Data Science Specialization

David Mukajanga

8/10/2021

Introduction

This course is taken as a starting point to learning data science. It is a beginner course and in it we will learn what is data science about, why we should learn it, where it is used and start with the basics.

Contents

Module 1: Introduction to Data Science

What is Data?

- It is a ***collection*** of ***rows*** and ***columns*** of collected information about ***something***.
- The emphasis is on *rows* and *columns* because it is usually a repetitive information about ***something*** from different ***sources*** or ***points in time***.
- It is also the format you will often receive data from or arrange data to.
- Apart from this, you might also get data in terms of lines of text, but most of the times you will end up working with it in a tabular form.
- So the rows and columns can be termed as the standard format of perceiving data. This format of data is also known as ***DataFrames***.
- The ***something*** is what is commonly known as the **variable**. It is the thing we want to measure from our objects of research.
- The objects of research are commonly known as the **data points**, because they are the ones we record data from.
- Variables are in columns while our objects of research are in rows.

Meaning of Data Science

Data science is the art of using data to answer scientific questions. Scientific questions/Data science questions are the type of questions that can be answered by using statistical data analysis tools.

Your Responsibilities As a Data Scientist

- Your main role as a Data Scientist is ***to create an infrastructure that makes it possible to extract valuable insights from a particular set of relevant data***.
- This means, apart from knowing how to code and program, you need to know what you are looking for and why.
- It also means knowing the type of data needed, how to make it usable (cleaning and organizing it), and how to derive meaningful information out of it.
- This is the reason why you need a ***domain of expertise***, an area of science that you are quite knowledgeable about apart from the technical aspects of data science.
- It can be sports, business, biology, design, development and so many other.
- Because data science is a tool, a good data scientist is the one who knows how to apply it adequately in different situations, regardless of the field of study they are in.
- This is why a data scientist should have good communication skills, especially listening, in order to be able to apply his trade in different fields of study.

- We are human beings, we don't have perfect knowledge of everything. That is why a data scientist should know how to communicate to the people who are best in their field of studies in order to be able to adequately apply his/her skills to solve their problems.

Data Science Process

1. Formulating a scientific question.
2. Determining the type of data that is the most adequate to answer that question.
3. Getting the data.
4. Cleaning and organizing the data a.k.a **Data Wrangling**.
5. Exploring the data.
6. Modelling the data .i.e. deriving prediction capabilities out of it.
7. Presenting the findings and sharing them with the relevant audience.
8. Referring the people who helped along the way.

It is important to understand that while these steps are outlined in a linear form, it is likely that at some point in your projects as a data scientist you will go forward in them and then back, in certain projects you will do.

In some cases, it might be the inadequacy of the sample size that brings you back to data collection, determining the type of data that brings you back to reforming your research question etc..

1. Formulating a Scientific Question

- A scientific question most of the times shows the relationship between one aspect and another.
- The relationships help our curious minds to understand ***why*** a certain phenomena is as it is.
- In answering the question of "***why***" we also learn "***what***" causes our environment to be as it is and ***how***.
- For example, a normal curious mind would ask itself, why are the people near the equator darker than those far from it?
- A scientific question would then be, Does geographical climate influence your skin color?
- From here one important aspect of a scientific question becomes important: **Scope**.
- A scientific equation should be narrow enough so that we as scientists know what data to collect and wide enough so that we can get meaningful information from the data we collect.
- In the question above, we have specifically narrowed our attention to climate and its effects on genetics.
- We know from climate we will research aspects like the amount of sunlight a place receives, rain, temperature, atmospheric pressure, wind, and humidity. The information from these elements of climate will guide us to know the vegetation and animals possible on the land (thus what people can

eat and farm) and show us the type of lifestyle the environment forces on the people.

- From this information, we can derive meaningful insights into how climate influences human biology in producing different shades of skin color.
- This brings us to the next most important character of a scientific question: **Context**.
- Context refers to how well you are informed about the things you are asking about.
- In our question, we are approaching the concept of skin color as a result of genetics rather than superstition for example.
- If, for example, we approached the question of skin color as a result of a curse, the data we would have to collect would have little to do with climate and more to do with folklore.
- Context therefore provides direction to which aspects of our topic are strong enough to base our research on.
- Gaining context requires doing research.
- This involves laying back and consuming relevant information about the object of your research question. Sometimes this requires looking at the data you already have about the object of interest.
- SQL is a method used by scientists to find data in big databases. More on this later.

2. Determining the Necessary Data

Context: What Kind of Data to Collect.

- After the question, you have to make an educated guess (a.k.a ***Hypothesis***) to answer your question.
- The hypothesis is important because it tells you what kind of data you should collect.
- In our example above, we basically have two questions to answer: Whether climate influences the skin color and how.
- The rule of thumb in the scientific research is to deny any relationship between the variables of interests during formulating the hypothesis. This type of hypothesis is known as the ***Null Hypothesis***
- This makes sense since our efforts in data collection will be to find the type of data that will ***disprove*** the null hypothesis and prove that there is actually a relationship between one aspect and the other.
- This also helps creating better studies since the researchers will have to go deep and understand other influencing factors if they are really going to disprove the null hypothesis and prove that there is a relationship between A & B.
- But even if we start our hypothesis by accepting that; 'Yes, climate does influence skin color.' for the first question and; 'Due to a prolonged exposure to high levels of sunlight' for our second question; we still have to look for all the data that can disprove this theory. This does turn us back to explore the other factors that facilitate different shades of skin color.

Scope: How much data to collect.

- This depends on how wide you want to derive conclusions depending on the data you collected.
- If you want to derive conclusions about all people in the world, you have to make sure that you collect the amount of data that is adequate enough to draw inference for more than 7 billion people.
- The rule of thumb is that the larger the sample the more confident we are about our conclusion about our object of interest.
- But the necessary number of sample is obtained when it has a similar descriptive statistics to the entire population.

3. Getting Data.

Data can be collected through:

- **Active Data Collection** :- This is when you are setting up specific conditions in which to get data. These includes using surveys and experiments.
- **Passive Data Collection** :- This is when you are looking for data that already exists. These include finding relevant datasets and ***web scrapping***.
- Important things to keep in mind during data collection include:
  - Recognizing the influence of bias and errors and trying to anticipate sources of these and avoid them.
  - Checking the adequacy of sample size.

4. Cleaning The Data.

- As a scientist, you will most of the time get ***raw data*** in a variety of file types and formats.
- Before one can work with the raw data at hand, they need to transform that raw data into ***tidy data***.
- A tidy data set is the type of data set that has been removed of all impurities, such as missing values, invalid data and unlabeled columns and has been organized such that all variables are clearly understood and can be manipulated (eg. merging, separated etc.,) with ease.
- The process of transforming the data into a tidy data set is known as Data Wrangling.
- ***Hadley Wickham*** of RStudio outlined in depth the principles of a tidy data set in his ***paper*** and ***video*** concerning the best practices in tidying-up data sets.
- The hard and fast rules include:
  - Each Variable Measured Should Be in One Column.
  - Each Different Observation of That Variable Should Be in A Different Row.
  - There Should Be One Table For Each "Kind" of Variable.
  - If There Are multiple tables, they should include a column that allows them to be joined or merged.

- Cleaning and organizing the data set is very important since it shortens the time it takes to do a meaningful analysis on the data.

- It also ensures that data can be shared between several statisticians and analyses being done faster without the need for every statistician to start from zero making sure that the data is in good shape to be analyzed on.

- ***Jeff Leek*** of the ***John Hopkins Data Science Lab*** wrote a good and quite short guide on how to prepare your data before sharing it with other data scientists. Check it **out**

- There are several tools that can be used to carry out data cleaning and organization in data science. They are going to be covered as we move on with the chapters.

5. Exploring the Data

- Exploring the data is the prerequisite to deriving desired analyses on the data.

- Exploring the data includes:
  - Summarizing the data by using descriptive statistics.
  - Visualizing the data to see the relationship between various aspects of the data.
- Summarizing the data by using descriptive statistics gives us a general understanding of the characteristics of the information we have.

- Here we find values such as mean, mode, median and standard deviation.

- Using Standard deviation for example, one can see whether their data points are wildly different or similar, from the data they have given.

- Visualizations helps us to identify things like outliers etc.

- This is important because it sheds a light on biases and errors in data collection or even our research question and it gives us a signal whether one should go back to data cleaning or collection or not.

6. Modeling and Analysis

- This is where you start to look whether the relationships you thought existed exist or not.

- The first step is to find the relationship that exists between the variables in your data.

- The nature of the research question and variables will determine whether you will use ***Dependence*** techniques or ***Independence*** techniques for data analysis.

- You will use ***Dependence*** techniques when there is clarity in which variables influence which ones.

- Dependence techniques include ***Linear Regressions***, ***Discriminant Analyses*** and others. More on these later.

- **Independence techniques** will be used when there is no clarity on which variable influences the other.

- In these cases you will use independent techniques such as ***Exploratory Data Analysis***, ***Cluster Analysis*** and others. More on them in later chapters.

- The number of variables in your study will determine whether you will either use ***Univariate*** analytic methods or ***Multivariate*** ones.

- You will use **Univariate** analytic methods when you are measuring the impact to one variable from **one** variable at a time.

- Taking our question for example, you will be doing a univariate analysis if you are measuring the impact of exposure to sunlight on skin color separately, food to skin color separately etc,.

- **Multivariate Analysis** is when you are measuring the impact of these different factors to skin color at the same time.

- After identifying the relationships, the next step building ***mathematical models*** that expresses them.

- The purpose of creating models is to create the ability to ***predict*** what would be the outcome when one of the variables of interest is altered.

7. Communicating The Findings

- Two main ways of communicating data include: ***Visualizing*** and ***Storytelling***
- The visualization component of communicating findings in data science is perhaps the easiest to understand.

- The assumption here is that you have used graphs that are easy to understand even to a layman.

- The main component of both visualizing and storytelling is that we assume you know who is going to consume your work.

- If your audience is the type that can understand the complexity of the analysis you have done and can interpret the more complex graphs that are best in capturing every little thing, then by all means be free.

- But if it is like most of the time, where your audience is just averagely educated in a wide range of subjects, using a simple language and easy to understand graphs goes a long way in making your work delicious to consume.

- The most valuable asset in story-telling is your ability of contextualizing your work so that the audience gets attention to the significance it brings to the table.

8. Automation and Reproducibility

- The end of your scientific quest in using data science is the ability of your work being ***reliably*** useful.

- This is achieved when, first, your work can be reproduced by other people and secondly (and even better), if your work can facilitate an automated process that can carry out a specific function without the presence of a human.

- This quality of good data science can be seen in applications of data science in aspects such as the generation of automated reports, natural language processing, and others.

Data Science Applications

Data Science is usually used in:

- Making Reports
- Creating Recommender Systems
- Dynamic Pricing
- Natural Language Processing

Reports

- Comes from the scientific research nature that Data Science has.
- Can be a scientific paper, product report etc,
- The report can also be automated, especially in a company/coprorative structure.

Recommendation Engines

- Building up on big data, data science uses machine learning to create programs that can predict what users will like based on their previous selection or selections of people in the same demographic group.
- Popular applications include recommendation systems as used by Youtube, Netflix etc..

Dynamic Pricing

- This is when pricing is changed depending on factors that affect demand, eg. Taxi's being more expensive at night than in day, hotels being more expensive in summer than in winter etc..
- The practice of dynamic pricing has been made fluent with the availability of big data in industries that can afford to collect large volumes of data concerning consumer demand on their products in a short period of time.
- Uber for example, has been practicing dynamic pricing, thanks to its ability to collect data from each and every one of their consumers depending on what time they need their services, at what locations etc..

Natural Language Processing

- This is the application and usage of computer programming to process and analyze text.
- It is very useful in enabling computers to analyze tone and emotions in speech and writing and thus enable the creation of computer programs that can compose poems for example, understand and reply to human wit etc..

- A good thing about using data science and big data at large in Natural Language Processing is the ability to create an artificially intelligent computer that can learn as it is interacts with human consumers.

- This is usually applicable in automated chat-boxes and programs of that nature.

Recap

In this chapter, we have looked at:

1. What is Data?
2. Meaning of Data Science
3. Data Science Process
4. Your Responsibilities As A Data Scientist
5. Data Science Tools
6. Installing R & RStudio
7. Version Control: Git & GitHub
8. Types of Data Science Questions
9. Applications of Data Science

Module 2: Tools of Data Science.

Contents

1. Data Science Tools
2. The Toolbox of Data Science
3. Introduction To Computer Programming
4. Version Control Systems
5. Markdown

Introduction

In this module we will look at the tools you need for Data Science. Welcome to explore the data science world.

Data Science Tools

- Data Science uses the following in its operations:
  - ***Descriptive and Inferential Statistics*** :- In cleaning, analyzing and exploring the data.
  - ***Probability*** :- In creating prediction models that work to minimize uncertainty about the future action of a data source.
  - ***Programming*** :- In creating tools that does both the statistics and probability so swiftly that the end result is delivered in a nick of time.

Computer Programming in Data Science

- As mentioned in the previous chapter, the job of a data scientist is to create infrastructures in which data processing and analyses can be done and deliver the final product directly.
- In this understanding, Data science differentiates itself from normal statistical analysis since, although the normal statistical analysis can be used to achieve almost a similar outcome to data science when it comes to report writing, it is usually a one time solution compared to the applications in which data science operates in.
- This is why, although data science and statistical analysis both use statistical inferencing and probability to carry out their operations, data science usually use computer programming while statistical analysis uses ready made statistical programs such as SPSS, STATA, Excel etc.
- With data science, the end result is to have a system that does a particular function with minimal human interference, while in statistical analysis, we constantly rely on human knowledge to be able to advise on a range of subjects of interests.
- A good example is the task of predicting the criminal risk of a particular person in a particular period of time, say 2 years for example. While a statistical analysis can enable us to see in a general scale the demographic groups likely to commit certain kinds of crimes in the specified timeline,

data science enables us to create a computer program that can predict as accurately as it is possible, the possibility of an individual person committing a certain crime in the specified amount of time.

- While the end result of a statistical analysis in such a question as one above is static in certain time intervals and thus more likely to give false conclusions, a predictor model made out of artificial intelligence through data science continuously updates itself with every new information that it receives, thus being a more sustainable solution in long-run.

- It is important to note that though the machine learning tools created through data science are marvelous tools in facilitating an automated and reliable execution of tasks related to prediction, they still function under human formulated boundaries and sometimes these boundaries are biased.

- So at some point, an artificial intelligence model can be unreliable if the underlying parameters are biased. A good video explaining this topic is attached ***here***.

- Computer programming in Data Science mainly uses two computer languages: **Python** & **R**.

- The following chapters will shed a light on how to use these tools as needed in their time.

Introduction to Computer Programming

- As it is slowly becoming clear by now, computer programming is the art of writing instructions for a computer to execute.

Why Learn Programming?

- Most of the time is either you want to create a computerized tool that will enable you to carry out a specific function or you want to analyze a particular set of data.

- Most of those who use programming as a means to create digital tools aim at becoming software engineers. Some of them do this as their full time job while others like a hobby.

- Most of those who learn programming without wanting to become professional software developers are people like librarians, lawyers, doctors/academicians and all professionals that deal with data in their process of decision making.

- Sometimes they outsource this function but most of the time they learn the art themselves because it gives them a driver's seat in being able to do their jobs as they are supposed to be exactly done.

- For this course, the notes will base in data analysis. If you get so much intrigued so that you want to explore more advanced courses in programming, I, the writer of these notes wishes you all the best in your journey of learning.

Characteristics of Programming

1. Precision

- The difference between instructing a human versus instructing a computer is that a human can correct errors in the instructions as long as they account them.

- A computer on the other hand, will stop and surrender if it encounters anything it doesn't understand. This is commonly known as a "Syntax Error".

- Syntax Errors usually come from either mis-spelling the characters in the code (command), or mis-arrangement of words inside the command.

- For this reason, your code can be failing because of something as simple as putting one spacing instead of 2 in your code.

- So keep in mind to be precise when you are writing your codes.

2. Hardware

- The computer hardware is the one responsible for executing all your instructions and make possible for whatever task you had in mind to be executable.

- The next question to ask oneself now is how does it do that? Well the process goes like this:-

  1. You write your instruction to the **Secondary** memory. This is what is commonly known as a Hard disk. It includes all readable media such as flash drives, cd etc.
     - The instructions you write here stay here even if the computer is turned off.
     - Sometimes the instructions come directly from input devices such as a gamepad. Most of the time the instructions will still go through your secondary memory because the main program lives there.

  2. When you switch on your computer, your instructions are uploaded to the **Main** memory. This is what is commonly known as RAM.
     - Your instructions are translated from a computer language like Python to a machine language which binary, by using a program called **Compiler**. More on this later.
     - Once in the **main** memory, they stay there temporary while they are being worked on by the CPU and then they get erased once the program is shut down or the computer is shut down.

  3. The **CPU** is ***just*** a very sophisticated hand calculator with the basic task of asking **-"What is Next!"-**. What it does is take your translated instructions, solve them, and return the answer back to the main memory.

  4. The answer gets outputed back via the output devices such as a screen or speaker.

3. Intimidating / Full of Bluffs.

- Your main enemy will be "Syntax Error".

- At some point it will be frustrating but do not get discouraged. The word only means that Python (or any other language of programming you are using) is not understanding what you are saying. Be patient with yourself and do the work bit by bit. In the end you will become fluent in it.

Reasons Why Your Code Might Not Work

- 90% of the time you would've made a mistake either in writing the code or choosing the method of analysis.

Platforms For Asking Help

- Coursera Community
- Stack Overflow
- Cross Validated

Guidelines For Getting Help

1. Read the manuals
2. Search the web
3. If you are going to ask from a forum, first search the archives of the forum you plan to ask.
4. Read the frequently asked questions.
5. Ask a skilled fiend: This usually entails your classmates.
6. Find an answer by inspection or experimentation.
7. If you are an advanced programmer, read the source code.

Guidelines In Asking Questions

1. Let other people know that you have done all the previous necessary steps.
2. Read the documentation :) :), just be patient and read it.
3. Describe the goal. This is more important than describing steps.
4. Be explicit about your question.
5. Provide as minimum amount of information as it possibly appropriate.
6. Follow-up with the answer if/when you find it.

Version Control

- It is a computer program that enables users to track, store & integrate changes that they make in their text files.

- There are several version control programs out there the most popular is called Git.
- Git is a free and open-source version control software developed in 2005.

- It has an internet interface system called **GitHub** that is also very popular among programmers.

- With Git, one can keep track on the changes he/she is making on files of a certain project only in his/her computer.

- GitHub provides both an online cloud storage space and the Git version control system where programmers can upload their projects online, make it visible to anyone they like and enable peer programming where one can see changes that people suggest on their work.

Important Terms:

Repository

- It is where you save your file for Git to access them (online).

2. Pulling

- Downloading the current version of files from a certain online repository to a personal computer.

3. Pushing

- Uploading the current version of files of a certain project to the repository of that project.

4. Committing Confirming changes to a file that is monitored by a version control system.

5. Staging Preparing to confirm changes to a file that is monitored by a version control system.

6. A Branch

- A copy of a repository made to exist under the same repository it was copied from.

- Branches are usually made by the collaborators in a project.
- Let's say a certain company is designing a car. The work flow will be like this:
  - A company will open a GitHub account and open a repository (A fancy name for a folder stored in GitHub) on that car model.
  - It will upload all the files containing design details of the car to the repository, and these files are said to be in the ***Main / Master*** branch.
  - Let's say the tire mechanic wants to experiment on different types of tires:
    * He will create another branch and name it "Tires"
    * The tire mechanic will play around with his tire designs until he gets what he wants, but because his new design has to be approved by everyone, he has to open a "**Pull request**".
    * A Pull request is the request to **merge** the changes made to the original branch using a different branch made by a contributor to the project.

    * When a Pull request is opened, all other contributors can view the suggested changes and comment.

    * If changes are accepted by the owner of the repository, he accepts the pull request and thus merges the two branches into one. Now the main branch will include the new tires added by the mechanic.

7. Forking

- It is equal to copying an existing repository and making in your own.
- The difference between Forking and making a new branch is that while a new branch exists under the same repository, Forking includes copying the whole thing, with its branches and everything, to a different user account.
- A forked repository will still show the name of the original author, but it will give the user who forked it the permission to edit even the main branch.
- The edited version of the forked repository will still need to be presented to the original author via a pull request for it to be able to be used in place of the original one.
- If the user intends to use the forked repository as his/her own, GitHub will still display that the repository was forked from someone else and thus there is no chance for plagiarism.

8. Cloning

Best Practises in Using Git & GitHub

1. Purposeful single-issue commits
2. Informative commit messages
3. Pulling from the repository **BEFORE** starting to make changes in the offline copy of project files.
4. Pushing Often
5. Branches should be concise & short-lived. Otherwise they will start to be confusing if the project involves many different parts.
6. Always make a branch if you want to edit shared projects.

GitHub Pages

- They are a static and easier to consume webpages that displays the contents of your specific repository.
- It is a good way to share the contents of your repositories with people who are not familiar with GitHub.
- To open them:-
    - Open your intended depository

- On the top of repository's bar, select "Setting"
- In Settings Scroll down to "Pages"
- On the interface that occurs, to to the 1st prompt that you want your pages to originate and select one of the branch of that current repository.

RMarkdown

- It is a computer language that allows the user to write text and computer code in one file and be able to display both the text, the code and the output of the code in the final document.
- R Markdown is important in the data science community since it enables easier communication from one data scientist to another by enabling them to see both the content of the research study and the steps taken to analyze findings both in the same file. This makes it easy if the other data scientist want to use the same steps to solve a similar problem.
- It is also popular for it's ease of revision and editing and the fact that the final output can be printed in PDF, HTML, a Web page and many other useful formats.
- It is also quite simple. It has few symbols for assigning headers, text formatting and specifying titles and codes in the document. The document containing instructions on what symbols are used to do what in R Markdown (A.K.A RMarkdown Cheat Sheet) is found ***here***.
- To open a new RMarkdown file in RStudio, go to File > New File > R Markdown..
- A new window will appear asking you for the name of the document, author, date and whether you want to save it in PDF, HTML or PowerPoint. Choose your settings and a new tab will open in the source quadrant of RStudio with the name of your file and a ".Rmd" file extension. You are good to go.
- After writing on your file, you can save the contents by clicking on the saving disk icon at the source panel of your Rstudio session. This will save what you have written so far but will not print the final output in the format you specified e.g. PDF, PowerPoint etc.
- To do this, find a knitting icon in the source panel or click "Ctrl + Shift + K" if you are using windows. More instructions are provided in the RMarkdown cheat sheet attached above.

Module 3: Programming With Python

This chapter will give you everything you need to know about python and how you will use it in data science. It is a detailed condensation of hands-on knowledge of python and you should be fairly comfortable and fluent with the Python language after finishing it. Good Luck!.

Introduction to Python

What is Python?

- It is a programming language made in the late 1980's as an attempt to creating a powerful but easy to learn programming language.
- It was so because programming languages of that time were very technical and had a steep learning curve.

So with Python you should be able to hit the ground running in your computer programming journey.

Requirements

1. No previous programming knowledge needed.
2. A computer
3. A text editor for programming such as Notepad++, Atom etc. Atom is recommended because it behaves similarly in Windows, Mac and Linux computers. Get it **here**
4. Python itself. You can download it **here**.

- The important thing to note before we continue is that we will be ***writing*** our code in the text editor, which in this case is Atom, and ***running*** that code through the *command prompt* in Windows computers and the *terminal* in Macs.

Important Terminologies.

1. Interactive Programming.

- This is when you are writing code and executing it in the same time.
- You usually do this when for example you are coding directly in the Windows or Mac's terminal.
- It is a little bit fun but it is not convenient in writing a huge chunk of coding lines in it since you lack most of the tools from programming text editors that enable you to write your code correctly and in the end your code is not saved.

2. The Python Interpreter (>>>) A.K.A *The Chevron Script*

- It is a sign that you can type and execute a python command.

- If this symbol also appears in the Window's Command Prompt or Mac's Terminal, it means that python has been opened and is running in the background.

- It also means you can program directly from the command prompt or terminal but your code will not be saved.

3. Errors.

1. ***Syntax Error*** = You should know this by now :) :).

2. ***ValueError*** = When you are trying to coerce a value of data to a class of data that it can not be coerced to. Eg, letters to integers.

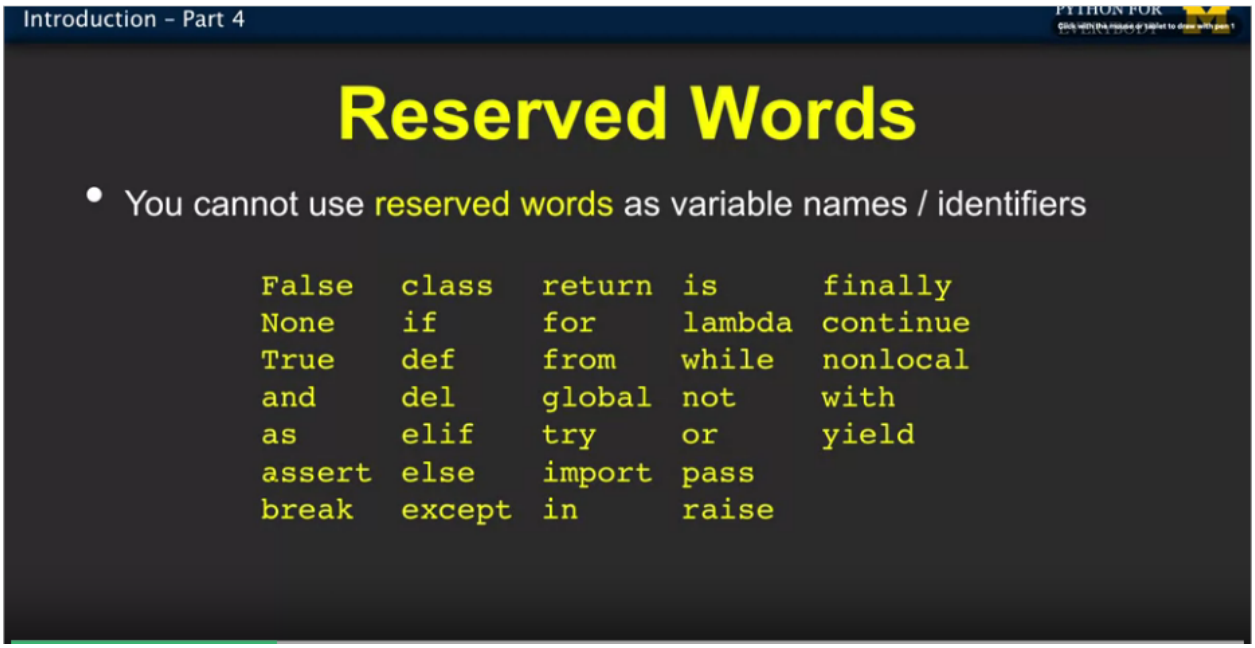
3. ***TypeError*** = When you are giving a mathematical relations between two different types of variables whose class of data don't relate mathematically. Eg. trying to sum a lettered variable and an integer variable.

4. Reserved Words

- These are function specific words that are used in Python to tell it to do a specific task.

- Under no situation can these words be used to mean something else than what the programming language understand them to be mean.

- This is the list of them:



5. Sentences / Lines

- Basically are lines of code

- Contains either the Assignment Statement (variable & its value), Assignment with an Expression (variable, its value + a command) or a function (eg. print(variable name goes here)).

6. Operators.

- These are symbols used to instruct the Programming language like Python to carry out a specific task.
- The following are common operators:
  - Assignment Operator (=) = Used to assign a specific value to a variable. Eg, Variable1 = 2.
  - Addition = (+)
  - Subtraction = (-)
  - Multiplication = (*)
  - Division = (/)
  - Power = (**)
  - Remainder (%) = This publishes **just** the remainders in the division equation.
    Eg, if you input the following code:

```
>>> x = 9 / 2
>>> print(x)
x = 4.5
```

But if you input this one:

```
>>> x = 9 % 2
>>> print (x)
x = 5 #(Because 5 is the remainder in 4.5)
```

- Larger than / Smaller than = “>” “<”
- Equal to = “==” (Basically asks the question ‘Is. . . equal to . . . ?’)
- Larger than or Equal to = “>=”
- Smaller than or Equal to = “<=”

7. Expressions

- These are commands in the right side of the assignment operator .i.e. ‘=’.
- They should not be confused with constants in such a code as: x = **2**. Here 2 is a constant (also used as “value” in the previous explanations), not an expression.
- The following is an example of an expression:-

```
>>> VarX = 2  #(VarX is assigned a constant)
>>> VarA = VarX + 2  #(VarA is an expression here)
```

8. Variables.

- A.K.A variable names, these are numbers, letters or string of words which are used to represent something in the programming codes.
  - Eg. In this small code "x = 2", x is a variable while 2 is a constant.
- The ***type*** of a variable depends on the ***class*** of data in that variable.
  - **String variables** are all variables whose data is enclosed by single quotes or double quotes .i.e. 'data here' / "" . In these type of variables, even if the data inside the quotes is numbers or has numbers, python will still read these as letters.
  - **Integers**. These are numbers that are presented only in a round form .i.e. 10 instead of 10.0 or 10.1 etc.
  - **Floating numbers (Real Numbers)**. These are the opposite of integers, that is, can be presented in fractions, decimals etc. Covers the whole range of numbers.
    N.B
- The difference between floating numbers and integers in Python is of great importance since some variables are supposed to only be presented in one class and not the other. Money for example should not be input in terms of floating numbers in python. That's just a rule of thumb.
- There is a way of *coercing* one type of variable to become another one, depending on the suitability of the class of data within that variable.
- For example, the '123' in code `>>> x = '123'` is a string variable and thus will be read as letters within python.
- So if we want to add 1, for example, by using the following code, the following error will occur in the execution of our code;

```
>>> x = '123'
>>> a = x + 1
>>> print(a)
TypeError: Can't convert 'int' object to str implicitly.
```

- To coerce this into an integer, the "**int**()" function is used. Eg:

```
>>> x = '123'
>>> y = int(x)
>>> a = x + 1
>>> print (a)
124.
```

- But not all variable types can be coerced in integers or floating numbers. (At least in python. In R it can be done.)

- In python, to use the **int()** or the **float()** function, there has to be numbers in that string for it to be transformed into an integer variable. Look at the following example:

```
>>> x = 'Hello Bob'
>>> a = int(x)
ValueError: Invalid literal for int() with base 10: 'x'
```

- To know the type of your variable, type the following function: type(Variable Name). Eg.

```
>>> x = '123'
>>> type(x)
<class 'int'>
```

- Integers can be coerced into floating numbers by using the "float()" function. Eg.

```
>>> x = 11
>>> float(x)
11.0
```

9. Mnenomic (Nimonik) Variable Names

- This is a system of naming variables in programming that makes the code easy to understand for a human being.
- For Example: The following variable names are completely fine for Python:

```
>>> x3ni21 = 2
>>> x3pi21 = 4
>>> x3pi12 = (x3ni21 * x3pi21) / x3pi21
```

- But a human reading this code will have a hard time telling the variables apart.
- This may lead to grave problems if they confuse one variable with another in debugging or a similar exercise.
- Instead, if variable names like the following are used, notice how it is now easier to digest the code:

```
>>> score = 2
>>> no.courses = 4
>>> gpa = (score * no.courses) / no.courses
```

- So Mnemonic naming is naming variables so that they can be clear, memorable & non-confusable.
- An important thing to keep in mind is to ***not*** use the 'reserved words' as variable names.

10. Scripts.

- These are text files containing all of the Python code.

Commenting in Python

- The symbol `#` is used for commenting in python.
- Essentially, everything that comes at the right side of that symbol in a python file is regarded as a comment and will thus be ignored in the execution of the code.
- This means that you can put instructions, comments and even reserve code under that symbol and the program will not look at it.
- This is useful in situations where you need to share your code to someone else or you are coming to read your own code after a while and don't remember why you did a particular function.
- It is therefore advisable to put instructions in the form of comments in your code in major steps so that it should be easy for you in the future.

Learning To Swim In Python

- In this chapter, you will start to learn the basics of python through writing simple python programs. By the end of this course, you should have known the following:
  - Likely sources of errors in your code
  - The "Input()" Function.
  - Purpose of the "Print()" Function
  - Indentations and Code Blocks
  - Custom Functions (a.k.a Store & Re-use Statements)
  - Try & Except Statements (a.k.a Error Catching)
  - Sequential Codes, Conditional Codes and Repeating/Loop Codes

Likely Sources of Errors In your Code.

Before we begin, I am necessitated to bring the following things to your attention, things that made my first codes not work. They are small stuff which you too can make and they can make your brain heat up when you don't notice them as the source of problems in your code.

1. Capitalizing the command words..

- Don't capitalize command words such as "if", "elif" etc. Your code will not work, the main reason being that the capitalized command words will be read as variable names instead of as commands.
- The sign for this is that, if you are using a programming text editor like Atom, the command words with capital letters won't be colored. If you notice that the command words are of the same color as the variable names then know that you didn't write them correctly.
- Don't even capitalize the first letter. Python wants to read the command statement like an "if" as "if ..." not "If..." :) :)

2. Not Coercing The User-Input into Integers or Real Numbers When Using Comparison Operators Against Numerical Values.

- Look at the following example:

```
>>> xscr = input("Please Enter Your Score: ")
>>> if xscr > 75:
        print ("You can enroll to the course")
```

- The code above will show a ***TypeError*** in the second line telling you this:

  `TypeError: '>' not supported between instances of 'str' and 'int'. .`

- This is because, at the second line, we are comparing the original value of the variable `xscr` which is still read as a string (even if the user inserted a number), to 75 which is a numeric. To solve this, the code should be re-written like so:

```
>>> xscr = float(input("Please Enter Your Score: "))
>>> if xscr > 75:
        print ("You can enroll to the course")
```

- Notice now that the value that will be compared to 75 is a floated numerical value of the variable `xscr` from the first line.

3. Putting an Instruction In Front of the "else" Statement.

- Look at the following code:

```
>>> xscr = float(input("Please Enter Your Score: "))
>>> if xscr > 75 :
      print('You can enroll to the course.')
>>> elif xscr > 50 :
        print('You have to take the preparation course.')
>>> else xscr < 50:
        print('You have to retake the entrance exam.')
```

- The code will give the following error in the third line, at the 'else' statement:
`SyntaxError: invalid systax`
- This is because, although the programmer had a good intention of giving very specific instructions to python, python already knows that if you use the "else" statement, you already made up your mind that anything else that is different from the 'if' or 'elif' statements you made earlier should be executed by the else statement.
- So, to avoid this problem, write your else statements like so:

```
>>> xscr = float(input("Please Enter Your Score: "))
>>> if Xscr > 75:
        print("You can enroll to the course")
>>> elif xscr > 50:
        print ("You have to take the preparation course")
>>> else :
      print ("You have to repeat the entrance exam")
```

Post your problem in online forums.

- Sometimes the only people that can see the problem in your code are anybody else but you :) :).
- This is usually because they are more experts or experienced than you :) :).
- Post your code in online developer communities like ***Stack Overflow***, ***Coursera Community*** and ***Cross Validated*** to get help.
- Sometimes the problem might be as complicated as having used different ***character codes*** in your code.

- The following is the guideline for asking questions in the forums:

1. Let other people know that you have done all the previous necessary steps.
2. Read the documentation :) :), just be patient and read it.
3. Describe the goal. This is more important than describing steps.
4. Be explicit about your question.
5. Provide as minimum amount of information as it possibly appropriate.
6. Follow-up with the answer if/when you find it.

Program 1: Pay Calculator.

The "input("")" Function

- It is a programming function that instruct the program to pause and ask for a user input before using that input to continue executing the written code.

- As it can be noticed, the inside of the brackets in the input function has double quotient marks which means that everything that gets entered into that function is a **STRING** data. This includes numbers.

- Therefore, in an example like this, let us write a program to prompt a user for hours and rate per hour in order to calculate the gross pay:

```
>>> hrs = input("Enter Hours:")
>>> rate = input("Enter Rate:")
>>> pay = float(hrs) * float(rate)
>>> print ("Pay:", pay)
```

In this code:

1. The user will be asked to input the number of hours worked through the first line. The words "Enter Hours:" are the ones that will be displayed in her screen. The input from the answer will be regarded as a string regardless of whether the answer is in numbers or text. In order to thus be able to force the answer into numbers, the programmer has to allow only a numerical input in the interface of the user.

2. The second line will get executed and the user will be asked to enter the amount of rate per hour. The same applies here.

3. The third variable is assigned the coerced values of var1 & var2

4. The output of the third variable is displayed to the user with the words "Pay:" preceding the value of the third variable. The output will look like this: - **Pay: 96.25**

The "print()" Function

- This displays the output of the code to the user of the program.

- It is a way in which the programmer displays the things he/she wants to display to the user.

Program 2: The Course Admission Checker

The Python Flow

- This is the order in which Python executes the written code.
- There are three types of Python execution flow depending on the type of the ***python statements*** written. They are:
  1. Sequential Execution
  - This is a default execution flow in python and it executes the lines of code written from the first to the last in a descending order.
    2. Conditional Execution

- This is when a conditional statement has been put in python code to instruct python to execute certain lines of code **only** when that condition is met.
  3. Repeated Execution / Loops
- This is when python is instructed to repeat executing a certain line of code until a specified condition is met.

- These executions are made possible by using the `while` and `for` statements. More on them later.

- This nature of python execution flow depends solely on **indentation**.

Indentation

- In python, indentation gives a signal to which code will be run first.
- Un-indented code lines are given the senior position by python and they are treated as a single instruction.
- An un-indented code line that is followed by several indented code lines is called a ***code chunk*** / ***code block***.
- A code chuck gets executed as a single line, with the indented lines treated as the pre-requisite building block of the main un-indented code line of the code block.
- The execution therefore, is inherently sequential, as python moves one level after the other through the written lines of code, only skipping or looping when instructed so within a code chunk.
- Different from other languages like C++ where indentation will not cause your code not to work, in Python, indentation can be a source of your code not being able to run.
- The main cause is usually mixing the indentation by using both the 'TAB' key and the 'SPACE' key to make indentation in the same document. The space created by these two keys are interpreted differently in the program, thus creating an error in execution.

- Indentation made by using the 'SPACE' key is known as **Soft Tab** while the one made by the 'TAB' key is known as the **Hard Tab**.

- It is a good practice to configure your text editor so that it uses soft tabs even when you press the 'TAB' key.

- To do this, in your text editor, go to: File > Setting > Editor > Tab Type. There are usually 3 options there, you should choose soft.

- The procedure might differ depending on the text editor you are using so if you are having a hard time finding those setting, searching on Google or YouTube might help.

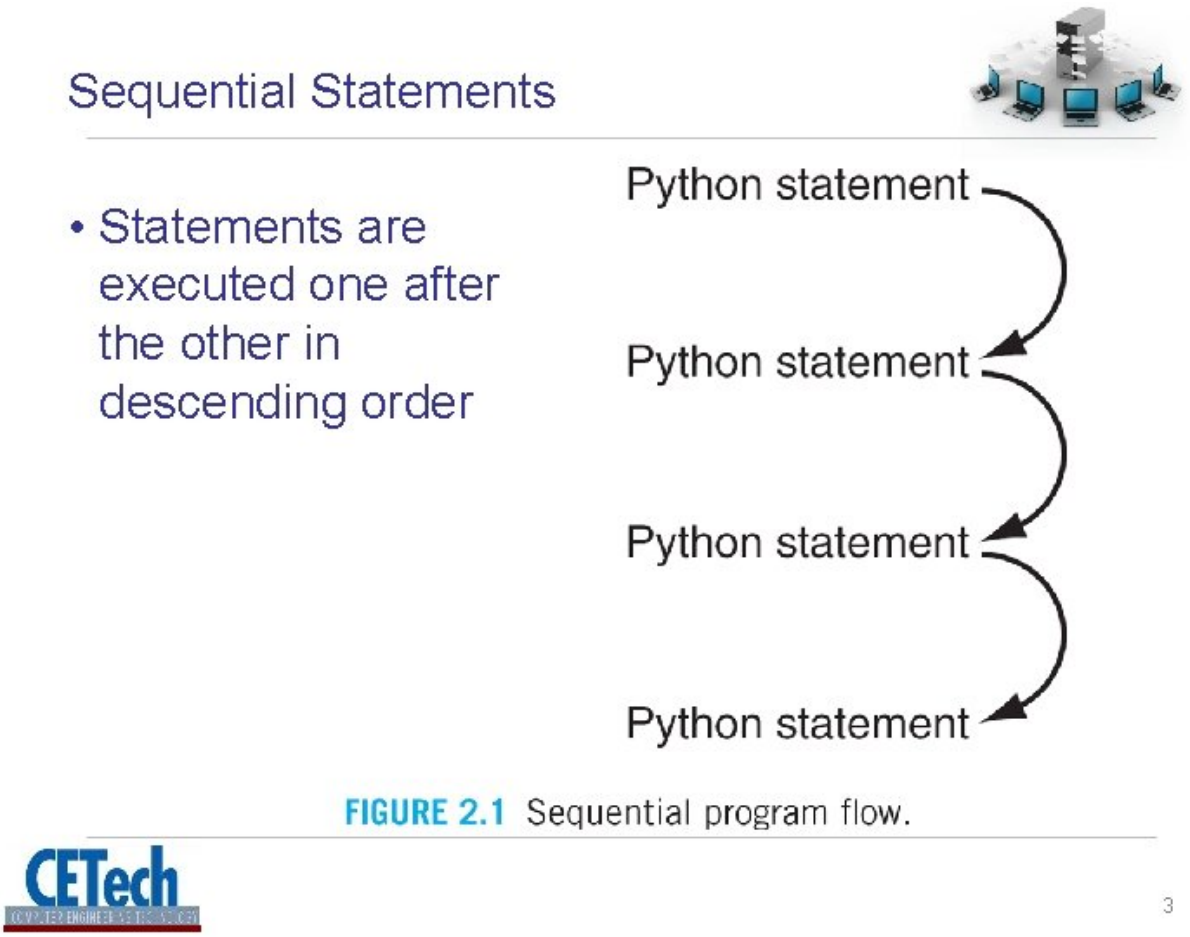
Types of Python Statements/Codes

Sequential Statements/Codes

- These are python codes written in such a manner that they are executed in a sequence from the first line to the last.

- It is the default execution pattern in python. All the other execution flows are embedded in a sequential execution of the code.

- The picture of the following chart flow explains this clearly:



Conditional Statements

- These are python codes that will not get executed unless a particular condition specified by the programmer is met.

- The use of conditional statements in programming is also known as the use of "**Logic**".

- As the name suggests, **Logic** as used in programming is the common sense that you put in how a computer program executes the task you give it to execute.

- Conditional statements are the building blocks of logic since they are the ones' giving a computer program a sort of rudimentary intelligence on what to do and when.

- They are made by using an "**if**" statement and further clarified by using the "**elif**" or "**else**" statements.

1. The "if" Statements

- Consider the following example:
  Eg.

```
>>> x = float(Input("Please Enter Your Score Here: "))
>>> if x > 75:
        print ('You can enroll in the program,please choose the starting day that is most appro
>>> if x < 75:
        print ('Please repeat the course')
>>> if x > 75:
        input("Enter The Date Here: ")
```

- In this code, if the user's score is above 75, the second line of code with the instructions "Please repeat the course" will not be executed. If the grade is smaller that 75 then the third line of code with the instructions "Enter The Date Here:") will not be executed.

2. The "elif" and "else" statements

- The word "elif" is a combination of "else" and "if".

- This statement is used to give more clarification on what should be done if the first condition created by using the "if" statement was not met.

- Consider the following example:

```
>>> xscr = float(input("Please Enter Your Score: "))
>>> if xscr > 75:
        print("You can enroll to the course")
>>> elif xscr > 50:
        print("You have to take the preparation course")
>>> else :
        print ("You have to repeat the entrance exam")
```

- Like it can be observed, we have basically instructed the program to tell our user to take the preparation course if our user enters any value less than 75 but more than 50 by using the ***elif*** statement.

- In this example, if our user indeed enters a value less than 75 but more than 50, the first code under the "If" statement will not be executed and the program will move on to the second instruction which is the "elif" statement.

- If that doesn't work too it will move to the 'else' statement.

- In short, anywhere where the three conditional statements are put together, only one condition will be executed.

- If, for example, we didn't put any clarification of what should be done if the value entered is less than 75, then the program will not do anything as it won't execute the "If" line and there is nothing else after that.

Repeating/Loop Statements.

- There are made by either using the 'while' statement or the 'for' statement.

1. Indefinite Loops

- These are made by using the `while` statement.

- They essentially repeat the same thing until a specific condition/logic created by the programmer is met.

- The condition is known as the "iteration statement".

- The iteration statement is usually placed in the 'while' statement line and what python essentially does with the statement is asking whether the iteration statement is true or false. If true, it executes the instructions in the 'while' block. If not, it skips to the next line of code.

- Consider the following example:

```
>>> x = 5
>>> while x > 0 :
        print (x)
        x = x - 1

>>> print('Blast off')
```

- In the code above, the iteration statement is `x > 0`.

- As long as the assigned value of x is greater than zero, python will continue to run the code in the indented portion of the while statement. Only when `x == 0` will python exit that code block and continue downwards.

- The output of the above code will be as following:

```
C:\Users\David\Desktop\DS\Programs>the_while_statement.py
5
4
3
2
1
Blast off

C:\Users\David\Desktop\DS\Programs>
```

1.1. Infinite Loops

- These are 'while' statements with either no iteration statement or an iteration statement which always remains true.

- Consider the following example :

```
>>> x = 5
>>> while x > 0 :
        print (x - 1)

>>> print (Blast off!)
```

- In this code, the iteration statement is always true. Python will print number 4 forever because:

  1. Python will see that x has been assigned a value of 5.

  2. It would have been told that as long as the value of x is greater than 0, it is supposed to be printing (x - 1) (that's how the 'while' statement functions).

  3. When it finishes to print (x - 1), it goes back to evaluate the iteration statement in the 'while' statement. The problem is that we have not put any instruction that changes the value of x and thus the value of 'x' in this block will always remain greater than zero and thus the while statement will run without stopping until the computer is shut down.

- Therefore, it is important to make sure that the iteration statement written in a loop statement has an end.

1.2. The 'break' statement.

- This is used to put another iteration statement within the 'while' statement.

- It basically tells python to "break" repetition and ***exit*** the 'while' statement if that second condition is met.

- Consider the following example :

```
>>> x = input('Do you need an appointment (y/n): ')
>>> while x == 'y' :
        y = input("Please select doctors' office numbers below: ")
        if (y == '3') or (y == '5') or (y == '7') :
            print('Your appointment has been accepted, please fill out your information')
            break
        else :
            print('This doctor is on a lunch break, please select another number')

>>> pname = input('Enter your full name please: ')
>>> quit ()
```

- In the code above, once the patient chooses 3, the 'else' statement won't be executed and instead python will "break" out of the while statement and move on to execute the variable name "pname".

1.3. The 'continue' statement.

- The 'continue' statement basically tells python to stop where it is at in the 'while' or 'for' statement and start again from the beginning of it.
- Like the 'break' statement, it also contains a specific condition that if met, the statement will instruct python to repeat the cycle of iteration by "continuing" to run the 'while' or 'for' statement.
- Take the following example:

```
>>> while True:
>>>   line = input ('> ')
      if line[0] == '#' :
          continue
      if line == 'done'
          break
      print (line)
>>> print('Done')
```

- In the code above, any input that starts with a "#" sign will not be printed and python will stop the iteration from there and start it from above, not executing any statements below the continue statement in the code block.
- So for example if we set empty answers in the survey to be "#", that line will be skipped whenever this code is run and instead python will go back up and ask the input as instructed in the variable "*line*"
-

2. The Definite Loops

- These are made by using the `for` statement
- Usually used for values that are predefined by the programmer.
- This includes things as lists or lines in a file or characters in a string.
- In the `for` statement, the iteration variable is assigned the particular values in the order they occur.
- This is done by the usage of the `in` statement along with the for statement.
- Consider the following example:
- Take the following example:

```
>>> for var1 in [5, 4, 3, 2, 1] :
        print (var1)
>>> print ('Blastoff!')
```

- The output will be the same as the previous one (`i.e. 5, 4, 3, 2, 1, Blastoff!`), but with this code, the machine knows which number starts/ends and in which order.

- The 'for' statement is made more straightforward in that it is told specifically which values a variable will take and in which order.

- It is like an automated version of the 'while' statement, but one that is carefully made to outline what does what. :) :).

Loop Idioms

- It is another way of saying; ***Common uses of Loops***.
- They include the following:

1. Finding The Largest / The Smallest Value In The Loop.

- Can be done by using both the `for` and `while` statement, depending on the situation.

- The `for` statement is used when there is a particular set of values already outlined in the program.

- The `while` becomes useful when the values are not specifically outlined, like when the values are coming directly from the user input.
- Because a computation like this is directly on the go, `while` is better because we don't really know how many values will be entered.
  - Consider the following example:

```
biggest = None
smallest = None
while True :
    usx = input('Enter a number: ').lower()
    if usx == 'done' :
        break
    try :
        int_usx = int(usx)
        if biggest is None :
            biggest = int_usx
        elif biggest < int_usx :
            biggest = int_usx
        if smallest is None :
            smallest = int_usx
        elif smallest > int_usx :
            smallest = int_usx
        else :
            continue
    except :
        print('Invalid input')

print('Maximum is', biggest)
print('Minimum is', smallest)
```

- In this code:

    1. There are "**comparison/initial**" variables. These are `biggest` and `smallest` and they are used to set the initial benchmark for the initial value of the loop.

    2. Iteration variable is set as `True`. This means that as long as the logical conditions we have put inside the loop statement are met, this loop will continue to run. The `True/False` statements are mainly used in indefinite loops.

- Notice the use of `is` line 9 and 13.

- `is` and `is not` are operators used to show absolute accuracy whether something `is` or `is not` the same as something else.

- Notice also the usage of the `None` variable.

- This variable is used to assign an empty space to a variable of choice.

- It is useful in these kinds of situations for enabling the comparison of the values in the loop against themselves so that an accurate answer of which one is the biggest/smallest can be obtained.

- If, for example, the programmer is finding a smallest number from a user input and sets the value of the comparison variable to a particular number, say 0, the output of the program will come out false if the user enters numbers that are all greater than zero since the smallest number that will be produced by the program will be zero instead of the actual smallest number of the user input.

- Consider the example below for a specified list, using the `for` statement.

```
>>> init = None
>>> for largest in (9, 54, 39, 12, 23, 43):
        if init == None :
            init = Largest
        elif largest > Init :
            init = Largest
        else :
            continue

>>> print (init)
```

The code above also uses the `None` variable to establish a logic that:
1. The first value in the list should be compared to an empty value from the comparison variable (`init`).
2. Be the value of the comparison variable.
3. Then be the value which the other values are compared to and being replaced by any value that will be less than it in the list.

2. Looping Through a Set.

- This is basically when you instruct the program to print every value that is in the loop.

- This is useful when you have a certain set of values that are either generated by the program or being begotten somewhere and you want to display them to the user.

- For example, the following code will print numbers from 0 to 10..

```
for it in range(0, 11)
    print(it)
```

- The `range()` function is used here to set the boundaries of what we want to print. The formula for using the `range()` is like this: `range(start value, stop value [, step (how many numbers you want to skip per iteration) value])`. Be sure to put the comma signs as they have been placed in the formula.

3. Counting, adding and finding the average in the values of a Loop

- To count in a loop, we introduce a counting ***initial*** variable assigned to start at zero and add 1 to itself every time the loop runs.

- To add, we do the same thing with the initial comparison variable and we instruct it to add itself to the value of the iteration variable every time the repetition takes place.

- To find an average, we divide the comparison variable for summation over the comparison variable for counting.

- By using these two comparison variables; .i.e. the one for addition and the one for counting, many other statistical operations can be derived from them.

- Consider the following example:

```
count = 0
tot = 0
for x in range(11, 41) :
    count = count + 1
    tot = tot + x
    mean = tot/count
    deviation = (x - mean)
    w = (deviation)**2
    totdev = 0
    totdev = totdev + w
    totdev = totdev/count
    def std(rsqdev)
        rsqdev = (totdev)**0.5

print('Count =', count, 'Mean =', mean, 'Standard Deviation = ', std(rsqdev))
```

- The above code uses the `for` loop to count the number of values available, sum them and eventually find mean and standard deviation.

4. Filtering in a loop.

- Can be done for multitude of purposes.

- Maybe certain numbers from a range of numbers need to be shown to the user, filtering can be done to achieve this.

- Or a programmer might be in need of certain numbers in a loop, filtering can be used to catch these numbers as the loop runs.

- Loops are set by using the `if` statement within the loop.

- Consider the following example:

```
for x in range(3, 43)
    if x == (13), (14), (17) :
        print(x)
```

5. Searching in a Loop.

- This is done by using the **Boolean** variable.

- This variable has only two values: `True` / `False`.

- The values `True` or `False` should be entered with the first letter being **capital letter**. Otherwise it will not be read by python.

- It is usually used with the `while` and `if` statements, to basically instruct python that as long as the certain piece of code/logic is "True / False", to continue to run the loop.

- To search by using the Boolean variable in a loop, the initial comparison variable should be set to `False` and put an instruction in the loop that whenever the value we want is found, this initial variable should be changed to `True`.

- Consider the following example:

```
init_var = False
for x in range(3, 10) :
    print(x)
    if x == 5 :
        init_var = True
        print('Found', init_var)
```

Custom Functions / Store and Re-use Statements

- While Python comes with a list of built-in functions to perform specific task (Eg, max(), min() etc), it also allows a user to create and store custom code chunks that are configured to perform a particular task.

- These custom code chunks, also known as ***functions***, can later be used anywhere in the program to execute that particular task.

- Functions are made by using the "def" statement. "def" means 'define'.
- Take the following example:

```
>>> def computepay(h, r):
        if h <= 40:
            print (h * r)
        elif h > 40 :
            return ((40*r) + ((h - 40)*(r * 1.5)))

>>> try :
       h = (float(input("Please Enter Hours: ")))
       r = (float(input("Please Enter Rate: ")))
>>> except :
      print("Please Enter Numeric Values Only")
      h = (float(input("Please Enter Hours: ")))
      r = (float(input("Please Enter Rate: ")))
>>> print ("Pay:", computepay(h, r))
```

- In the code above:
  1. "computepay()" = *Function Name*
  2. (h, r) = *Parameters*
- From now on we can use the function `computepay()` above whenever we want to use display the value of the amount of pay to be calculated.

Parameters

- These are variable names that we put into the 'def' statement to represent user input.
- They can sometimes be variables that only exist in the 'def' statement. When they are used in this manner, the goal is usually to assign ***arguments*** to them.
- They also help us programmers understand what is going on in the 'def' statement. Check the following example:

```
>>> def greeting(lang) :
        if lang == 'en' :
            print ('Hello!')
        elif lang == 'es' :
            print('Hola!')
        elif lang == 'sw' :
            print ('Habari!')

>>> greeting('sw') :
      print ('greeting()', Bob)
>>> greeting ('es') :
      print ('greeting()', Juma)
>>> greeting ('en') :
      print ('greeting()', Sarah)
```

- In the code above:

- 'lang' = Parameter
- 'en', 'sw', 'es' = Arguments

- The thing to note is the following:
  - When we first run the program, the whole 'def' block will be skipped. Python will only take note that there is a function called 'greeting' in that block of code and that whenever that function gets mentioned later in the code it should come back and execute the code therein.

Arguments

- These are the **values** that we give to parameters when we invoke a custom function.
- They can be: *Positional Arguments*, *Keyword Arguments* or *Default arguments*.
- ***Positional Arguments*** are arguments assigned to parameters in accordance to the position of the parameters in the function.

- The arguments `2`, `0`, `1` and `0` are an example of positional arguments in the following code:

```
def personal_xstics(no_of_investmnts, smoking_status, no_of_children, criminal_record):
    income_status = no_of_investments - criminal_record - no_of_children - smoking_status


personal_xstcs(2, 0, 1, 0)
```

- ***Keyword Argument*** is when we assign an argument to a parameter by completely call the name of the parameter and assign it a value when invoking our custome function.
- In the code below, keyword arguments have been used to assign value to `miles_to_travel`, `rate` and `discount`.

```
  def calculate_taxi_price(miles_to_travel, rate, discount):
      to_user = (miles_to_travel * rate) - discount
      print(to_user)


  calculate_taxi_price(rate = 10, miles_to_travel = 2, discount = 5)
```

Note: In keyword arguments, the position of the parameter does not matter.

- ***Default*** arguments are arguments that are set directly in the function statement before calling the function in question.
- This looks like this:

```
def calculate_taxi_price(miles_to_travel = 12, rate = 10, discount =5):
      to_user = (miles_to_travel * rate) - discount
      print(to_user)
```

- The values of the parameters in our function above have been expressed directed in the function statement, so they become default argument by default.
- Although they exist by default, the default arguments can be over-written in a function call by using positional arguments.
- In the following code for example, the values that will be used in the execution of the function during the function call will be the values entered in the function call.

```
def calculate_taxi_price(miles_to_travel = 12, rate = 10, discount = 5):
      to_user = (miles_to_travel * rate) - discount
      print(to_user)

calculate_taxi_price(2, 4, 3)
```

It is important to know that since arguments are essentially user-input, the type of data that can be entered as an argument can be a string or a numerical value.

A string argument will be surrounded by quotes as usual while a numerical argument will have a plain number.

When an argument has

Return statements...

- This is a statement used to give an answer to the defined 'def' statement.
- It is used in a situation where the 'def' function has been given its arguments, has done the computation needed and has to ***return*** a value to be used as the value of the function call in the ***calling expression***.

- For example:
  -> def greet() return "Hello"
  -> print(greet(), "Glenn") -> print(greet(), "Sally")

- Here whenever the *greet()* function has been invoked, the answer that will be returned is "*Hello*" OR

```
    -> def greet(lang):
           if lang == 'es':
              ***return*** 'Hola'
           elif lang == 'fr':
              ***return*** 'Bonjour'
           else:
              ***return*** 'Hello'
    -> print(greet('en'), 'Glenn')
    Hello Glenn
    -> print (greet('es'), 'Sally')
    Hola Sally
    -> print(greet('fr'). 'Michael')
    Bonjour Michael.
```

- 'def' functions that have to return an answer after it has executed its function is called a ***fruitful function***.

- ***Unfruitful Functions*** are 'def' statements that do not have a return value.
- These usually are the 'def' statements which have been ordered to execute a specific task without the need to provide a value in the end.
- Eg.

```
>>> def end() :
        print('Thanks to the Almighty you are all well. Have a good time.')
        quit()

>>> x = input('Hey! Welcome. I am a virtual doctor appointment secretary. Do you need an appointmen
>>> if x == 'Yes'
        y = input('Ok. Select your most appropriate date from the list of available dates below: ')
>>> elif x == 'No'
        end()
```

- The 'def' statement here just closes the program. It does not return any value.

Data Types and Structures in Python

- In this part of chapter 3, we look at the types of data entered and processed in Python.
- We will look on several operations that can be done to them in python and we will explore for what purpose those operations are done.
- After that we will look at how these different data types are organized in python, a more occupational name being Data Structuring :), :).

Data Type 1: STRINGS.

Basic Features of String Variables

- Can be concatenated (joined) together by using the "+" operator.
- This is to remind you that the "+" operator serves a different purpose on strings compared to numbers.
- The "+" operator cannot be used between numerical and string variables.
- Take the following example:

```
>>> a = 'String'
>>> numerical = 10
>>> print ('I just merged a ' + a + 'with a ' + numerical 'variable'.)
TypeError: must be str, not int
```

Operations That Are Done On Strings.

- There are several things that can be done to text in python. The following are among the major ones.

1. Indexing The Letters in a Block of Text.

- By default, python assigns a number to each letter of a string variable.
- The numbers starts with zero in the first letter and ascends to the length of the text.
- Indexing is calling out a specific letter or group of letters in python by using the assigned numbers.
- This is done by using the square brackets "`[]`", with the *index*(the number of the letter), inside the brackets.
- The brackets are put in the end of the string variable.
- Consider the following example:

```
>>> ex = 'Example'
>>> ind = ex[0], ex[1], ex[2], ex[3]
>>> print(ind)

('E', 'x', 'a', 'm')
```

2. Looping Through a String

- Can be used with an indexing variable.

```
>>> ex = 'Example'
>>> ind = ex[0], ex[1], ex[2], ex[3]
>>> for letter in ind :
        print(letter)
E
x
a
m
```

3. Counting the Number of Letters in a String.

- This is done by using the `len()` function.
- The variable name of the string variable in question goes into the brackets.
- Consider the following example:

```
>>> name = 'David, Mukajanga'
>>> count = len(name)
>>> print(count)
16
```

- The code above counted all the letter, including all white spaces and all the commas.

4. Slicing

5. Search & Replace

6. Stripping White Space

String Library

- A library in Python is a list of all functions that can be used to on different types of variables, eg., string variables have their own libraries, numerical variables their own etc.
-

Prefixes

Parsing and Extracting

Data Type 2: NUMBERS.

- There are 2 types of Numbers known to python. They are ***Integers*** and ***Floating Numbers***.

1. Integers

- These are all numerical values that don't have fractions or decimals.
- They represent quantities of objects that cannot be divided into fractional quantities like a number of phones, people etc..

2. Real Numbers

- Known as "***Floating Numbers***" in python, they are all numbers that have decimals and fractions.
- They represent measurements of objects that can be divided into fractional units such as litres of water, lenght of a wall etc.
- One important thing to note about real (from hence forth we'll call them ***floating numbers***) numbers in python is that: Python will store ***the nearest approximation*** of the floating number entered / displayed in the program and ***not*** the actual value of the number itself.
- This is due to the problems of ***base*** number conversion where by base 2 is the default arithmetic system on the computer and base 10 the default representation system in the computer.
- For this reason, the value 0.1 as displayed in the program is not the actual value of 1/10 but the nearest representation of it.
- For most use cases of floating numbers, this is not a big problem. But for people who are seeking absolute accuracy out of their computer calculations, the floating point inaccuracy might become a problem as the computer will have a slightly higher or lower value of the intended calculation in its mind, differently on what is displayed on the screen.
- ***This*** article on floating point inaccuracies and ***This*** python documentation on floating numbers have explained this problem in detail.
- Another thing to remember also is that; ***Money is never represented as a floating number in Python***. This is a rule of thumb.

Data Type 3: BOOLEANS

- These are data is said to either be `True` or `False`

Data Structures

Difference Btn Algorithms and Data Structures

- Algorithms are set of computer codes written in form of one line after the other, with each variable containing a specific value, while collections / data structures contains several values in one variable.

Concept of a Collection

Data Structure 1: LISTS

- This is the organized version of collections.

Creating a List

- `[]`

Indexing items in a list

- Zero-indexing and negative indexing

Slicing and Dicing Lists

- This refers to selecting multiple elements from your list.
- Slicing is equivalent to indexing items in a list, only this time you are giving a certain range where the ***starting*** value will be included by the ending value won't.
- Take a following example:

```
list_example = ["Item 1", "Item 2", "Item 3", 4, 5, 6, 7]
sliced_list = list_example[2:6]
print(sliced_list)


["Item 2", "Item 3", 4, 5]
```

- The printed `sliced_list` displays values 2 to 5, excluding value six which was put as a boundary for the slicing.
- When the beginning or the ending value in the slicing statement has not been specified, python will automatically start from the beginning or go up to the end.

For example:

```
>>> family = ["John Smiths", "Adam Smiths", "Jerry Smiths", "Anna Smiths", "Aden Smiths"]
>>> children_in_the_family = family[ :3]
>>> parents_of_the_family = family[3: ]
>>> print("Children of the Smiths' family are: ", children_in_the_family)
>>> print("Parents in the Smiths' family are: ", parents_of_the_family)
```

```
Children of the Smiths' family are: "John Smiths", "Adam Smiths", "Jerry Smiths"

Parents in the Smiths' family are: "Anna Smiths", "Aden Smiths"
```

Indexing the Items of a list within a list

- Done using the following format:
  `a_list[Item in the fist list][Item in the list inside the first list]`

For example:

```
>>> random_list = [["a", "b", "c"], ["d", "e", "f"], ["g", "h", "i"]]
>>> print(random_list[2][2])
'e'

>>> print(random_list[2][:2])
['d', 'e']
```

Changing Items in a List 1. Changing an Element in A list

* Can be done by indexing a list item and assigning a new value to it.

For example:

```
>>> random_list = ["a", "b", "c", "d", "e"]
>>> random_list[3] = "Letter of my name"
>>> print(random_list)

["a", "b", "c", "Letter of my name", "e"]
```

2. Adding an Element

* Done by using the + sign.
*

For example:

```
>>> possessions = ["Laptop", '2 Sofas', 'Working Table', 'Bed', 'Car', 'House', 'DVD Set']
>>> possessions = possessions + ['PlayStation 4', 'New Speaker System']
>>> print(possessions)

['Laptop', '2 Sofas', 'Working Table', 'Bed', 'Car', 'House', 'DVD Set', 'Playstation 4', 'New Speaker
```

3. Deleting an Element

- It is done by using the `del()` statement.

For example:

```
>>> possessions = ["Laptop", "2 Sofas", "Working Table", "Bed", "Car", "House", "DVD Set", 'PlayStation
>>> possessions_new = del(possessions[-2])
>>> print(possessions_new)

["Laptop", "2 Sofas", "Working Table", "Bed", "Car", "House", "PlayStation 4"]
```

4. Making a New List Out of an Old One * Can also serve the purpose of separating lists.
* Uses the indexing mechanism.

For example:

```
>>> possessions = ["Laptop", "2 Sofas", "Working Table", "Bed", "Car", "House", "PlayStation 4"]
>>> of_david_possessions = possessions[:2]
>>> of_isaac_possessions = possessions[3:]
>>> print(of_isaac_possessions)

['Bed', 'Car', 'House', 'PlayStation 4']
```

4. Copying a List * Done by assign a different variable to an old list after putting under the `list()` function.

For example:

```
>>> possessions = ["Laptop", "2 Sofas", "Working Table", "Bed", "Car", "House", "PlayStation 4"]
>>> of_david_possessions = possessions[:2]
>>> of_isaac_possessions = possessions[3:]
>>> print(of_isaac_possessions)

['Bed', 'Car', 'House', 'PlayStation 4']

>>> of_david_possessions_updated = list(of_david_possessions)
>>> of_david_possessions_updated[0] = 'Desktop Computer'
>>> print(of_david_possessions_updated)

['Desktop Computer', '2 Sofas', 'A Working Table']
```

- The entire list can also be copied by using the `[:]` like so:

```
>>> possessions = ["Laptop", "2 Sofas", "Working Table", "Bed", "Car", "House", "PlayStation 4"]
>>> of_david_possessions = possessions[:2]
>>> of_isaac_possessions = possessions[3:]
>>> print(of_isaac_possessions)

['Bed', 'Car', 'House', 'PlayStation 4']

>>> of_david_possessions_updated = of_david_possessions[:]
>>> of_david_possessions_updated[0] = 'Desktop Computer'
>>> print(of_david_possessions_updated)

['Desktop Computer', '2 Sofas', 'A Working Table']
```

Lists and Definite Loops

Indexing and Lookup

Mutability of Lists

Function: Len, Min, Max, Sum

Slicing Lists

List Methods: Append, Remove

Creating An Empty List

Sorting Out Lists

The Split Fuction

Splitting Strings Into Lists of Words

Using Split to Parse Strings

Data Structure 2: Tuples

Features of Tuples

Tuples in Assignment Statements

Immutability of Tuples

Comparability of Tuples

Sorting Tuples

Data Structures 3: Dictionaries

- An unorganized version of collections in python.
- It is a list-like accumulation of values that are given specific labels.
- The labels are used to identify the value later in the program.
- Consider the following example:

```
>>> purse = {}
>>> purse['PaperMoney'] = 12
>>> purse ['Bank Card'] = 14
>>> purse['Coins'] = 13
>>> purse['Bus Card'] = 39
>>> print(purse)
{'Money': 12, 'key': 14, 'perfume': 13, 'Bus Card': 39}
```

- From the code above, a wallet has been taken to exemplify what a python dictionary is.

    1. A new empty dictionary has been made. This is done by assigning empty curly brackets (`{}`) or using the `dict()` function on a new variable.

    2. We added values and assigned keys to them.

    3. Values and Keys can be added like ***this***;

    ```
    variableName = {'Key': Value, 'Key': Value}
    ```

    or like this:

    ```
    VariableName = {}
    VariableName['key'] = Value
    VariableName['key'] = Value
    ```

    So in our code above, values are 12, with key "Money", 14 "Bank Card", 13 "Coins" and 39 "Bus Card".

- So, in a normal wallet, colors and textures would be used to identify the value of paper money / a coin. In python the "Keys" are used for that purpose.

Functions used in Dictionaries

Looping Through Dictionaries

Common Applications of Dictionaries

Counting The Frequency

Reading Files into Python

Components of the Computer You've Been Utilizing So Far..

Opening A File

File Handles..

File Structure

Newline Character

Reading a File Using the `For` Loop

Searching For Lines

Reading The File Names

Dealing With Bad File Names

Debugging

The First Step

- This involves putting a print statement just above the line that has traceback.
- This helps to understand what the program was doing before it encountered the error in the problematic line.
- Sometimes the error is not in the line itself but rather the logic that is put on the code.
- Printing out the results before the traceback helps to know whether it is the line or the logic that is faulty.
- Take the following example:

The Guardian Pattern

The Short-Circut Evaluation

- This is the practice of using a compound guardian statement.
- A Compound Guardian Statement is a guardian statement that has two or more conditions in it.
- It is made by using the `if` and `or` statements in the following manner:

```
if (first condition goes here) or (second condition goes here) :
    # Do something
```

- Conditions in this statement are executed in sequence, .i.e, if the first condition is met, the second condition will not be run.
- This is the **most important** thing to know about the short-circuit evaluation.
- It is also the main reason why it is used in the first place.
- Take the following example:

```
att = ['Gvt Officials', 'Organizers', 'Students', 'Public', '99']
for num in att :
    if
```

Regular Expressions

Model 4: R Programming

Contents:

1. Nuts and Bolts of R
2. Control Structures, Functions, Scoping Rules, Dates and Times.
3. Loop Functions, Debugging Tools.
4. Simulation, Code Profiling.

Nuts and Bolts of R

What is R?

- It is a both a computer language and a program used for statistical analysis and at an advanced stage programming.
- It is a derivative of another coding language called "S" which was developed in 1976. "S" was originally developed as an internal statistical analytical tool for a technological company called *Bell Technologies* but later adopted by the computer programming community as both a language for statistical analysis and programming.
- The original goal of the creator of this language was to enable a statistician to be able to carry-out advanced statistical analysis on data without having the need to know any major computer programming language.
- The goal is achieved as both "S" and later "R" became popular computer languages for statistical analysis. The syntax between the 2 languages is not different at all but R has got more functioning added onto it that makes it popular in the open source coding community.

What is RStudio

- It is a platform which makes using R simpler and more interactive. This is by enabling it to display tables, charts, graphs and other things.

How R and RStudio works?

- First you have to download R. The instructions for doing so are in the next heading.
- Then you download RStudio.
- You can use R without RStudio, just that your workflow will be a little bit constricted and not pleasing to the eye.
- The basic R installation comes with several tools that enable you to carry out statistical analysis right away but it doesn't contain all tools for statistical analysis. These missing tools are called **Packages**. They basically add functionality to R as a program and enable it to do more complicated stuff.

- For example, when you install a new operating system in your computer, the operating system might come with native programs for things like text editing etc, but if you want serious functionality, you will still have to download 3rd party programs such as Microsoft Word and VLC Media Player. These third party programs are what Packages are in R.

- R Packages are stored in cloud-based online databases called "Repositories". In these particular websites, R Packages can be downloaded and installed at any time free of cost.

- Because R is an open source project, R Packages are developed and updated by an online open source community which means that anybody can create a new package and modify an existing one without any restrictions.

- The open-source part of it also allows for problems within packages to be detectable quickly and also solved quickly as anyone with the ability to do so can do so.

- The main 3 big repositories for R Packages are **CRAN** (Comprehensive R Archive Network), **Bio-Conductor** (For BioInfo statistical packages), and GitHub (Which is not only specific for R). To know these is just a formality as installing R packages does not require you to visit these sites first.

Download Sources:

- Both two are free software so just type "R free download for Windows/Mac etc" in Google and it should lead you to right places.

- The same goes for RStudio.

Tour of RStudio

- The interface has four main quadrants/panels:
  1. Source = Top Left
  2. Console = Bottom Left
  3. Environment = Top Right
  4. Files = Bottom Right
- The work flow is like the following:
  1. You input the code into the **CONSOLE** quadrant
  2. It gets interpreted in the **ENVIRONMENT** quadrant
  3. If the output is not a plot, it is shown in **SOURCE** quadrant. If it is a plot it is shown in the **Plots** tab of the **FILE** quadrant
  4. The **FILES** quadrant in the bottom right will show you the folder in which all the files of your current project are saved. The **Packages** tab in the same quadrant will show all R packages that you have installed and will display also which are currently active via a green tick at a check box in the left side of the package name.
- To find the packages for any task you have in mind, there are 2 main ways:-
  1. rdocumentation.org

2. Google that task with “R package” at the end. Eg. “R package for statistical plots”

- To install packages from BioConductor:

1. 1ˆ{st} run this code:- source(“https://bioconductor.org/bioclite.R”)
   - This will make the main install function of Bioconductor: bioclite (), availabe to you.

2. Then you can install any package from Bioconductor like so:-
   - Bioclite (“Package Name Goes Here”)

- Rare occasions:-
  - You might not find the package you are looking for in CRAN or Bioconductor.
  - In such a case you’ll need to install your package from GitHub. To do this:-
    1. Find the package name **AND** the name of the author of the package you want from GitHUb.
    2. Install the “devtools” package in R and load it.
    3. Use the following command to install the package from GitHub:-
    * install_github (“author/package”)

- To load a package, click on the checkbox in the “Packages” tab or type: **library (Package Name)** in the console.
  *To get help on how to use a package or the functions it contains:-
  - Either go to the **Files** quadrant and select the éHelp" tab.
  - OR type:- **help(“Package Name”)** in the console to get the general description on what the package is about.

- Or type:- **browsevignettes(“PackageName”)** to access detailed information on how to use the package.

R Programming.

R Input & Evaluation

If you are working directly from R without RStudio, the first step is to set a **working** directory.
* In windows, go to an open R-GUI and open file > change dir & then choose the location of the folder you want R to read and write from. For Mac users, please search the web :) :).
* To know what is your working directory type this command in R console:- getwd()

Data Types

R Objects & Attributes

Matrices

Factors

Missing Values

Data Frames

Name Attribute

Reading Tabular Data Into R

N.B :- Before starting this part, please read **this page**, and have it memorized.

- There are main two functions used to paste tabular data into R:-
  - read.table() : Used primarily for a tables that are on normal text files .i.e. MS Word etc.
  - read.csv() : Used primarily for data that has been written in a spreadsheet or saved in a similar file structure.
- Depending on the size of your dataset, you might or might not need to attach the following attributes into your read.table/csv function. These attributes specify for R the number of rows in the data, the class of elements in the table & so on so that R doesn't need to carryout those calculations by itself. This is very handy when you are handling big data because if you don't tell R these attributes of your data, it will eat a whole chunk of your time counting them and in the process slow you down.
- It is also necessary for you to count and know these attributes about your data if you are working with big data because they will enable you to know if you have a large enough RAM to print that data into R or not. This is because, by default, R stores all the tables that you import into it in the RAM of the computer. So if, for example, printing your data set into R will require 3GB of your RAM while your computer has only two, you will not be able to do anything with that data through R.
- It will also assist you on deciding how many application you will need open during that analysis as all applications also use RAM to store temporary data into them. In the above example, even if your computer had 4GB of RAM, you will still be having a hard time doing that particular analysis since an operating system like windows 10 itself uses at least 1GB in it's operations. If your R takes the remaining 3GB then the performance of that analysis will be considerably slow.
- Let's take an example of a moderately large data set with 1.5 million rows and 120 columns. To get the amount of RAM needed for this analysis, we will do the following calculation:-

  1. The Number of Elements x The number of Bytes to store each element.

  - No. of Elements = 1.5mil x 120.

  - Assuming all elements are numeric, the standard storage is 8 bytes per numeric.
    Hence:- 1.5mil x 120 x 8 = 1.44 billion Bytes.
  - 1 GB = 1 Billion Bytes
  - 1.44 Billion Bytes = 1.44 GB.

- Now although the calculated amount is 1.44GB, it is usually recommended to count double the amount to account for a smoother operation. So in our example, we will need not less that 3GB to run our analysis smoothly, not accounting for the influence of other software that may be running at the time.
- But if you are doing an analysis on a small or medium data set, say with a 1000 or less rows, you need not trouble yourself with specifying to R the classes of data or other attributes in the table. It will do the work itself and it won't take long.
- The arguments in question are following:-
  - colClasses"" :- This indicates the class of data in each column in the dataset. It is super useful in big data because specifying this saves you a lot of time and processing requirement that would take R to do this by default.
  - nrow"" :- Also super useful in terms of memory as it will tell R outrightly how many rows are there thus making it easier to calculate the amount of storage needed rather than keeping quiet. You can roughly overestimate the number, R will still count the right number of rows in the end.
  - skip"" :- It will tell R the number of rows to skip from the beginning.
  - Sep"" :- This indicates how columns are separated.
  - StringAsFactors :- You use this when you want to code character variables as factors.
  - File :- This tells the name fo a file or connection.
  - Header :- This is a logical indicating that the table has a header line.
  - Comment.char :- A character string indicating the comment character.
- It is usually advisable to make sure that here are no comments on your data set. You can account for this by using the "comment.char" argument but it is just good practice to remove comments from a data set.

Textual Data Format

- This is a way of saving a data set such that the metadata concerning that data set, data like the class of elements in a column, number of rows etc also gets saved with the file.
- The data is saved not necessarly in a tabular form but as a sort of textual data that can be read back to R with all the metadata instructions intact. The metadata instruction is saved as a code so when you read in back into R, those instructions concerning number of rows and classes of data get executed directly without the need for you to re-specify anything from the beginning.
- The function to create that kind of data is called: dput () or dump (); while the function to read that kind of data is called "dget ()".
- The difference between dput() and dump() is that dput () is only used for encoding one R object/table while dump() can be used for multiple R objects.
- Dump-ing or dput-ting is useful in saving data that might have originally been in an excel format etc, as an R readable text file and include that metadata instruction that is very important.

- Another benefit is the ability of fixing the problem if there has been a corruption in the data. This makes the data saved this way more long living.
- The only down side of saving files this way is that they eat a lot of space and generally need to be compressed.

Connections: Interfaces to The Outside World