**PROJECT REPORT**

**DREAM TEAM FANTASY CRICKET GAME**

Submitted in partial fulfillment of the

Requirements for the award of

**Degree of Bachelor in computer applications**

**Under the surveillance of:**

**Submitted By:**

Name:Shivam raj

University Roll No. : 192880200003



**P.N.S INSTITUTE OF TECHNOLOGY AND MANAGEMENT**

**DECLARATION**

I hereby declare that the project report(“Dream Team Fantasy Cricket”) is an authentic record of my own work .

SHIVAM RAJ 192880200003

Date:

Head of Department

(Signature and Seal)

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

**TABLE OF CONTENTS**

1. About
2. Motivation
3. Results
4. Learning outcomes and importance
5. Screenshots of Each module with working
6. Conclusion
7. Appendix

**About**

In this training program we have learnt about Python (Basics of programming, OOP’s concept in Python, GUI toolkit), Database connectivity etc.

This training introduces object-oriented concepts and the Python programming language. It is divided into different modules. The module begins with a brief explanation of basic programming with Python and Object-Oriented concepts. This training covered essential concepts on the building blocks of Python, object-oriented programming, the use of SQLite database and development of GUIs for Python applications.

In this module we learn about:

* INTRODUCTION TO PYTHON.
* BASICS OF PROGRAMMING IN PYTHON.
* PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING.
* SQLITE DATABASE CONNECTIVITY.
* DEVELOPING A GUI WITH PYQT.

Python was conceived in the late 1980s by Guido can Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum’s long influence on Python is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

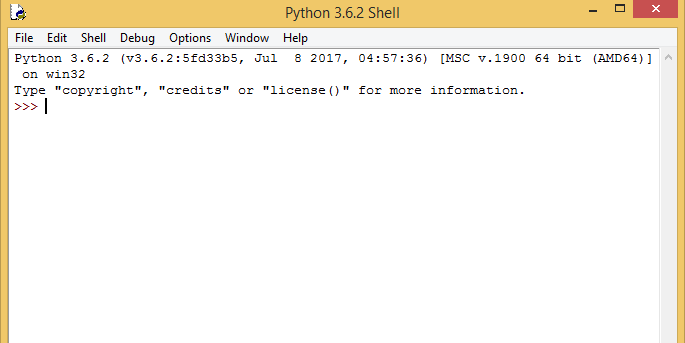
Python 3.0 was released on 3 December 2008. It was a major version of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6x and 2.7x version series. Release of Python 3 includes the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

**1.Using the python interpreter IDLE**

Every Python installation comes with an **Integrated Development and Learning Environment**, which you’ll see shortened to IDLE or even IDE. These are a class of applications that help you write code more efficiently. While there are many [IDEs](https://realpython.com/python-ides-code-editors-guide/) for you to choose from, Python IDLE is very bare-bones, which makes it the perfect tool for a beginning programmer.

Python IDLE comes included in Python installations on Windows and Mac. If you’re a Linux user, then you should be able to find and download Python IDLE using your package manager. Once you’ve installed it, you can then use Python IDLE as an interactive interpreter or as a file editor.

****

**1.1.Salient features of python**

There are many features in Python, some of which are discussed below –

**1. Easy to code:**  
Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

**2. Free and Open Source:**  
Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword.  
[Download Python](https://www.python.org/downloads/)  
Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

**3. Object-Oriented Language:**  
One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

**4. GUI Programming Support:**  
Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.  
PyQt5 is the most popular option for creating graphical apps with Python.

**5. High-Level Language:**  
Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

**6. Extensible feature:**  
Python is a **Extensible** language. We can write us some Python code into C or C++ language and also we can compile that code in C/C++ language.

**7. Python is Portable language:**  
Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

**8. Python is Integrated language:**  
Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.

**9. Interpreted Language:**  
Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called **bytecode**.

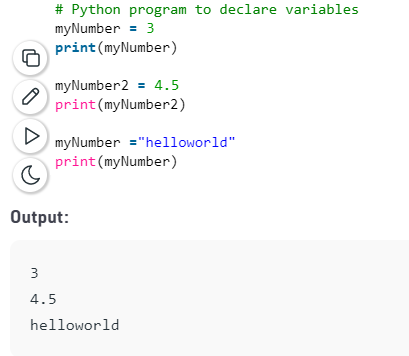
**10. Large Standard Library**  
Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

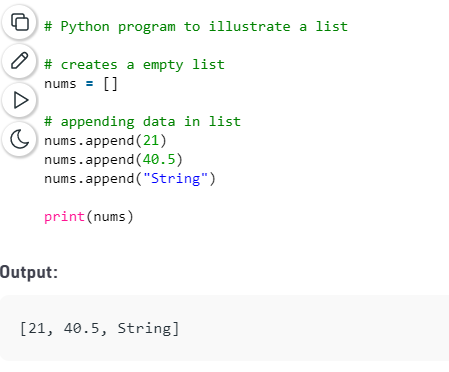
**11. Dynamically Typed Language:**  
Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

**Basics syntax of python**

[Python](https://www.geeksforgeeks.org/python-tutorial-learn-python-3-with-examples/) was developed by Guido van Rossum in the early 1990s and its latest version is 3.7.1, we can simply call it as Python3. Python 3.0 was released in 2008. and is interpreted language i.e it’s not compiled and the interpreter will check the code line by line. This article can used to learn very basics of [Python programming language](https://www.geeksforgeeks.org/python-programming-language/).

So before moving on further.. let’s do the most popular ‘HelloWorld’ tradition  and hence compare Python’s Syntax with C, C++, Java ( I have taken these 3 because they are most famous and mostly used languages).





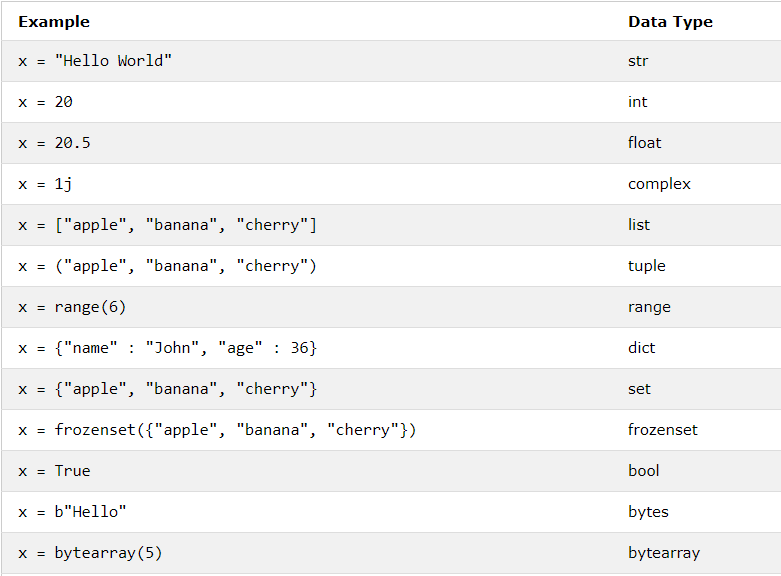
**2. Using variables in python**

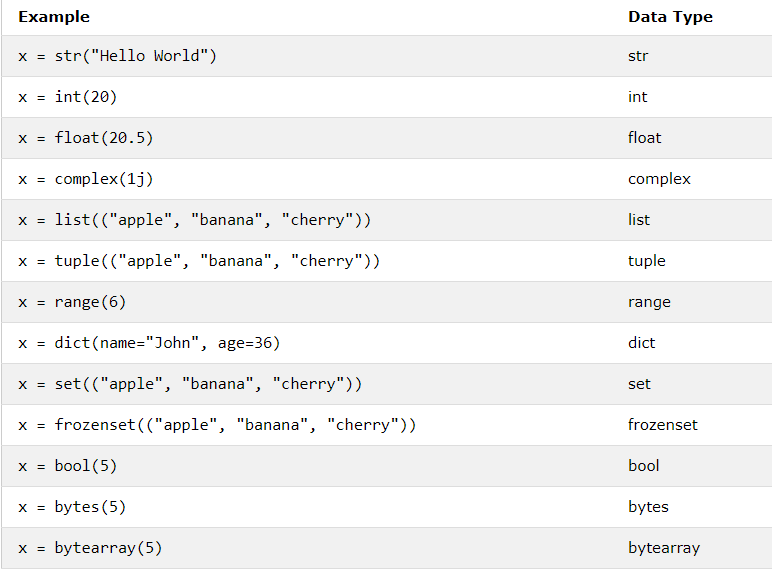
**2.1 Data types and variables**

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |

****

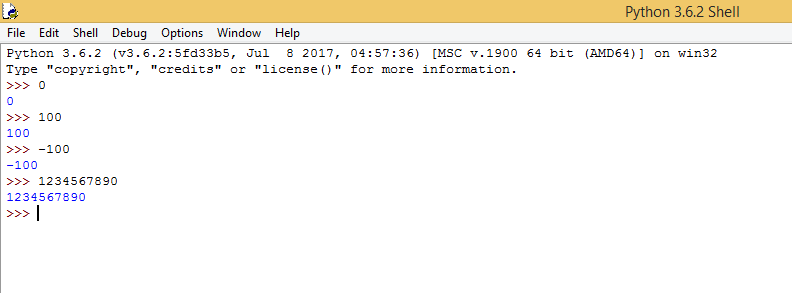
****

**2.2 Numeric data types**

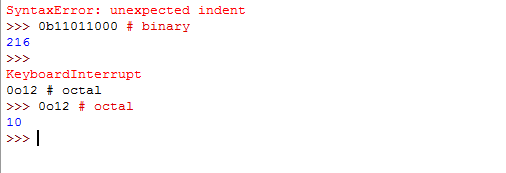
Python includes three numeric types to represent numbers: integers, float, and complex number.

## Int

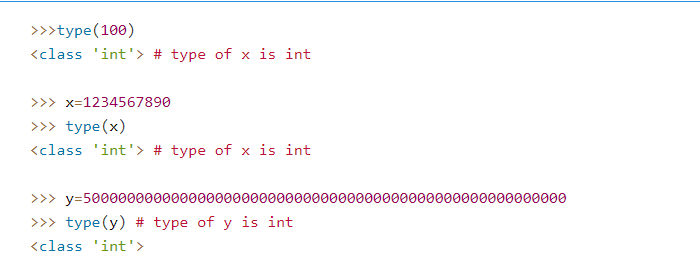
In Python, integers are zero, positive or negative whole numbers without a fractional part and having unlimited precision, e.g. 0, 100, -10. The followings are valid integer literals in Python.

****

Integers can be binary, octal, and hexadecimal values.

****

All integer literals or variables are objects of the int class. Use the type() method to get the class name, as shown below.

****

Leading zeros in non-zero integers are not allowed e.g. 000123 is invalid number, 0000 is 0.

### Binary

A number having **0b** with eight digits in the combination of 0 and 1 represent the binary numbers in Python. For example, 0b11011000 is a binary number equivalent to integer 216.



### Octal

A number having **0o** or **0O** as prefix represents an **octal** number. For example, 0O12 is equivalent to integer 10.



### Hexadecimal

A number with **0x** or **0X** as prefix represents **hexadecimal** number. For example, 0x12 is equivalent to integer 18.



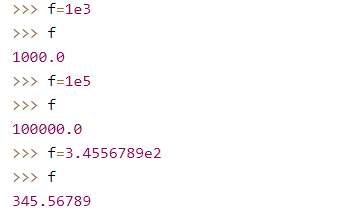
## Float

In Python, floating point numbers (float) are positive and negative real numbers with a fractional part denoted by the decimal symbol . or the scientific notation E or e, e.g. 1234.56, 3.142, -1.55, 0.23.

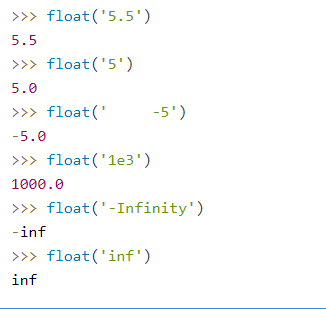


Floats can be separated by the underscore \_, e.g. 123\_42.222\_013 is a valid float.

Floats has the maximum size depends on your system. The float beyond its maximum size referred as "inf", "Inf", "INFINITY", or "infinity". Float 2e400 will be considered as infinity for most systems. Scientific notation is used as a short representation to express floats having many digits. For example: 345.56789 is represented as 3.4556789e2 or 3.4556789E2



Use the float() function to convert string, int to float.



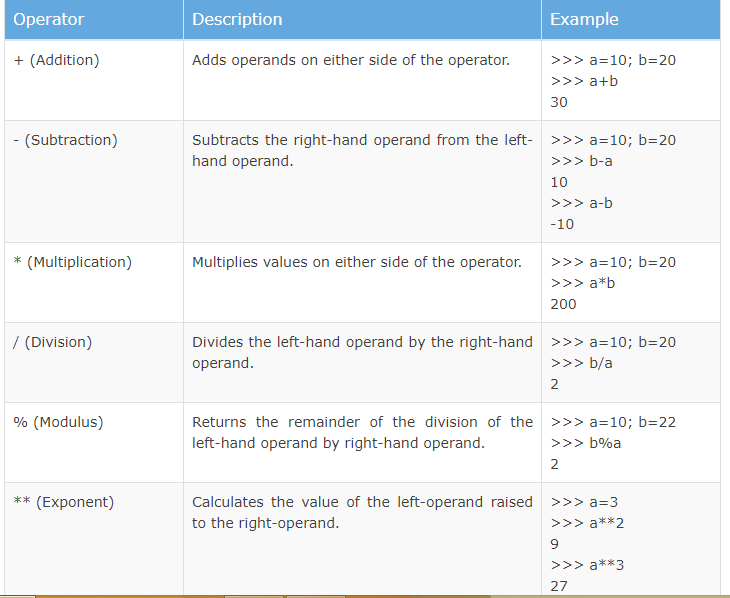
## Complex Number

A complex number is a number with real and imaginary components. For example, 5 + 6j is a complex number where 5 is the real component and 6 multiplied by j is an imaginary component.



You must use j or J as imaginary component. Using other character will throw syntax error.

**Arithmetic operators**



**2.3 String data types**

## What is a String?

String can be defined as a **sequence of characters**, and that's the most basic explanation of string that you can provide. In this definition, we can see two important terms, first being **sequence** and other is **characters**. If you are here after finishing the [last tutorial](https://www.studytonight.com/python/data-types-in-python), then there, we already explained - What is **Sequence** data type and how **Strings** are a type of sequence. Just for revision, in python, Sequence is a data type which is made up of several elements of same type, i.e., integers, float, characters, strings etc.

**Note:***There is a unique code provided to all existing characters. The coding convention had been labelled as****Unicode format****. It consists of characters of almost every possible languages and in fact emoticons too (yes, emoticons had been declared as characters too).*

Hence, strings can be considered as a special type of sequence, where all its elements are characters. For example, string "Hello, World" is basically a sequence **['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd']** and its length can be calculated by counting number of characters inside the sequence, which is 12.

**Note:** *Yes, space, comma everything inside those quotes will be a character if the length is 1.*

Generally in programming languages there is a different data type dedicated to characters only, while in Python, there is no character data type. Instead characters are just treated as a string of length 1.

### Escape Sequence

Suppose you want a string to store a quote by Mahatma Gandhi.

**"You must be the change you wish to see in the world" - Gandhi**

This is the exact line you want to display in the console. And you also wish to have the quotes surrounding this sentence. As you go ahead and print the statement, you will see that it isn't that simple.

Python will instantly return a syntax error. This is because of those extra double quotes that we added. In above image you can notice that Gandhi's quoted text is in black colour, while” **Godse "** is in green. Also, if you have used IDLE enough you might know that all the characters inside the string are highlighted in green in the IDLE (it can be other colours too depending upon text editor, python version, OS etc). This clearly means that Python isn't treating *You must be the change you wish to see in the world* part of the sentence as a string. Therefore, this concludes that whenever we open a quote and close it, to declare a string, whatever we write after the closing quote, is just considered as some python keyword.

Like for the above quotation text, we started the string with two double quotes and wrote **You must be the change you wish to see in the world** just next to it, since double quote was already closed before this phrase, hence Python considered the entire sentence as some non-understandable python keywords. After the phrase, another double quote started, then came **- Gandhi** after that and finally the closing double quote, since ***- Gandhi*** part is within a pair of double quotes hence its totally legitimate.

**2.4 Sequence data type**

## Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

* [String](https://www.tutorialsteacher.com/python/python-string): A string value is a collection of one or more characters put in single, double or triple quotes.
* [List](https://www.tutorialsteacher.com/python/python-list): A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
* [Tuple](https://www.tutorialsteacher.com/python/python-tuple): A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

## Mapping Type

[Dictionary](https://www.tutorialsteacher.com/python/python-dictionary): A dictionary Dict() object is an unordered collection of data in a key:value pair form. A collection of such pairs is enclosed in curly brackets. For example: {1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}

## Set Types

* [set](https://www.tutorialsteacher.com/python/python-set): Set is mutable, unordered collection of distinct hashable objects. The set is a Python implementation of the set in Mathematics. A set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc.
* **frozenset**: Frozenset is immutable version of set whose elements are added from other iterables.

### Mutable and Immutable Types

Data objects of the above types are stored in a computer's memory for processing. Some of these values can be modified during processing, but contents of others can't be altered once they are created in the memory.

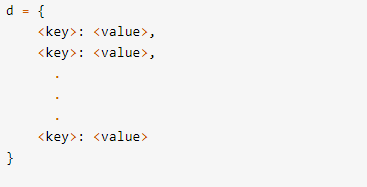
[Numbers](https://www.tutorialsteacher.com/python/python-number-type), [strings](https://www.tutorialsteacher.com/python/python-string), and [Tuples](https://www.tutorialsteacher.com/python/python-tuple) are immutable, which means their contents can't be altered after creation.

On the other hand, items in a [List](https://www.tutorialsteacher.com/python/python-list) or [Dictionary](https://www.tutorialsteacher.com/python/python-dictionary) object can be modified. It is possible to add, delete, insert, and rearrange items in a list or dictionary. Hence, they are mutable objects.

**2.5 Dictonary data type**

Dictionaries are Python’s implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

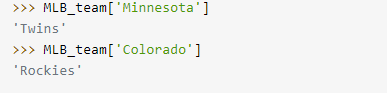
You can define a dictionary by enclosing a comma-separated list of key-value pairs in curly braces ({}). A colon (:) separates each key from its associated value:

****

## Accessing Dictionary Values

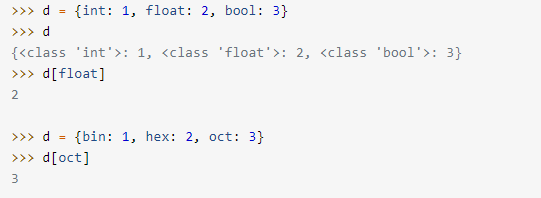
Of course, dictionary elements must be accessible somehow. If you don’t get them by index, then how do you get them?

A value is retrieved from a dictionary by specifying its corresponding key in square brackets ([]):

****

## Restrictions on Dictionary Keys

Almost any type of value can be used as a dictionary key in Python. You just saw this example, where integer, float, and Boolean objects are used as keys:

****

However, there are a couple restrictions that dictionary keys must abide by.

First, a given key can appear in a dictionary only once. Duplicate keys are not allowed. A dictionary maps each key to a corresponding value, so it doesn’t make sense to map a particular key more than once.

**3. Basic programming in python**

There can be various python programs on many topics like basic python programming, conditions and loops, functions and native data types. A list of top python programs are given below which are widely asked by interviewer.

**3.1 Understanding programs and programming**

Computers are all around us in the 21st century, and promise to become ever more ubiquitous as time goes on. They are having an ever increasing impact on our lives, and we are becoming less and less able to even conceive of living without them.

As Charles Severance says in the beginning of his [Python for Everybody](https://www.py4e.com/) course, “It is important that we learn to tell these computers what to do rather than just let them increasingly control our lives.”

Information in the 21st century is now almost exclusively distributed through the [Internet](https://en.wikipedia.org/wiki/Internet) and the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web). We do research, entertain ourselves, make purchases, interact with our government, and study and learn online.

If we aspire to have democratic participation in the world of the 21st century, at least a basic understanding of computer programming, what it is and what it can do, and the workings of the Web are essential for all informed citizens.

That, in a nut shell, is what this book is about. It has two goals: to teach you to think like a computer programmer, so that you can better understand what computers are and what they can and can not do, and to help you apply this understanding in creating web applications.

Learning to think like a computer programmer is a good end in itself, since the processes used in computer programming combine some of the best features of mathematics, engineering, and natural science. Like mathematicians, computer programmers use formal languages to denote ideas (specifically computations). Like engineers, they design things, assembling components into systems and evaluating tradeoffs among alternatives. Like scientists, they observe the behavior of complex systems, form hypotheses, and test predictions.

The single most important skill for a computer programmer is **problem solving**. Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately. As it turns out, the process of learning to program is an excellent opportunity to practice problem-solving skills.

On one level, you will be learning to program, a useful skill in itself. On another level, you will use programming as a means to an end - creating web applications, which can open the door to you to play an active part in building the tools of the information age.

## 1.1. The Python programming language

The programming language you will be learning is [Python](http://python.org/). Python is an example of a [high-level language](http://en.wikipedia.org/wiki/High_level_language); other high-level languages you might have heard of are C++, PHP, and Java.

As you might infer from the name high-level language, there are also [low-level languages](http://en.wikipedia.org/wiki/Low_level_language), sometimes referred to as machine languages or assembly languages. Loosely speaking, computers can only execute programs written in low-level languages. Thus, programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a small disadvantage of high-level languages.

But the advantages are enormous. First, it is much easier to program in a high-level language. Programs written in a high-level language take less time to write, they are shorter and easier to read, and they are more likely to be correct. Second, high-level languages are [portable](http://en.wikipedia.org/wiki/Software_portability), meaning that they can run on different kinds of computers with few or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten to run on another.

Due to these advantages, almost all programs today are written in high-level languages. Low-level languages are used only for a few specialized applications requiring exact control of the computer.

Two kinds of programs process high-level languages into low-level languages: [interpreters](http://en.wikipedia.org/wiki/Interpreter_%28computer_software%29) and [compilers](http://en.wikipedia.org/wiki/Compiler). An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.

## 1.2. What is a program?

A [computer program](http://en.wikipedia.org/wiki/Computer_program) is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations or finding the square root of a number, but it can also be a symbolic computation, such as searching and replacing text in a document or (strangely enough) compiling a program.

The details look different in different languages, but a few basic instructions appear in just about every language:

input

Get data from the keyboard, a file, or some other device.

output

Display data on the screen or send data to a file or other device.

assignment

Set the value of a storage location denoted by a variable name.

math and logic

Perform basic mathematical operations like addition, and multiplication, and logical operations like and, or, and not.

conditional execution

Check for certain conditions and execute the appropriate sequence of statements.

repetition

Perform some action repeatedly, usually with some variation.

Believe it or not, that’s pretty much all there is to it. Every program you’ve ever used, no matter how complicated, is made up of instructions that look more or less like these. Thus, we can describe programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with sequences of these basic instructions.

That may be a little vague, but we will come back to this topic later when we talk about [algorithms](http://en.wikipedia.org/wiki/Algorithm).

## 1.3. What is debugging?

Programming is a complex process, and because it is done by human beings, it often leads to errors. Programming errors are called [bugs](http://en.wikipedia.org/wiki/Programming_bug) and the process of tracking them down and correcting them is called **debugging**. Three kinds of errors can occur in a program: [syntax errors](http://en.wikipedia.org/wiki/Syntax_error), [runtime errors](http://en.wikipedia.org/wiki/Runtime_error), and [semantic errors](http://en.wikipedia.org/wiki/Logic_error). It is useful to distinguish between them in order to track them down more quickly.

## 1.4. Syntax errors

**Syntax** refers to the structure of a program and the rules about that structure. For example, in English, “Do you like programming?” is a syntactically correct sentence, while “Like do programming you?” is not. Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message.

For most speakers of a natural language, a few syntax errors are not a significant problem, so if a someone new to English asked you, “Like do programming you?”, you would probably understand what they meant.

Python is not so forgiving. If there is a single syntax error anywhere in your program, Python will display an error message and quit, and you will not be able to run your program. During the first few weeks of your programming career, you will probably spend a lot of time tracking down syntax errors. As you gain experience, though, you will make fewer errors and find them faster.

## 1.5. Runtime errors

The second type of error is a runtime error, so called because the error does not appear until you run the program. These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.

Runtime errors are uncommon in the simple programs you will see in the first few chapters, so it might be a while before you encounter one.

## 1.6. Semantic errors

The third type of error is the **semantic error**. If there is a semantic error in your program, it will run successfully, in the sense that the computer will not generate any error messages, but it will not do what you wanted it to do. It will do something else. Specifically, it will do what you told it to do.

The problem is that the program you wrote is not the program you wanted to write. The meaning of the program (its semantics) is wrong. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

## 1.7. Experimental debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.

In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

Debugging is also like an experimental science. Once you have an idea what is going wrong, you modify your program and try again. If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one. As Sherlock Holmes pointed out, “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” (A. Conan Doyle, The Sign of Four)

For some people, programming and debugging are the same thing. That is, programming is the process of gradually debugging a program until it does what you want. The idea is that you should start with a program that does something and make small modifications, debugging them as you go, so that you always have a working program.

For example, Linux is an operating system kernel that contains millions of lines of code, but it started out as a simple program Linus Torvalds used to explore the Intel 80386 chip. According to Larry Greenfield, one of Linus’s earlier projects was a program that would switch between displaying AAAA and BBBB. This later evolved to Linux (The Linux Users’ Guide Beta Version 1).

Later chapters will make more suggestions about debugging and other programming practices.

## 1.8. Formal and natural languages

**Natural languages** are the languages that people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.

**Formal languages** are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules. And more pertinent to our subject:

Programming languages are formal languages that have been designed to express computations.

Formal languages tend to have strict rules about syntax. For example, 3+3=6 is a syntactically correct mathematical statement, but 3=+6$ is not. H2O is a syntactically correct chemical name, but 2Zz is not.

Syntax rules come in two flavors, pertaining to **tokens** and structure. Tokens are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with 3=+6$ is that $ is not a legal token in mathematics (at least as far as we know). Similarly, 2Zz is not legal because there is no element with the abbreviation Zz.

The second type of syntax rule pertains to the **structure** of a statement— that is, the way the tokens are arranged. The statement 3=+6$ is structurally illegal because you can’t place a plus sign immediately after an equal sign. Similarly, molecular formulas have to have subscripts after the element name, not before.

When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is (although in a natural language you do this subconsciously). This process is called **parsing**.

For example, when you hear the sentence, “The old man kicked the bucket”, you understand that man is the subject and kick is the verb, and bucket the direct object. Once you have parsed a sentence, you can figure out what it means, or the **semantics** of the sentence. Assuming that you know what a man and a bucket are and what it means to kick, you will understand the general implication of this sentence.

Of course one of the challenges of natural languages is that they are full of [idioms](http://en.wikipedia.org/wiki/Idiom). When we say “The old man [kicked the bucket](http://en.wikipedia.org/wiki/Kick_the_bucket)”, there is actually no bucket at all and we actually are saying that the old man died. You won’t see this kind of thing in formal languages like Python.

Although formal and natural languages have many features in common — tokens, structure, syntax, and semantics — there are many differences:

**ambiguity**

Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.

**redundancy**

In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.

**literalness**

Formal languages mean exactly what they say. On the other hand, natural languages are full of idiom and metaphor. If someone says, “The old man kicked the bucket”, there is probably no bucket and nothing was actually kicked.

People who grow up speaking a natural language — everyone — often have a hard time adjusting to formal languages. In some ways, the difference between formal and natural language is like the difference between poetry and prose, but more so:

**poetry**

Words are used for their sounds as well as for their meaning, and the whole poem together creates an effect or emotional response. Ambiguity is not only common but often deliberate.

**prose**

The literal meaning of words is more important, and the structure contributes more meaning. Prose is more amenable to analysis than poetry but still often ambiguous.

**computer program**

The meaning of a computer program is unambiguous and literal, and can be understood entirely by analysis of the tokens and structure.

Here are some suggestions for reading programs (and other formal languages). First, remember that formal languages are much more dense than natural languages, so it takes longer to read them. Also, the structure is very important, so it is usually not a good idea to read from top to bottom, left to right. Instead, learn to parse the program in your head, identifying the tokens and interpreting the structure. Finally, the details matter. Little things like spelling errors and bad punctuation, which you can get away with in natural languages, can make a big difference in a formal language.

## 1.9. Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.

For this reason, it is sometimes helpful to add notes to your programs to explain in natural language what the program is doing.

A **comment** in a computer program is text that is intended only for the human reader - it is completely ignored by the interpreter.

**3.2 Using conditional**

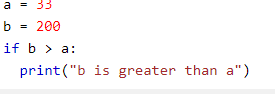
## Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

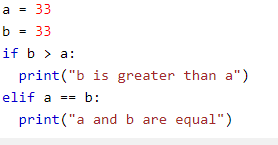
These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

****

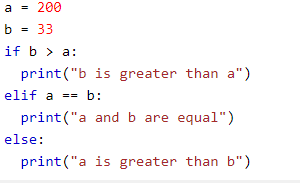
## Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

****

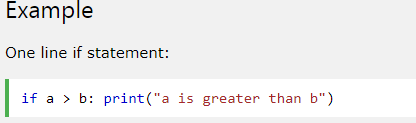
## Else

The else keyword catches anything which isn't caught by the preceding conditions.

****

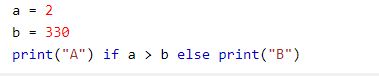
## Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

****

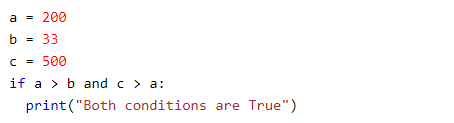
## Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

****

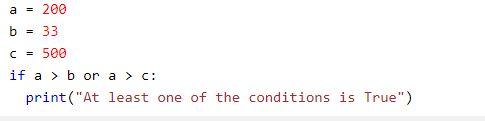
## And

The and keyword is a logical operator, and is used to combine conditional statements:

****

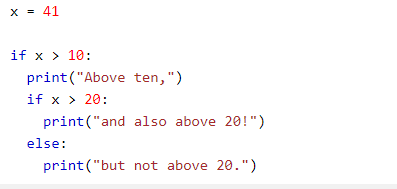
## Or

The or keyword is a logical operator, and is used to combine conditional statements:

****

## Nested If

You can have if statements inside if statements, this is called nested if statements.

****

## The pass Statement

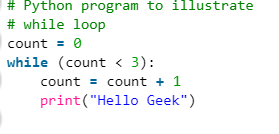
if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

****

**3.3 Using loops**

Python programming language provides following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. **While Loop:**
2. In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

****

1. **for in Loop:**

 For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to [for each](https://www.geeksforgeeks.org/g-fact-40-foreach-in-c-and-java/) loop in other languages. Let us learn how to use for in loop for sequential traversals.

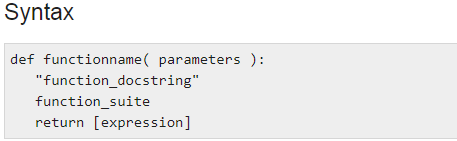
1. **Using else statement with for loops:**

We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.

**Nested Loops:**Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

**3.4 Using functions**

* Function blocks begin with the keyword **def** followed by the function name and parentheses ( ( ) ).
* Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
* The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
* The code block within every function starts with a colon (:) and is indented.
* The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

****

**Python OOPs concepts**

In this section, we will learn about basics of OOPs. Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism, inheritance etc.

**OOPs (Object Oriented Programming System)**

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

**Object**

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike, etc.

It can be physical and logical.

Objects can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing details of each other’s data or code, the only necessary thing is that the type of message accepted and type of response returned by the objects.

**Class**

Collection of Objects is called class. It is a logical entity. A class can also be defined as a blueprint from which we can create an individual object. Class doesn’t store any space.

**Inheritance**

When one object acquires all the properties and behaviours of parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Terms used in Inheritance**

**Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

**Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

**Super Class/parent Class:** There was a fair amount of churn in the terminology during the first years of object oriented programming as various people worked in the area and published papers and books and developed Object Oriented Languages. It was all quite new and exciting and people were trying to decide the proper vocabulary to use so they were trying out various words and phrases to express Object Oriented concepts.

**Python - Basic Syntax**

Here, you will learn the basic syntax of Python 3.x.

Just like natural languages, a computer programming language comprises of a set of predefined words which are called keywords. A prescribed rule of usage for each keyword is called a syntax.

Python 3.x interpreter has 33 keywords defined in it. Since they have a predefined meaning attached, they cannot be used for any other purpose. The list of Python keywords can be obtained using the following help command in Python shell.

## Python Identifiers

Apart from keywords, a Python program can have variables, functions, classes, modules, packages etc. Identifier is the name given to these programming elements. An identifier should start with either an alphabet letter (lower or upper case) or an underscore (\_). After that, more than one alphabet letters (a-z or A-Z), digits (0-9) or underscores may be used to form an identifier. No other characters are allowed.

* Conventionally, the name of the class begins with an uppercase alphabet letter. Others start with lowercase alphabet letters.
* Use of one or two underscore characters has a special significance when naming the instance attributes of a class. More about this will follow in the discussion about inheritance.
* Two leading and trailing underscores are used in the language itself for a special purpose. For example (e.g. \_\_add\_\_, \_\_init\_\_)

## Python Statement

By default, the Python interpreter treats a piece of text terminated by hard carriage return (new line character) as one statement. It means each line in a Python script is a statement. (Just as in C/C++/C#, a semicolon ; denotes the end of a statement).

# Python Data Types

Data types are the classification or categorization of data items. Data types represent a kind of value which determines what operations can be performed on that data. Numeric, non-numeric and Boolean (true/false) data are the most used data types. However, each programming language has its own classification largely reflecting its programming philosophy.

Python has the following standard or built-in data types:

## Numeric

A numeric value is any representation of data which has a numeric value. Python identifies three types of numbers:

* **Integer:** Positive or negative whole numbers (without a fractional part)
* **Float:** Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation
* **Complex number:** A number with a real and imaginary component represented as x+yj. x and y are floats and j is -1(square root of -1 called an imaginary number)

## Boolean

Data with one of two built-in values True or False. Notice that 'T' and 'F' are capital. true and false are not valid booleans and Python will throw an error for them.

## Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

* **String**: A string value is a collection of one or more characters put in single, double or triple quotes.
* **List** : A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
* **Tuple**: A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

## Dictionary

A dictionary object is an unordered collection of data in a key:value pair form. A collection of such pairs is enclosed in curly brackets. For example: {1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}

## Mutable and Immutable Objects

Data objects of the above types are stored in a computer's memory for processing. Some of these values can be modified during processing, but contents of others can't be altered once they are created in the memory.

Number values, strings, and tuple are immutable, which means their contents can't be altered after creation.

On the other hand, collection of items in a List or Dictionary object can be modified. It is possible to add, delete, insert, and rearrange items in a list or dictionary. Hence, they are mutable objects.

**Python - While Loop**

Loop is a very popular phrase in programming jargon. A program, by default, follows a sequential execution of statements. If the program flow is directed towards any of the earlier statements in the program, it constitutes a loop. However, sending it unconditionally causes an infinite loop, which is not desired.

Python uses the while and for keywords to constitute a conditional loop, by which repeated execution of a block of statements is done until a Boolean expression is true. Here, you will learn about the while loop.

The following is the while loop syntax.

Python keyword while has a conditional expression followed by the : symbol to start a block with increased indent. This block has statements to be executed repeatedly. Such a block is usually referred to as the body of the loop. The body will keep executing till the condition remains true. If and when it turns out to be false, the program will come out of the loop. Consider the following example.

Example: while loop

 Copy

num =0

while num< 5:

num = num + 1

print("num =", num)

Result:

num = 1

num = 2

num = 3

num = 4

num = 5

Here the repetitive block after the while statement involves incrementing the value of an integer variable and printing it. Before the block begins, the variable num is initialized to 0. Till it is less than 5, num is incremented by 1 and printed to display the sequence of numbers, as above.

Let us develop a Python program that successively takes a number as input from the user and calculates the average, as long as the user enters a positive number. Here, the repetitive block (the body of the loop) asks the user to input a number, adds it cumulatively and keeps the count if it is non-negative.

Example: while loop

 Copy

num=0

count=0

sum=0

while num>=0:

num=int(input("enter any number .. -1 to exit\n"))

if num>=0:

count=count+1 #this counts number of inputs

sum=sum+num # this adds input number cumulatively.

avg=sum/count

print ("numbers input:",count, "average:",avg)

When a negative number is provided by the user, the loop terminates and displays the average of the numbers provided so far. A sample run of the above code is below:

Result:

enter any number .. -1 to exit

10

enter any number .. -1 to exit

20

enter any number .. -1 to exit

30

enter any number .. -1 to exit

-1

numbers input: 3 average: 20.0

**SQLITE DATABASE CONNECTIVITY**

Installation. **SQLite3** can be integrated with **Python** using **sqlite3** module, which was written by Gerhard Haring. It provides an SQL interface compliant with the **DB**-API 2.0 specification described by PEP 249. You do not need to install this module separately because it is shipped by default along with **Python** version 2.5

It consist of following steps:

1. Creating a SQLITE database.
2. Query
3. INSERT query
4. DELETE query
5. Accesing through python
6. Connection method and creating database.
7. Creating a new record.
8. To accept user input
9. Error handling in python.
10. Retrieving new record with fetchone(),fetchall()
11. Update a record

**DEVELOPING A GUI WITH PYQT**

PyQt is a GUI widgets toolkit. It is a Python interface for **Qt**, one of the most powerful, and popular cross-platform GUI library. PyQt was developed by RiverBank Computing Ltd. The latest version of PyQt can be downloaded from its official website − [riverbankcomputing.com](https://riverbankcomputing.com/software/pyqt/download)

PyQt API is a set of modules containing a large number of classes and functions. While **QtCore** module contains non-GUI functionality for working with file and directory etc., **QtGui** module contains all the graphical controls. In addition, there are modules for working with XML **(QtXml)**, SVG **(QtSvg)**, and SQL **(QtSql)**, etc

In this section we will learn about:

1.GUI and Event driven Qt Designer

2. Using Common Widgets

3.Geometry Management

4.Designing Menu programming systems.

**MOTIVATIONS**

Over 15 million people participate in online fantasy sports. Applying a uses and gratifications framework, we use Q-methodology, a quantitative means for developing typologies of people, to examine types of online fantasy sports users and their motivations. Five types of players emerged, with casual players, skilled players, and isolationist thrill-seekers being the three most common types. Differences among types of users were primarily associated with two motivations—arousal and surveillance—while entertainment, escape, and social interaction motivations were judged to be less important. The minimal importance of social interaction to fantasy sports users in this study was unexpected, based on previous research, and implies that not all online communities build or maintain relationships.

**RESULTS**

**PYTHON FANTASY CRICKET GAME**

It is Python GUI Game where you create a virtual team of real cricket players and score points depending on how your chosen players perform in real life matches.

**REQUIREMENTS FOR RUNNING PROGRAM**

1. Install PyQT5 Package

* Open Command Prompt by using Shortcut (Window key+R) and type cmd.
* Type following command in cmd :-

pip install pyqt5

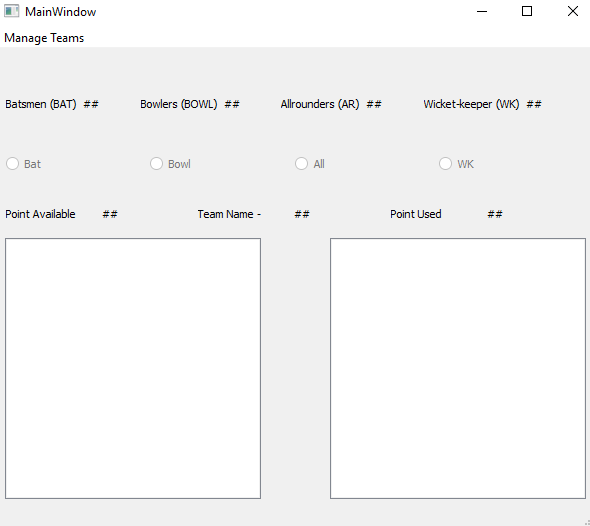
1. Install sqlite3 Package

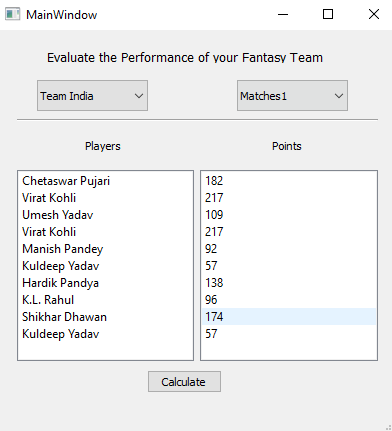
* Open Command Prompt by using Shortcut (Window key+R) and type cmd.
* Type following command in cmd :-
  + pip install db-sqlite3

1. Install Sqlite3 Studio

### Run main.py file on CMD.

**SCREENSHOTS**



****

**FEATURES:**

* Create your own dream team in Auction
* Save your team and evaluate it later as per **Match**
* Initially a Owner will be provided with 1000 Points to purchase there players
* Team selection follows basic cricketing rules
* Not more than 5 batsman/bowler
* Only a Single Wicket Keeper

**TOOLS USED**

1.ATOM

2.SQLITE STUDIO

3.INTERNSHALA SUMMER TRAININGS

4.PYTHON

**LEARNING OUTCOMES AND IMPORTANCE**

There is no denying the fact that [sports](https://www.analyticsindiamag.com/why-sports-analytics-is-more-than-just-a-numbers-game/) in India is a big industry. Started in 2008, Indian Premier League is a [cricket](https://www.analyticsindiamag.com/data-redefining-game-of-cricket-sports-technologies-lend-umpires-a-helping-hand/) league that has taken cricket to a whole new world, making India the hotbed for cricket enthusiasts.

To push the sports industry in India ahead, technology over the years has not left any stone unturned, and one of the most popular concepts is fantasy sports. Fantasy sports is a type of online game where participants form an imaginary or virtual team of real players of a professional sport and then compete based on the statistical performance of those players’ in actual games.

There is no denying the fact that [sports](https://www.analyticsindiamag.com/why-sports-analytics-is-more-than-just-a-numbers-game/) in India is a big industry. Started in 2008, Indian Premier League is a [cricket](https://www.analyticsindiamag.com/data-redefining-game-of-cricket-sports-technologies-lend-umpires-a-helping-hand/) league that has taken cricket to a whole new world, making India the hotbed for cricket enthusiasts.

To push the sports industry in India ahead, technology over the years has not left any stone unturned, and one of the most popular concepts is fantasy sports. Fantasy sports is a type of online game where participants form an imaginary or virtual team of real players of a professional sport and then compete based on the statistical performance of those players’ in actual games.

What started as a concept, today is one of the [rising markets in India](https://www.analyticsindiamag.com/scoring-analytics-top-sports-analytics-companies-india/). Fantasy sports platforms are getting significant funding from prominent VCs and the popularity is getting bigger and better. Started in 2012, Dream11 is India’s Biggest Sports Gaming platform with more than 4 crore users playing Fantasy Cricket, Football, Kabaddi and NBA. The Series D funded startup’s platform allows Indian sports fans to showcase their sports knowledge.

### When Sports Meets AI

We all know that AI over the years has found applications in several industries and time again the tech has proved that it has a major business impact — serving clients faster, better and make more money.

However, it is not just the businesses that are using artificial intelligence, sports gaming platforms are also in the rat race to become top AI-driven platforms. And being a top sports gaming platform, Dream 11 is leveraging the superpowers of AI to make the experience better for its users.

When it comes to fantasy sports in India, the IPL season is something that is really critical for platforms like Dream 11 —not only in terms of performance but also in terms of customer support. In order to deliver top-notch customer support, Dream 11 uses an AI-based bot. Created by[Haptik](https://haptik.ai/), the Dream 11[customer support bot](https://haptik.ai/casestudy-dream11) is a conversational AI that handles the massive scale of incoming requests by delivering instant responses without making anxious users wait. Also, the bot was designed and trained in such a way that it was able to answer most of the game related queries.

**APPENDIX**

**Code for team evaluation**:

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'Team\_Evaluation.ui'

#

# Created by: PyQt5 UI code generator 5.6

#

# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

import sqlite3

selectedteam = 'Select Team'

selectedmatch = 'Select Match'

class Ui\_evaluate\_team(object):

def CalculatePlayerScore(self, l):

"""

INPUT: A tuple containing the stats of a player

OUTPUT: An integer denoting the score of the player

"""

#INITIALIZE ALL THE VARIABLES REQUIRED TO CALCULATE THE SCORE

score = 0

name = l[0]

runsscored = l[1]

ballsfaced = l[2]

fours = l[3]

sixes = l[4]

bowled = l[5]

runsgiven = l[6]

wickets = l[7]

catches = l[8]

stumpings = l[9]

runouts = l[10]

#CALCULATE THE BATTING SCORES

score += runsscored//2

if runsscored >= 50:

score += 5

if runsscored >= 100:

score += 10

if ballsfaced > 0:

strike\_rate = (runsscored / ballsfaced) \* 100

if strike\_rate > 100:

score += 4

elif strike\_rate >= 80:

score += 2

score += fours + (2 \* sixes)

#CALCULATE THE BOLWLNG SCORES

score += (10 \* wickets)

if wickets >= 3:

score += 5

if wickets >= 5:

score += 10

if bowled > 0:

economy\_rate = (runsgiven / bowled) \* 6

if economy\_rate >= 3.5 and economy\_rate <= 4.5:

score += 4

elif economy\_rate >= 2 and economy\_rate < 3.5:

score += 7

elif economy\_rate < 2:

score += 10

#CALCULATE THE FIELDING SCORES

score += 10 \* (catches + stumpings + runouts)

#DISPLAY THE SCORES OF THE PLAYERS

self.players.addItem("{}".format(name));

self.scores.addItem("{}".format(str(score)));

return score

def CalculateMatchScore(self):

"""

INPUT: The Team Name and Match no.

OUTPUT: The total SCORE of the TEAM in the given MATCH

"""

\_translate = QtCore.QCoreApplication.translate

global selectedteam

global selectedmatch

self.players.clear()

self.scores.clear()

conn = sqlite3.connect('Matches.db')

c = conn.cursor()

totalscore = 0

c.execute("SELECT \* from Teams WHERE teamname = ?", (selectedteam,))

t = c.fetchone()

t = t[1:]

for i in t:

c.execute("SELECT \* from {} WHERE name = ?".format(selectedmatch), (i,))

l = c.fetchone()

totalscore += self.CalculatePlayerScore(l)

self.points.setText(\_translate("evaluate\_team", "{}".format(str(totalscore))))

c.close()

conn.close()

def TeamChanged(self):

global selectedteam

global selectedmatch

\_translate = QtCore.QCoreApplication.translate

selectedteam = self.select\_team.currentText()

selectedmatch = self.select\_match.currentText()

if selectedteam != 'Select Team' and selectedmatch != 'Select Match':

self.Evaluate\_btn.setEnabled(True)

self.players.clear()

self.scores.clear()

self.points.setText(\_translate("evaluate\_team", "0"))

else:

self.Evaluate\_btn.setEnabled(False)

def MatchChanged(self):

global selectedteam

global selectedmatch

\_translate = QtCore.QCoreApplication.translate

selectedmatch = self.select\_match.currentText()

selectedteam = self.select\_team.currentText()

if selectedmatch != 'Select Match' and selectedteam != 'Select Team':

self.Evaluate\_btn.setEnabled(True)

self.players.clear()

self.scores.clear()

self.points.setText(\_translate("evaluate\_team", "0"))

else:

self.Evaluate\_btn.setEnabled(False)

def setupUi(self, evaluate\_team):

evaluate\_team.setObjectName("evaluate\_team")

evaluate\_team.resize(633, 452)

self.label = QtWidgets.QLabel(evaluate\_team)

self.label.setGeometry(QtCore.QRect(130, 10, 361, 51))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(12)

self.label.setFont(font)

self.label.setObjectName("label")

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(10)

self.select\_team = QtWidgets.QComboBox(evaluate\_team)

self.select\_team.setGeometry(QtCore.QRect(90, 70, 141, 21))

self.select\_team.setFont(font)

self.select\_team.setCurrentText("Select Team")

self.select\_team.setObjectName("select\_team")

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(10)

self.select\_match = QtWidgets.QComboBox(evaluate\_team)

self.select\_match.setGeometry(QtCore.QRect(360, 70, 141, 22))

self.select\_match.setFont(font)

self.select\_match.setCurrentText("Select Match")

self.select\_match.setObjectName("select\_match")

''''''

conn = sqlite3.connect('Matches.db')

c = conn.cursor()

c.execute("SELECT \* FROM Teams")

l = c.fetchall()

list1 = ['Select Team']

for teamname in l:

list1.append(teamname[0])

c.execute("SELECT name FROM sqlite\_master WHERE type='table';")

l = c.fetchall()

list2 = ['Select Match']

for matchname in l:

if matchname[0].startswith('Match'):

list2.append(matchname[0])

c.close()

conn.close()

''''''

''''''

self.select\_team.addItems(list1)

self.select\_match.addItems(list2)

''''''

''''''

self.select\_team.activated.connect(self.TeamChanged)

self.select\_match.activated.connect(self.MatchChanged)

''''''

self.line = QtWidgets.QFrame(evaluate\_team)

self.line.setGeometry(QtCore.QRect(40, 95, 551, 41))

self.line.setStyleSheet("color: rgb(0, 0, 0);")

self.line.setFrameShape(QtWidgets.QFrame.HLine)

self.line.setFrameShadow(QtWidgets.QFrame.Sunken)

self.line.setObjectName("line")

self.label\_2 = QtWidgets.QLabel(evaluate\_team)

self.label\_2.setGeometry(QtCore.QRect(80, 160, 81, 21))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(12)

font.setBold(False)

font.setWeight(50)

self.label\_2.setFont(font)

self.label\_2.setObjectName("label\_2")

self.label\_3 = QtWidgets.QLabel(evaluate\_team)

self.label\_3.setGeometry(QtCore.QRect(340, 160, 81, 21))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(12)

font.setBold(False)

font.setWeight(50)

self.label\_3.setFont(font)

self.label\_3.setObjectName("label\_3")

self.points = QtWidgets.QLabel(evaluate\_team)

self.points.setGeometry(QtCore.QRect(390, 160, 61, 20))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(12)

self.points.setFont(font)

self.points.setStyleSheet("color: rgb(7, 200, 168);")

self.points.setObjectName("points")

self.Evaluate\_btn = QtWidgets.QPushButton(evaluate\_team)

self.Evaluate\_btn.setEnabled(False)

self.Evaluate\_btn.setGeometry(QtCore.QRect(240, 410, 121, 31))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(10)

self.Evaluate\_btn.setFont(font)

self.Evaluate\_btn.setObjectName("Evaluate\_btn")

''''''

self.Evaluate\_btn.clicked.connect(self.CalculateMatchScore)

''''''

self.players = QtWidgets.QListWidget(evaluate\_team)

self.players.setGeometry(QtCore.QRect(80, 180, 191, 221))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(10)

font.setBold(True)

font.setWeight(75)

self.players.setFont(font)

self.players.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)

self.players.setDefaultDropAction(QtCore.Qt.IgnoreAction)

self.players.setSelectionMode(QtWidgets.QAbstractItemView.NoSelection)

self.players.setObjectName("players")

self.scores = QtWidgets.QListWidget(evaluate\_team)

self.scores.setGeometry(QtCore.QRect(340, 180, 191, 221))

font = QtGui.QFont()

font.setFamily("Comic Sans MS")

font.setPointSize(10)

font.setBold(True)

font.setWeight(75)

self.scores.setFont(font)

self.scores.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)

self.scores.setSelectionMode(QtWidgets.QAbstractItemView.NoSelection)

self.scores.setObjectName("scores")

self.retranslateUi(evaluate\_team)

QtCore.QMetaObject.connectSlotsByName(evaluate\_team)

def retranslateUi(self, evaluate\_team):

\_translate = QtCore.QCoreApplication.translate

evaluate\_team.setWindowTitle(\_translate("evaluate\_team", "Dialog"))

self.label.setText(\_translate("evaluate\_team", "Evaluate the Performance of your Fantasy Team"))

self.label\_2.setText(\_translate("evaluate\_team", "Players"))

self.label\_3.setText(\_translate("evaluate\_team", "Points"))

self.points.setText(\_translate("evaluate\_team", "####"))

self.Evaluate\_btn.setText(\_translate("evaluate\_team", "Calculate Score"))

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

evaluate\_team = QtWidgets.QDialog()

ui = Ui\_evaluate\_team()

ui.setupUi(evaluate\_team)

evaluate\_team.show()

sys.exit(app.exec\_())

1. **Code for team details**

# -\*- coding: utf-8 -\*-

import sqlite3

selectedteam = 'Fantasy11'

conn = sqlite3.connect('Matches.db')

c = conn.cursor()

'''

c.execute('drop table if exists Match1')

c.execute("""CREATE TABLE Match1 (

name text,

runsscored integer,

ballsfaced integer,

fours integer,

sixes integer,

bowled integer,

runsgiven integer,

wickets integer,

catches integer,

stumpings integer,

runouts integer

)""")

'''

'''

c.execute("""INSERT INTO Match5 VALUES

('Virat Kohli', 36, 54, 2, 0, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Yuvraj Singh', 12, 20, 1, 0, 48, 36, 1, 0, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Ajinkya Rahane', 8, 18, 0, 0, 0, 0, 0, 0, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Shikhar Dhawan', 34, 23, 8, 0, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('M.S.Dhoni', 13, 17, 1, 0, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Axar Patel', 8, 4, 2, 0, 48, 35, 1, 0, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Hardik Pandya', 0, 1, 0, 0, 54, 30, 2, 0, 0, 1)""")

c.execute("""INSERT INTO Match5 VALUES

('Ravindra Jadeja', 18, 10, 1, 1, 60, 50, 2, 1, 0, 1)""")

c.execute("""INSERT INTO Match5 VALUES

('Kedar Jadhav', 2, 4, 0, 0, 60, 57, 4, 0, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Ravichandran Ashwin', 23, 42, 3, 0, 56, 43, 2, 0, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Umesh Yadav', 50, 28, 5, 1, 0, 0, 0, 0, 2, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Jasprit Bumrah', 0, 0, 0, 0, 42, 22, 1, 0, 1, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Bhuwaneshwar Kumar', 19, 20, 2, 0, 42, 43, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Rohit Sharma', 115, 126, 11, 4, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match5 VALUES

('Dinesh Kartick', 29, 42, 3, 0, 0, 0, 0, 2, 0, 1)""")

'''

'''

c.execute("""INSERT INTO Match6 VALUES

('Virat Kohli', 102, 98, 8, 2, 0, 0, 0, 0, 0, 1)""")

c.execute("""INSERT INTO Match6 VALUES

('Yuvraj Singh', 12, 20, 1, 0, 48, 36, 1, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Ajinkya Rahane', 49, 75, 3, 0, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Shikhar Dhawan', 32, 35, 4, 0, 0, 0, 0, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('M.S.Dhoni', 56, 45, 3, 1, 0, 0, 0, 3, 2, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Axar Patel', 8, 4, 2, 0, 48, 35, 1, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Hardik Pandya', 42, 36, 3, 3, 30, 25, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Ravindra Jadeja', 18, 10, 1, 1, 60, 50, 2, 1, 0, 1)""")

c.execute("""INSERT INTO Match6 VALUES

('Kedar Jadhav', 65, 60, 7, 0, 24, 24, 0, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Ravichandran Ashwin', 23, 42, 3, 0, 60, 45, 6, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Umesh Yadav', 0, 0, 0, 0, 54, 50, 4, 1, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Jasprit Bumrah', 0, 0, 0, 0, 60, 49, 1, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Bhuwaneshwar Kumar', 15, 12, 2, 0, 60, 46, 2, 0, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Rohit Sharma', 46, 65, 5, 1, 0, 0, 0, 1, 0, 0)""")

c.execute("""INSERT INTO Match6 VALUES

('Dinesh Kartick', 29, 42, 3, 0, 0, 0, 0, 2, 0, 1)""")

'''

'''

c.execute("SELECT \* FROM Match5")

l = c.fetchall()

for i in l:

print(i)

'''

'''

c.execute("SELECT \* FROM PlayerDetails")

l = c.fetchall()

for i in l:

print(i)

c.execute("SELECT \* FROM Teams")

l = c.fetchall()

for i in l:

print(i)

'''

'''

c.execute("SELECT \* FROM sqlite\_master WHERE type='table';")

l = c.fetchall()

#for i in l:

# print(i)

for i in l:

if i[1].startswith('Match'):

print(i)

'''

'''

for name in res:

if name[0].startswith('Match'):

print(name[0])

'''

#c.execute("SELECT \* FROM Teams WHERE teamname = ?", (selectedteam,))

#print(c.fetchone())

#c.execute("SELECT \* FROM PlayerDetails")

#print(c.fetchall())

#c.execute("DELETE FROM Match6")

#c.execute('drop table if exists Match6')

#c.execute("SELECT \* FROM Teams")

#print(c.fetchall())

#Team\_name = 'Fantasy11'

'''

c.execute("SELECT \* FROM Teams")

l = c.fetchall()

flag = 0

for i in l:

if i[0] == Team\_name:

flag = 1

break

if flag == 0:

Team = ('Fantasy11', 'Virat Kohli', 'Yuvraj Singh', 'Ajinkya Rahane', 'Shikhar Dhawan', 'M.S.Dhoni', 'Axar Patel', 'Hardik Pandya', 'Ravindra Jadeja', 'Kedar Jadhav', 'Ravichandran Ashwin', 'Umesh Yadav')

c.execute("INSERT INTO Teams VALUES {}".format(Team))

else:

print("Teamname already exists")

'''

'''

Team = ('MightyLions', '', '', '', '', '', '', '', '', '', '', '')

c.execute("INSERT INTO Teams VALUES {}".format(Team))

'''

'''

c.execute("SELECT \* FROM Teams")

print(c.fetchall())

'''

#conn.commit()

c.close()

conn.close()