** for upper & lower cases and capitalizing first letter of the string.

```
word = "Hello World"
print (word.upper())
print (word.lower())
print (word.capitalize())
```



Strip: Use the ***strip*** functions to remove most left or right character from a given string.

```
word = "Hello World"
print (word.strip('H'))
print (word.strip('d'))
print (word.strip('W'))
```



For more detail, please visit the following websites.
https://www.pythonforbeginners.com/basics/string-manipulation-in-python
https://www.w3schools.com/python/python_ref_string.asp

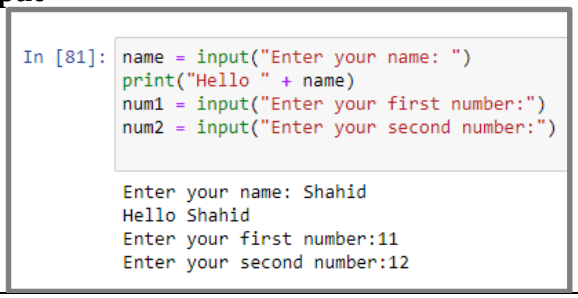
12. **Input function:**

- Programs can accept inputs from the user using the ***input()*** function.
- By default, the input is considered as string datatype.
- For integer and floating type data, you will need to explicitly convert them.

Example:

```
name = input("Enter your name: ")
print("Hello " + name)
num1 = input("Enter your first number:")
num2 = input("Enter your second number:")
```

Output



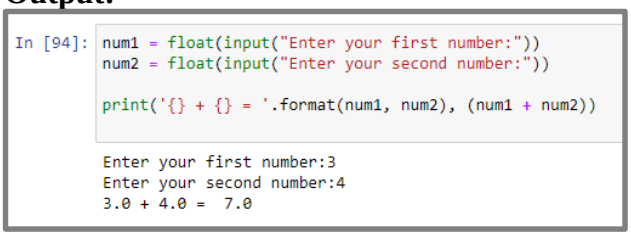
13. **Format function:**

- We can also combine strings and numbers by using the ***format()*** function that takes the arguments, formats them as variables and places them in the ***string placeholders {}***.

Example:

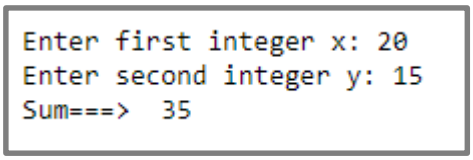
```
num1 = float(input("Enter your first number:"))
num2 = float(input("Enter your second
number:"))

print('{} + {} = '.format(num1, num2))
print(num1 + num2)
```

Output:



Homework:

1. **Define two variables x and y. Use input function to get the integer inputs, add them and print the output. Your output should look like:**



2. **What is the output of the following string comparison?**

```
print("John" > "Jhon")
print("Emma" < "Emm")
```

3. **Test the output of the following code and explain str1[1:4] str1[:5], str1[4:]?**

```
str1 = "Hello World"
print(str1[1:4], str1[:5], str1[4:])
```

Python Lecture-6

1) List data types
2) List manipulation
3) Tuple,
4) Set
5) Set manipulation
6) Dictionary
7) Dictionary manipulation

1. List data types

- Lists are written within ***square brackets [].***

```
# Define a list
z = [3, 7, 4, 2]
```

- Lists store an ordered collection of items which can be of different types.

z =	[3,	7,	4,	2]
index	0	1	2	3

- The list defined above has items that are all of the same type ***(int),*** but all the items of a list ***do not need to be of the same type*** as you can see below.

```
heterogenousElements = [3, True, 'Michael', 2.0]        # Define a list
```

Example 1:

```
z = [3, 7, 4, 2]
print (z)
print('element at index 0: ', z[0],'\nelement at index 1: ',z[1],'\nelement at index 2: ',z[2],'\nelement at index 2: ',z[3])
```

Output:

```
In [105]: z = [3, 7, 4, 2]
          print (z)
          print('element at index 0: ', z[0],'\nelement at index 1: ',z[1],'\nelement at index 2: ',z[2],'\nelement at index 2: ',z[3])

          [3, 7, 4, 2]
          element at index 0:  3
          element at index 1:  7
          element at index 2:  4
          element at index 2:  2
```

Example 2:

```
heterogenousElements = [3, True, 'Michael', 2.0]
print (heterogenousElements)
print('element at index 0: ', heterogenousElements[0])
print('element at index 1: ',heterogenousElements[1])
print('element at index 2: ',heterogenousElements[2])
print('element at index 2: ',heterogenousElements[3])
```

Output:

```
In [107]: heterogenousElements = [3, True, 'Michael', 2.0]
          print (heterogenousElements)
          print('element at index 0: ', heterogenousElements[0])
          print('element at index 1: ',heterogenousElements[1])
          print('element at index 2: ',heterogenousElements[2])
          print('element at index 2: ',heterogenousElements[3])

          [3, True, 'Michael', 2.0]
          element at index 0:  3
          element at index 1:  True
          element at index 2:  Michael
          element at index 2:  2.0
```

2. List manipulation

Accessing element: Access the element from left to right with positive index and right to left with negative index.

```
# Define a list
z = [3, 7, 4, 2]
# Access the first item of a list at index 0
print(z[0])
print(z[-1])
```

Output:

```
In [111]: # Define a list
          z = [3, 7, 4, 2]
          # Access the first and last item of a list at index 0, -1
          print(z[0])
          print(z[-1])

          3
          2
```

Slice of Lists: The slice helps to get a subset of the list.

```
# Define a list
z = [3, 7, 4, 2]
print(z[0:2]) # first two elements
print(z[:3])  # First to third elements
print(z[0:])  # first to last elements
print(z[1:3]) # second to second last elements
```

Output

```
In [120]: # Define a list
          z = [3, 7, 4, 2]
          print(z[0:2]) # first two elements
          print(z[:3])  # First to third elements
          print(z[0:])  # first to last elements
          print(z[1:3]) # second to second last elements

          [3, 7]
          [3, 7, 4]
          [3, 7, 4, 2]
          [7, 4]
```

Update list: Update the elements of a list.

```
# Defining a list
z = [3, 7, 4, 2]
# Update the item at index 1 with the string "fish"
z[1] = "fish"
print(z)
```

Output

```
In [121]: # Defining a list
          z = [3, 7, 4, 2]
          # Update the item at index 1 with the string "fish"
          z[1] = "fish"
          print(z)

          [3, 'fish', 4, 2]
```

Index function: Find the index of an element using the ***index()*** function. In the case of multiple same values, the index() return the first index. However, you can specific the starting point of the search.

```
# Define a list
z = [4, 1, 5, 4, 10, 4]
print(z.index(4))
print(z.index(4, 4))
```

Output:

```
In [128]: # Define a list
          z = [4, 1, 5, 4, 10, 4]
          print(z.index(4))
          print(z.index(4, 4))

          0
          5
```

Count function: It counts the number of times a value occurs in a list.

```
random_list = [4, 1, 5, 4, 10, 4]
print(random_list.count(5))
```

Output:

```
In [130]: random_list = [4, 1, 5, 4, 10, 4]
          print(random_list.count(4))

          3
```

Sort function: Sort the elements in ascending and descending orders

Example1

```
z = [4, 1, 5, 2, 10, 0]
print('Origional list:', z)
print('.....')
z.sort() # Ascending order
print('Asceding order:', z)
print('.....')
z.sort(reverse = True) # Descending order
print('Decending order:', z)
```

Output:1

```
In [141]: z = [4, 1, 5, 2, 10, 0]
          print('Origional list:', z)
          print('.....')
          z.sort() # Ascending order
          print('Asceding order:', z)
          print('.....')
          z.sort(reverse = True) # Descending order
          print('Decending order:', z)

          Origional list: [4, 1, 5, 2, 10, 0]
          .....
          Asceding order: [0, 1, 2, 4, 5, 10]
          .....
          Decending order: [10, 5, 4, 2, 1, 0]
```

Example 2

```
z = ['Elle', 'Miles', 'Kratos', 'Joel', 'Peter', 'Nathan']
print('Origional list:', z)
print('.....')
z.sort() # Ascending order
print('Asceding order:', z)
print('.....')
z.sort(reverse = True) # Descending order
print('Decending order:', z)
```

Output:2

```
In [142]: z = ['Elle', 'Miles', 'Kratos', 'Joel', 'Peter', 'Nathan']
          print('Origional list:', z)
          print('.....')
          z.sort() # Ascending order
          print('Asceding order:', z)
          print('.....')
          z.sort(reverse = True) # Descending order
          print('Decending order:', z)

          Origional list: ['Elle', 'Miles', 'Kratos', 'Joel', 'Peter', 'Nathan']
          .....
          Asceding order: ['Elle', 'Joel', 'Kratos', 'Miles', 'Nathan', 'Peter']
          .....
          Decending order: ['Peter', 'Nathan', 'Miles', 'Kratos', 'Joel', 'Elle']
```

Append function: The append method adds an element to the end of a list.

```
z = [7, 4, 3, 2]
z.append(10)
print(z)
```

Output:

```
In [146]: z = [7, 4, 3, 2]
          z.append(10)
          print(z)

          [7, 4, 3, 2, 10]
```

Remove function: The remove function removes an element from the list.

```
z = [7, 4, 3, 2, 5, 8]
print('Original list: ',z)
print('.....')
print('After removing 4: ',z)
z.remove(4)
print(z)
```

Output:

```
In [153]: z = [7, 4, 3, 2, 5, 8]
          print('Original list: ',z)
          print('.....')
          print('After removing 4: ',z)
          z.remove(4)
          print(z)

          Original list:  [7, 4, 3, 2, 5, 8]
          .....
          After removing 4:  [7, 4, 3, 2, 5, 8]
          [7, 3, 2, 5, 8]
```

Pop function: The pop method removes an item at the index you provide.

```
z = [7, 4, 3, 2, 5, 8]
```

Output:

```
print('Original list: ',z)
print('.....')
print('After popup: ',z.pop(4))
print(z)
```

```
In [157]: z = [7, 4, 3, 2, 5, 8]
          print('Original list: ',z)
          print('.....')
          print('After popup: ',z.pop(4))
          print(z)

          Original list:  [7, 4, 3, 2, 5, 8]
          .....
          After popup:  5
          [7, 4, 3, 2, 8]
```

Extend function: The extend() function extends a list by appending items. The same can be achieved by adding two lists.

Example 1

```
z = [7, 4, 3, 2, 5, 8]
print('Original list: ',z)
print('.....')
z.extend([20,40])
print('Extended list: ',z)
```

Output:1

```
In [162]: z = [7, 4, 3, 2, 5, 8]
          print('Original list: ',z)
          print('.....')
          z.extend([20,40])
          print('Extended list: ',z)

          Original list:  [7, 4, 3, 2, 5, 8]
          .....
          Extended list:  [7, 4, 3, 2, 5, 8, 20, 40]
```

Example 2

```
z = [7, 4, 3, 2, 5, 8]
z1 = [20,40]
print('First list: ',z)
print('.....')
print('Second list: ',z1)
print('.....')
print('extended: ',z+z1)
```

Output:2

```
In [160]: z = [7, 4, 3, 2, 5, 8]
          z1 = [20,40]
          print('First list: ',z)
          print('.....')
          print('Second list: ',z1)
          print('.....')
          print('extended: ',z+z1)

          First list:  [7, 4, 3, 2, 5, 8]
          .....
          Second list:  [20, 40]
          .....
          extended:  [7, 4, 3, 2, 5, 8, 20, 40]
```

Insert function: The ***insert()*** function inserts an item before the index you provide.

```
z = [7, 3, 3, 4, 5]
print('Original list: ',z)
print('.....')
z.insert(4, 10)
print('After inserting 10: ',z)
```

Output

```
In [168]: z = [7, 3, 3, 4, 5]
          print('Original list: ',z)
          print('.....')
          z.insert(4, 10)
          print('After inserting 10: ',z)

          Original list:  [7, 3, 3, 4, 5]
          .....
          After inserting 10:  [7, 3, 3, 4, 10, 5]
```

3. **Tuple:**
   - Python provides another type that is an ***ordered collection*** of objects, called a ***tuple***.
   - Tuples are identical to lists in all respects, except for the following properties:
     - Tuples are defined by enclosing the elements in ***parentheses ().***
     - Tuples are ***immutable***

```
In [172]: t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
          print (t)
          print (type(t))

          ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
          <class 'tuple'>
```

List:

```
t = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
print (t)
print (type(t))
print('.............................')
t[0] = 'zoo'
print (t)
```

Output:

```
In [174]: t = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
          print (t)
          print (type(t))
          print('.............................')
          t[0] = 'zoo'
          print (t)

          ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
          <class 'list'>
          .............................
          ['zoo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

Tuple:

```
t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
print (t)
print (type(t))
print('.............................')
t[0] = 'zoo'
print (t)
```

Output:

```
In [175]: t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
          print (t)
          print (type(t))
          print('.............................')
          t[0] = 'zoo'
          print (t)

          ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
          <class 'tuple'>
          .............................

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-175-3e6c16dc72c6> in <module>
                3 print (type(t))
                4 print('.............................')
          ----> 5 t[0] = 'zoo'
                6 print (t)

          TypeError: 'tuple' object does not support item assignment
```

4. Set:

- A set is created using curly braces *{}*, with elements separated by comma.
- The elements may be of different types (integer, float, tuple, string etc.).
- The set elements are immutable
- Set elements can't be duplicated

Example 1: Creating different types of sets

```
# Different types of sets in Python
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Example 2:

```
# set cannot have duplicates
# Output: {1, 2, 3, 4}
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)
```

Example 3: Set from list

```
# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1, 2, 3, 2])
print(my_set)
```

Example 4:

Output: 1

```
In [180]: # Different types of sets in Python
          # set of integers
          my_set = {1, 2, 3}
          print(my_set)
          print(type(my_set))

          # set of mixed datatypes
          my_set = {1.0, "Hello", (1, 2, 3)}
          print(my_set)
          print(type(my_set))

          {1, 2, 3}
          <class 'set'>
          {1.0, 'Hello', (1, 2, 3)}
          <class 'set'>
```

Output:2

```
In [178]: # set cannot have duplicates
          # Output: {1, 2, 3, 4}
          my_set = {1, 2, 3, 4, 3, 2}
          print(my_set)

          {1, 2, 3, 4}
```

```
# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.
my_set = {1, 2, 3}
print(my_set)
my_set = {1, 2, 3, [6,7]}
```

Output: 3

```
In [185]: # we can make set from a list
          # Output: {1, 2, 3}
          my_set = set([1, 2, 3, 2])
          print(my_set)

          {1, 2, 3}
```

Output: 4

```
In [186]: # set cannot have mutable items
          # here [3, 4] is a mutable list
          # this will cause an error.
          my_set = {1, 2, 3}
          print(my_set)
          my_set = {1, 2, 3, [6,7]}

          {1, 2, 3}

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-186-8e04ab24eb96> in <module>
                4 my_set = {1, 2, 3}
                5 print(my_set)
          ----> 6 my_set = {1, 2, 3, [6,7]}

          TypeError: unhashable type: 'list'
```

Example 5: Empty set

```
# Distinguish set and dictionary while creating empty set

# initialize a with {}
a = {}

# check data type of a
print(type(a))

# initialize a with set()
a = set()

# check data type of a
print(type(a))
```

Output: 5

```
In [187]: # Distinguish set and dictionary while creating empty set

          # initialize a with {}
          a = {}

          # check data type of a
          print(type(a))

          # initialize a with set()
          a = set()

          # check data type of a
          print(type(a))

          <class 'dict'>
          <class 'set'>
```

5. Set manipulation

Adding element and update set: The ***add()*** and ***update()*** methods are used to modify the set.

Example:

```
# initialize my_set
my_set = {1, 3}
print('Original set', my_set)
print('.............................')
# add single element
my_set.add(2)
print('Modified set', my_set)
print('.............................')
# adding multiple element
my_set.update({1, 6, 8})
print('Modified set', my_set)
```

Output:

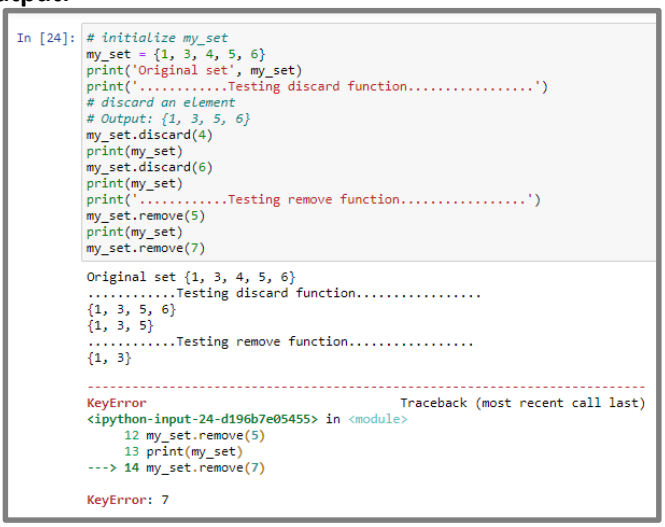
```
In [18]: # initialize my_set
         my_set = {1, 3}
         print('Original set', my_set)
         print('.............................')
         # add single element
         my_set.add(2)
         print('Modified set', my_set)
         print('.............................')
         # adding multiple element
         my_set.update({1, 6, 8})
         print('Modified set', my_set)

         Original set {1, 3}
         .............................
         Modified set {1, 2, 3}
         .............................
         Modified set {1, 2, 3, 6, 8}
```

Removing element: Use the ***remove()*** and ***discard().*** IF an element doesn't exist the ***remove()*** function will raise an error, while the ***discard()*** will do nothing.

Example:

```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print('Original set', my_set)
print('.....Testing discard function.....')
# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)
my_set.discard(6)
print(my_set)
print('.....Testing remove function.....')
my_set.remove(5)
print(my_set)
my_set.remove(7)
print(my_set)
```

Output:

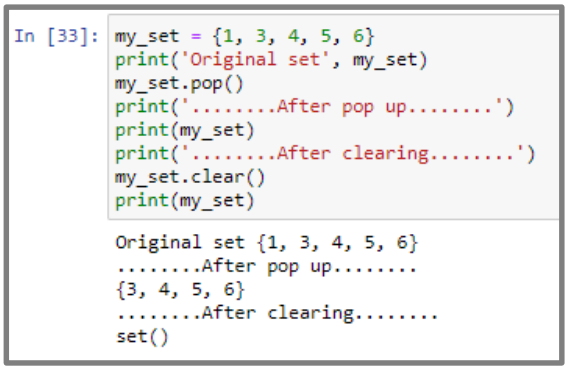


POP up element and clearing set: The ***pop()*** function popup the first element from the set, while the ***clear()*** function remove all the elements from a set.

Example:

```
my_set = {1, 3, 4, 5, 6}
print('Original set', my_set)
my_set.pop()
print('.....After pop up.....')
print(my_set)
print('.....After clearing.....')
my_set.clear()
print(my_set)
```

Output:

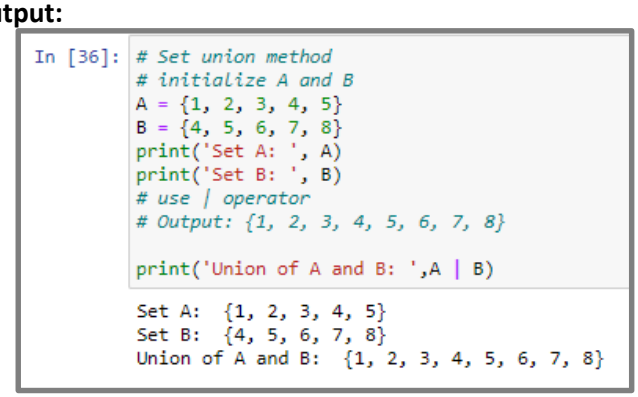


Set operations:

- **Union:**

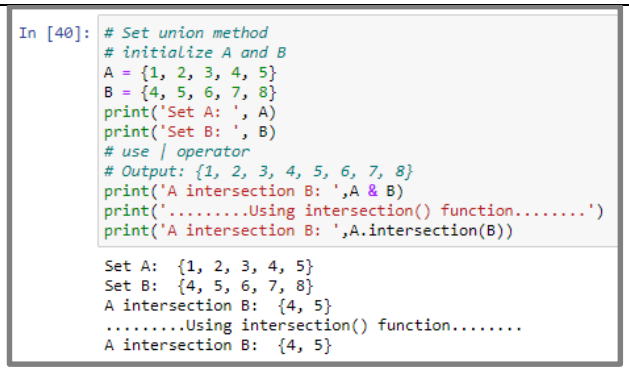
```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print('Set A: ', A)
print('Set B: ', B)
# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print('Union of A and B: ',A | B)
```

Output:



- **Intersection:**

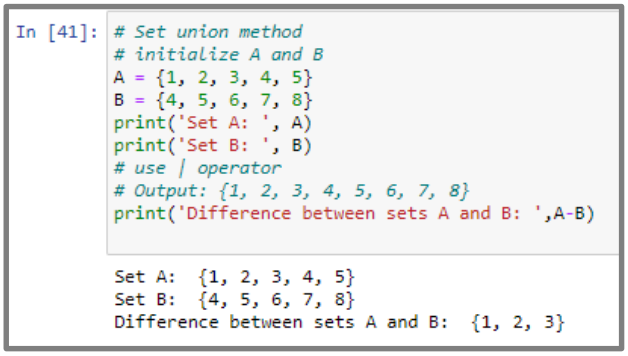
```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print('Set A: ', A)
print('Set B: ', B)
# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print('A intersection B: ',A & B)
print('.....Using intersection() function.....')
print('A intersection B: ',A.intersection(B))
```



```
In [40]: # Set union method
         # initialize A and B
         A = {1, 2, 3, 4, 5}
         B = {4, 5, 6, 7, 8}
         print('Set A: ', A)
         print('Set B: ', B)
         # use | operator
         # Output: {1, 2, 3, 4, 5, 6, 7, 8}
         print('A intersection B: ',A & B)
         print('.....Using intersection() function.....')
         print('A intersection B: ',A.intersection(B))

         Set A:  {1, 2, 3, 4, 5}
         Set B:  {4, 5, 6, 7, 8}
         A intersection B:  {4, 5}
         .....Using intersection() function.....
         A intersection B:  {4, 5}
```

- **Difference:**

```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print('Set A: ', A)
print('Set B: ', B)
# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print('Difference between sets A and B: ',A-B)
```



```
In [41]: # Set union method
         # initialize A and B
         A = {1, 2, 3, 4, 5}
         B = {4, 5, 6, 7, 8}
         print('Set A: ', A)
         print('Set B: ', B)
         # use | operator
         # Output: {1, 2, 3, 4, 5, 6, 7, 8}
         print('Difference between sets A and B: ',A-B)

         Set A:  {1, 2, 3, 4, 5}
         Set B:  {4, 5, 6, 7, 8}
         Difference between sets A and B:  {1, 2, 3}
```

For further study on set operations please check.

https://www.programiz.com/python-programming/set

6. Dictionary:

- Dictionaries are Python's implementation of a data structure that is more generally known as an associative array.
- A dictionary consists of a collection of key-value pairs.
- Each key-value pair maps the key to its associated value.
- You can also construct a dictionary with the built-in ***dict()*** function.
- The argument to ***dict()*** should be a sequence of ***key-value pairs***.

```
d = dict([
    (<key>, <value>),
    (<key>, <value),
      .
      .
      .
    (<key>, <value>)
])
```

Example:

```
months = {} # Create empty dictionary
```

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print('Dictionary :', months)

print ('The dictionary contains the following keys: ', months.keys())

print ('The dictionary contains the following keys: ', months[3])
```

Output:

```
In [48]: months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "Septemb
         print('Dictionary :', months)

         print ('The dictionary contains the following keys: ', months.keys())

         print ('The dictionary contains the following keys: ', months[3])

Dictionary : {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June', 7: 'July', 8: 'August', 9: 'September',
10: 'October', 11: 'November', 12: 'December'}
The dictionary contains the following keys:  dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
The dictionary contains the following keys:  March
```

7. Dictionary manipulation

Accessing Element: A value is retrieved from a dictionary by specifying its corresponding key in square brackets [].
If you refer to a key that is not in the dictionary, Python raises an exception:

Example: 1

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }

k = int(input("Enter the key value in range [1-12]"))
print ('The dictionary value at ' ,  k, ' is ', months[k])
```

Example: 2

Enter the key value in range [1-12]**13**

Output:1

```
In [61]: months = { 1 : "January", 2 : "February", 3 : "March", 4 :

         k = int(input("Enter the key value in range [1-12]"))
         print ('The dictionary value at ' ,  k, ' is ', months[k])

Enter the key value in range [1-12]1
The dictionary value at  1  is  January
```

Output:2

```
In [62]: months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May"

         k = int(input("Enter the key value in range [1-12]"))
         print ('The dictionary value at ' ,  k, ' is ', months[k])

Enter the key value in range [1-12]13

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-62-f22dd1aec68d> in <module>
      2
      3 k = int(input("Enter the key value in range [1-12]"))
----> 4 print ('The dictionary value at ' ,  k, ' is ', months[k])

KeyError: 13
```

Update an entry: If you want to update an entry, you can just assign a new value to an existing key:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
k = int(input("Enter the key value in range [1-12]: "))
print ('Current value at ' ,  k, ' is ', months[k])
print('.....................................................................')
stV = input("Enter the value: ")
months[k] = stV
```

Output:

```
In [67]: months = { 1 : "January", 2 : "February", 3 : "March", 4 :
         k = int(input("Enter the key value in range [1-12]: "))
         print ('Current value at ' ,  k, ' is ', months[k])
         print('.....................................................
         stV = input("Enter the value: ")
         months[k] = stV
         print ('Modified value at ' ,  k, ' is ', months[k])

Enter the key value in range [1-12]: 3
Current value at  3  is  March
.....................................................
Enter the value: Mar.
Modified value at  3  is  Mar.
```

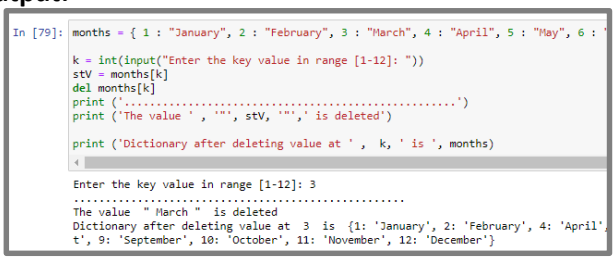
print ('Modified value at ' , k, ' is ', months[k])

Delete: To delete an entry, use the del statement, specifying the key to delete:

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }

k = int(input("Enter the key value in range [1-12]: "))
stV = months[k]
del months[k]
print ('.....')
print ('The value ' , '"', stV, '",' is deleted')

print ('Dictionary after deleting value at ' , k, ' is ', months)
```

Output:

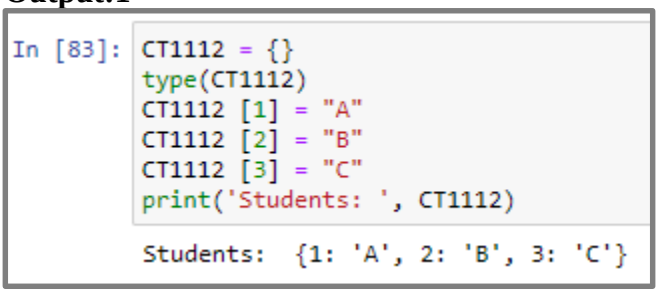


Dynamic Dictionary: You can start by creating an empty dictionary, which is specified by empty curly braces. Then you can add new keys and values one at a time:

Example: 1

```
CT1112 = {}
type(CT1112)
CT1112 [1] = "A"
CT1112 [2] = "B"
CT1112 [3] = "C"
print('Students: ', CT1112)
```

Output:1



Example: 2

```
CT1112 = {}
CT1112[1] = "John"
print(CT1112[1], ' registered')
print('.....')
print('Total students: ', CT1112)
print('.....')
CT1112[2] = "Paul"
print(CT1112[2], ' registered')
print('.....')
print('Total students: ', CT1112)
CT1112[3] = "Connor"
print(CT1112[3], ' registered')
print('.....')
print('Total students: ', CT1112)
```

Output:2

```
CT1112[4] = "Jack"
print(CT1112[4], ' registered')
print('.....................................')
print('Total students: ', CT1112)
```

Multiple element at single key:

Example:1

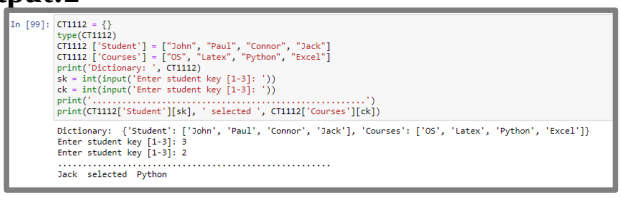
```
CT1112 = {}
type(CT1112)
CT1112 ['Student'] = ["John", "Paul", "Connor", "Jack"]
CT1112 ['Courses'] = ["OS", "Latex", "Python", "Excel"]
print('Dictionary: ', CT1112)

print('.....................................................')
print(CT1112['Student'][1], ' is studing: ', CT1112['Courses'][2])
```

Output:1

Example: 2

```
CT1112 = {}
type(CT1112)
CT1112 ['Student'] = ["John", "Paul", "Connor", "Jack"]
CT1112 ['Courses'] = ["OS", "Latex", "Python", "Excel"]
print('Dictionary: ', CT1112)
sk = int(input('Enter student key [1-3]: '))
ck = int(input('Enter student key [1-3]: '))
print('.....................................................')
print(CT1112['Student'][sk], ' selected ', CT1112['Courses'][ck])
```

Output:2



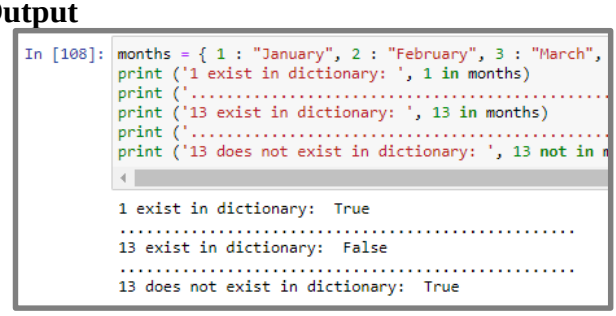
Built-in functions:

in* and *not in: The ***in*** and ***not in*** returns True and False values according to the values in the dictionary.

Example:

```
months = {1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print ('1 exist in dictionary: ', 1 in months)
print ('.....................................................')
```

Output



```
print ('13 exist in dictionary: ', 13 in months)
print ('.....')
print ('13 does not exist in dictionary: ', 13 not in months)
```

Len(): The ***len()*** function returns the number of key-value pairs in a dictionary:

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print ('Length of dictionary: ', len(months))
```

Output:

```
In [109]: months = { 1 : "January", 2 : "February", 3 :
          print ('Length of dictionary: ', len(months))

          Length of dictionary:  12
```

Clear(): The clear() function empties all the entries in a dictionary

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print ('Length of dictionary: ', len(months))
print('.....')
months.clear()
print ('Length of dictionary: ', len(months))
```

Output:

```
In [110]: months = { 1 : "January", 2 : "February", 3 :
          print ('Length of dictionary: ', len(months))
          print('.....
          months.clear()
          print ('Length of dictionary: ', len(months))

          Length of dictionary:  12
          .....
          Length of dictionary:  0
```

Get(): The ***get()*** function returns the value for a given key, it exist otherwise it return **none**.

Example: 1

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print ('Length of dictionary: ', len(months))
print('.....')
v = int(input('Enter a key: '))
print('Value for ', v, 'is: ', months.get(v))
```

Output:1

```
In [119]: months = { 1 : "January", 2 : "February", 3 : "M
          print ('Length of dictionary: ', len(months))
          print('.....
          v = int(input('Enter a key: '))
          print('Value for ', v, 'is: ', months.get(v))

          Length of dictionary:  12
          .....
          Enter a key: 3
          Value for  3 is:  March
```

Example: 2

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print ('Length of dictionary: ', len(months))
print('.....')
v = int(input('Enter a key: '))
print('Value for ', v, 'is: ', months.get(v,'Not found'))
```

Output:2

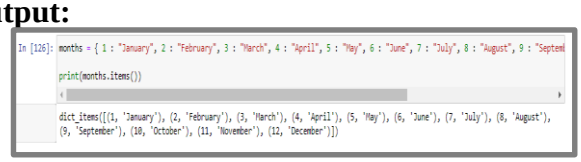
```
In [120]: months = { 1 : "January", 2 : "February", 3 : "March", 4 :
          print ('Length of dictionary: ', len(months))
          print('.....
          v = int(input('Enter a key: '))
          print('Value for ', v, 'is: ', months.get(v,'Not found'))

          Length of dictionary:  12
          .....
          Enter a key: 13
          Value for  13 is:  Not found
```

Items(): The ***item()*** function returns a list of tuples containing the key-value pairs. The first item in each tuple is the key, and the second item is the key's value:

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.items())
```

Output:

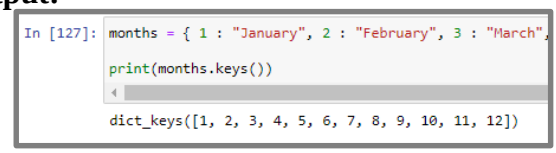


Keys(): The ***keys()*** function returns all the keys of a dictionary.

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.keys())
```

Output:



Values(): The ***values()*** functions returns all the values of a dictionary.

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.values())
```

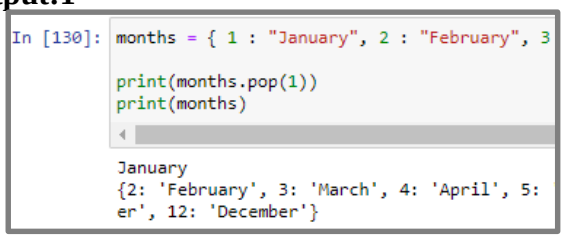
Output:

Pop(): The ***pop()*** function remove the specified key and returns the its value. It will raise and error if the key doesn't exist.

Example: 1

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.pop(1))
print(months)
```

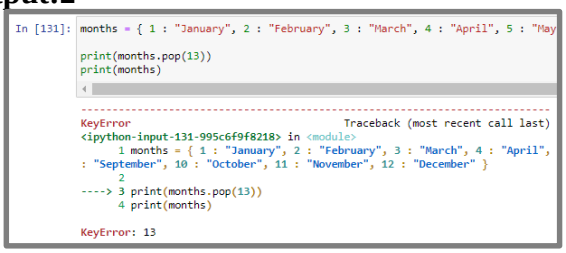
Output:1



Example: 2

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.pop(13))
print(months)
```

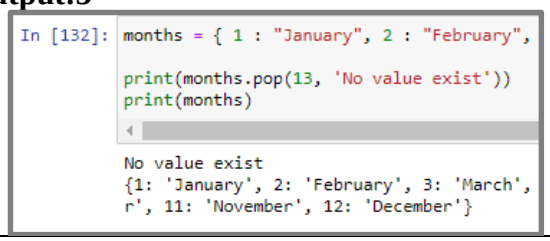
Output:2



Example: 3. Handling error

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.pop(13, 'No value exist'))
print(months)
```

Output:3

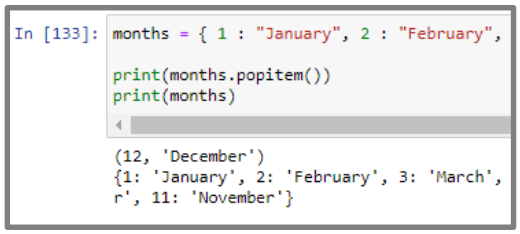


Popitem(): This function remove the last key-value pair and return them.

Example:

```
months = { 1 : "January", 2 : "February", 3 : "March", 4 : "April", 5 : "May", 6 : "June", 7 : "July", 8 : "August", 9 : "September", 10 : "October", 11 : "November", 12 : "December" }
print(months.popitem())
print(months)
```

Output:

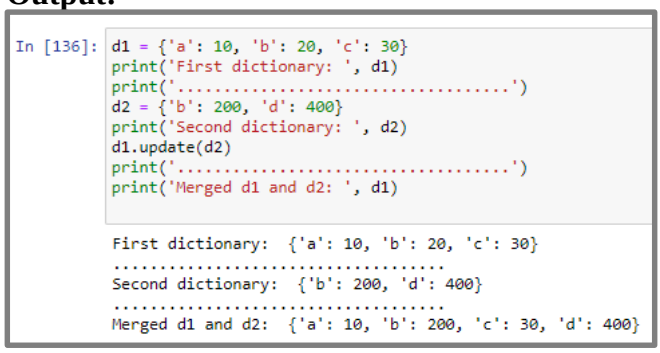


Update(): The ***update()*** function merge two dictionaries.

Example:

```
d1 = {'a': 10, 'b': 20, 'c': 30}
print('First dictionary: ', d1)
print('.....')
d2 = {'b': 200, 'd': 400}
print('Second dictionary: ', d2)
d1.update(d2)
print('.....')
print('Merged d1 and d2: ', d1)
```

Output:



Python Lecture-7

1) Conditional statement (If-Else-if statements)
2) Loops
3) Function
4) Scope of a variable

1. **Conditional statement:**
   - Decision making is required when we want to ***execute a code only if a certain condition is satisfied.***
   - The ***if…elif…else*** statement is used in Python for decision making.

1.1. If statement: Syntax

```
if test expression:
    statement(s)
```

The program ***text expression*** and execute ***statement(s)*** if the condition is true
If the condition is ***false***, the statement(s) will ***not execute***

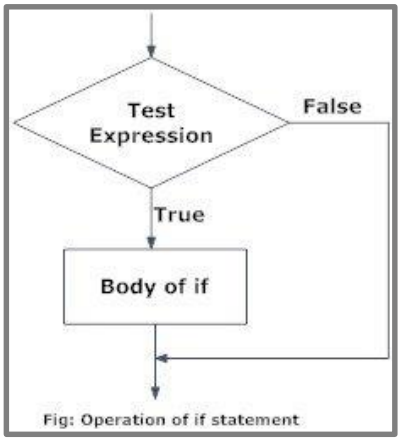


Fig: Operation of if statement

Example:

```
#Testing a positive number
num = int(input('Enter a number: '))
if num > 0:
    print(num, "is a positive number.")
```

Output:1 True conditon

```
In [141]: #Testing a positive number

num = int(input('Enter a number: '))
if num > 0:
    print(num, "is a positive number.")

Enter a number: 5
5 is a positive number.
```

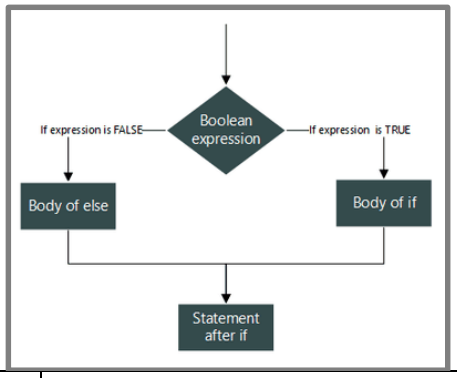
Output:2 False condition

```
In [142]: #Testing a positive number

num = int(input('Enter a number: '))
if num > 0:
    print(num, "is a positive number.")

Enter a number: -10
```

1.2. If Else statement: Syntax

```
if test expression:
    Body of if
else:
    Body of else
```

If the text expression is true, then execute ***body of if*** otherwise execute ***body of else***



Example:

```
#Testing a positive or negative number

num = int(input('Enter a number: '))
if num > 0:
   print(num, "is a positive number.")
else:
   print(num, "is a negative number.")
```

Output: True conditon

```
In [145]: #Testing a positive or negative number

num = int(input('Enter a number: '))
if num > 0:
    print(num, "is a positive number.")
else:
    print(num, "is a negative number.")

Enter a number: 10
10 is a positive number.
```

Output: False condition

```
In [146]: #Testing a positive or negative number

num = int(input('Enter a number: '))
if num > 0:
    print(num, "is a positive number.")
else:
    print(num, "is a negative number.")

Enter a number: -5
-5 is a negative number.
```

1.3. if...elif...else Statement: Syntax

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The ***elif*** is short for ***else-if.***

If the condition for ***if*** is ***False***, it checks the condition of the next ***elif*** block and so on.

If all the conditions are ***False***, the body of ***else is executed***.

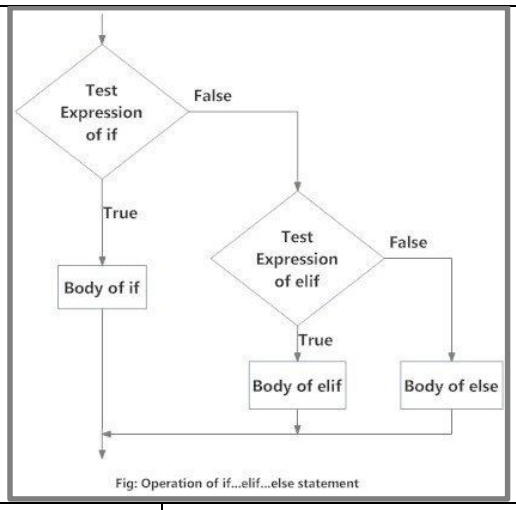


Fig: Operation of if...elif...else statement

Example:

```
num1 = float(input("Enter first number: "))
op = input("Enter the operator: ")
num2 = float(input("Enter second number: "))

if op == "+":
    print("The answer is: ", num1 + num2)
elif op == "-":
    print("The answer is: ", num1 - num2)
elif op == "*":
    print("The answer is: ", num1 * num2)
elif op == "/":
    print("The answer is: ", num1 / num2)
else:
    print("Invalid operator. Run programme again")
```

Output:

```
In [158]: num1 = float(input("Enter first number: "))
          op = input("Enter the operator: ")
          num2 = float(input("Enter second number: "))

          if op == "+":
              print("The answer is: ", num1 + num2)
          elif op == "-":
              print("The answer is: ", num1 - num2)
          elif op == "*":
              print("The answer is: ", num1 * num2)
          elif op == "/":
              print("The answer is: ", num1 / num2)
          else:
              print("Invalid operator. Run programme again")

          Enter first number: 10
          Enter the operator: +
          Enter second number: 15
          The answer is:  25.0
```

2. **Loops:** Loops provides a way to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.
   Common types: For Loop , While loop, Nested loop

2.1 For Loop: Syntax

```
for val in sequence:
    loop body
```

- Here, **val** is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we ***reach the last item*** in the sequence.
- The body of for loop is separated from the rest of the code using ***indentation***.

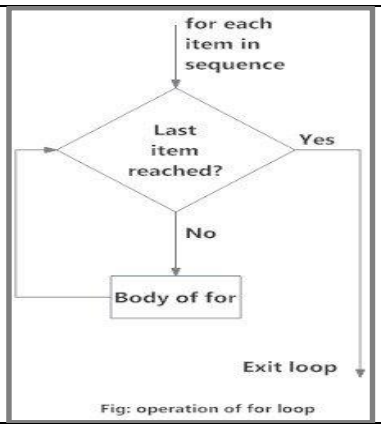


Fig: operation of for loop

Example:

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
    sum = sum+val
print("The sum is: ", sum)
```

Output:

```
In [159]: # Program to find the sum of all numbers stored in a list

          # List of numbers
          numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

          # variable to store the sum
          sum = 0

          # iterate over the list
          for val in numbers:
              sum = sum+val

          print("The sum is: ", sum)

          The sum is:  48
```

Loop with range() function:

- The ***range()*** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- The range() function supports different parameters for start, stop, and step size. i.e., ***range(start, stop, step_size)***

Example: 1

```
for x in range(6):
  print(x)
```

Example: 2

```
for x in range(2, 30, 3):
  print(x)
```

- List with range():

Example:3

```
print(range(10))
print(list(range(10)))
print(list(range(2, 8)))
print(list(range(2, 20, 3)))
```

Output: 1

```
In [160]: for x in range(6):
            print(x)

          0
          1
          2
          3
          4
          5
```

Output: 2

```
In [161]: for x in range(2, 30, 3):
            print(x)

          2
          5
          8
          11
          14
          17
          20
          23
          26
          29
```

Output: 3

Example:4

```
# create a list lst with start:2, stop:20, and step size:3
lst = list(range(2,20,3))
# iterate over the list using index
print('The list values are:')
for i in range(len(lst)):
    print(lst[i])
```

Example: 5

```
# Program to iterate through a list using indexing
genre = ['OS', 'Latex', 'Python', 'Excel']
# iterate over the list using index
for i in range(len(genre)):
    print("I like", genre[i])
```

```
In [163]: print(range(10))

          print(list(range(10)))

          print(list(range(2, 8)))

          print(list(range(2, 20, 3)))

          range(0, 10)
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
          [2, 3, 4, 5, 6, 7]
          [2, 5, 8, 11, 14, 17]
```

Output: 4

```
In [167]: # create a list 1 with start:2, stop:20, and step size:3
          lst = list(range(2,20,3))
          # iterate over the list using index
          print('The list values are:')
          for i in range(len(lst)):
              print(lst[i])

          The list values are:
          2
          5
          8
          11
          14
          17
```

Output: 5

```
In [168]: # Program to iterate through a list using indexing

          genre = ['OS', 'Latex', 'Python', 'Excel']

          # iterate over the list using index
          for i in range(len(genre)):
              print("I like", genre[i])

          I like OS
          I like Latex
          I like Python
          I like Excel
```

For loop with else and break:

Example:1

```
digits = [0, 2, 5, 6, 7]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Example:2

```
digits = [0, 2, 5, 6, 7]
for i in digits:
    if i == 5: break
    print(i)
else:
    print("Loop break point")
```

Output:1

```
In [177]: digits = [0, 2, 5, 6, 7]

          for i in digits:
              print(i)
          else:
              print("No items left.")

          0
          2
          5
          6
          7
          No items left.
```

Output:2

```
In [179]: digits = [0, 2, 5, 6, 7]

          for i in digits:
              if i == 5: break
              print(i)
          else:
              print("Loop break point")

          0
          2
```

For loop with Dictionary, if, and break

Example:

```
# program to display student's marks from record
student_name = input('Enter student name: ')
marks = {'James': 90, 'Jules': 55, 'Arthur': 77}
for student in marks:
    if student == student_name:
        print(marks[student])
        break
else:
    print('No entry with that name found.')
```

Output

```
In [184]: # program to display student's marks from record
          student_name = input('Enter student name: ')

          marks = {'James': 90, 'Jules': 55, 'Arthur': 77}

          for student in marks:
              if student == student_name:
                  print(marks[student])
                  break
          else:
              print('No entry with that name found.')

          Enter student name: Jules
          55
```

2.1. While loop:

- The while loop is used to iterate over a block of code as long as the ***test expression*** (condition) is ***true***.
- Generally, the While loop is used when the ***number of iteration*** beforehand is ***not known.***
- Syntax:

```
while test_expression:
    Body of while
```

If the ***text_experssion*** is true run the body of the while loop.

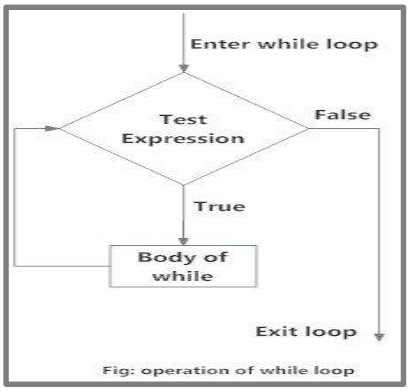


Fig: operation of while loop

Example:1

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n
# To take input from the user,
n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

Output:1

```
In [189]: # Program to add natural
          # numbers up to
          # sum = 1+2+3+...+n

          # To take input from the user,
          n = int(input("Enter n: "))

          # initialize sum and counter
          sum = 0
          i = 1

          while i <= n:
              sum = sum + i
              i = i+1    # update counter

          # print the sum
          print("The sum is", sum)

          Enter n: 20
          The sum is 210
```

Example: 2

Output:2

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
In [190]: i = 1
          while i < 6:
            print(i)
            i += 1

1
2
3
4
5
```

While loop with break statement:

Example:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output:

```
In [191]: i = 1
          while i < 6:
            print(i)
            if i == 3:
              break
            i += 1

1
2
3
```

While loop with continue statement:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Output:

```
In [192]: i = 0
          while i < 6:
            i += 1
            if i == 3:
              continue
            print(i)

1
2
4
5
6
```

While loop with else statement:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Output:

```
In [194]: i = 1
          while i < 6:
            print(i)
            i += 1
          else:
            print("i is no longer less than 6")

1
2
3
4
5
i is no longer less than 6
```

2.3. Nested loop:

Example:

```
for i in range(3):
    for j in range(5):
        print(j)
```

Output:

```
In [230]: for i in range(3):
              for j in range(5):
                  print(j)

0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
```

3. Function:

- **Syntax:**

- Function is a ***block of code*** (subprogram) that is ***written once***
- Called/used as ***many time*** as required
- Performs ***specific task***
- Pass data to function known as ***arguments***
- Function ***returns*** the result
- Main programming is known as ***calling program***, the function is known as ***called program***

Types of function:

- Built-in function
- User defined function

Components:

- Function definition
- Function call

Note: Built-in functions have definitions in the libraries and import the libraries, call the function with arguments and the function return the result.

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

1. Keyword ***def*** that marks the start of the function header.
2. A ***function name*** to uniquely identify the function.
3. ***Parameters (arguments)*** through which we pass values to a function.
4. A ***colon (:)*** to mark the end of the function header.
5. Optional documentation string ***(docstring)*** to describe what the ***function does***.
6. One or more valid python ***statements*** that make up the ***function body***.
7. An optional ***return*** statement to return a value from the function

Example: Simple function

```
def fun(): # Function definition
    print('This is a simple function!')

fun()  # Function call
```

Output:

```
In [295]: def fun(): # Function definition
              print('This is a simple function!')

          fun()  # Function call

          This is a simple function!
```

1.1. Passing arguments

Example:

```
Def greet(name): # function definition
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, ", name, ". Good afternoon!")
greet('James') # function call
```

Output:

```
In [258]: def greet(name): # function definition
              """
              This function greets to
              the person passed in as
              a parameter
              """
              print("Hello, ", name, ". Good afternoon!")
          greet('James') # function call

          Hello,  James . Good afternoon!
```

Passing two integer arguments

Example:

```
def sum(x, y):  # Function definition
    s = x+y
    print('Sum: ', s)
```

Output:

```
In [299]: def sum(x, y):  # Function definition
              s = x+y
              print('Sum: ', s)


          sum(10, 15)  # Function call

          Sum:  25
```

```
sum(10, 15)  # Function call
```

Passing integer and string arguments

Example:

```
# Function definition is here
def printinfo(name, age ):
   "This prints a passed info into this function"
   print ('Name: ', name)
   print ('Age: ', age)

# Now you can call printinfo function
printinfo( age=35, name='James' )
```

Output:

```
In [303]: # Function definition is here
          def printinfo(name, age ):
             "This prints a passed info into this function"
             print ('Name: ', name)
             print ('Age: ', age)


          # Now you can call printinfo function
          printinfo( age=35, name='James' )

          Name:  James
          Age:  35
```

Default arguments: The default arguments are the arguments that are considered if no arguments are passed to the function.

Example:

```
# Function definition is here
def printinfo(name, age = 35 ):
   "This prints a passed info into this function"
   print ('Name: ', name)
   print ('Age: ', age)
# Now you can call printinfo function
print('Both the arguements are passed')
printinfo(age=50, name='James') # both the arguements are passed
print('.....Age has a defult value.....')
printinfo( name='James' )
```

Output:

```
In [308]: # Function definition is here
          def printinfo(name, age = 35 ):
             "This prints a passed info into this function"
             print ('Name: ', name)
             print ('Age: ', age)


          # Now you can call printinfo function
          print('Both the arguements are passed')
          printinfo(age=50, name='James') # both the arguements are passed
          print('.....Age has a defult value.....')
          printinfo( name='James' )

          Both the arguements are passed
          Name:  James
          Age:  50
          .....Age has a defult value.....
          Name:  James
          Age:  35
```

Variable arguments:

- You can pass variable number arguments to the function.
- An asterisk (*) is placed before the variable name that holds the values of all non-keyword variable arguments.

Example: 1

```
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print (arg1)
   for var in vartuple:
      print (var)

# Now you can call printinfo function
print ('Fixed arguement passed is: ')
printinfo( 10 )
```

Output:1

```
In [316]: # Function definition is here
          def printinfo( arg1, *vartuple ):
             "This prints a variable passed arguments"
             print (arg1)
             for var in vartuple:
                print (var)


          # Now you can call printinfo function
          print ('Fixed arguement passed is: ')
          printinfo( 10 )
          print ('Variable arguement passed is: ')
          printinfo( 70, 60, 50 )

          Fixed arguement passed is:
          10
          Variable arguement passed is:
          70
          60
          50
```

```
print ('Variable argument passed is: ')
printinfo( 70, 60, 50 )
```

Example:2

```
def calculateTotalSum(*arguments):
    totalSum = 0
    for number in arguments:
        totalSum += number
    print(totalSum)

# function call
calculateTotalSum(5, 4, 3, 2, 1)
```

Output:2

```
In [317]: def calculateTotalSum(*arguments):
              totalSum = 0
              for number in arguments:
                  totalSum += number
              print(totalSum)

          # function call
          calculateTotalSum(5, 4, 3, 2, 1)

          15
```

3.2. Return statement:

- The ***return*** statement is used to passing back an ***expression*** to the caller.
- **Syntax:**
  return [expression_list]
- This statement can contain an expression that gets evaluated and the value is returned.
- If there is nothing to return and the return statement is used, then the function will return the ***None*** object.

Example:

```
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2
   return total; # Return the total

# Now you can call sum function
total = sum( 10, 20 );
print ('Returned total : ', total)
```

Output:

```
In [323]: # Function definition is here
          def sum( arg1, arg2 ):
             # Add both the parameters and return them."
             total = arg1 + arg2
             return total; # Return the total

          # Now you can call sum function
          total = sum( 10, 20 );
          print ('Returned total : ', total)

          Returned total :  35
```

Assignment: Write a function ***fun*** that ask the user to enter the ***first and second integers*** and sum them, the function then ***returned the output***.

Output:

```
Enter first integer: 12
Enter second integer: 13
Returned total :  25
```

Scope of a variable:

- The scope of a variable defines the ***accessibility*** of the variable.
- Variables that are defined ***inside a function*** body have a ***local scope***, and those defined ***outside*** have a ***global scope***.

Example:

```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
```

Output:

```
In [330]: total = 0; # This is global variable.
          # Function definition is here
          def sum( arg1, arg2 ):
             # Add both the parameters and return them."
             total = arg1 + arg2; # Here total is local variable.
             print ('Inside the function local total : ', total)

          # Now you can call sum function
          sum( 10, 20 );
          print ('Outside the function global total : ', total)

          Inside the function local total :  30
          Outside the function global total :  0
```

total = arg1 + arg2; # Here total is local variable.
print ('Inside the function local total : ', total)
 	
# Now you can call sum function 	
sum(10, 20);
print ('Outside the function global total : ', total)	

Python Lecture-8

1) External files
2) Classes and objects
3) Modules/libraries

1. Files:

- Files are named locations on ***disk to store*** related information.
- They are used to ***permanently store data in a non-volatile memory*** (e.g. hard disk).
- Hence, in Python, a file operation takes place in the following order:
  - Open a file
  - Read a file
  - Close the file
  - Write to the file
  - Creating a new file

1.1. Open a file: The built-in ***open()*** function is used to open a file in python.

- **Syntax:**

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt")
```

1.2. Read a file: The ***read()*** function can be used to read from the file. The optional argument ***r*** specifies the read mode.

- **Mode:** Mode defines the purpose for opening a file. For instance, reading, writing to the file, or appending etc. A detailed of different modes are given below.

Mode	Description
r	Opens a file for reading. (default)
w	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Opens a file for exclusive creation. If the file already exists, the operation fails.
a	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in text mode. (default)
b	Opens in binary mode.
+	Opens a file for updating (reading and writing)

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures NUIG/Lectures/Python/test.txt", "r")
print(f.read())
```

Output:

```
In [336]: # specifying full path
          f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
          print(f.read())

          This text is from file.
```

- **Readline function:**
  - The ***readline()*** function is used to read the first line from the file.
  - If it is called twice it will read the first two lines.

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.readline())
print(f.readline())
```

Output:

```
In [342]: # specifying full path
          f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
          print(f.readline())
          print(f.readline())

          This text is from file.

          This is second line.
```

- **Reading file through loop:**
  - Multiple lines/entire file contents can be read through looping.

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
for x in f:
    print(x)
```

Output:

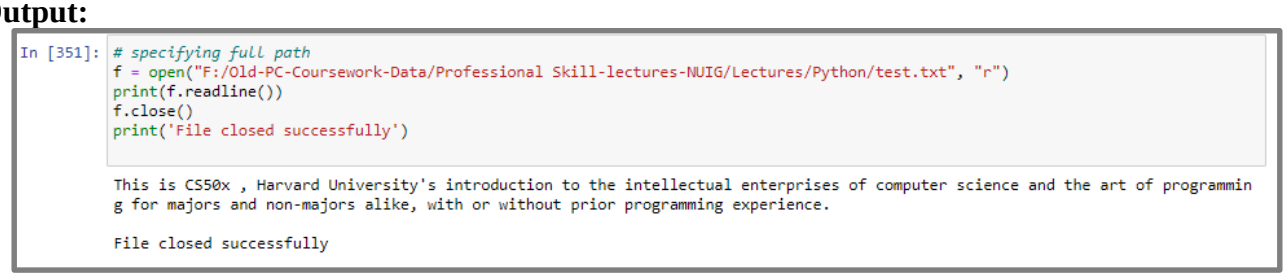
```
In [345]: # specifying full path
          f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
          for x in f:
              print(x)

          This is CS50x , Harvard University's introduction to the intellectual enterprises of computer science and the art of programmin
          g for majors and non-majors alike, with or without prior programming experience.

          An entry-level course taught by David J. Malan, CS50x teaches students how to think algorithmically and solve problems efficien
          tly.

          Topics include abstraction, algorithms, data structures, encapsulation, resource management, security, software engineering, an
          d web development.

          Languages include C, Python, SQL, and JavaScript plus CSS and HTML. Problem sets inspired by real-world domains of biology, cry
          ptography, finance, forensics, and gaming.

          The on-campus version of CS50x , CS50, is Harvard's largest course.
```

1.3. Closing a file: The ***close()*** function is used to close a file.

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.readline())
f.close()
print('File closed successfully')
```

Output:



```
In [351]: # specifying full path
          f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
          print(f.readline())
          f.close()
          print('File closed successfully')

          This is CS50x , Harvard University's introduction to the intellectual enterprises of computer science and the art of programmin
          g for majors and non-majors alike, with or without prior programming experience.

          File closed successfully
```

1.4. Write to the file: The ***write()*** function is used to write to the file. There are two mode parameters namely ***"a" and "w"***. The "a" parameter is for ***appending the text without overwriting*** the previous contents; however, the "w" ***overwrite the file***.

- **Writing to an existing file:** Writing an existing file requires to be opened using the ***open()*** function. Two examples with "a" and "w" are illustrated below.

  - **Writing with "a" option:**

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "a")
f.write("This is the new contents added from python")
f.close()

# Reopen the file with "r" mode and check the added contents
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.read())
```

Output:

```
In [355]: # specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "a")
f.write("This is the new contents added from python")
f.close()

# Reopen the file with "r" mode and check the added contents
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.read())

This is CS50x , Harvard University's introduction to the intellectual enterprises of computer science and the art of programmin
g for majors and non-majors alike, with or without prior programming experience.
An entry-level course taught by David J. Malan, CS50x teaches students how to think algorithmically and solve problems efficien
tly.
Topics include abstraction, algorithms, data structures, encapsulation, resource management, security, software engineering, an
d web development.
Languages include C, Python, SQL, and JavaScript plus CSS and HTML. Problem sets inspired by real-world domains of biology, cry
ptography, finance, forensics, and gaming.
The on-campus version of CS50x , CS50, is Harvard's largest course. This is the new contents added from python
```

- **Writing with "w" option:**

Example:

```
# specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

# Reopen the file with "r" mode and check the added contents
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.read())
```

Output:

```
In [356]: # specifying full path
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

# Reopen the file with "r" mode and check the added contents
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/test.txt", "r")
print(f.read())

Woops! I have deleted the content!
```

1.5. Creating a new file: To create a new file in Python, use the ***open()*** function, with one of the following parameters:

- "x" - Create - will create a file, returns an error if the file exist

Example:

```
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/CT1112.txt", "x")
f.write("This is a new created file")
f.close()
# Reopen the file with "r" mode and check the added contents
f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/CT1112.txt", "r")
print(f.read())
```

Output:

```
In [357]: f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/CT1112.txt", "x")
          f.write("This is a new created file")
          f.close()

          # Reopen the file with "r" mode and check the added contents
          f = open("F:/Old-PC-Coursework-Data/Professional Skill-lectures-NUIG/Lectures/Python/CT1112.txt", "r")
          print(f.read())

          This is a new created file
```

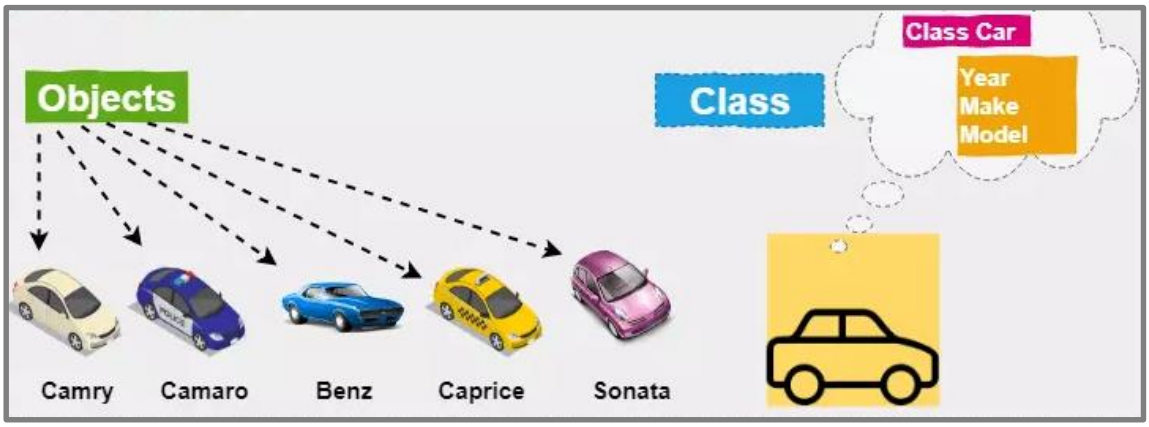
Note: There are various methods available with the file object. Some of them have been used in the above examples. Please read and practice if interested.

Method	Description
close()	Closes an opened file. It has no effect if the file is already closed.
detach()	Separates the underlying binary buffer from the `TextIOBase` and returns it.
fileno()	Returns an integer number (file descriptor) of the file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns `True` if the file stream is interactive.
read(`n`)	Reads at most `n` characters from the file. Reads till end of file if it is negative or `None`.
readable()	Returns `True` if the file stream can be read from.
readline(`n` =-1)	Reads and returns one line from the file. Reads in at most `n` bytes if specified.
readlines(`n` =-1)	Reads and returns a list of lines from the file. Reads in at most `n` bytes/characters if specified.
seek(`offset` , `from` = `SEEK_SET`)	Changes the file position to `offset` bytes, in reference to `from` (start, current, end).
seekable()	Returns `True` if the file stream supports random access.
tell()	Returns the current file location.
truncate(`size` = `None`)	Resizes the file stream to `size` bytes. If `size` is not specified, resizes to current location.
writable()	Returns `True` if the file stream can be written to.
write(`s`)	Writes the string `s` to the file and returns the number of characters written.
writelines(`lines`)	Writes a list of `lines` to the file.

2. Classes and objects:

- A class represents an ***abstract name*** for group of physical entities that holds common properties. For example, Human, Animals, Car etc. are classes that represents specific group.
- An object is a physical instance of the class, for example ***"James"*** is an object of class ***Human***, ***"Benz"*** is an object of ***Car*** and so on.

- Creating an object for the class allocate a space in the memory.
- **Example:** A general example of Class **(i.e., Car),** with objects.



- Considering the object-oriented programing, the classes are name representations for the objects and does not hold any space in memory.

- **Syntax:**

```
class Car:  # define class
""" Body (Variables and functions) of the Car class here. """

obj = Car() # define object
""" Different attributes and behavior of parrots are manipulated through object obj"""
```

Example:

```
class Car:  # Define class Car

    # class attribute
    model_name
    model_year
    model_color

    # instance attribute
    #The self keyword help to access the attribute of class
    def fun(self, mn , my, mc): # Function fun() definition
        self.model_name = mn
        self.model_year = my
        self.model_color = mc
        print('Model===> ', self.model_name)
        print('Year===>', self.model_year)
        print('Color===>', self.model_color)

# instantiate the Parrot class
obj = Car() # Create object obj for Car class
```

```
var1 = input('Enter the model name: ')
var2 = input('Enter the model year: ')
var3 = input('Enter the color: ')
print('.....................................................')
obj.fun(var1, var2, var3) # Call function fun
```

Output:

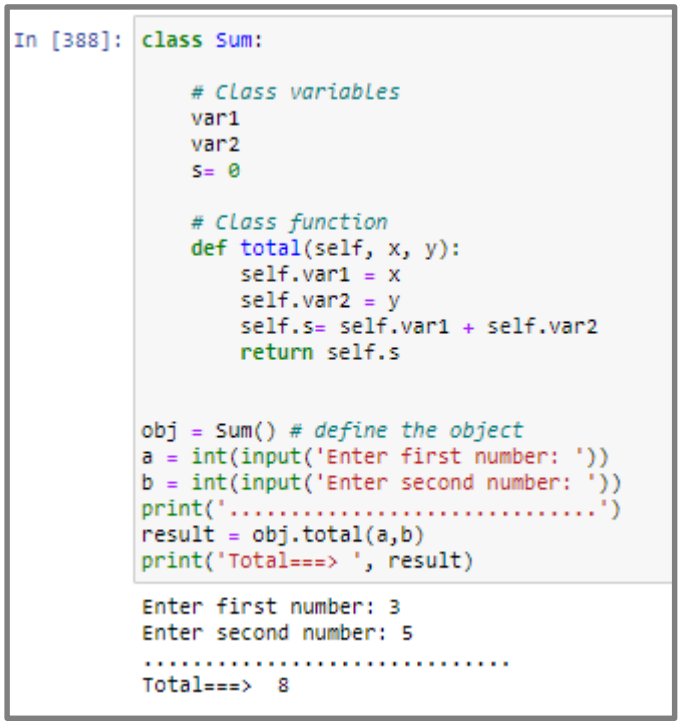
```
In [384]: class Car:  # Define class Car

              # class attribute
              model_name
              model_year
              model_color

              # instance attribute
              #The self keyword help to access the attribute of class
              def fun(self, mn , my, mc): # Function fun() definition
                  self.model_name = mn
                  self.model_year = my
                  self.model_color = mc
                  print('Model===> ', self.model_name)
                  print('Year===>', self.model_year)
                  print('Color===>', self.model_color)

          # instantiate the Parrot class
          obj = Car() # Create object obj for Car class
          var1 = input('Enter the model name: ')
          var2 = input('Enter the model year: ')
          var3 = input('Enter the color: ')
          print('.....................................................')
          obj.fun(var1, var2, var3) # Call function fun

          Enter the model name: Benz
          Enter the model year: 2021
          Enter the color: Blue
          .....................................................
          Model===>  Benz
          Year===> 2021
          Color===> Blue
```

Example: 2

```
class Sum:

   # Class variables
   var1
   var2
   s= 0

   # Class function
   def total(self, x, y):
      self.var1 = x
      self.var2 = y
      self.s= self.var1 + self.var2
      return self.s
```

```
obj = Sum() # define the object
a = int(input('Enter first number: '))
b = int(input('Enter second number: '))
print('.............................')
result = obj.total(a,b)
print('Total===> ', result)
```

Output:



3. Modules/Libraries:

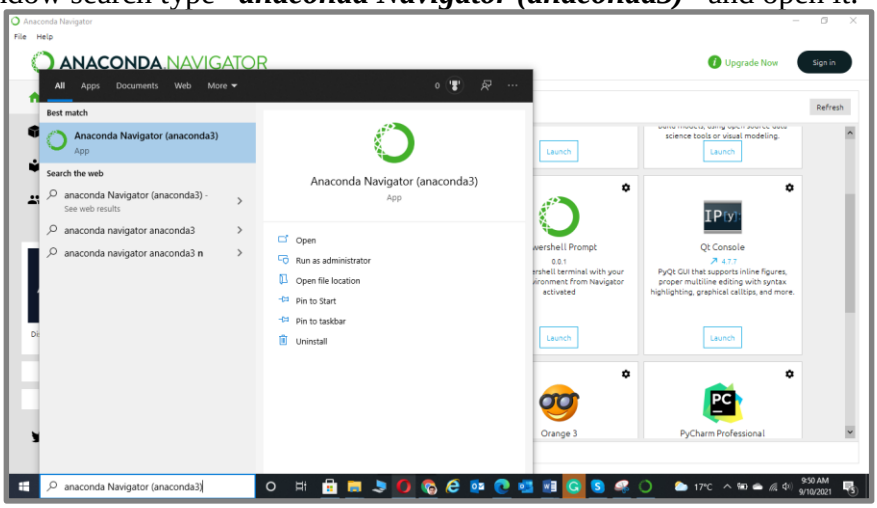
- Modules refer to a file/libraries containing ***Python statements and definitions***.
- We use modules to ***break down large programs*** into small ***manageable and organized files***.
- Furthermore, modules provide ***reusability of code***.
- We can define our most used ***functions in a module and import it***, instead of copying their definitions into different programs.
- There is a slight difference in module and libraries. A library is a collection of files while a module is a single file. We can say module is a subset of library. However, we considered the module as libraries.
- **Example of libraries:**

```
import myown_code            # My own reusable functions, variables, etc …
import NumPy                 # Package for scientific computation
import Tkinter               # Package for GUI
import scikit-learn          # Package for machine learning and AI
```

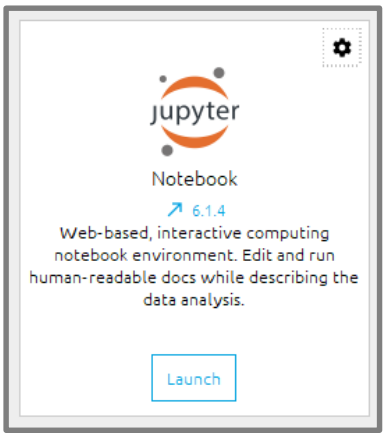
- There are two types of modules

3.1. **User define module:** Python support to create your own modules and reuse them. Following are the required steps.

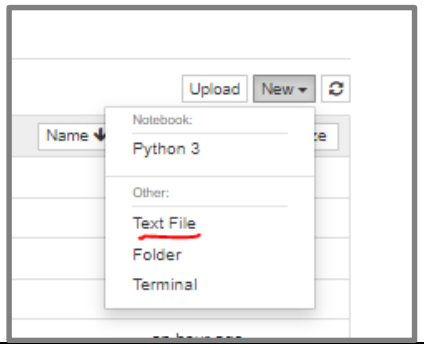
Step. 1. From window search type ***"anaconda Navigator (anaconda3)"*** and open it.



Step. 2. From anaconda navigator lunch ***Jupyter Notebook***.



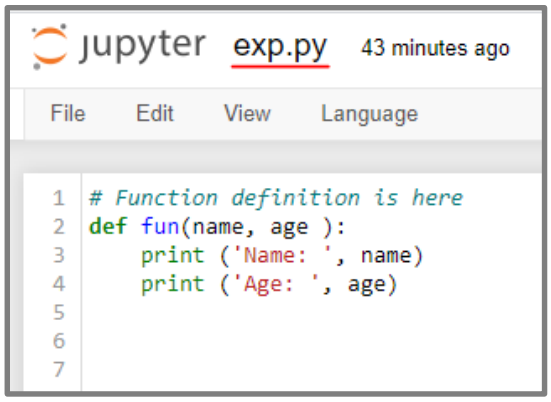
Step. 3. In Jupyter Notebook, click ***new*** and ***Text File***.



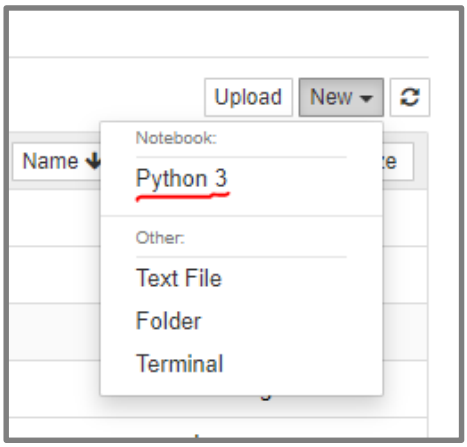
Step. 4. In the text file write your code. In our case we write the following code and save it as ***exp.py (i.e., it is our own defined name, you can use any name).***

```
# Function definition is here
def fun(name, age ):
    print ('Name: ', name)
    print ('Age: ', age)
```

It will look like the shown in figure below.



Step. 5. Now, from Jupyter Notebook open Python (i.e., Python 3 in our case)

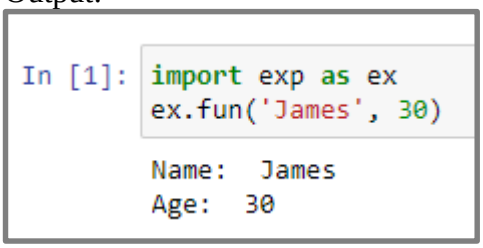


Step. 6. Import the module ***exp.py*** and run the program. Use the following code.

Example:

```
import exp as ex
ex.fun('James', 30)
```

Output:



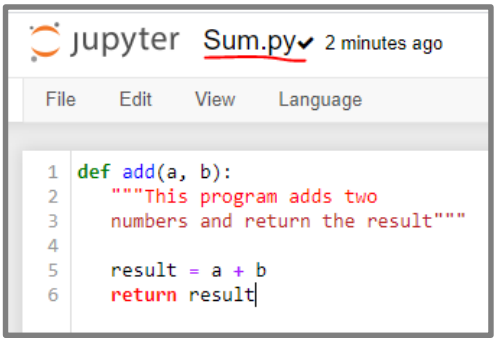
Example: 2
Put the following code in the text file and save it as ***Sum.py***

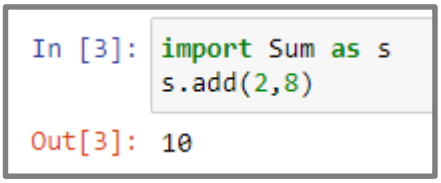
```
def add(a, b):
    """This program adds two
    numbers and return the result"""

    result = a + b
return result
```

Now, from python3, import the Sum.py module by running the following code.

```
import Sum as s
s.add(2,8)
```

Output:





3.2. **Built-in Modules/Libraries**

- Python has a huge collection of modules/libraries.
- .Some of the common libraries are presented as follows.

Library name	**Description**
Pillow	Python ***Image Library*** and can supports a lot of file types such as PDF, WebP, PCX, PNG, JPEG, GIF, PSD, WebP, PCX, GIF, IM, EPS, ICO, BMP, and many others as well.
Matplotlib	Matplotlib is a Python library that uses Python Script to write ***2-dimensional graphs and plots.***
Numpy	Numpy is a ***popular array*** – processing package of Python. It provides good support for different dimensional array objects as well as for matrices.
OpenCV Python	Open Source Computer Vision is a python package for ***image processing***.
Requests	Requests is a rich Python ***HTTP library***. Requests is focused on making HTTP requests more responsive and user-friendly.
Keras	Keras is an open-source ***deep neural network library***.
TensorFlow	TensorFlow is a free, open-source python ***machine learning library***.
Theano	Theano is a python library and a compiler for feasible computer programs – a.k.a an ***optimizing compiler***.
NLTK (Natural Language Toolkit)	NLTK a.k.a ***Natural language toolkit*** is one of the most popular python NLP libraries.
Arrow	Arrow is a practical python library. It is a friendly library that basically works with ***dates and times***.

FlashText	FlashText is another python library that offers easy ***search and replacement of words from documents.***
Scipy	Scipy is an open-source python library that is used for ***both scientific and technical computation.***
PyTorch	PyTorch is an open-source python ***machine learning library***.
Bokeh	Bokeh is a ***data visualization library*** for python. It allows interactive data visualization.
Pandas	It is a must to learn for ***data-science*** and dedicatedly written for Python language.
Scikit Learn	Scikit learn is a simple and useful python ***machine learning library***.
PyGame	It is a set of python functions and classes dedicated to writing ***video games*** mainly.

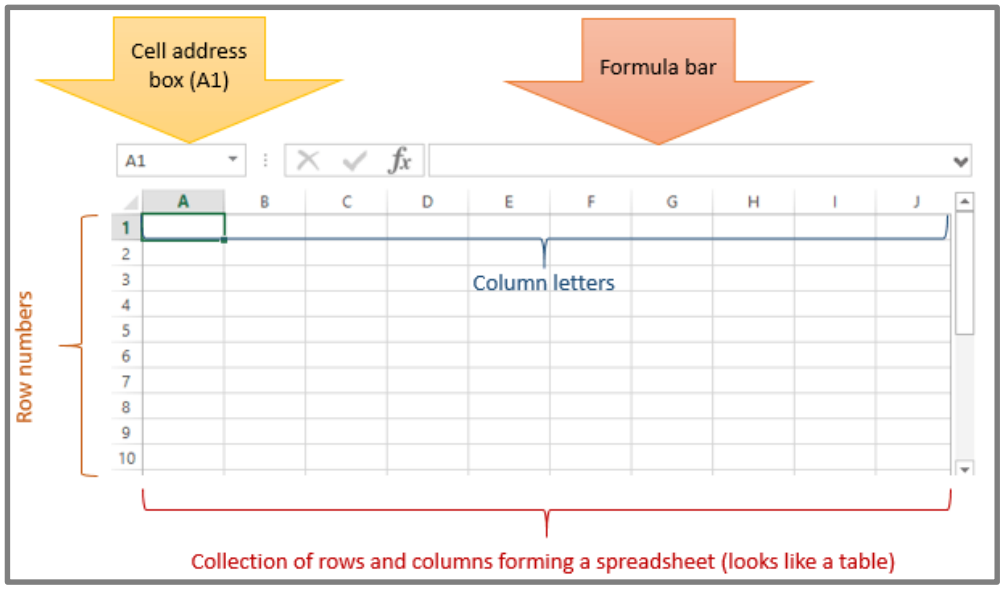
Further Information

https://www.python.org/

Microsoft Excel Lecture-9

1) Introduction
2) Basic components
3) Worksheets and workbook
4) Customization Microsoft Excel Environment
5) Important Shortcut keys
6) Math operations
7) Make column names bold
8) Align data to the left
9) Enclose data in boxes
10) Setting the print area and printing (Print View) & Page Layout
11) Data validation
12) Data filters
13) Group and Ungroup
14) Adding images to spreadsheets
15) Excel Formula
16) Excel Functions
17) Condition and logical statements (IF, AND, OR etc).

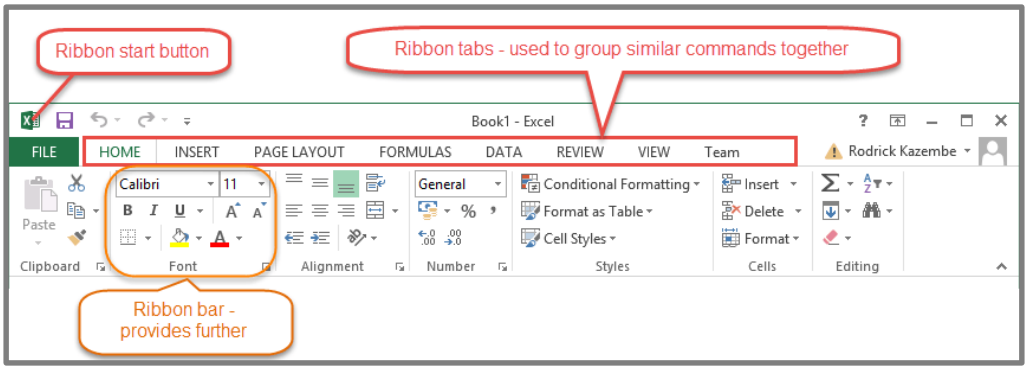
1. Introduction:
   - Microsoft Excel is a spreadsheet program used to ***record and analyze numerical and statistical data***.
     - It provides ***multiple features*** to perform various operations like ***calculations, tables, and graph tools***, etc.
     - Compatible with ***Windows, macOS, Android and iOS***
     - Basic layout of excel sheet.



2. **Basic components:**

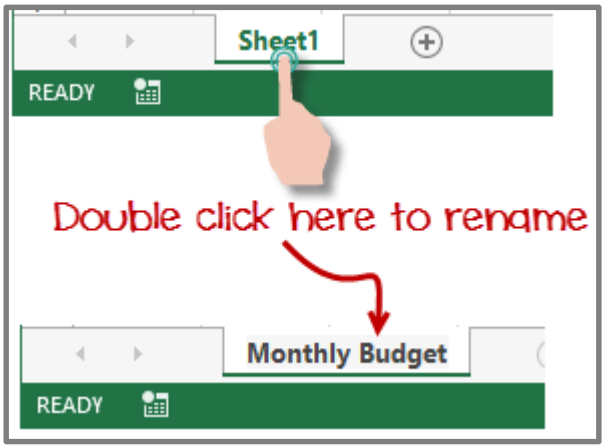
2.1. Understanding Ribbon: The ribbon provides shortcuts to commands in Excel. A command is an action that the user performs.

- **Ribbon start button** – It is used to access commands i.e. ***creating new documents, saving existing work, printing,*** accessing the options for customizing Excel, etc.
- **Ribbon tabs** – The tabs are used to ***group similar commands together***. The home tab is used for basic commands such as formatting the data to make it more presentable, sorting and finding specific data within the spreadsheet.
- **Ribbon bar** – The bars are used to group similar commands together such as ***Alignment ribbon bar is used to group all the commands that are used to align data together***.



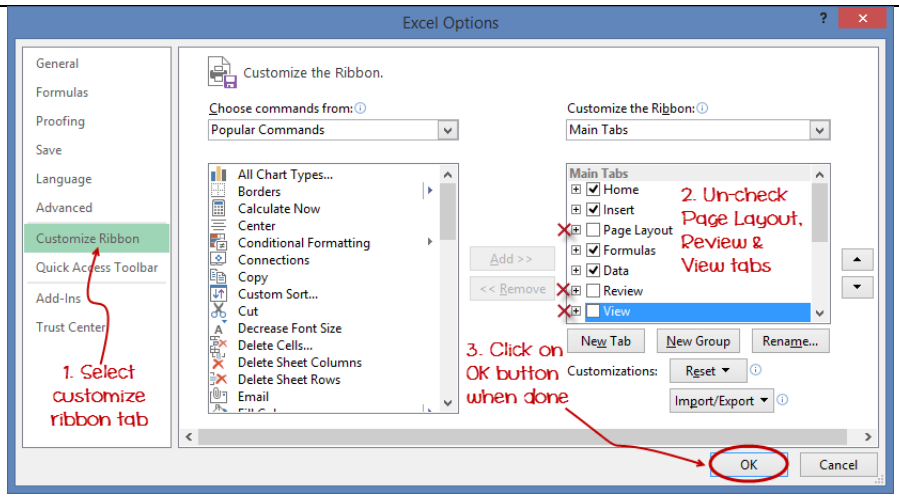
3. Worksheet and workbook:

- A worksheet is a collection of rows and columns. When a row and a column meet, they form a cell. Cells are used to record data.
- A workbook is a collection of worksheets.

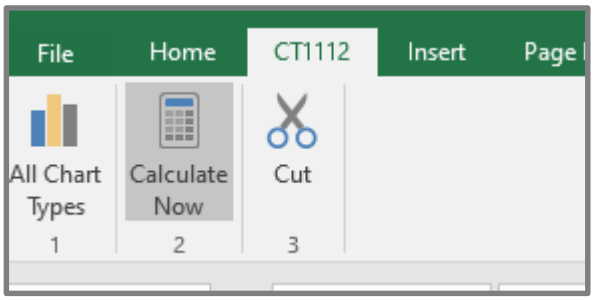


4. Customization Microsoft Excel Environment:

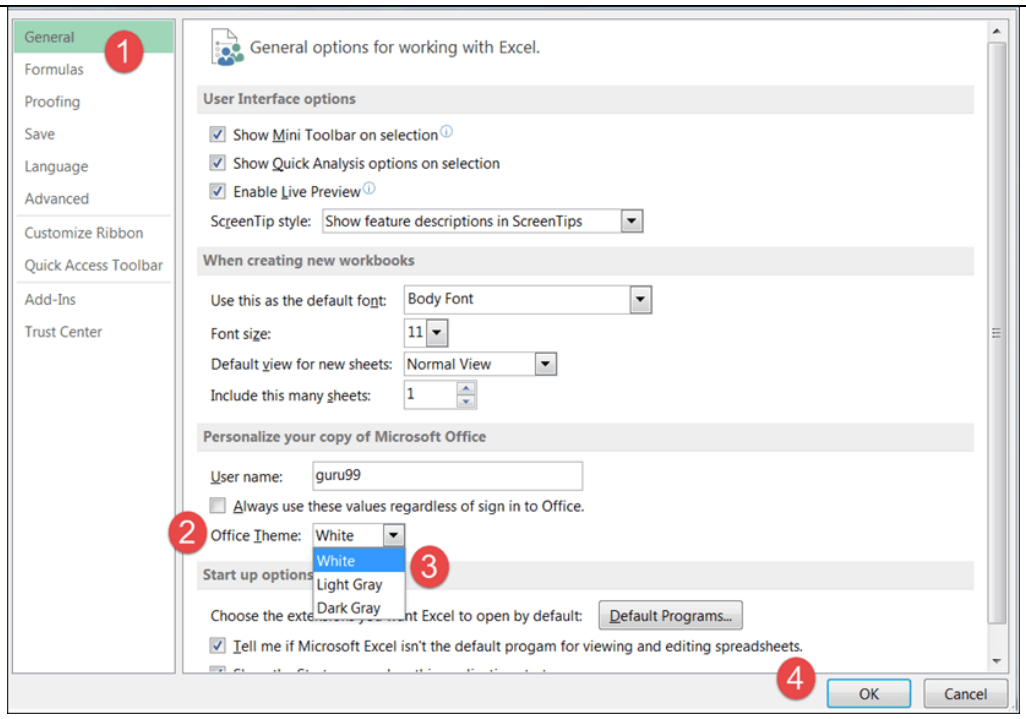
- Customization the ribbon: If you wish to customize the default setting of ribbon, click on ***file options and select the customize ribbon.***



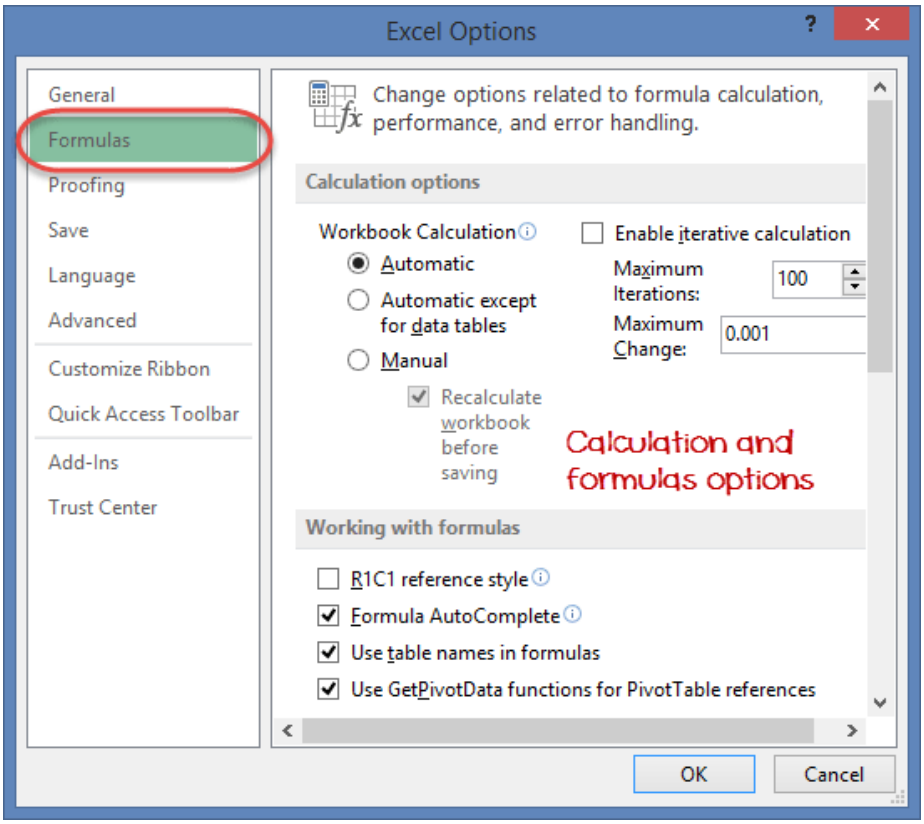
- **Adding custom tabs to the ribbon:** You can also add your own tab, give it a custom name and assign commands to it.
  Step. 1. Right click on the ribbon and select ***Customize the Ribbon***. The dialogue window shown above will appear.
  Step. 2. Click on ***new tab button***
  Step. 3. Select the ***newly created tab***
  Step. 4. Click on ***Rename button***
  Step. 5. Give it a name of ***CT1112***
  Step.6. Add icon and test



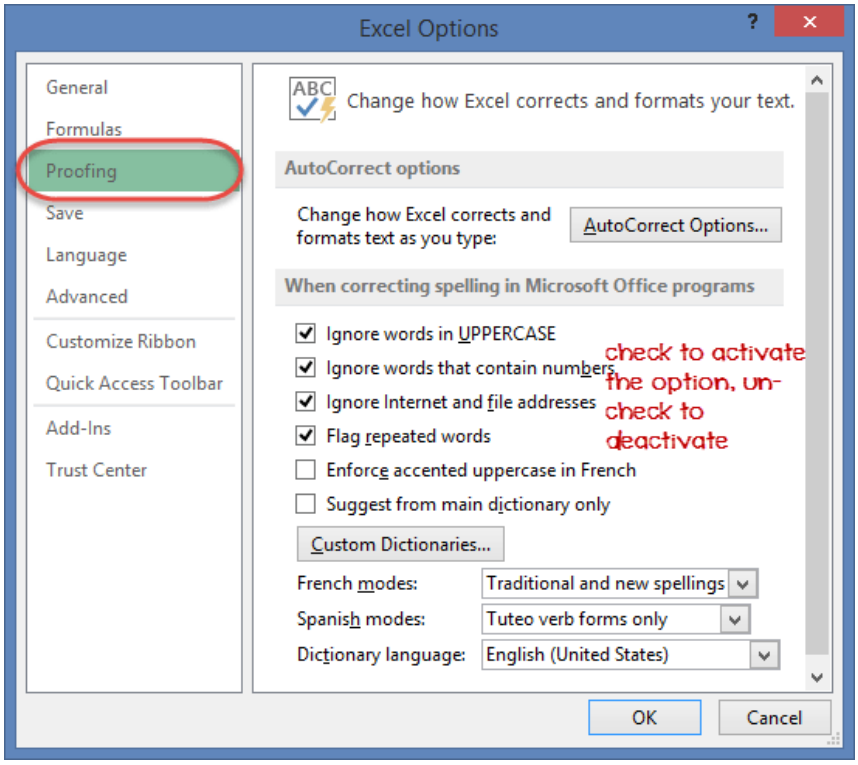
- **Setting the colour theme:** File→ Options→General



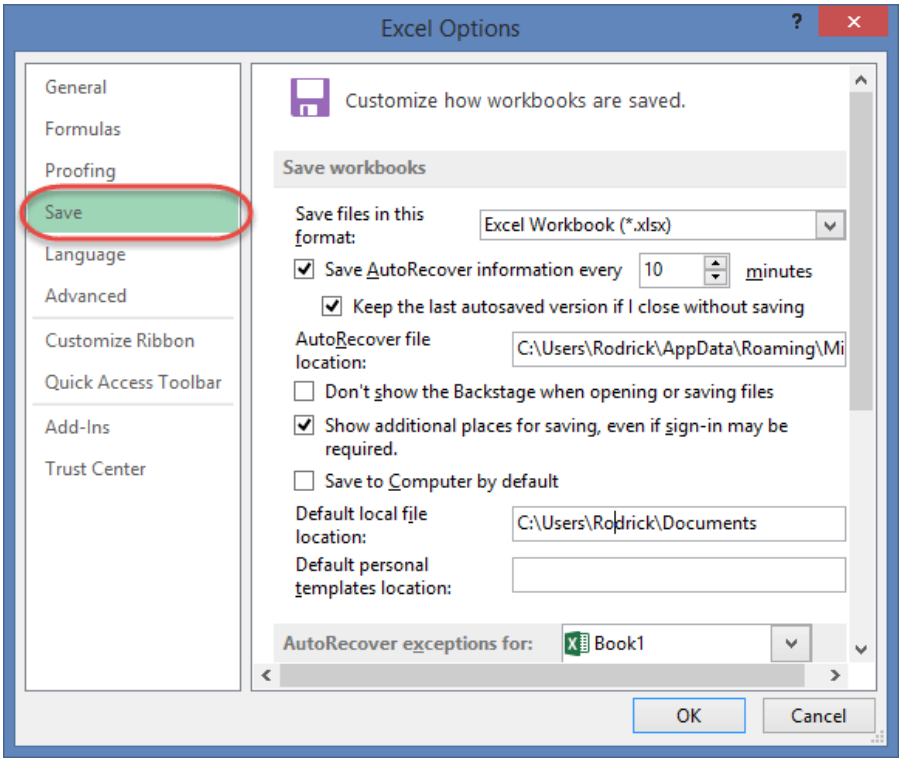
- **Settings for formulas: File→Options→Formula**



- **Proofing settings:**



- **Save settings:**



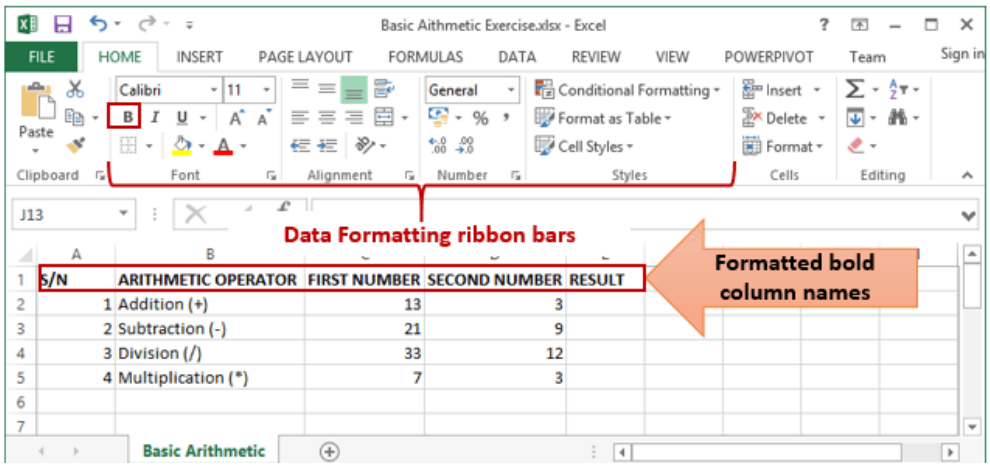
5. **Imported Shortcuts:**

Ctrl + P	used to open the print dialogue window
Ctrl + N	creates a new workbook
Ctrl + S	saves the current workbook
Ctrl + C	copy contents of current select
Ctrl + V	paste data from the clipboard
SHIFT + F3	displays the function insert dialog window
SHIFT + F11	Creates a new worksheet
F2	Check formula and cell range covered

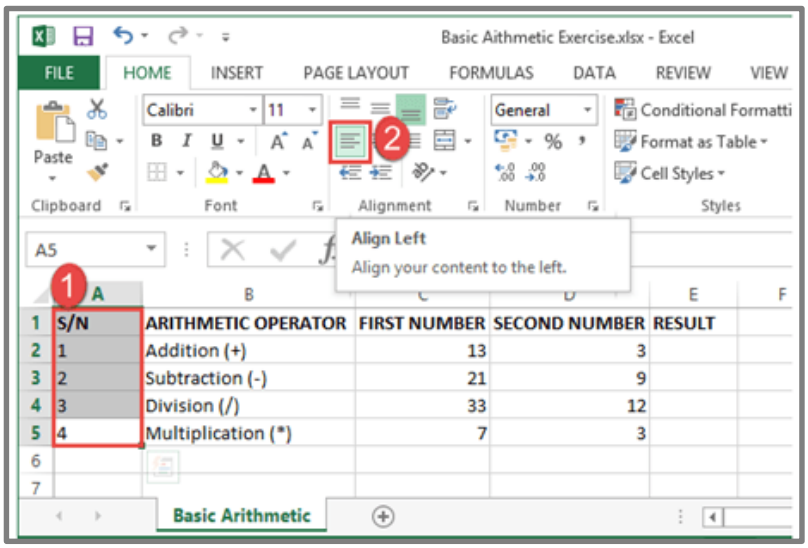
6. **Math operations:**

	A	B	C	D	E
	S/No	Arthmetic operators	First number	Second number	Result
	1	Addition (+)	30	5	35
	2	Subtraction(-)	10	2	8
	3	Multiplication(*)	2	15	30
	4	Division(/)	15	3	5

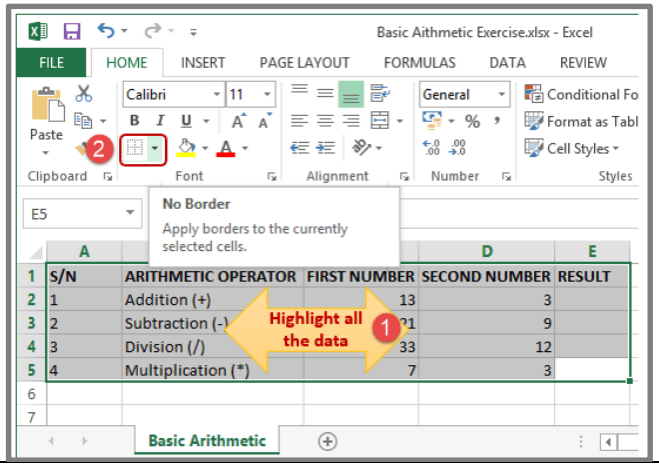
7. **Make column names bold**

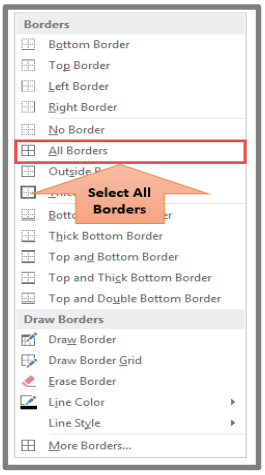


8. Align data to the left

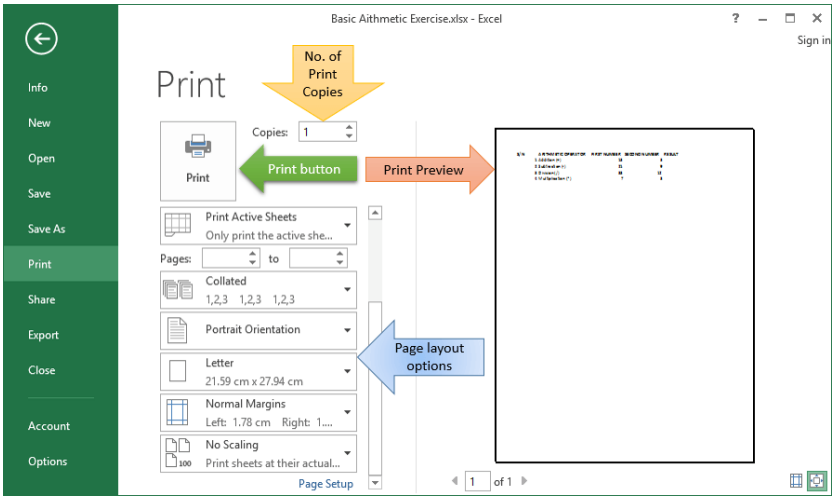


9. Enclose data in boxes



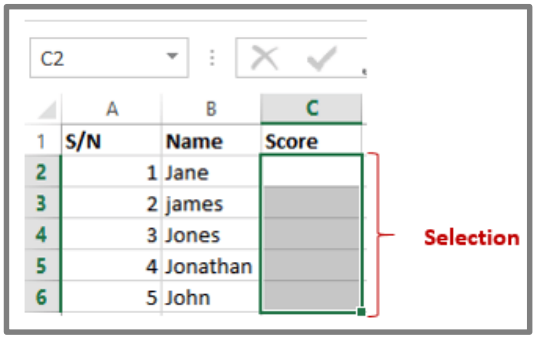


10. Setting the print area and printing (Print View) & Page Layout

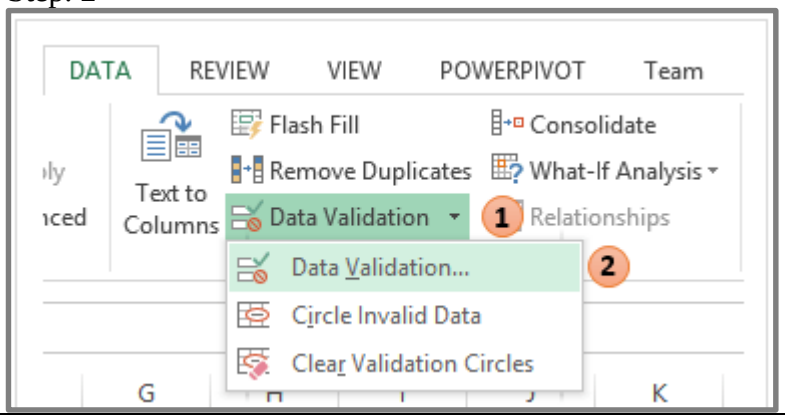


11. **Data validation:** Data validation is very important in the sense that it helps us avoid mistakes that can be avoided.
Example: Consider the following table where the student's marks minimum and maximum values are 0 and 100. To ensure the data must be between 0-100, we use the data validation feature.

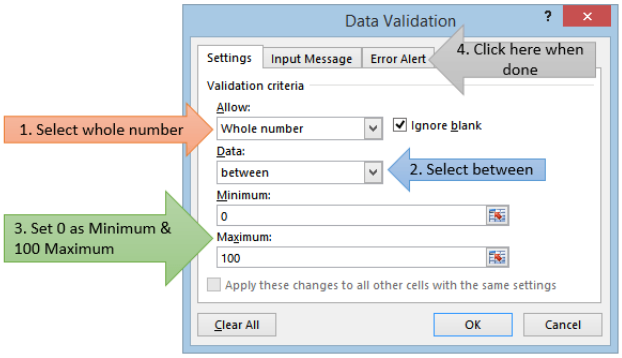
Step. 1



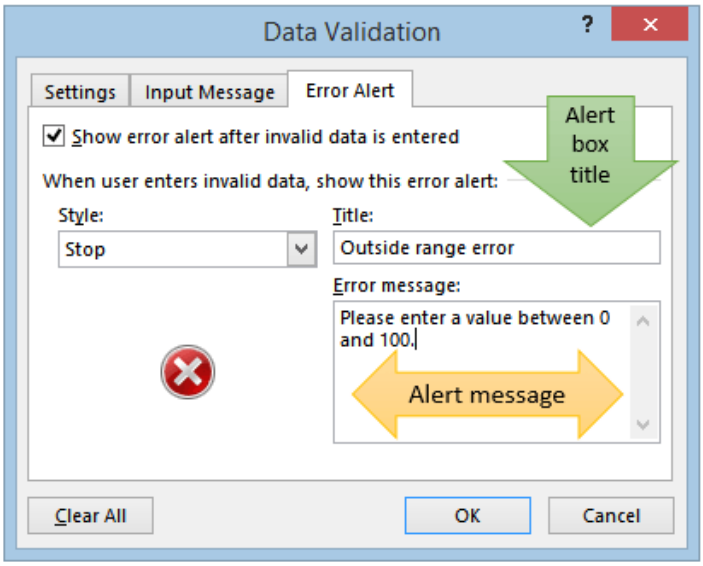
Step. 2



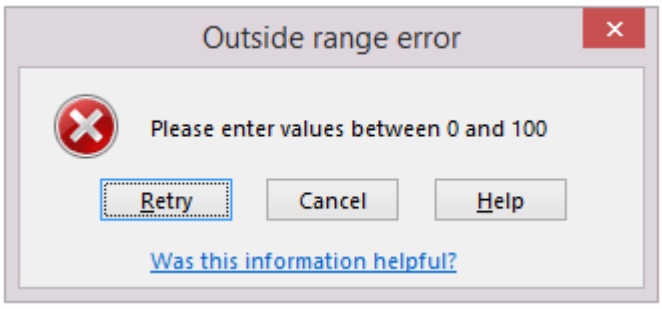
Step. 3



Step. 4

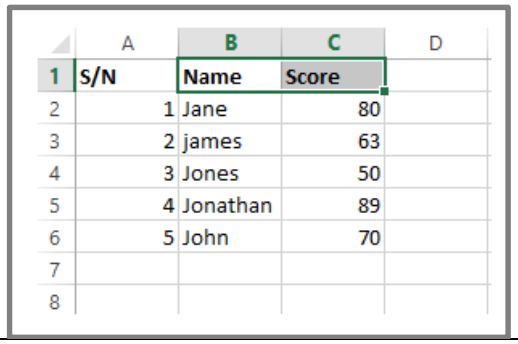


Step. 5. Try to enter 200



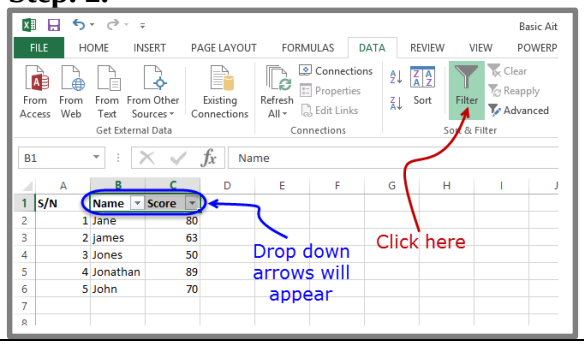
12. Data filters: Data filters allow us to get data that matches our desired criteria. Let's say we want to show the results of all the students whose names start with "ja". To do so, we use the following steps.

Step. 1.

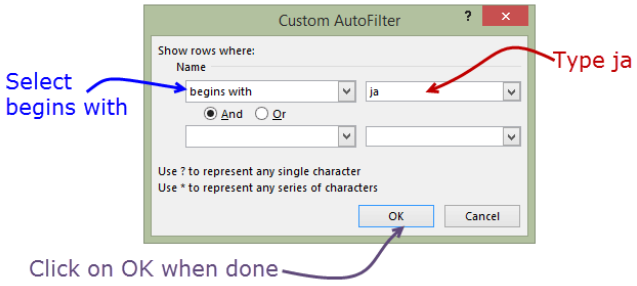


	A	B	C	D
1	S/N	Name	Score	
2	1	Jane	80	
3	2	james	63	
4	3	Jones	50	
5	4	Jonathan	89	
6	5	John	70	
7				
8				

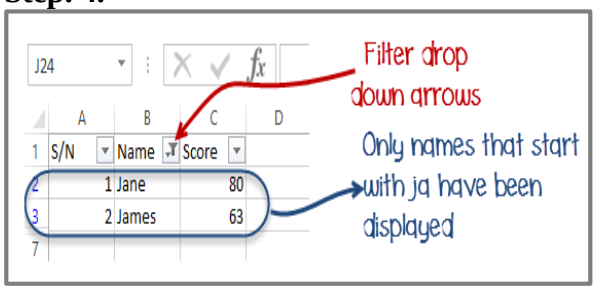
Step. 2.



Step. 3.

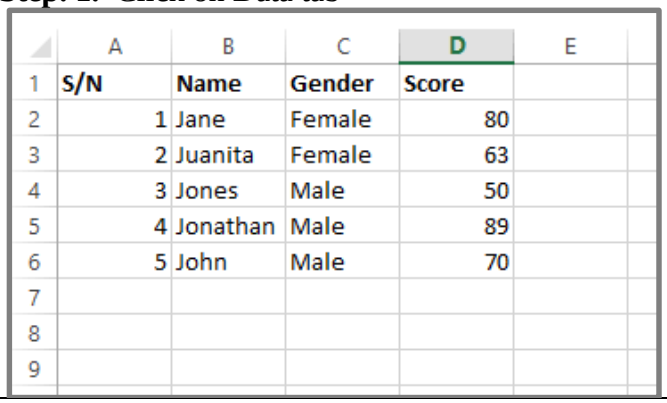


Step. 4.



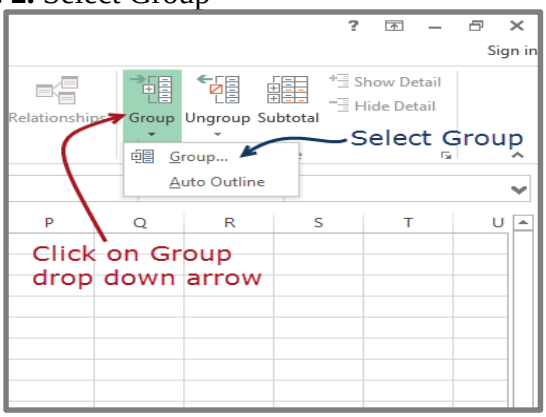
13. Group and Ungroup:

Step. 1. Click on Data tab

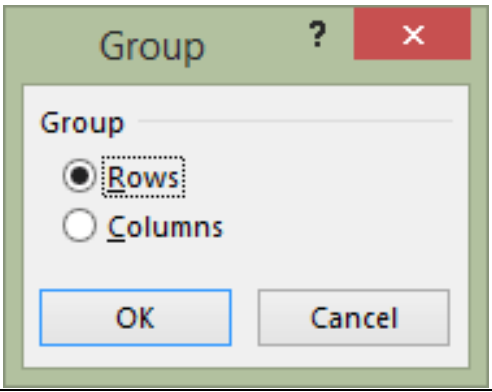


	A	B	C	D	E
1	S/N	Name	Gender	Score	
2	1	Jane	Female	80	
3	2	Juanita	Female	63	
4	3	Jones	Male	50	
5	4	Jonathan	Male	89	
6	5	John	Male	70	
7					
8					
9					

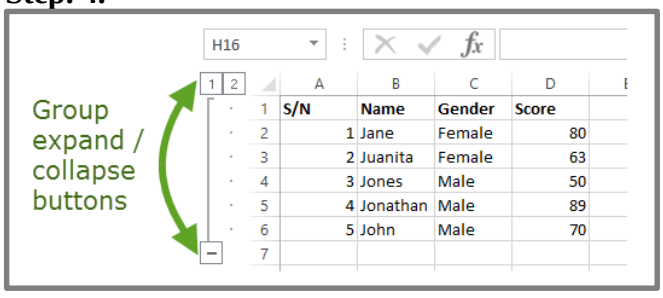
Step. 2. Select Group



Step.3.

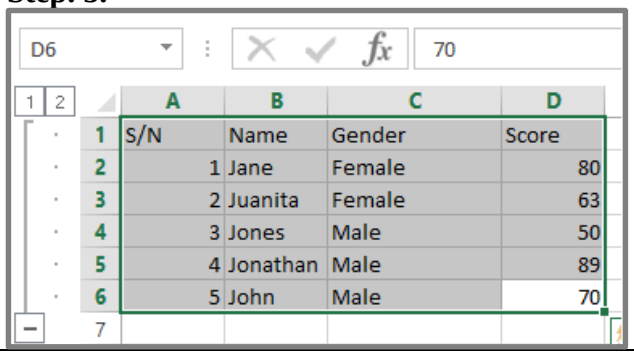


Step. 4.

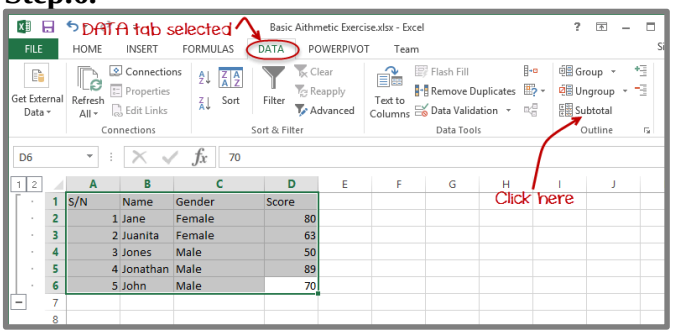


	A	B	C	D
1	S/N	Name	Gender	Score
2	1	Jane	Female	80
3	2	Juanita	Female	63
4	3	Jones	Male	50
5	4	Jonathan	Male	89
6	5	John	Male	70
7				

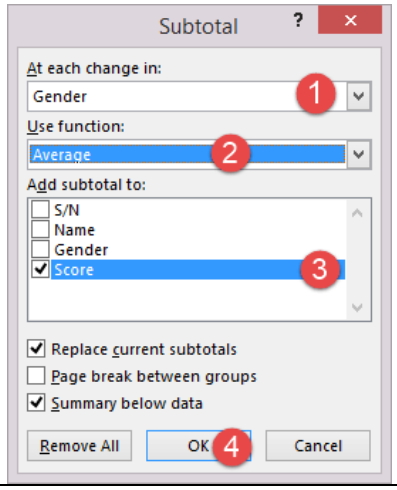
Step. 5.



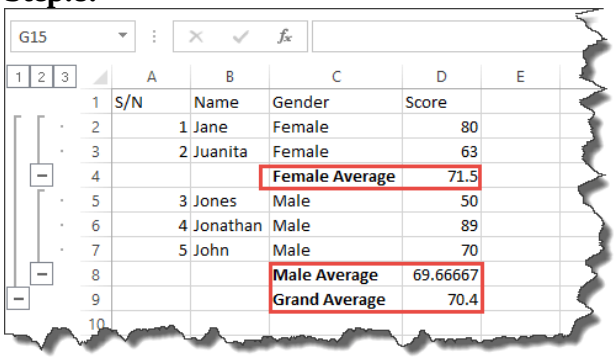
Step.6.



Step.7.

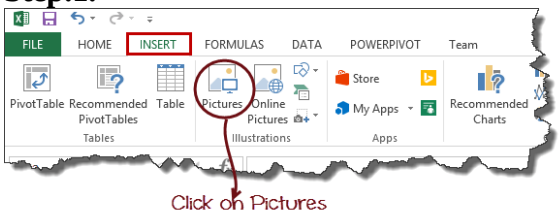


Step.8.

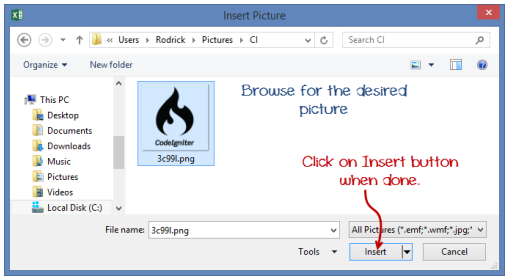


14. Adding images to spreadsheets:

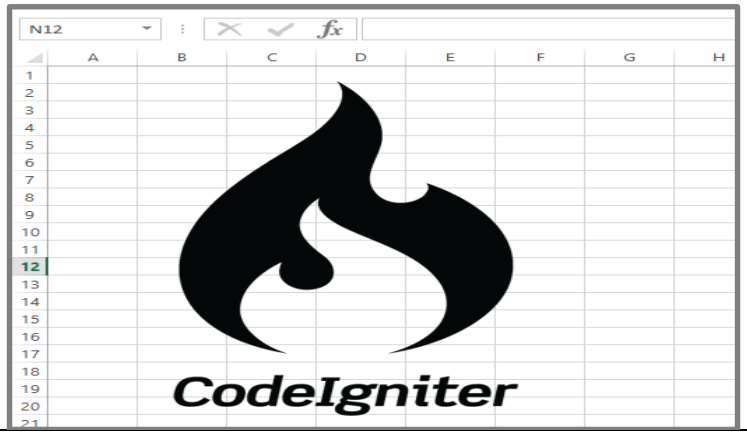
Step.1.



Step.2.



Step.3.

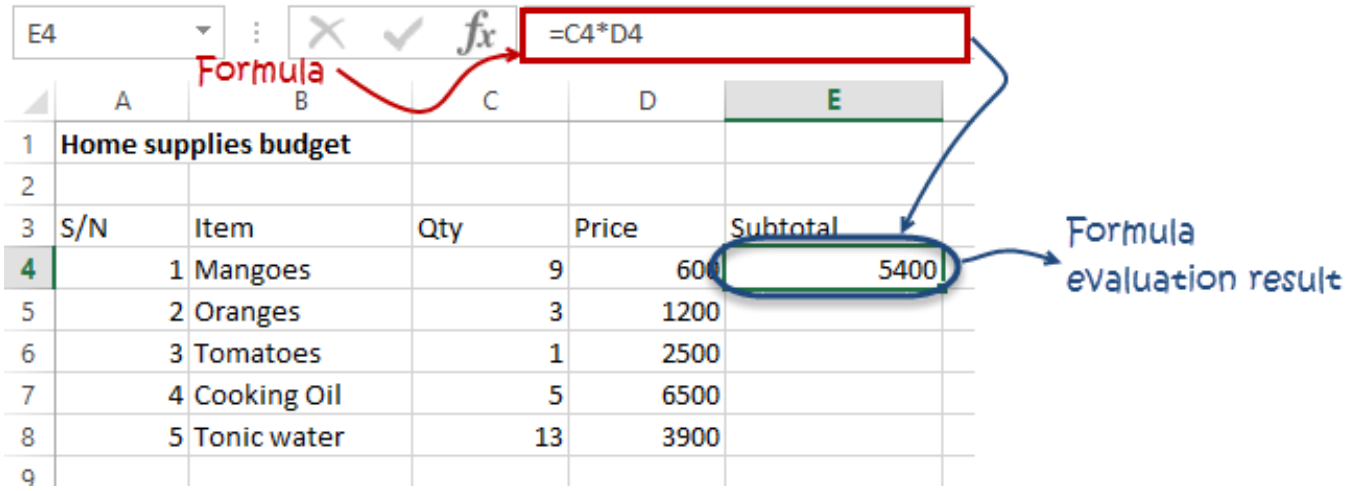


15. Excel Formulas: FORMULAS IN EXCEL is an expression that operates on values in a range of cell addresses and operators.

An example of a formula made up of discrete values lik **`=A2 * D2 / 2`**

- `"="` tells Excel that this is a formula, and it should evaluate it.
- `"A2" * D2"` makes reference to cell addresses A2 and D2 then multiplies the values found in these cell addresses.
- `"/"` is the division arithmetic operator
- `"2"` is a discrete value

Example:



	A	B	C	D	E
1	**Home supplies budget**				
2					
3	S/N	Item	Qty	Price	Subtotal
4	1	Mangoes	9	600	5400
5	2	Oranges	3	1200	
6	3	Tomatoes	1	2500	
7	4	Cooking Oil	5	6500	
8	5	Tonic water	13	3900	
9					

16. Excel Functions: FUNCTION IN EXCEL is a predefined formula that is used for specific values in a particular order.

Examples

- **SUM** for summation of a range of numbers
- **AVERAGE** for calculating the average of a given range of numbers
- **COUNT** for counting the number of items in a given range

- **Common functions.**

S/N	FUNCTION	CATEGORY	DESCRIPTION	USAGE
01	SUM	Math & Trig	Adds all the values in a range of cells	=SUM(E4:E8)
02	MIN	Statistical	Finds the minimum value in a range of cells	=MIN(E4:E8)
03	MAX	Statistical	Finds the maximum value in a range of cells	=MAX(E4:E8)
04	AVERAGE	Statistical	Calculates the average value in a range of cells	=AVERAGE(E4:E8)
05	COUNT	Statistical	Counts the number of cells in a range of cells	=COUNT(E4:E8)
06	LEN	Text	Returns the number of characters in a string text	=LEN(B7)
07	SUMIF	Math & Trig	Adds all the values in a range of cells that meet a specified criteria. =SUMIF(range,criteria,[sum_range])	=SUMIF(D4:D8,">=1000",C4:C8)
08	AVERAGEIF	Statistical	Calculates the average value in a range of cells that meet the specified criteria. =AVERAGEIF(range,criteria,[average_range])	=AVERAGEIF(F4:F8,"Yes",E4:E8)
09	DAYS	Date & Time	Returns the number of days between two dates	=DAYS(D4,C4)
10	NOW	Date & Time	Returns the current system date and time	=NOW()

- **Number functions**

S/N	FUNCTION	CATEGORY	DESCRIPTION	USAGE
1	ISNUMBER	Information	Returns True if the supplied value is numeric and False if it is not numeric	=ISNUMBER(A3)
2	RAND	Math & Trig	Generates a random number between 0 and 1	=RAND()
3	ROUND	Math & Trig	Rounds off a decimal value to the specified number of decimal points	=ROUND(3.14455,2)
4	MEDIAN	Statistical	Returns the number in the middle of the set of given numbers	=MEDIAN(3,4,5,2,5)
5	PI	Math & Trig	Returns the value of Math Function PI(π)	=PI()
6	POWER	Math & Trig	Returns the result of a number raised to a power. **POWER(number, power)**	=POWER(2,4)
7	MOD	Math & Trig	Returns the Remainder when you divide two numbers	=MOD(10,3)
8	ROMAN	Math & Trig	Converts a number to roman numerals	=ROMAN(1984)

- **Strng functions**

S/N	FUNCTION	CATEGORY	DESCRIPTION	USAGE	COMMENT
1	LEFT	Text	Returns a number of specified characters from the start (left-hand side) of a string	=LEFT("GURU99",4)	Left 4 Characters of "GURU99"
2	RIGHT	Text	Returns a number of specified characters from the end (right-hand side) of a string	=RIGHT("GURU99",2)	Right 2 Characters of "GURU99"
3	MID	Text	Retrieves a number of characters from the middle of a string from a specified start position and length. **=MID (text, start_num, num_chars)**	=MID("GURU99",2,3)	Retrieving Characters 2 to 5
4	ISTEXT	Information	Returns True if the supplied parameter is Text	=ISTEXT(value)	value – The value to check.
5	FIND	Text	Returns the starting position of a text string within another text string. This function is case-sensitive. **=FIND(find_text, within_text, [start_num])**	=FIND("oo","Roofing",1)	Find oo in "Roofing", Result is 2
6	REPLACE	Text	Replaces part of a string with another specified string. **=REPLACE (old_text, start_num, num_chars, new_text)**	=REPLACE("Roofing",2,2,"xx")	Replace "oo" with "xx"

- **Date time functions**

S/N	FUNCTION	CATEGORY	DESCRIPTION	USAGE
1	DATE	Date & Time	Returns the number that represents the date in excel code	=DATE(2015,2,4)
2	DAYS	Date & Time	Find the number of days between two dates	=DAYS(D6,C6)
3	MONTH	Date & Time	Returns the month from a date value	=MONTH("4/2/2015")
4	MINUTE	Date & Time	Returns the minutes from a time value	=MINUTE("12:31")
5	YEAR	Date & Time	Returns the year from a date value	=YEAR("04/02/2015")

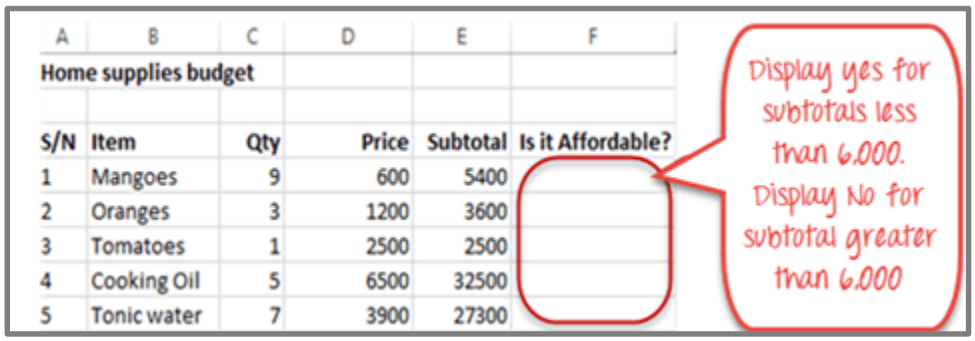
Microsoft Excel Lecture-9

1) Condition and logical statements (IF, AND, OR etc).
2) Charts
3) Advance charts
4) XML data
5) CVS data
6) Data entry form
7) Solver

help

1. **Condition and logical statements:** This feature us to take decision if a condition is true or false while executing the formulations and functions. Excel support the ***IF conditional statement***.

   Example: Consider the data given in Figure below. We use the ***IF function to determine if an item is expensive or not,*** assuming that items with a value ***greater than 6,000 are expensive.***

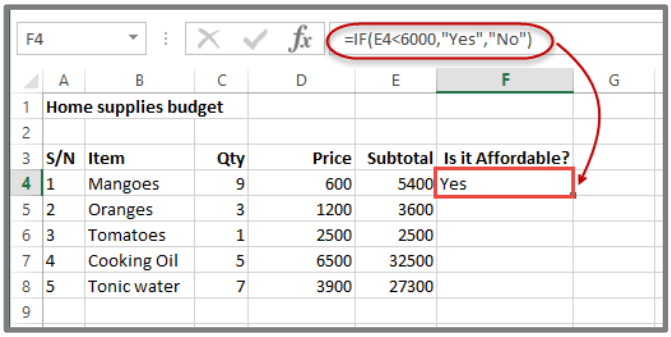


	A	B	C	D	E	F
1	Home supplies budget					
2						
3	S/N	Item	Qty	Price	Subtotal	Is it Affordable?
4	1	Mangoes	9	600	5400	
5	2	Oranges	3	1200	3600	
6	3	Tomatoes	1	2500	2500	
7	4	Cooking Oil	5	6500	32500	
8	5	Tonic water	7	3900	27300	

Here is the IF conditional statement

=IF(E4<6000,"Yes","No")

- **"=IF(…)"** calls the IF functions
- **"E4<6000"** is the condition that the IF function evaluates. It checks the value of cell address E4 (subtotal) is less than 6,000
- **"Yes"** this is the value that the function will display if the value of E4 is less than 6,000 True condition
- **"No"** this is the value that the function will display if the value of E4 is greater than 6,000 False condition



	A	B	C	D	E	F	G
1	Home supplies budget						
2							
3	S/N	Item	Qty	Price	Subtotal	Is it Affordable?	
4	1	Mangoes	9	600	5400	Yes	
5	2	Oranges	3	1200	3600		
6	3	Tomatoes	1	2500	2500		
7	4	Cooking Oil	5	6500	32500		
8	5	Tonic water	7	3900	27300		
9							

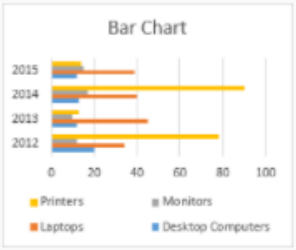
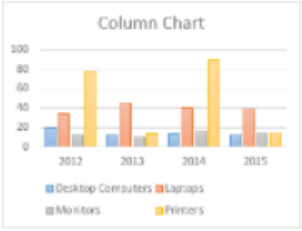
List of logical functions:

S/N	FUNCTION	CATEGORY	DESCRIPTION	USAGE
01	AND	Logical	Checks multiple conditions and returns true if they all the conditions evaluate to true.	=AND(1 > 0,ISNUMBER(1)) The above function returns TRUE because both Condition is True.
02	FALSE	Logical	Returns the logical value FALSE. It is used to compare the results of a condition or function that either returns true or false	FALSE()
03	IF	Logical	Verifies whether a condition is met or not. If the condition is met, it returns true. If the condition is not met, it returns false. **=IF(logical_test,[value_if_true], [value_if_false])**	=IF(ISNUMBER(22),"Yes", "No") 22 is Number so that it return Yes.
04	IFERROR	Logical	Returns the expression value if no error occurs. If an error occurs, it returns the error value	=IFERROR(5/0,"Divide by zero error")
05	IFNA	Logical	Returns value if #N/A error does not occur. If #N/A error occurs, it returns NA value. #N/A error means a value if not available to a formula or function.	=IFNA(D6*E6,0) N.B the above formula returns zero if both or either D6 or E6 is/are empty
06	NOT	Logical	Returns true if the condition is false and returns false if condition is true	=NOT(ISTEXT(0)) N.B. the above function returns true. This is because ISTEXT(0) returns false and NOT function converts false to TRUE
07	OR	Logical	Used when evaluating multiple conditions. Returns true if any or all of the conditions are true. Returns false if all of the conditions are false	=OR(D8="admin",E8="cashier") N.B. the above function returns true if either or both D8 and E8 admin or cashier
08	TRUE	Logical	Returns the logical value TRUE. It is used to compare the results of a condition or function that either returns true or false	TRUE()

2. **Charts:** A chart is a visual representative of data in both columns and rows. Charts are usually used to analyse trends and patterns in data sets.
   Example: Consider the following data table. The different types of charts are then illustrated, subsequently.

Item	2012	2013	2014	2015
Desktop Computers	20	12	13	12
Laptops	34	45	40	39
Monitors	12	10	17	15
Printers	78	13	90	14

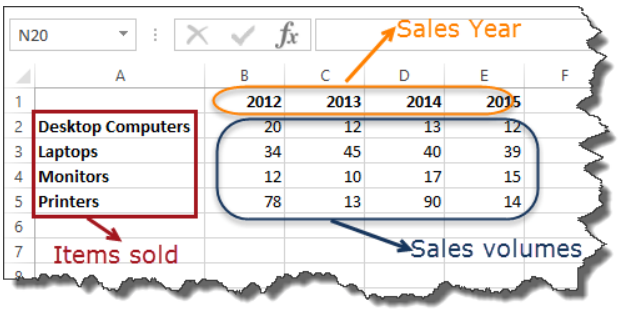
Different types of charts in Excel

S/N	CHART TYPE	WHEN SHOULD I USE IT?	EXAMPLE
1	Pie Chart	When you want to quantify items and show them as percentages.	
2	Bar Chart	When you want to compare values across a few categories. The values run horizontally	
3	Column chart	When you want to compare values across a few categories. The values run vertically	
4	Line chart	When you want to visualize trends over a period of time i.e. months, days, years, etc.	
5	Combo Chart	When you want to highlight different types of information	

Creating charts: The steps used to create charts in excel

Step.1.

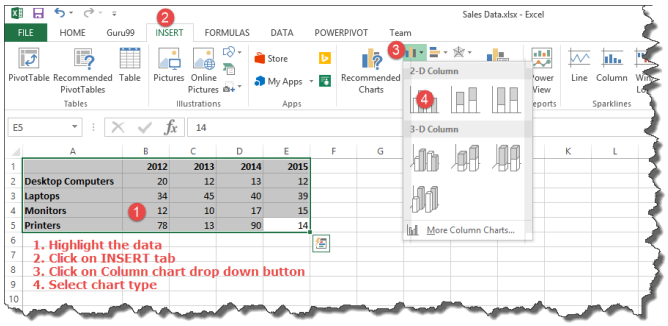
- Open Excel
- Enter the data from the sample data table above
- Your workbook should now look as follows



	A	B	C	D	E
1		2012	2013	2014	2015
2	Desktop Computers	20	12	13	12
3	Laptops	34	45	40	39
4	Monitors	12	10	17	15
5	Printers	78	13	90	14

Step. 2.

- Select the data you want to represent in graph
- Click on INSERT tab from the ribbon
- Click on the Column chart drop down button
- Select the chart type you want



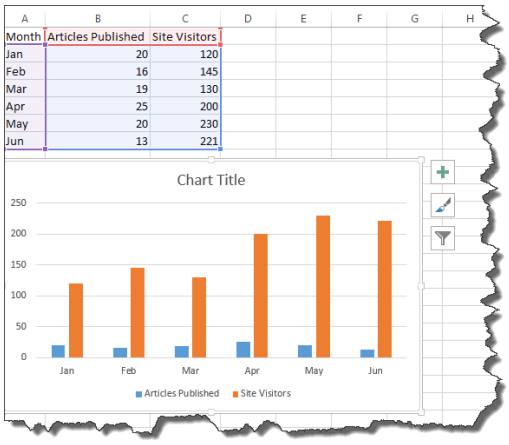
3. Advance charts: To create advanced charts, we consider the following data table.

Month	Articles Published	Site Visitors
Jan	20	120
Feb	16	145
Mar	19	130
Apr	25	200
Jun	20	230
Jul	13	221

Example:

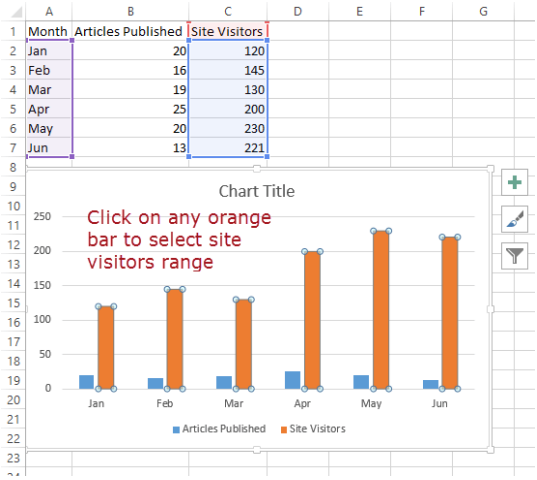
Step 1.

- Create a new workbook in Excel.
- Enter the data shown above
- Create a basic column chart as shown below. If you do not know how to create a basic chart, then read the article on charts.



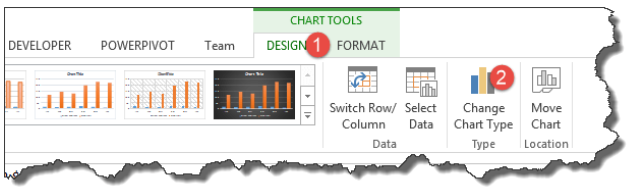
Step.2.

- Now, it's time for our charts and complex graphs in Excel to take beyond the basics.
- Select the orange bars representing traffic

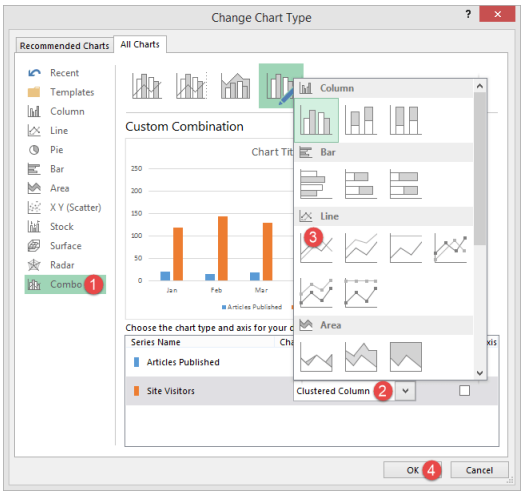


Step. 3.

- Click on change chart type as shown below

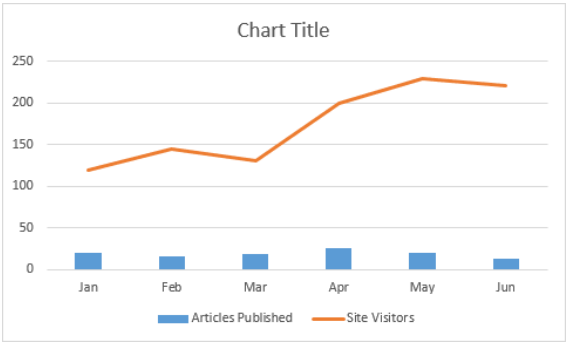


- You will get the following dialog window

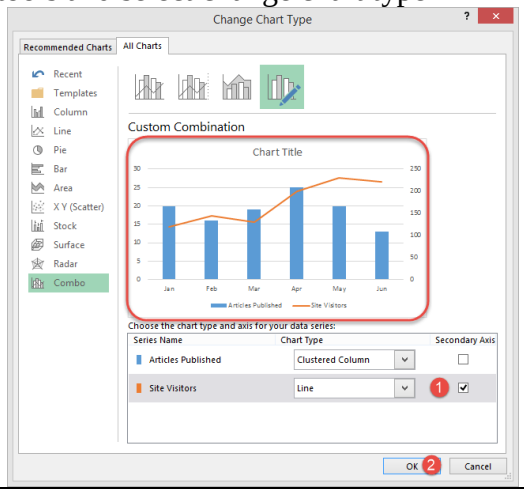


Step.4. Select Combo and,

- Click on the clustered column
- Select Line chart
- Click on OK button, you will see the following figure.



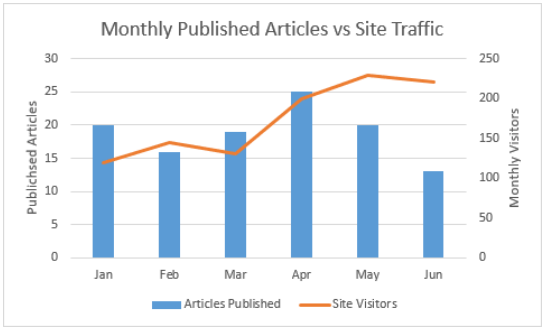
- Select the chart
- Click on Design under chart tools and select change chart type



Step. 5.

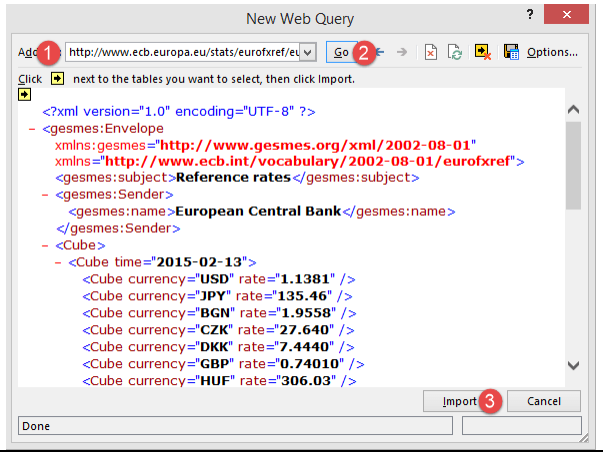
- Edit the chart through the steps shown in figure below.

- Finally, you will get the chart given below.

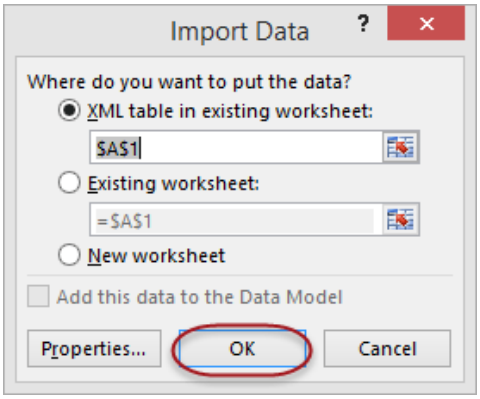


4. **XML(Website) data:** Consider the http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml

- Open a new workbook
- Click on the DATA tab on the ribbon bar → Get External Data
- Click on "From Web" button
- You will get the following window



- Enter http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml in the address
- Click on Go button, you will get the XML data preview
- Click on Import button when done



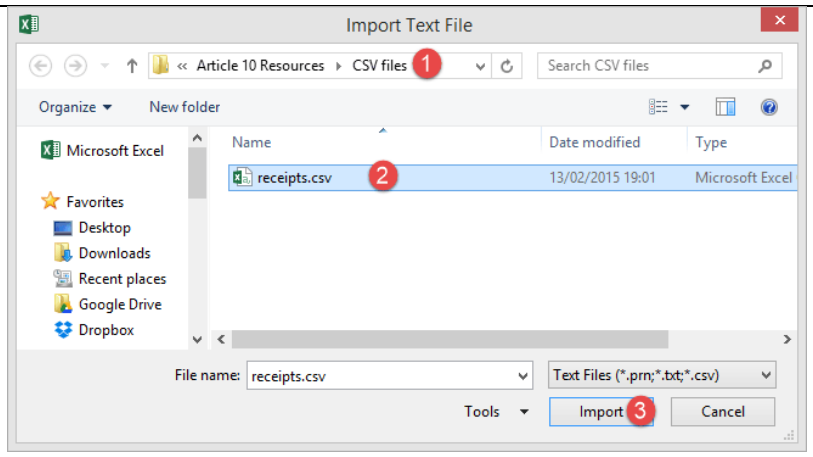
- Click on OK button
- You will get the following Excel import XML data

	A	B	C	D	E	F
1	ns1:subject	ns1:name	time	currency	rate	
2	Reference rates	European Central Bank	13/02/2015	USD	1.1381	
3	Reference rates	European Central Bank	13/02/2015	JPY	135.46	
4	Reference rates	European Central Bank	13/02/2015	BGN	1.9558	
5	Reference rates	European Central Bank	13/02/2015	CZK	27.64	
6	Reference rates	European Central Bank	13/02/2015	DKK	7.444	
7	Reference rates	European Central Bank	13/02/2015	GBP	0.7401	
8	Reference rates	European Central Bank	13/02/2015	HUF	306.03	
9	Reference rates	European Central Bank	13/02/2015	PLN	4.1768	
10	Reference rates	European Central Bank	13/02/2015	RON	4.4431	
11	Reference rates	European Central Bank	13/02/2015	SEK	9.5887	
12	Reference rates	European Central Bank	13/02/2015	CHF	1.0576	
13	Reference rates	European Central Bank	13/02/2015	NOK	8.6535	
14	Reference rates	European Central Bank	13/02/2015	HRK	7.7128	
15	Reference rates	European Central Bank	13/02/2015	RUB	72.989	

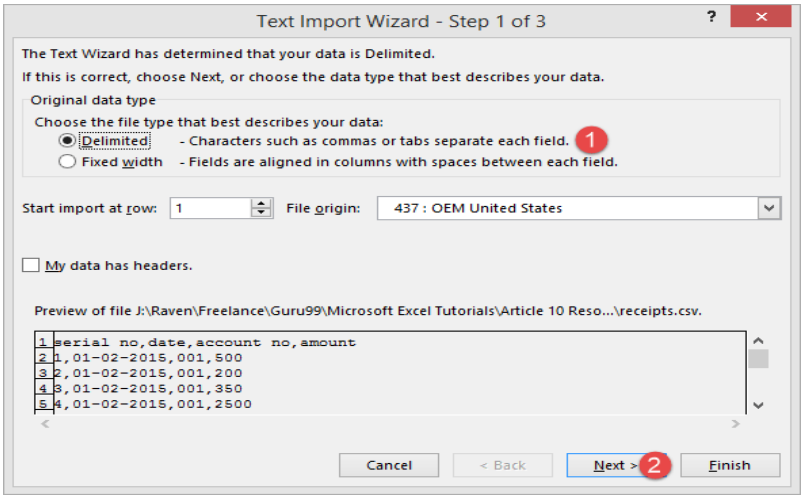
1. **CSV data.** Download any CVS (Comma Separated Values) file from internet and follow the following steps.

Steps. 1.

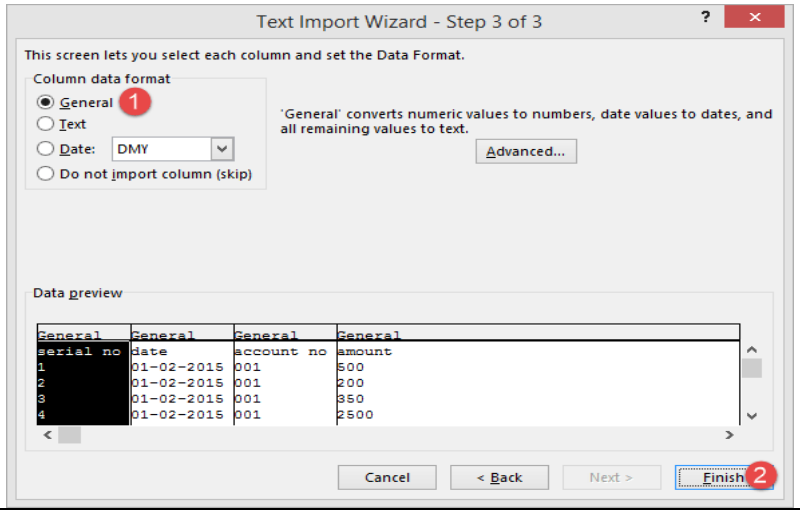
2. Open a new workbook
3. Click on DATA tab on the ribbon
4. Click on From Text button
5. You will get the following window
6. Browse to the folder where you downloaded the CSV file
7. Select da.csv file
8. Click on Import button
9. You will get the following import text file wizard



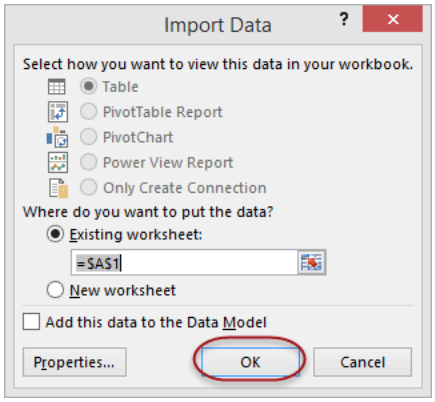
- Click on Next button
- Select Comma on the Delimiters panel
- Click on Next button



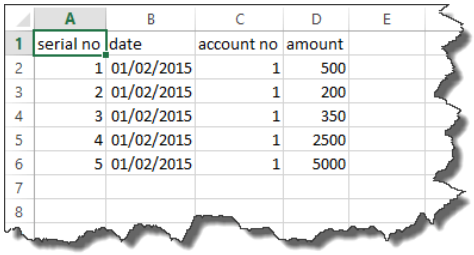
- Click on Finish button



- Click on OK button



- You will get the data, look like as given in the following

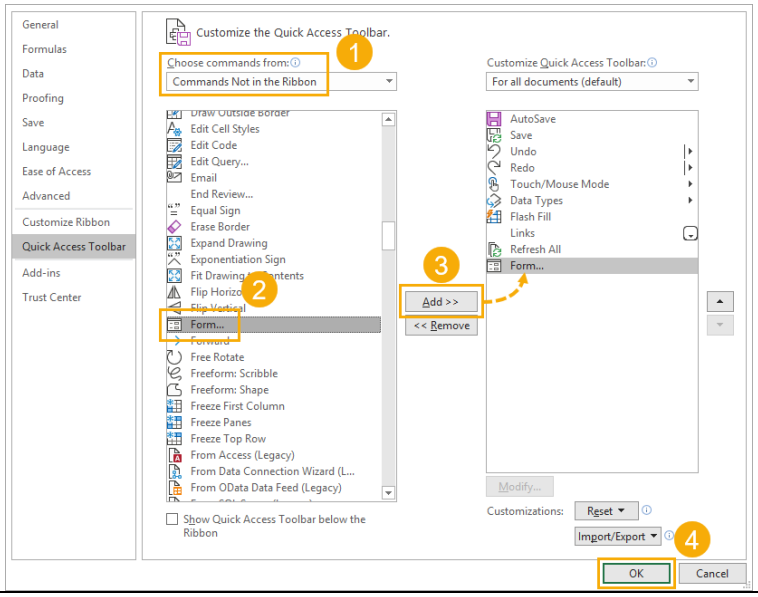


	A	B	C	D	E
1	serial no	date	account no	amount	
2	1	01/02/2015	1	500	
3	2	01/02/2015	1	200	
4	3	01/02/2015	1	350	
5	4	01/02/2015	1	2500	
6	5	01/02/2015	1	5000	
7					
8					

6. Data Entry Form:

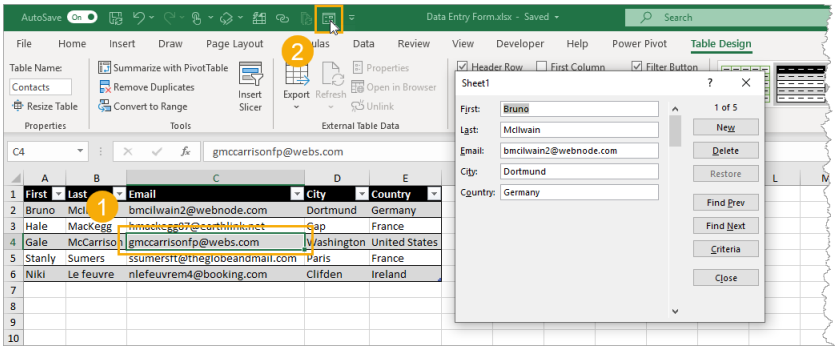
Steps. 1.

- Right click anywhere on the quick quick access toolbar.
- Select ***Customize Quick Access Toolbar*** from the menu options.



Step.2.

- Select Commands Not in the Ribbon.
- Select Form from the list of available commands. Press F to jump to the commands starting with F.
- Press the Add button to add the command into the quick access toolbar.
- Press the OK button.
- Select a cell inside the data which we want to create a data entry form with.
- Click on the Form icon in the quick access toolbar area.



Components of the form:

- **New:** This will clear any existing data in the form and allows you to create a new record.
- **Delete:** This will allow you to delete an existing record.
- **Restore:** If you're editing an existing entry, you can restore the previous data in the form (if you haven't clicked New or hit Enter).
- **Find Prev:** This will find the previous entry.
- **Find Next:** This will find the next entry.
- **Criteria:** This allows you to find specific records.
- **Close:** This will close the form.
- **Scroll Bar:** You can use the scroll bar to go through the records.

7. Excel Solver:

- Solver is a Microsoft Excel add-in program that find an ***optimal (maximum or minimum).***
- The formula cell is known as ***objective cell*** which is subject to ***constraints, or limits***, on the values of other formula cells on a worksheet.

Optimization problem:

- Optimization problem is the problem of finding the ***best solution*** from all ***feasible solutions*** under the ***bounded constraints***. OR
- A ***computational problem*** in which the ***object*** is to find the best of all possible solutions.
- ***Mathematical relationships*** between the ***objective and constraints and the decision variables*** is hard to solve; therefore, designated ***algorithms (Solver)*** that comprehensively solve such complex relationship are used.

Problem statement:

A corporation plans on building a maximum of 11 new stores in a large city. They will build these stores in one of three sizes for each location – a **convenience store** (open 24 hours), **standard store**, and an **expanded services** store. The **convenience store** requires **$4.125 million** to build and **30 employees** to

operate. The **standard store** requires **$8.25 million** to build and **15 employees** to operate. The **expanded-services store** requires **$12.375 million** to build and **45 employees** to operate. The **corporation** can dedicate **$82.5 million** in construction capital, and **300 employees** to staff the stores. On the average, the **convenience store nets $1.2 million** annually, the **standard store nets $2 million** annually, and the **expanded services store nets $2.6 million** annually. **How many of each should they build to maximize revenue?**

Assign variables:

x_1: Number of convenience stores
x_2: Number of standard stores
x_3: Number of expanded stores

Constraints:

a. $x_1 + x_2 + x_3 \leq 11$
b. $4.125x_1 + 8.25x_2 + 12.375x_3 \leq 82.5$
c. $30x_1 + 15x_2 + 45x_3 \leq 300$
d. ~~$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$~~ x1≥0, x2≥ 0, x3≥0

Profit/Revenue function:

$N(x_1, x_2, x_3) = 1.2x_1 + 2x_2 + 2.6x_3$

Objective function:

$$\max_{x_1,x_2,x_3} (N)$$

Decision variables: x_1, x_2, x_3

Solution: First of all, add the solver in the excel following the following steps.

1. In Excel 2010 and later, go to **File** > **Options**

   Note: For Excel 2007, click the **Microsoft Office Button**  , and then click **Excel Options**.

2. Click **Add-Ins**, and then in the **Manage** box, select **Excel Add-ins**.

3. Click **Go**.

4. In the **Add-Ins available** box, select the **Solver Add-in** check box, and then click **OK**.

   Notes:

   - If the **Solver Add-in** is not listed in the **Add-Ins available** box, click **Browse** to locate the add-in.
   - If you get prompted that the Solver Add-in is not currently installed on your computer, click **Yes** to install it.

5. After you load the Solver Add-in, the **Solver** command is available in the **Analysis** group on the **Data** tab.

Once the Solver add-in is installed, we solve the problem using the following procedure, the subsequent figure shows the setup in excel worksheet.

Formulas in cells:

	Cell	Formula
constraint a	D15	= D6 + F6 + H6
constraint b	D16	= 4.125*D6 + 8.25*F6 + 12.375*H6
constraint c	D17	=30* D6 + 15*F6 + 45*H6
Maximize	D19	=1.2* D6 + 2*F6 + 2.6*H6

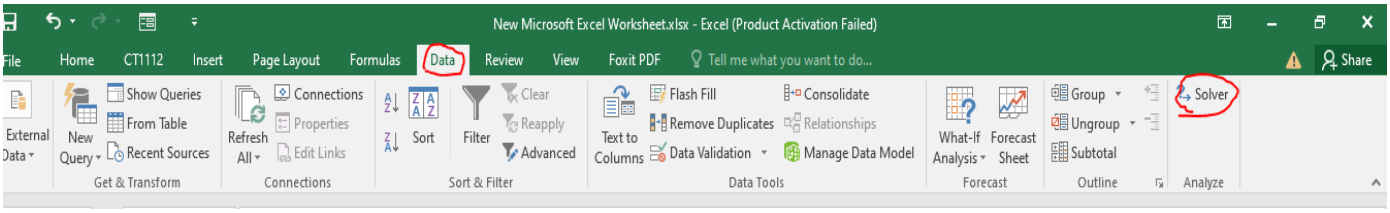
Type in variable assignments at the top of the spreadsheet.

Assign decision variable cells.
Decision variable cells:
D6, F6, and H6

Construct table from data in problem. How you set up the table is a matter of personal preference.
Not in table: the constraint which shows the sum is less than or equal to eleven.

	A	B	C	D	E	F	G	H	I
1									
2			x1 = no. convenience stores						
3			x2 = no. standard stores						
4			x3 = no. expanded-services stores						
5									
6			x1 =		x2 =		x3 =		
7									
8			Conven.	Stand.	Exp-Serv.		Limit		
9		Construct.	4.125	8.25	12.375		82.5		
10		Staffing	30	15	45		300		
11									
12		Revenue	1.2	2	2.6				
13									
14		Constraints							
15				0	<=	11			
16				0	<=	82.5			
17				0	<=	300			
18									
19		Maximize		0					
20									

The table in excel is setup, to access the solver, click on ***Data → Solver***, as shown in the following figure.



Setup the solver using the following steps.

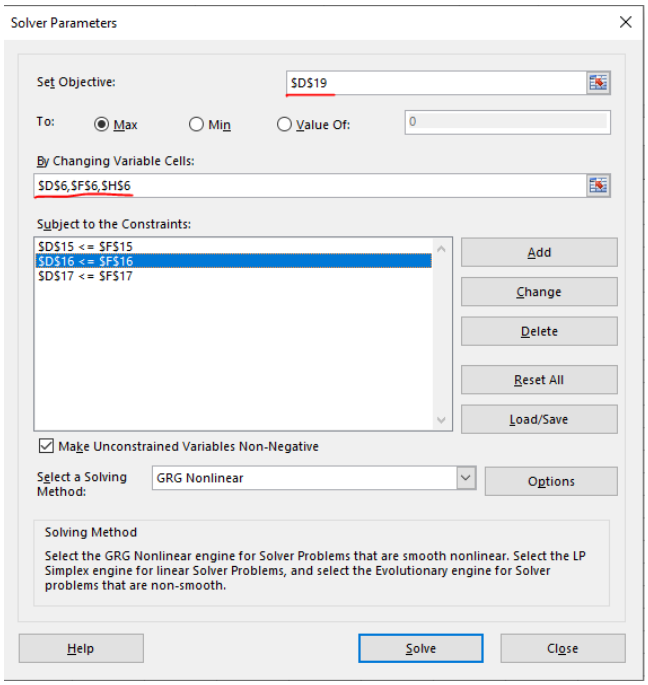
Step. 1. Set the objective cell ***D19***

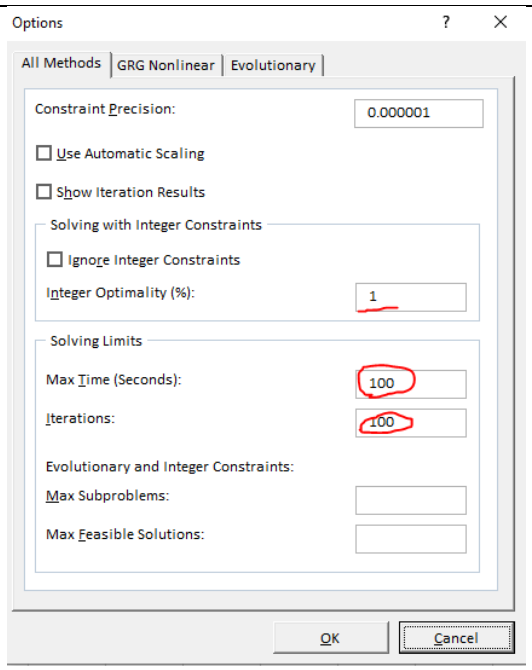
Step. 2. Add the three constraints a, b, c one by one, by clicking the add button.

Step. 3. Choose the Max radio button

Step. 4. Click on the options button and set the maximum time (i.e., 100) and iterations (i.e., 100) and click OK as shown in the second figure.

Step. 5. Finally, click on the ***Solve button***





Output:

	A	B	C	D	E	F	G	H
1								
2			**x1 = Number of convenience stores**					
3			**x2 = Number of standard stores**					
4			**x1 = Number of expended stores**					
5								
6			x1	2	x2	9	x3	0
7			Conv.	Std.	Exp.		Limit	
8		**Constracts**	4.125	8.25	12.375		82.5	
9		**Staffing**	30	15	45		300	
10								
11		**Revenue**	1.2	2	2.6			
12								
13								
14		**Constraints**						
15		**a**		11	<=	11		
16		**b**		82.5	<=	82.5		
17		**c**		195	<=	300		
18								
19		**Maximize**		20.4				
20								

Assignment: Using Excel solver, slove the following optimization problem.

$$
\begin{aligned}
\text{maximize} \quad & 4x_1 - x_2 \\
\text{subject to} \quad & 7x_1 - 2x_2 \le 14 \\
& x_2 \le 3 \\
& 2x_1 - 2x_2 \le 3 \\
& x_1, x_2 \ge 0 \text{ and integer}
\end{aligned}
$$