UPGRAD CERTIFICATION



DATA SCIENCE AND DATA ANALYTICS

Contents

**No table of contents entries found.**

# Module 0 – Problem Solving

Problem Solving is one of the essential skills needed in the corporate world right now. It is what the companies pay they employees for; it is what teams work day-and-night for.

## Introduction to Problem Solving

Problem Solving is at the centre-stage of all the kinds of revolutions that have happened in our history. It is what we, as humans, strive to do. It is important in both professional and personal lives.

**Problem Solving** is defined as the skill / ability of an individual to identify, evaluate, and find an optimal solution to a problem. This skill cannot be learned by watching any videos, or doing any course. It is gained from experience, from guidance, and from frequent discussions. However, there are some basic principles / elements of problem-solving, that are used by successful people around the globe.

Before we start rolling out these terminologies, we need to understand the meaning of a problem, and a solution.

**What is a Problem and a Solution?**

A *Problem*, in simple words, is a situation that is preventing a particular goal to be achieved. It is something being thrown at you in the form of a tricky situation, or a question.

A *Solution*, originate from the latin word, *Solverě*, means to find a way to solve a problem. Its etymology begins from old French word *Solucion*, and then developed in latin.

Both Problem and Solution are components of problem solving. Before we begin to understand problem solving, it is essential to understand these 2 components.

**Benefits of Problem Solving**

It might be argued to understand the importance and relevance of problem solving. It all boils down to one thing – What is the benefit of problem solving ?

problems help us see patterns to human needs and create innovations that impact people at scale. Whether it is the home decor lab by IKEA or a virtual showroom by Diesel, all entail the ability of a team to discover patterns and seize an opportunity.

Some key advantages of Problem solving are -

* **Identify a course of action to solve a problem**: Problem Solving helps us identify patterns, and according to those, design the correct course of action in order to solve a problem.
* **Addressing Risks Proactively**: A Good Problem Solver addresses risks proactively. It allows you to fix things when they break, and predict what risks or problems could arise in the future.
* **Seize Opportunities**: Problem Solving is not only about identifying how to solve a problem, or to pre-empt it, but also to use them as opportunities that no one else saw, and seize them.
* **Improve Professional Performance:** Problem Solving skills can make you a lynchpin anywhere you work. It can help you become a person your supervisors or colleagues can rely on, to address any issue. This will automatically improve your professional performance.
* **Advance in Career:** Having good problem-solving skills will differentiate you from the rest. Companies are looking for potential employees with excellent skills in problem solving. If you are strong in this area, any company will hire you, or your current employer will appreciate your value.

**Types of Problems**

There are many different types of problems that one encounter when they embark on a journey of problem-solving. These categories are not actually textually defined, but they are used by many people across the globe. These are –

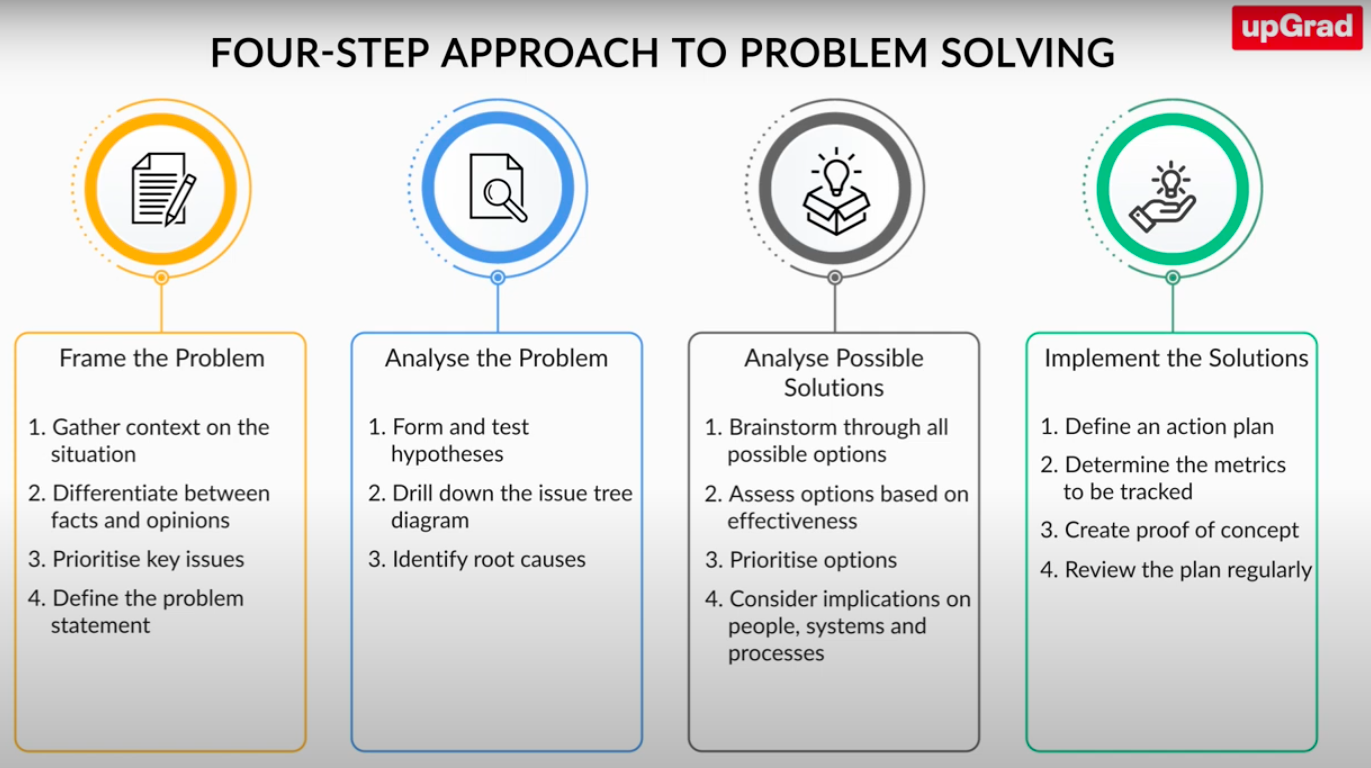
* **Unknown Causes Problems:** Many problems come across us, to which we have no clue how to approach. These problems are brand-new, completely unprecedented, and no one knows much about it. The best example of such problem is the Covid-19 virus, that we came across in late 2018.
* **Irrelevant Causes Problems:** Such problems are the ones that do not focus much on its cause, but rather on its resolution. For example, if you are playing a sport, and injure yourself, but your opponent is trying to win the game for their team, you will forget the cause, and try to resolve the problem.
* **Known Solutions but Uncertain Outcome Problems:** These problems are the ones where you know how to solve the problem, but you don’t know what will be the aftermath of the solution, and you cannot predict it. For eg. If a company wants to expand their business to a global market, and looking for various options, then there is no way to tell what an actual success would look like, for them.
* **Known Solutions and Certain Outcome Problems:** These are the easiest problems to solve. Most of the work of a problem solver is already done, before they even begin to solve it. Only thing they need to do, is to check which solution is the best-suited for them.

If we talk about the specificity of a business, then there are 3 types of problems in business:

* **Increasing the Profitability Of the Business**: How should the business become more profitable, is the main target here. It can be looked at a perspective of finding out a way to increase revenue, while reducing costs, and another way is to look at investment opportunities.
* **Challenge of Growth:** How could the company grow more, in a way of expansion, or in a way of more hiring, or entering new markets.
* **Efficiency of Value Chain:** The Value chain is the chain of activities a business needs to perform to serve their customers in the best way. Hence, its efficiency is very important.
* **Improving Customer Experience:** Customer is the god of each business. Hence, another main area to focus is to improve the customer’s experience by the company.
* **Conducting Mergers and Acquisitions:** How to merge and acquire other companies is another problem, that businesses need to address.

## Steps in Problem Solving

Now that the types of problems are defined, it is time to move to the steps of Problem Solving. There are usually 4 steps in this whole process, defined by the 4-Step Approach to Problem Solving Framework.



Now that there is a general idea on how this framework works, let us deep dive into all these steps, starting with framing the problem.

***Step 1 - Framing the Problem***

The first and one of the most important steps of this approach is to frame the problem precisely. If you are looking to solve a problem but are unable to frame the problem statement precisely, there is a high probability that you will end up not getting the desired results.

First of all, Do Not Jump To Solutions Immediately. These may work on a temporary basis, but for a real, authentic solution, you need to follow the method properly.

You can frame out a problem precisely by carrying out some key activities –

* **Establish the need for a solution**: You have to ask yourself what do you need to solve the problem. This is the most beginning phase of the framing the problem.
* **Justify the need for the solution**: You have to find reasons to justify your need for the solution. Once you have identified what do you need to solve the problem, you have to justify why you need to solve it.
* **Contextualize the problem:** After you have established why you need to solve the problem; you need to see what existing information you do have regarding the problem. This includes its origin and relevance of cause of the problem, failed solutions and many other things. Using them, it is time to finalize your problem.
* **Framing the final problem statement**: The final phase is to frame the problem statement on the basis of contextualization. This is where you define your target audience, your objectives, and requirements, based on all the information you gathered from contextualization.

While these phases help in creating the final problem statement, there are some questions in each phase that need to be answered –

|  |  |
| --- | --- |
| **Activity** | **Questions** |
| Establish the need for a Solution | * What do you need to find a solution? * What is the desired outcome? * Who benefits from the solution, and why? |
| Justify the need for the solution | * Is the effort aligned with the business strategy? * What are the desired benefits to the company, and how to measure them? * How to ensure the implementation of the solution? |
| Contextualize the problem | * What approaches have been tried, by the company, and by others? * What are the internal and external constraints on implementing a solution? |
| Forming the Problem Statement | * Is the problem a combination of many problems? * What requirements must the solution meet? * Which problem-solvers should we engage? |

**Facts and Opinions**

Whenever you are trying to gain context on a problem, you will come across several facts and opinions. But how would you determine which piece of information is a fact and which is just an opinion? To differentiate between a fact and an opinion, you first need to understand what they mean. The following table can be referred:

|  |  |  |
| --- | --- | --- |
| **Basis of Difference** | **Facts** | **Opinions** |
| Meaning | Facts refer to information that can be measured, observed, validated and proven | Opinions refer to personal views, which are influenced by the individual |
| Basis of Information | Facts are based on observation, research, or historical truth. Hence, they are objective reality. | Opinions are based on assumptions or personal views. Hence, they are subjective reality. |
| Representation | A fact represents something that happened / happen or was recorded. | An opinion, on the other hand, represents the perception of something by someone. |
| Verifiability | Facts can be conclusively verified. It can be through scientific journals, historical records etc. | Opinions are imprecise, and biased, and cannot be verifijed. Hence, they are always debatable. |

However, when you are solving a problem, you might face several issues other than the identification of facts and opinions. And you must understand that you do not have to make efforts to solve all these issues; some of them might just be causes of other major issues.

**Prioritizing Key Issues**

The next important step is to prioritise the key issues out of all the issues that you have identified while contextualising the problem.

Not all of these issues are even real; some are just symptoms. Others may be issues, but not significant ones. Hence, prioritization of key issues is important. **Deciding the criteria for prioritization is the key to problem solving**. It can be a number of criterions, like:

* Probability – chance of issue occurring more often
* Frequency – number of times issue has occurred
* Damage – the extent to which issue is damaging the company
* Impact – the scale of its impact on the company
* Cost – how much will it cost to rectify the issue
* Effort – how much effort will be required to solve the issue
* Time – how much time will it take to solve the issue
* Resources – how much of your resources (HR, finance) could be consumed while rectifying this issue.
* Feasibility – is the issue easy to fix, and readily feasible or not
* Payoff – how much could you gain from solving the issue

Once you have shortlisted the prioritisation criteria, it is important to get an opinion from all the stakeholders before finalising it and incorporate their suggestions if they fit the requirements.

**The S.M.A.R.T. Framework**

it is critical to frame the problem statement clearly. And we are not the only ones emphasising the importance of framing the problem; even Albert Einstein, who is one of the greatest problem solvers of all times, thinks the same. There is a famous quote by him that says:

‘If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.’

Now, you must be wondering how you can define the problem statement appropriately. Can you use a technique, or is it just trial and error? Thankfully, there is a technique for this, called the **SMART Framework.**



**Specific**: The problem statement should be very specific in language. Any generalizing statements should be avoided. You should narrow down the problem to a specific problem statement.

**Measurable**: The problem statement you define must be measurable, meaning that it should be quantified as much as possible. You should define the metrics associated with measuring the progress, and success of the solution.

**Attainable**: The problem statement should clearly define the nature of actions that need to be taken to solve it.

**Relevant**: The problem statement should be relevant to the problem. It should exclude anything irrelevant or insignificant relevance (lack of materiality).

**Time-Based**: The problem statement should clearly define the deadlines by which the solution should produce the desired outcome.

# Module 1 – Introduction to Python

Python can be considered as the default programming language for data science. Although there are multiple such languages, including R, Matlab, Scala, and many others, but python is preferred over others in the world currently.

## History of Python

Python is a general-purpose programming language that was named after Monty Python. It is simple and incredibly readable since it closely resembles the English language. But still, **why should you use Python?**

Python is a language that finds use in nearly every domain possible. Its official website will give you an overview of this. In addition, its simplicity, and the way it ensures tasks can be performed using fewer lines of code is encouraging many developers across the world to take it up.

It was developed by **Guido van Rossum**, who dedicated this name to the Monty Python’s British Circus performance that led him to create the language. It was started as a hobby, but soon, it became a general-purpose programming language.

## Features of Python

Python is the most preferred language for beginners to start programming, since the last 10 years. This is due to some of its features:

* It is easy to learn, since it resembles the general English language.
* It is a general-purpose language, unlike JavaScript, JAVA or C++, that can be used just for developing applications and not for other purposes such as testing, data analysis and many more.
* It is FOSS (Free and Open-Source Software), which makes it accessible to a wide range of communities.
* It has an increasingly large support for libraries, which makes building python packages much simpler and faster than other languages.
* It integrates very smoothly with other languages, so it is remarkably easy to convert code into python to other language, or to convert code from other language into python.
* It is an interpreter-based language, which means that it does not need compilation before every execution. The interpreter interprets each program as sequence of one or more sub-readings, all of them already compiled into machine code.
* It supports multiple programming paradigms, from Object Oriented Programming to Functional programming etc.

## Jupyter Notebook and Jupyter Lab

Notebook documents (or “notebooks”, all lower case) are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis.

The **Jupyter Notebook** App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. They have the extension of .ipynb(short for IPython Notebook).

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

However, recently, standalone Jupyter Notebooks have been going redundant. They are now re-branded as **Jupyter-Lab**. It is a lightweight online IDE, that opens various Jupyter notebooks in the browser.

**Elements of Jupyter Lab**

* Kernel: A notebook kernel is a “computational engine” that executes the code contained in a Notebook document. The Ipython kernel, executes python code. Kernels for many other languages exist (official kernels).

When you open a Notebook document, the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results. Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down.

* Notebook Dashboard: The Notebook Dashboard is the component which is shown first when you launch Jupyter Notebook App. The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown).

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files.

Jupyter Notebooks can be used for documenting your data science project, as it supports markdown texts as well as elements like images, colors, equations (using LaTEx).

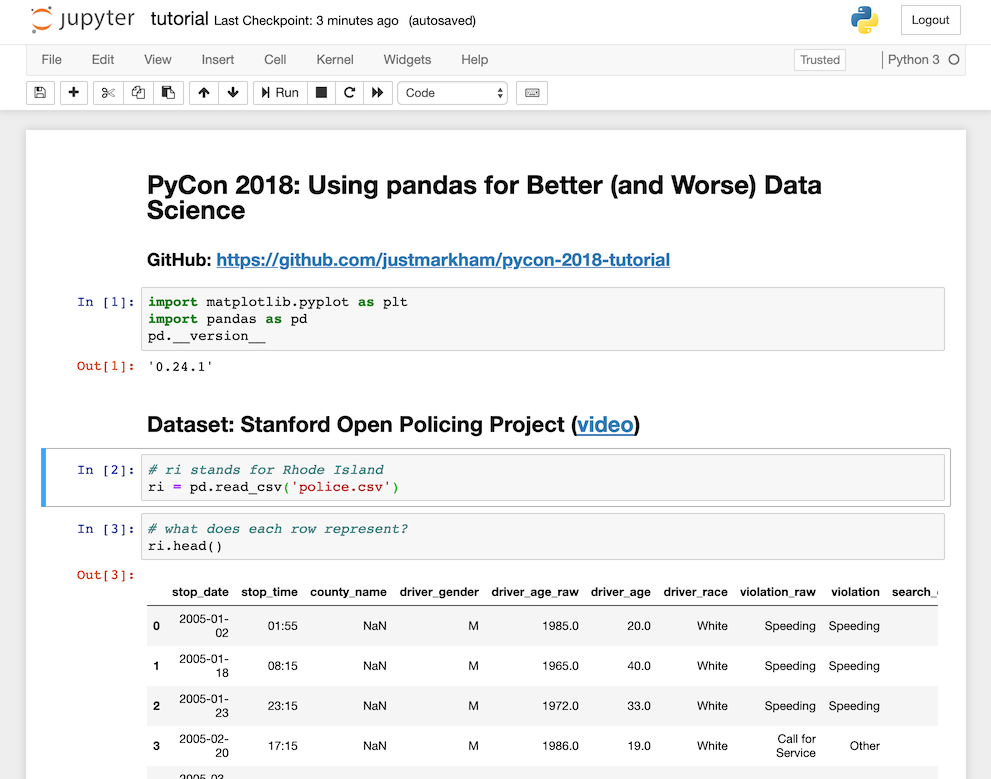
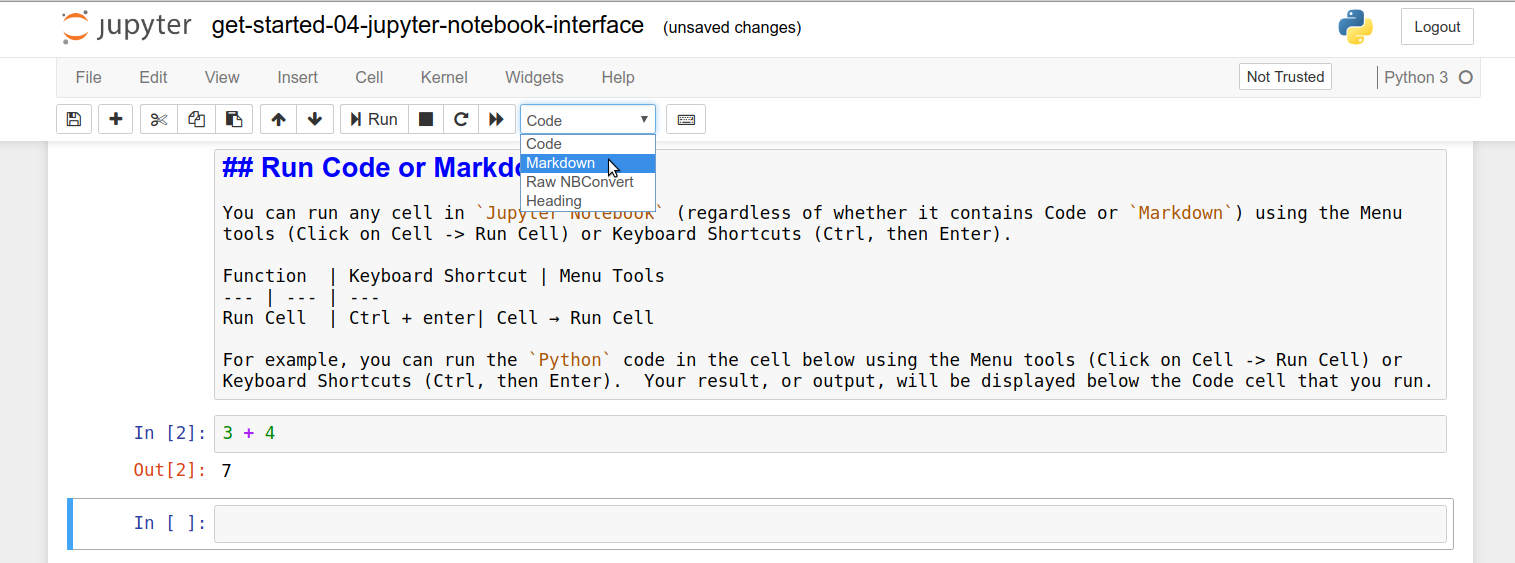


Figure 1. Jupyter Notebook Interface

**Markdown in Jupyter Notebook**

If you want to add markup in Jupyter Notebook, it is very simple. All you have to do, is go to code in the quick access toolbar, and change it to markdown, like this –



This will treat your inputs as plain text / markdown. To format things further, you can use the following notations, to add images, equations and other elements –

* ***Headings***

# for the titles

## for the main headings

### for the subheadings

#### for the smaller subheadings

##### for the italic subheadings

* ***Emphasis***

\_\_string\_\_ or \*\*string\*\* for bold text

\_string\_ or \*string\* for italic text

* ***Monospace fonts:*** A back single quotation mark ` on both sides to get monospace fonts
* ***Line breaks:***  <br> wherever you want a line break, as the notebook sometimes doesn't give you the required line break where you want it
* ***Indenting:***

> to indent the text

>> for further indenting it, and so on

* ***Bullets and numbering:*** A single dash, i.e. (-) followed by two spaces to make bullet points; A number and a dot followed by a space, i.e. 1. to make numbered lists.
* ***Colouring***

<font color = blue, yellow, red, pink, green, etc.> String </font> to give

your font any colour that you want

* ***LaTeX Equations:*** $ on both the sides of the text to write LaTeX equations.

Now that we have a basic idea of how to set up our workstation for python programming, let us move ahead and start learning python.

## Python Data Types

Now, it is time to start learning Python Code. Like any ordinary language, we will start with the very basic syllables that are defined in python – **Data Types.**

Before data, types, let us see what variables are, not just in python, but in any programming language.

**Variables**

Variables are like containers. Data is stored in memory locations, and variables are the references to those locations. Whenever you want to call that value, you use the variable. If the variable’s value changes, then the value in the memory also changes.

Variables are identified using **identifiers**, that are used to give definition to them. These identifiers are the names of the variables that help python to link the value to the variables.

Syntax: Identifier assignment-operator value;

*Eg.1. Name=“Yash”;*

*Assignment operators* are the operators that are used to define how to allot space to the variable in the python memory. These operators define what kind of space it will take. There are 2 types of operators –

* Arithmetic operators: plus (+), minus (-), multiply (\*), divide (/), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=).
* Logical Operators: AND, OR, NOT

To display the value of a variable, we use the print() function. This function displays the value inside the parenthesis to the console (output window). It has the following syntax –

Print(value);

Let us look at some examples now:

Name = ‘Yash’;

Print(Name); // referring to the value using the variable. If name

changes, so does the value.

Print(‘Yash’);

Print(“Yash”);

Output:

Yash

Yash

Yash

As the output shows, it does not matter if the value inside the parenthesis is under single quote or double quote; the value will be the same. This is another of python’s great features.

Some rules regarding variables that you need to remember are –

* The variable name (or identifier) cannot start with a number, i.e., declaring something like 2name = 7; it throws an error. Although, you can create an identifier having an alphanumeric value where number is anywhere but the 1st position, i.e. name1, na1me, nam1e are accepted.
* Python is case sensitive, or in other words, the python variables are case sensitive. This means that declaring name= 2 and Name= 4 would create two different variables.
* You can use hashes (-) and underscores(\_) while naming your identifiers. It is allowed, but they follow the same restriction that numbers do – they don’t come at the beginning of the name.

Coming on to **Data Types**, these can be considered as the syllables of the python language. Each data type has an allocated memory value, and so when a variable is defined, the data type tells python how much memory should the variable take.

In earlier versions (<Python 3), the use of the data type keywords (int, float etc) was necessary to declare a variable. Since python 3, it is not. You just have to assign the value to the variable, and python will automatically detect its data type. Another reason python is so popular !

There are 4 data types in python –

* Integer (int): As the name of the data type suggests, this data type is for the integer values, i.e. absolute numbers without decimals or fractions.
* Float (float): This data type is to define real numbers, that include integers, rational numbers, irrational numbers, decimals.
* String (str): This data type is used to represent strings – collection of characters in programming. It can have alphanumeric values, symbols and even whitespaces, but everything inside a string will be stored as a string character in the memory by python. It can be defined using either the single quotations (‘ ’) or the double quotations (“ “).
* Boolean (bool): This data type is used to represent Boolean values. It can have values True and False. The values are case-sensitive, and need to be written in sentence-case only.

To check the data type of a variable in python, we use the type() function. This function can check only 1 variable’s type at a time. It has the following notation-

Syntax: type(identifier);

Let us see an example of this to understand better.

x1=2;

x2=3.4;

x3=‘This is a String’;

x4=True;

print(type(x1),type(x2),type(x3),type(x4));

output:

<class int> <class float> <class str> <class bool>

As it is visible, python automatically detected the data types of all the variables without us mentioning their data types. Tells us how smart it actually is !

**TypeCasting**

If we want to change the data type of a variable, we can easily do it. This process is called TypeCasting. You can change from any data type to any other data type in python.

For typecasting, some rules need to be remembered:

* The Type of String is very flexible in terms of input. It can convert any input into a string value.
* The type of Int type can convert only strings having nothing but numbers. It can convert the float values also, but the values after the decimal will be removed.
* The Type of Bool has a restriction in input as well. Only the values of 1, 1.0, 0, 0.0 will be converted to boolean values. Others will return error. This is because the keywords of True and False have values of 1 and 0 in the memory.
* The Float has a restriction that strings should only contain decimal values, or integer values.

For typecasting, we use the inbuilt python functions int(value), float(value), str(value), and bool(value) to change to the desired data type.

Let us see an example for more information.

x1=1; x2=3.14;

x3=3.92; x4='This is Typecasting 101';

x5='True';

x6=str(x1); x7=float(x1);

x8=bool(x1); x9=0.0;

print(type(x6),type(x7),type(x8),type(x9));

print(x6,x7,x8,x9);

output:

<class 'str'> <class 'float'> <class 'bool'> <class 'float'>

1 1.0 True 0.0

Here, you can see that only the value 1 was converted to boolean value of true. Any other value will give error, since boolean does not accept any other inputs.

## Arithmetic Operations in Python

Performing simple arithmetic operations in Python is no big task. The print() function itself offers so much flexibility for us to display variables of multiple data types in one statement, that we just need to understand the principles.

There are 7 basic arithmetic operators in Python –

* **Addition (+)** – This operator performs simple arithmetic addition between variables or inputs. It could be float or int values. If you add the boolean values, they will just give the output of their numeric counterparts 1 and 0. You can only add strings to strings here, not strings to any other data type.
* **Subtraction (-)** – This operator performs simple arithmetic subtraction between the variables or inputs. The same rule as addition applies here, except you cannot subtract two strings.
* **Multiplication (\*)** – This operator performs simple arithmetic multiplication between variables or inputs. It behaves the same as addition. If you multiply a string with an integer, then it will appear that many times in the output.
* **Division (/) –** This operator performs simple division between variables or inputs. It behaves the same as subtraction does. But, it will always give a float output, even if the inputs are integers, and their quotient is also integer.
* **Exponential (\*\*)** – This function performs a simple exponential on the number. The number before is the base, whereas the number after is the exponent.
* **Floor Division (//)** – This function performs a division between variables, or inputs, but always gives an integer output, unless one of the values are in float, excluding the output, is in float.
* **Modulus (%)** – This function performs a remainder operation between two variables. It will behave similar to the floor division in terms of data type of the output.

In case of multiple operators being used at a same time, we need to remember the following ***operator precedence***:

Brackets ( ) > exponential (∗∗) > floor division (//) > modulus (%) > division (/) > multiplication (\*) > addition (+) > subtraction (−)

*Eg.1*

Print(4 % (1 + 9)\*\*2 - 60 // (7 + 2));

/\*

Step 1: Brackets: 4%10\*\*2 – 60//9;

Step 2: Exponential: 4%100 – 60//9;

Step 3: Floor Division: 4%100 – 6;

Step 4: Modulus: 4 – 6

Step 5: Subtraction: -2 -> Ans

\*/

Output:

-2

## String Operations and Methods in Python

A string is perhaps one of the most important data types, and hence, learning how to work with strings using Python is crucial. Performing string operations is a very crucial part of any programming language. Strings are immutable, which means they cannot be changed once created, and strings can be modified by slicing a part of it and concatenating. These operations include:

* Indexing: This means to access a specific element in a string. Always remember that forward indexing starts with 0, and reverse indexing starts with -1.
* Slicing: This means to get a substring from the original string.

Everything enclosed in a string is a collection of characters. Even the numbers, symbols and whitespaces included in a string are characters.

**The len()function**

A string is like a set of characters, in a given order. Hence, we can move to any character we want to, by navigating inside the string. But, for that, we need to first know how long the string actually is. We use the len() function for that.

Eg. If there is a string name=’Yash Jain’, then it has the characters – ‘Y’, ‘a’, ‘s’, ‘h’, ‘ ’, ‘J’, ‘a’, ‘i’, ‘n’. The length will be 9.

This function is used to calculate the number of characters in a string. It has the following syntax :

len(string);

**String Indexing**

Indexing is how string stores characters. Each character is given an index, in the same sequence that the string has them arranged, when the variable was declared.

When the string is shown in the output, the indexes are sorted, and the string gets the output in the same sequence. We have 2 ways to do indexing:

***Forward Indexing***: This method starts indexing from the first character, and the index starts at 0. Hence, if we have a string –

Str= ‘Yash Jain’

Then the indexing will be like this:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Char | Y | a | s | h |  | J | a | i | n |

To go to a specific character in the string, we use square brackets [ ] after the variable name, and in the square brackets, we write the index number.

For eg. Input: str[1], output: ‘a’

Input: str[4], output: ‘ ’ (blank space is also stored)

***Reverse Indexing***: Reverse Indexing is when we start indexing from the last element. The indexing starts from -1. The syntax remains the same for reverse indexing.

When you use a negative value inside the square brackets, python will automatically start negative indexing. Hence, in the same example, we have the reverse-indexing as:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| Char | Y | a | s | h |  | J | a | i | n |

Eg. Input: str[-3], Output: ‘a’;

Input: str[-5], Output: ‘ ’;

**String Immutability**: It is the property of the string that does not allow it to change the value of any character in the string. You can change the whole string by assigning it to some other string value, but you cannot change any character value.

*Eg.*

Str= ‘Python’;

Print(str);

Str[2]; //output will be ‘t’

Str[2]= ‘j’; //error – this is not allowed.

Str= ‘Pyjon’; // this is allowed

**String Concatenation**: This property of the string variables joins two more strings into a new string. It can be assigned to a variable, or it can be directly shown in the output using the print() function.

*Eg.*

str1= ‘Welcome’;

str2= ‘Python’;

str= str1 + ‘to’ + str2;

print(str); // method 1 – using a variable

print(str1 + ‘ to ’ + str2); // method 2 – direct output

**String Slicing**: This is an operation to extract a given part of a string. It can be used to take out a part of the string, and store it in some variable, or to create a new string, using a part of this string. It is used with the following syntax –

Identifier[start : end : step];

* *Start*: The first parameter inside the square brackets is *start*, which lets us define the index from which the slicing should start.
* *End*: The next parameter inside the square brackets is *end*, which lets us define the index at which slicing should end.
* *Step*: The last parameter is the *step*, which defines the jump between characters while navigating the element. The higher the value, the more characters it will jump in one go. Eg. If step is 2, then every 2nd characters will be displayed; if step is 3, then every 3rd character will be displayed and so on.

*Eg.1*

Name= ‘Yash Jain’;

Name[3:]; // starting from index 3 to the end, without any jumps

Name[:6] // ending at index 6 from the starting, without any jumps

Name[3:7:2] // starting at index 3 and ending at index 7, it will

jump 2 characters, i.e. alternate characters will be

displayed.

Name[:-2]; // ending at 2nd last character, using negative

indexing

Name[:-7:-2:2] // starting from 7th last character to 2nd last

character, while extracting every 2nd element.

**String Memberships**

The String Memberships are managed using the membership operators IN and NOT IN, that navigates through a string to see if the asked value is there in the string or not. It will always return a boolean value TRUE or FALSE, unless typecasting is used. These operations are case sensitive, and they will match for the exact value.

Syntax: value IN string;

value NOT IN string;

*Eg.1*

Str= ‘Indian Cricket Team’;

print(‘India’ IN Str); // boolean value TRUE

print(‘Delhi’ IN Str); //boolean value FALSE

print(‘Delhi’ NOT IN Str); //boolean value TRUE

print(‘India’ NOT IN Str); //boolean value FALSE

print(‘INDIA’ NOT IN Str); //boolean value TRUE due to case

sensitivity

**String Repetition**

Before, we learnt about arithmetic operators, where we saw that multiplication of an integer with a string will repeat it that many times. This is String Repetition. Let us see a very quick example of this:

*Eg.*  print(‘Ha ’\*4+‘ !’); // this will print 4 times Ha with a space

as well, since it is included in the

string.

Even if we add the numeric value before the asterisk(\*), and string after it, the output would be the same. This is due to the precedence of operators, and since addition is the only operation allowed with strings. Hence, the following code would also yield the same result :

print(4\*‘Ha ’+‘ !’);

**String Methods**