Contents

1. Python overview

As days pass on demand of the technology changes rapidly. Over the past few years, Python has become a Buzz word in the IT Industry especially in the area of data science and Artificial Intelligence. And this python programming has occupied in the top 3 lists of the language popularity. In today's world, data is a heart in the IT industry. But only a few people in the IT industry have the capability to process the data. And today many people called this as hottest job in the IT industry. Overview, This python overview makes people easy to learn **python for data science** as well as algorithm implementation.

According to the recent statistics we were 50% lagging in the supply of data scientist w.r.t the demand for a data scientist. And this python is one of the highest paying jobs in the IT industry. The average salary of the python developer ranges from $80000 to $95000. And managers make as much as $250000 per annum.

This tutorial will help you to learn the various concepts of programming language. And finally, helps you in clearing the python certification.

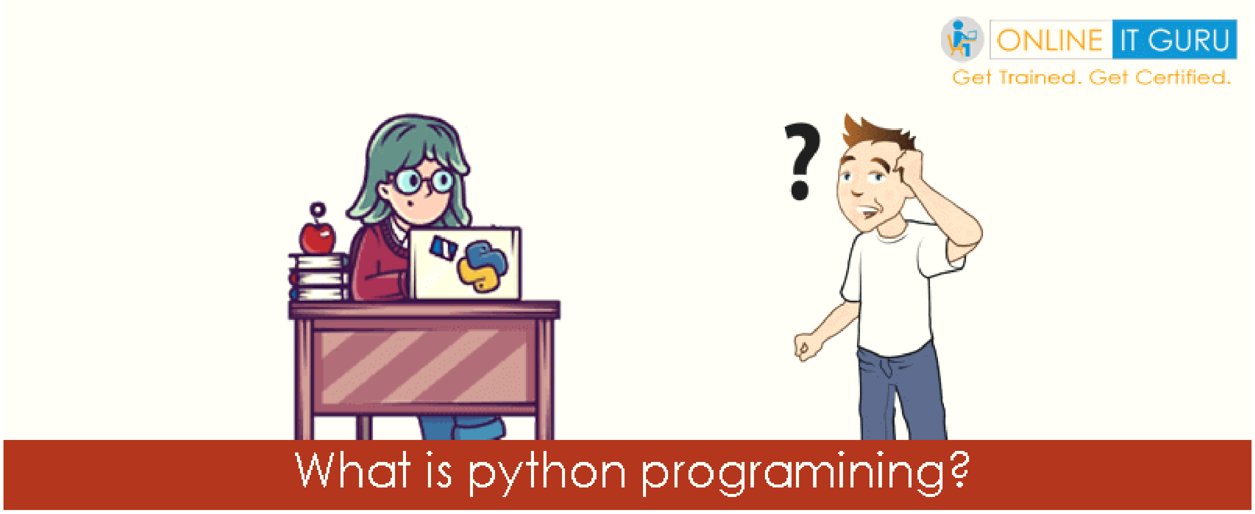
In this tutorial covers the basics of python programming, Control functions, data structures, modules, files, I/O functions, errors exceptions, oops concepts and so on.

Read the complete information about why do people prefer python?

2. Python Basics

What is python Programming?

Python is a high-level object-oriented programming language created by Guido Rossum in 1989. It provides constructs that enable clear programming on both large scale and small scale companies. Python. Its high-level built-in data structures combined with dynamic typing and dynamic binding makes it attractive for Rapid application development. Simple and easy to learn the syntax increases the readability and reduces the cost of the maintenance. Additionally, in python basics says that there is no compilation step and edit step debug is incredibly fast.



Click **what is python programming** for more elaborate details.

Version History:

Python has three major versions. Let us talk about one by one in detail.

Version 1.0:

Python 1.0 released in January 1994. This python version includes major new features like filters, lambda, and maps etc.

Version 2.0:

Python 2.0 was released in October 2000. In this versions features like full garbage collector, list comprehensions were included and support Unicode.

Version 3.0:

The latest version released in 2008. So, the major changes of these versions shown below:

In this python version, the print is a function.

Instead of lists, we have views and iterators.

More simplified rules for ordering the comparisons

Here when we divide two integers, the resultant results a float instead of an integer.

One *drawback of this versions it is not backward compact-able with python 2.x.*

Check once what are the major difference between python 2 & 3

Python Interpreter:

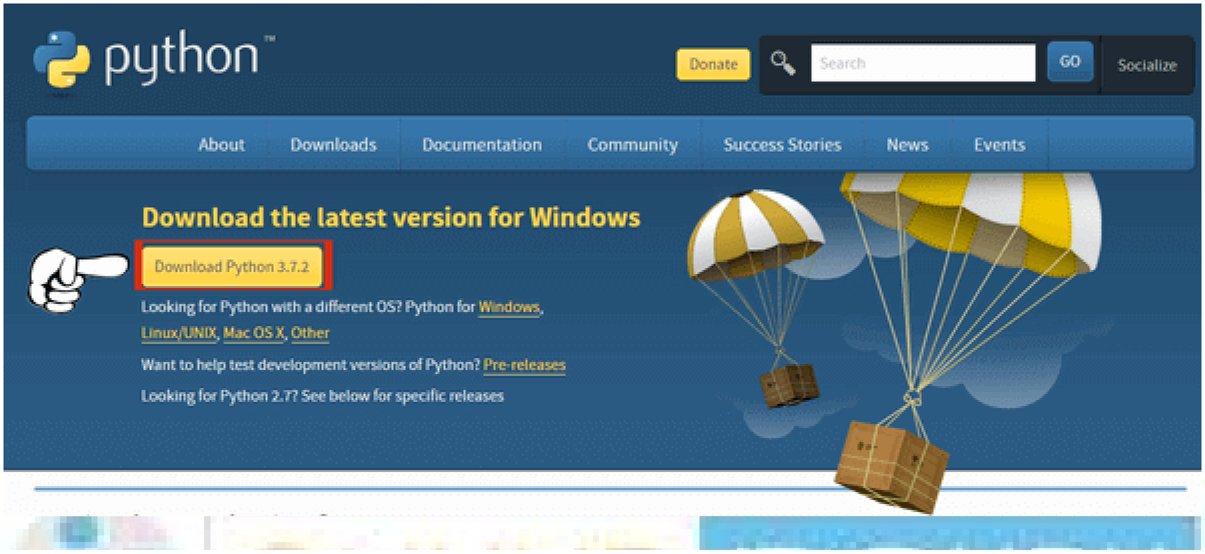
An interpreter is basically a computer programming languages that execute the program directly without compilation. So, it performs instructions written in programming (or) scripting language. Unlike JAVA, python uses interpreter

So now let me take to how to **install python** in the local system.

3. Install Python

We can install python on our local computer through the following steps:

1. Visit https://www.python.org/downloads/



Download the latest version for Windows

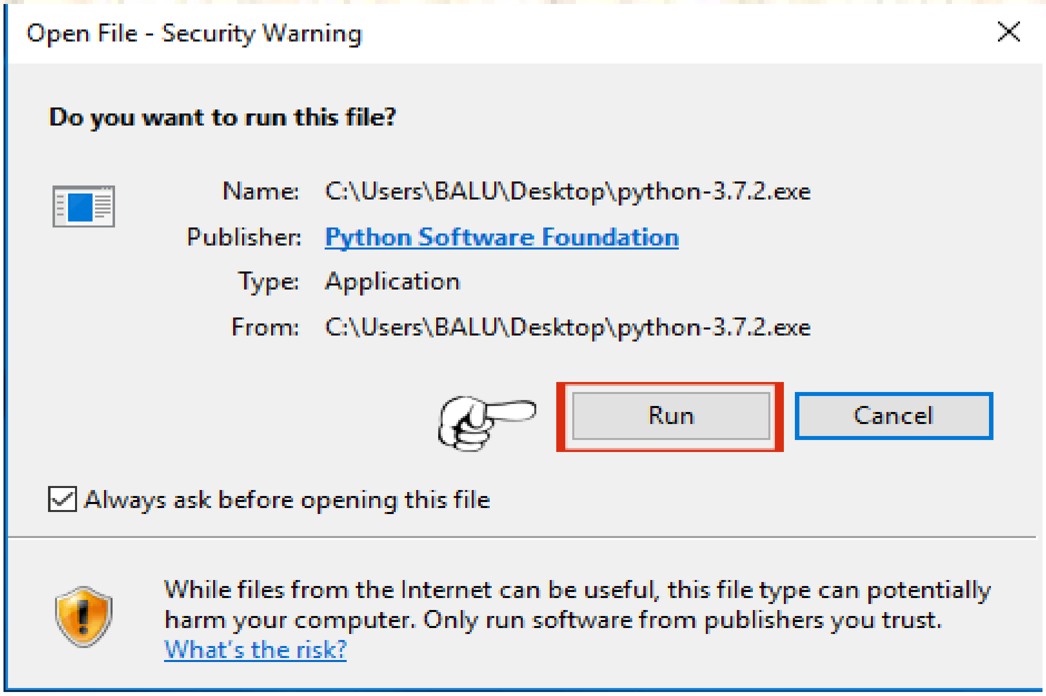
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for Python 2.7? See below for specific releases

1. Download the supported latest version (The installer is available for Windows, Mac and Linux operating systems)

2. Once you have downloaded click on **Run** to install the environment



Open File - Security Warning

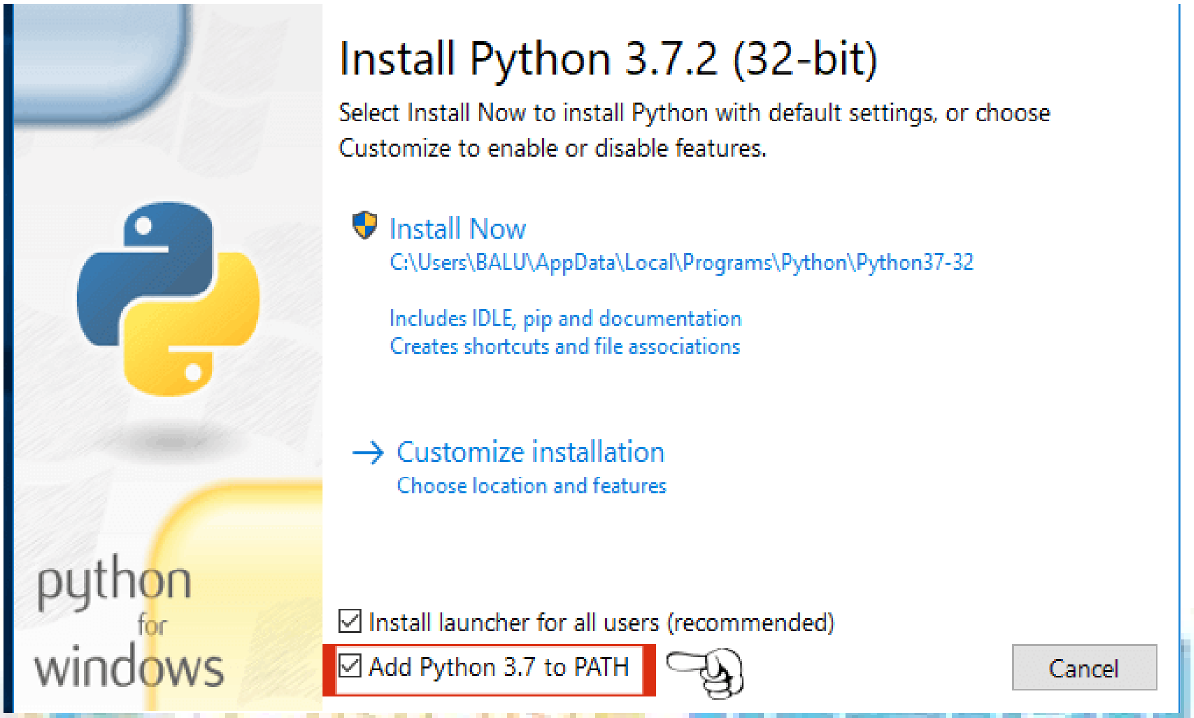
Do you want to run this file?

Name: C:\Users\BALU\Desktop\python-3.7.2.exe
Publisher: Python Software Foundation
Type: Application
From: C:\Users\BALU\Desktop\python-3.7.2.exe

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

3. Tick Add 3.7 to the path



Install Python 3.7.2 (32-bit)

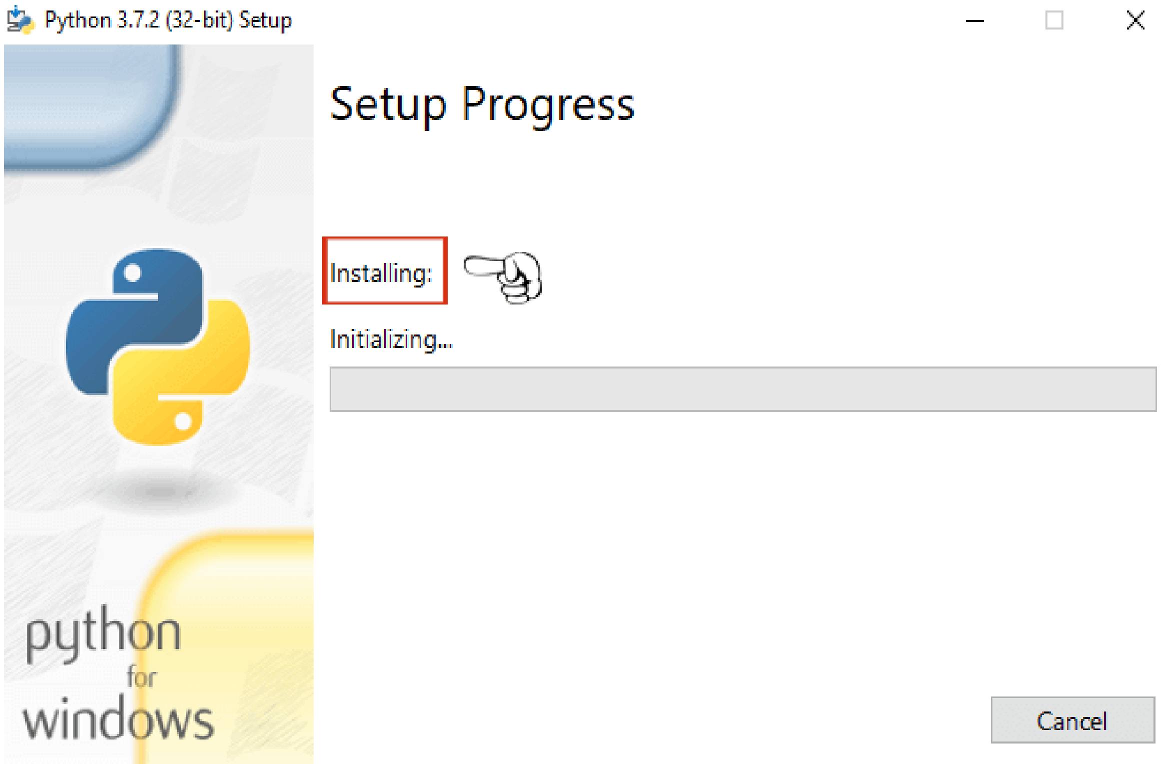
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\BALU\AppData\Local\Programs\Python\Python37-32
Includes IDLE, pip and documentation
Creates shortcuts and file associations
→ Customize installation
Choose location and features
python for windows
Cancel

☑ Install launcher for all users (recommended)

☑ Add Python 3.7 to PATH

5) Now click on install now to install the environment.

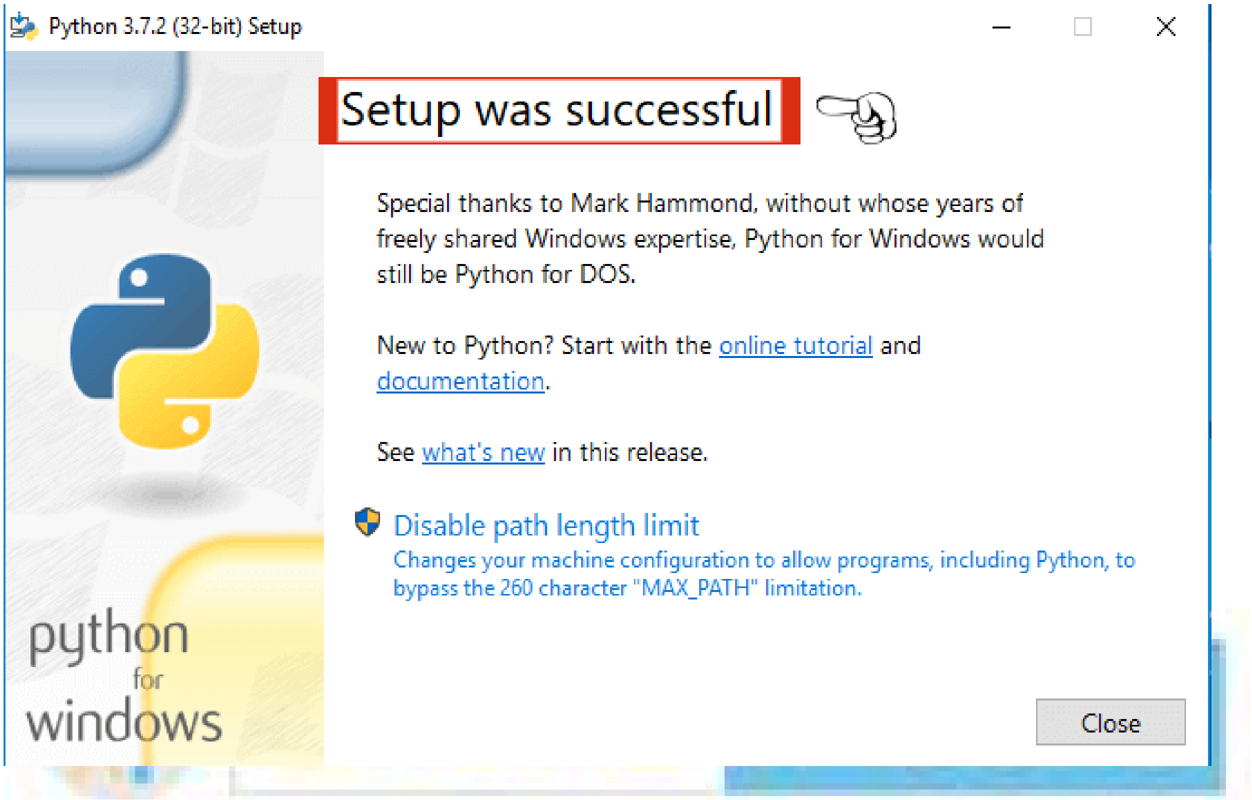


Setup Progress

Installing:

Initializing...

6) Finally, you will be seeing setup was successful.



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

IDLE is preferred scripting for Python. So go to your operating system search bar and search for IDLE. Then you will be entering into the following screen

This IDLE suits best for beginners. But this **Python IDLE** does not fit well for developing the application. So we need an alternative editor like IDE to develop the application using python. Today in the market, we do have many IDE's to learn Python **3.** But I prefer **Pycharm** as an editor to develop the applications.

But before going to know about pycharm installation, let us briefly discuss

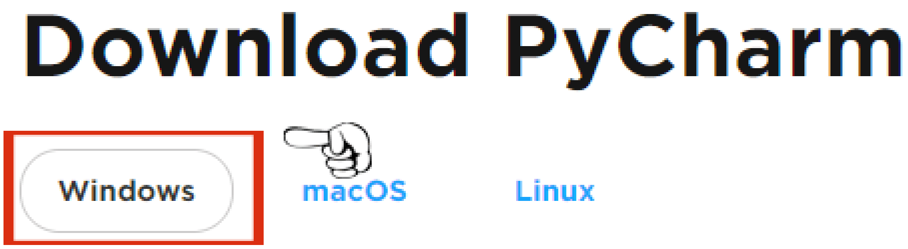
What is meant but IDE (Integrated Development Environment):

IDE typically provides Code editor, Compiler and debugger in one Graphical use Interface (GUI). It integrates the entire process of code creation, compilation, and testing. This finally increases the developer's productivity

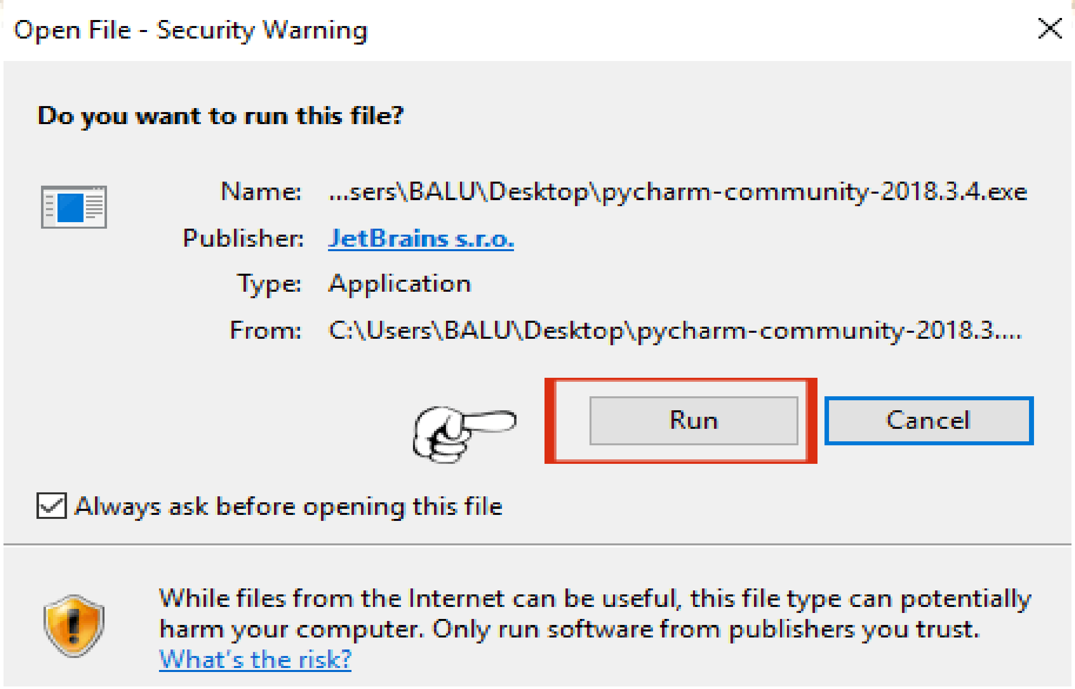
A developer working with IDE starts with a model. The IDE then converts the model into the code. Additionally, with a high level of automation, the IDE then debugs and then test the model driven code. And if once build is successful and properly tested, it can be deployed for the further testing through the IDE.

Now it's time to move into the installation of Pycharm IDE:

Visit https://www.jetbrains.com/pycharm/ to download the latest version of IDE.



Now it's time to select the operating systems. As of now, I opt **WINDOWS.** And this **pycharm** has two versions. (Community version is available for free and the professional version is a licensed version.) As a beginner, we would opt the community version. Now it's time to download the Community version. After downloading the file, click on **RUN** to install the file.



Open File - Security Warning

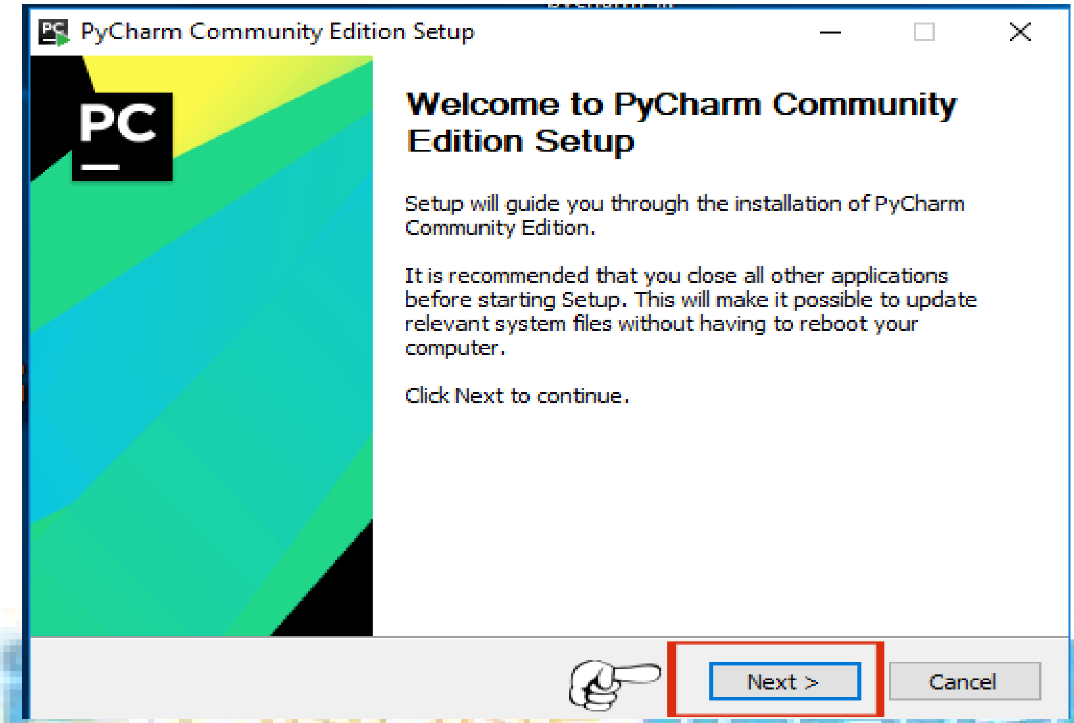
Do you want to run this file?

Name: ...sers\BALU\Desktop\pycharm-community-2018.3.4.exe
Publisher: JetBrains s.r.o.
Type: Application
From: C:\Users\BALU\Desktop\pycharm-community-2018.3....

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Now click on **NEXT**



PyCharm Community Edition Setup

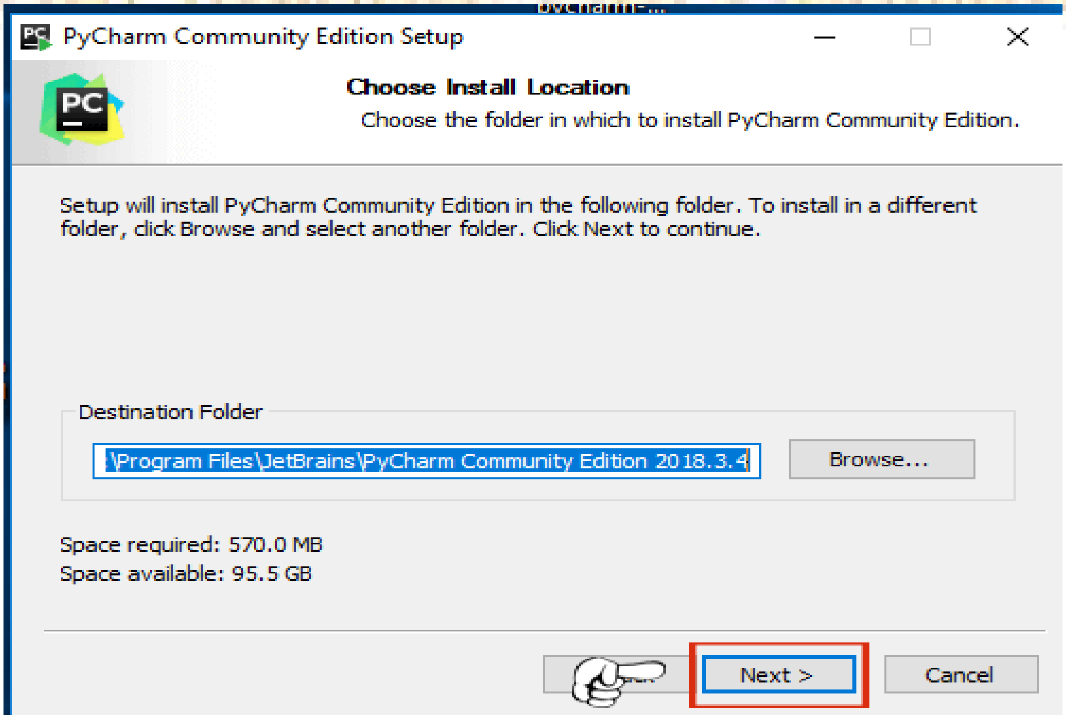
Welcome to PyCharm Community Edition Setup

Setup will guide you through the installation of PyCharm Community Edition.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Select the path for installation. (A default path would be provided and can be changed if required)



PyCharm Community Edition Setup

Choose Install Location

Choose the folder in which to install PyCharm Community Edition.

Setup will install PyCharm Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

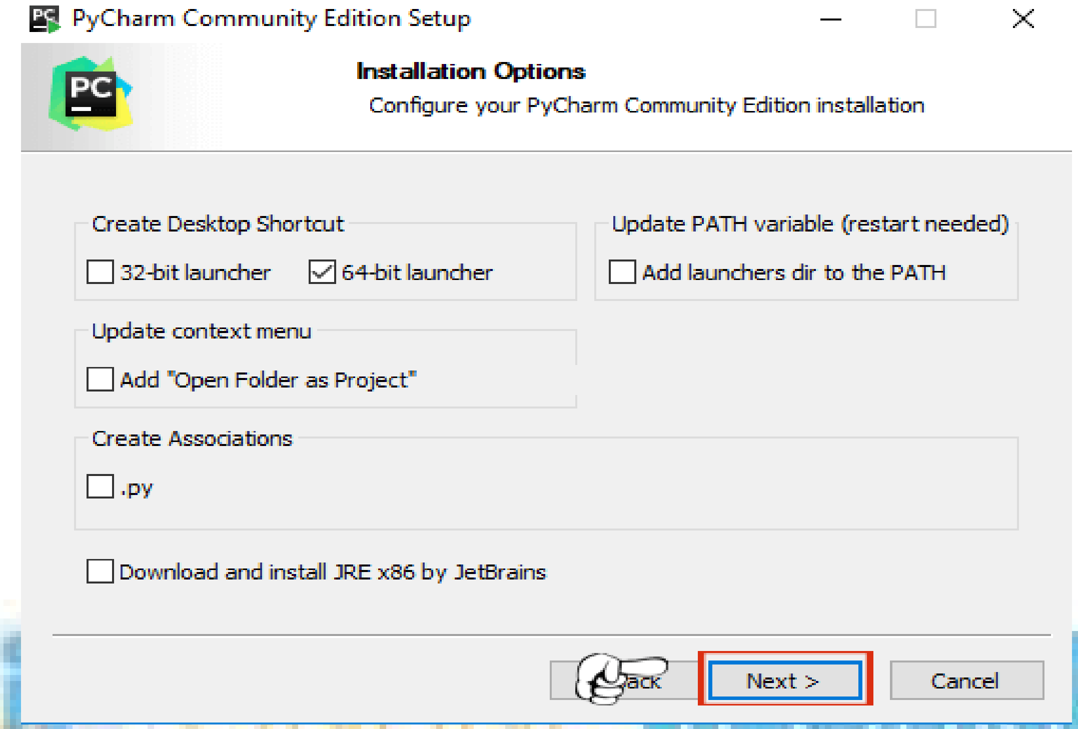
Destination Folder

\Program Files\JetBrains\PyCharm Community Edition 2018.3.4

Space required: 570.0 MB

Space available: 95.5 GB

Now select the operating system of the Local system.



PyCharm Community Edition Setup

Installation Options

Configure your PyCharm Community Edition installation

Create Desktop Shortcut

□ 32-bit launcher ☑ 64-bit launcher

Update PATH variable (restart needed)

□ Add launchers dir to the PATH

Update context menu

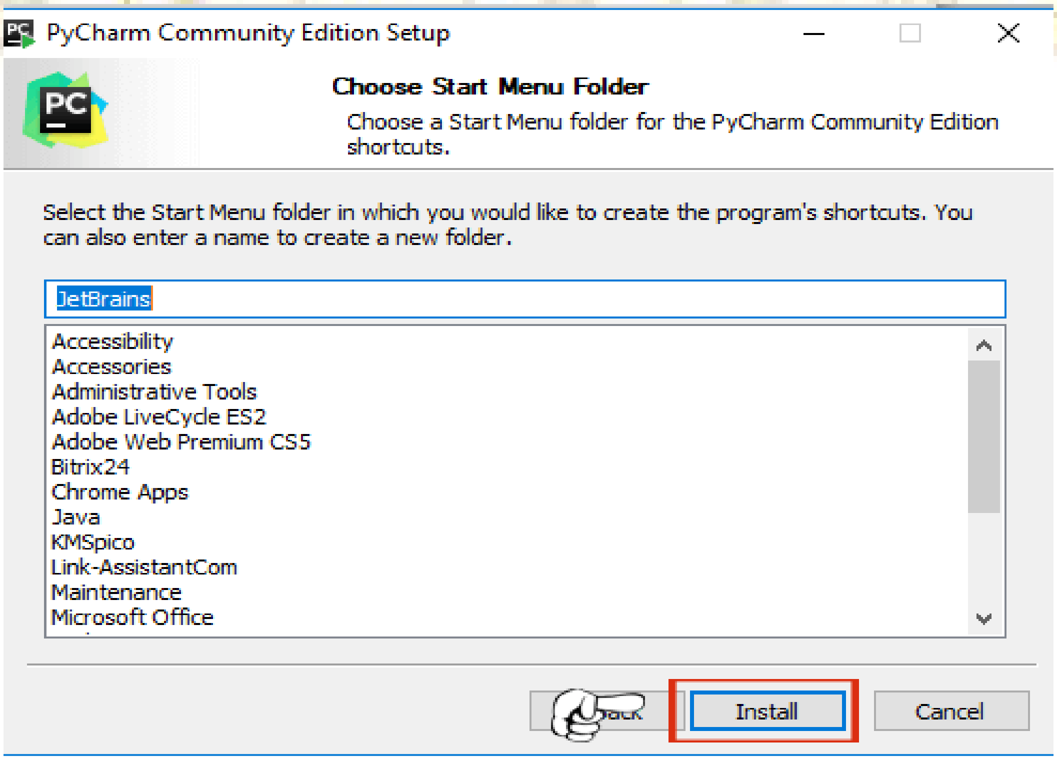
□ Add "Open Folder as Project"

Create Associations

□ .py

□ Download and install JRE x86 by JetBrains

Select the project location path to create the program shortcuts



PyCharm Community Edition Setup

Choose Start Menu Folder

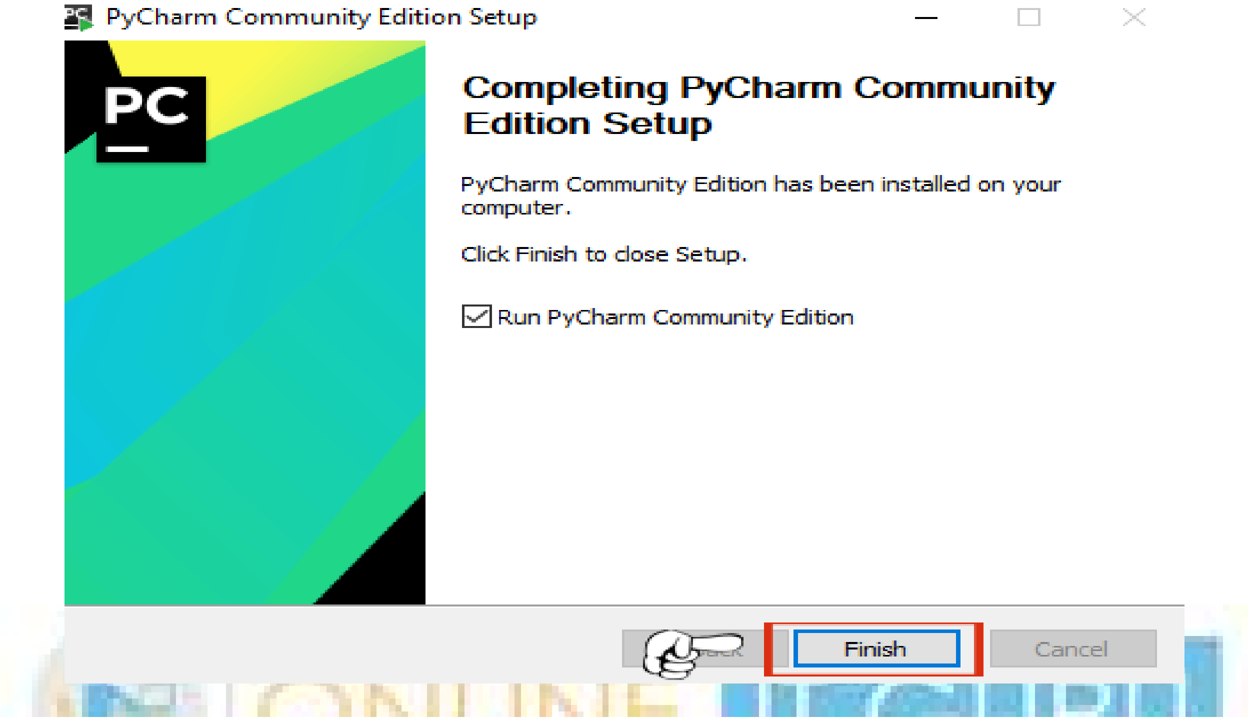
Choose a Start Menu folder for the PyCharm Community Edition shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

JetBrains

Accessibility
Accessories
Administrative Tools
Adobe LiveCycle ES2
Adobe Web Premium CS5
Bitrix24
Chrome Apps
Java
KMSpico
Link-AssistantCom
Maintenance
Microsoft Office

Now click on Install. And tick RUN Pycharm Community Edition


PyCharm Community Edition Setup

Completing PyCharm Community Edition Setup

PyCharm Community Edition has been installed on your computer.

Click Finish to close Setup.

☑ Run PyCharm Community Edition

Finally, tick Finish completing the installation.

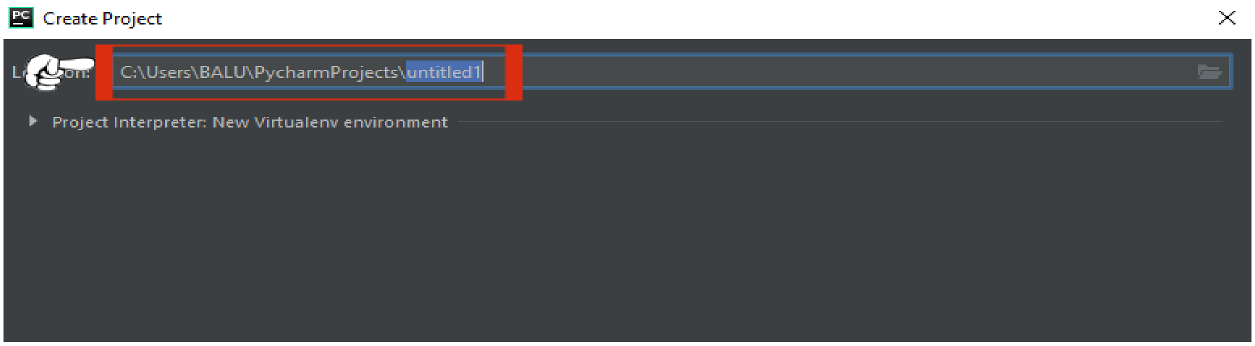
Visit **Essential Python IDE's to learn** to know all the IDE's available for Python

First program:

Hope you have completed the installation of python on your local computer. Now let us run the first program.

Initially open Pycharm shortcut that was installed in your computer.

Go to File menu and then click on the new project.


Create Project

Location: C:\Users\BALU\PycharmProjects\untitled1

Project Interpreter: New Virtualenv environment

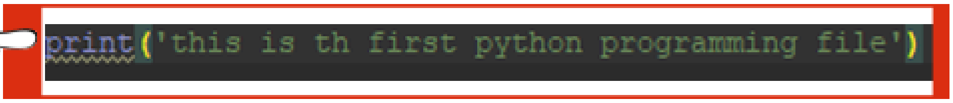
Then once again go to the file menu and then open the project. Now open the respective project and then create the new Python (.py) file.

Initially, I would like to print the user input statement.

```
this is th first python programming file
```

So after writing some code, it’s time to execute the piece of code. So to do this go to the terminal in the pycharm and the press **shift +F10.**

Finally, you will be getting the output as



```
print('this is th first python programming file')
```

So till now, you have installed python environment in your system. Now its time to discuss **features of python.**

Python features:

Python is said to be the best programming language because of the following features.

1) Simple and easy to learn

2) interpreted expressive and cross dependent language

3) Open source and free

4) Object-oriented language

5) Extensible and portable

6) GUI programming support

7) Standard library and integrated language.

Also check the Factors that will improve python growth**.**

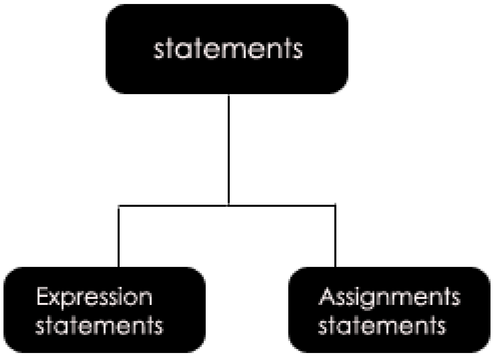
4. Python Fundamentals

So until now, we have seen the python installation in our systems. Now let us start coding with python fundamentals.

Statements:

Python statements are nothing, but the logical instructions that the interpreter execute. Hence the statements can be in single line (or) multi line statements.

Python statements were categorized into two statements:



Expression statement and Assignment statements.

Expression statements:

The expression statements include several operator expressions like Addition (+), Subtraction (-), multiplication (*), division (/) and modulo (%). Probably In other words, the expression statement returns a value.

An expression is something that returns a value.

Arithmetic expressions:

(10/2)*3

Functional expression:

Every Interpreter/compiler have some predefined function like POW (), sqrt () and so on. And Python interpreter supports this tool.

POW (2.3)

8

Sqrt (81)

3

Assignments statements:

These types of statements create a variable and assign a value to it. Hence the assignment statements logically operate and store the results in the variable (it operates on LHS and stores on RHS).

Examples:

Test1 ='I love coding'

Id (test1)

Output: 70809568

Test2='I love coding'

Id (test2)

Output: 70809568

If you seriously noticed the above two variables, namely test1 and test2, therefore the interpreter allocates same memory allocation for both the variables.

Want to become a master in Python visit python online training

Note:
Almost Python allocates same memory location in two cases

If the string is less than 20 characters (white spaces were included)

Integers ranging between -5 to + 255.

RHS operations:
In this, we have some operations on RHS. Hence, Let us discuss with some examples.

```
Test =7*2
```

```
Type (test)
```

Output: input

```
Test = 7*2/10
```

```
Type (test)
```

Output: float

Comments:

So, these are nothing but the non – executable line of code. While, we have two types of comments.

Single line comments:
The single line comments start with #

```
# I don't execute
```

Multiple line comments:
Finally, the multiple line comments start and end with "".

Would you like to know **is python compiled or interpreted?**

5. Python variables

Python Variables are nothing but the reserved memory locations to store the values. In other words, it is the name of the memory location where the data is stored. And if once the variable is stored a separate space is allocated in the memory. Moreover, we can define the variable name using the combination of letters, number, and underscore.

Click the here to know the **programming structure of python**

Assigning values to the variables:

In fact, there is no explicit declaration to store the memory. And we can easily define a variable using the '='operator.

Example:

```
a = 20

b = 30

Print a
```

Output:

```
20
```

Besides python **online training** suggest another type of declaring the variables.

Example:

```
A, b,c = 2,10,python
```

Note:

Python interpreter automatically identifies the types of variables

Can we re-declare a variable?

Today many starters have this question? Hope you got the answer after reading this. And the answer to the question is yes, we can re declare a variable. But the Variable stores the most recent value.

Ex:

```
x = ‘10’

Print (‘x’)

X=’abc’

Print (‘x’)
```

Output:

```
10

ABC
```

Constants:

These are fixed values that cannot be altered.

Ex: pi= 22.7

Gravity =9.8

Place these values separate files like constant.py

Now we can execute this by the following command

```
Import constant

Print (constant.pi)

Print (constant. Gravity)
```

So after running this command, we would get the following output

Output:

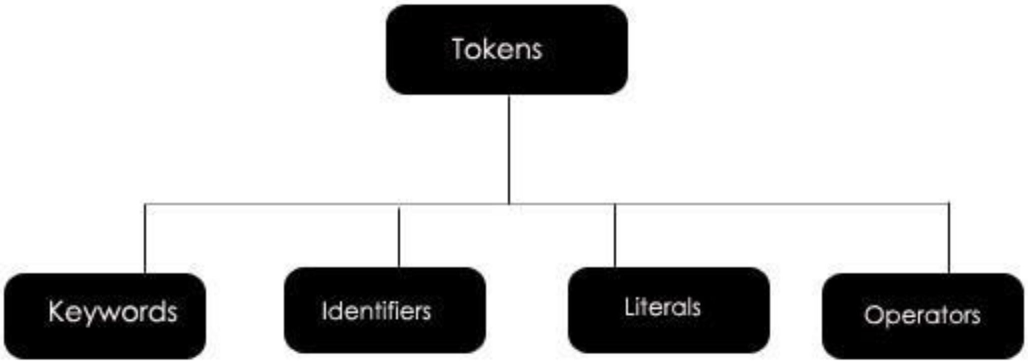
22.7

9.8

So till we seen the variable types, declarations. Now, it's time to discuss tokens

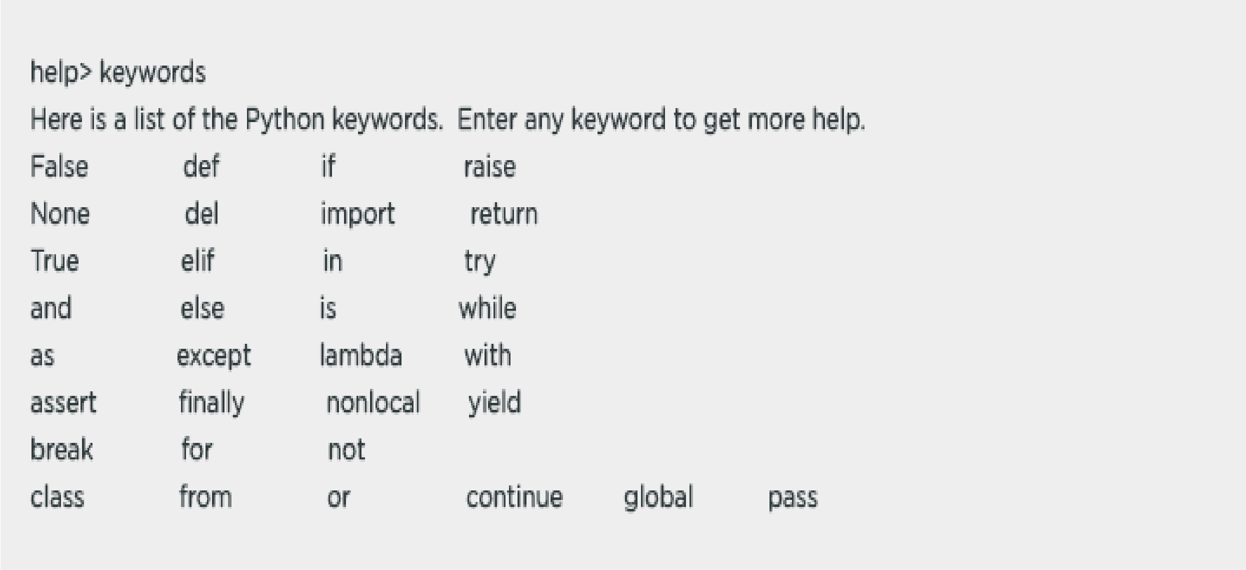
6. Python tokens

Basically, python tokens are the small units of the programming language. Python supports 4 types of Tokens



Keywords:

These are the dedicated words that have special meaning and functions. Moreover, compiler defines these words. Moreover, it does not allow users to use these words. Python compiler has the following words



```
help> keywords
Here is a list of the Python keywords.  Enter any keyword to get more help.
False        def         if          raise
None         del         import      return
True         elif        in          try
and          else        is          while
as           except      lambda      with
assert       finally     nonlocal    yield
break        for         not
class        from        or          continue     global      pass
```

Identifiers:

Identifiers represent the programmable entities. The programmable entities include user-defined names, variables, modules and other objects. Moreover, python defines some rules in defining the identifiers. Now let us discuss some of them.

An identifier can be a sequence of lower case (or) upper case (or) integers (or) a combination of any.

The identifier name should start with the lower case (or) upper case (It must not start with digits)

The identifier name should not be a reserved word.

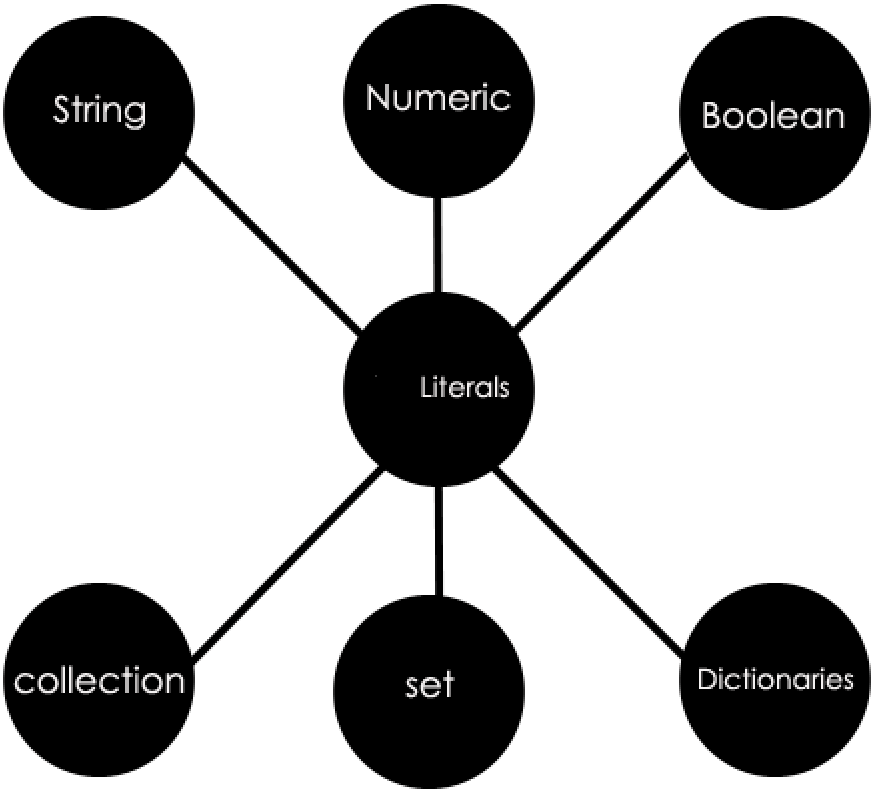
Only Underscore (_) is allowed to use as a special character in identifier names.

The length of the identifier name should not be more than 79 characters

Would you like to know how **Machine learning using python**

Literals:

Literals are used to define the data as a variable (or) constants. Python has 6 literals tokens.



String:

The string is a sequence of characters defined between quotes. (Both single and double quotes are applicable to define the string literals.). And these strings perform several operations let us discuss some of them.

Syntax	Operation
Len(String_name)	String length
String_name.index(char)	Locate the character in the string
String_name.count(char)	
String_name[::-1]	Reverse the string
String_name.upper()	Converts the strings to upper case
String-name. Lower	Coverts the string to lower case

Numeric:

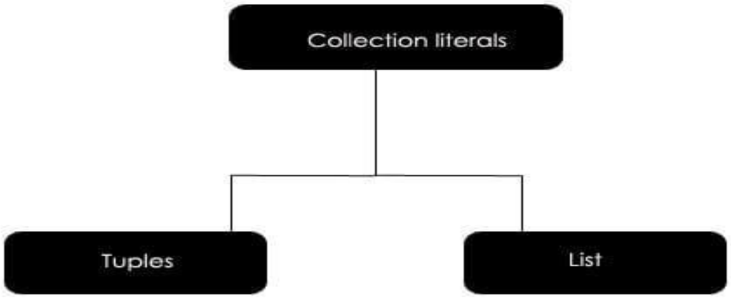
These are immutable (unchangeable) literals. We basically have 3 different numerical types namely integer, float, and complex

Boolean:

This has only two values. i.e true / false.

Collection literals:

A collection literal is a syntactic expression form that evaluates to an aggregate type such as array list (or) Map. Python supports 2 types of collection literal tokens



List Literals:

You can consider the python lists as arrays in C. But the difference between the Arrays and lists is that arrays hold homogeneous data type and lists holds the heterogeneous data types. Basically, this list is the most versatile data type in python. Python literals are separated by comma in []

Note:

If comma is not provided between the values, the output does not contain spaces

Example:

List = [‘a’,’b’,’c’]

Print (list)

Output:
[‘a’,’b’,’c’]

The other possible outputs shown below

Code	Output	Explanation
List	[‘a’,’b’,’c’]	This would print all the input values
List[0]	A	Like arrays, the index of the elements starts with 0
List[2]=’d’	[‘a’,’b’,’d’]	It will update the list at index[2] by d with c
Del list[1]	[‘a’, ’d’]	It would delete the value at index [1]
Len(list)	2	It returns the length of the tuple

List*2	[‘a’, ’d’] [‘a’, ’d’]	It prints the output as the number of times the input was given
List[::-1]	[‘d’,’a’]	It would print the result in the reverse order.

Tuples:

Tuples were similar to list. But like list tuples cannot changed the values. Beside, tuples are enclosed in parenthesis. Whereas lists are enclosed in square brackets.

And as said earlier, these tuples performs all the operations like lists. So I would like to leave the operation for you as a practice. And if you struck up anywhere clarify at python training**.**

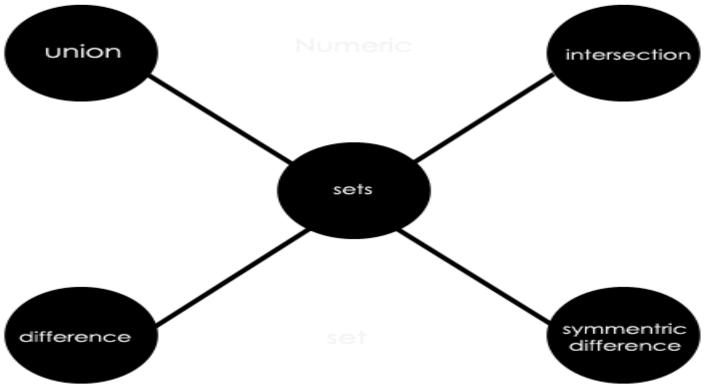
Set:

A set is a well-defined collection of elements. And the elements in the set are placed in a curly braces separated by comma. In the set every element is unique

Set 1 = {1, 2, 3}

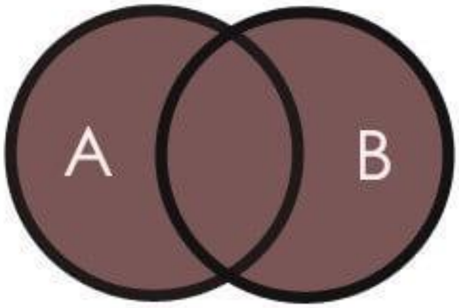
Set 2 = {1, 2, 2, 3}

In the above example the element 2 is taken twice. Now let us discuss the various set operation



Union:

It combines all the elements in the string. And the union operation performed using the pipe (|) operator tokens.



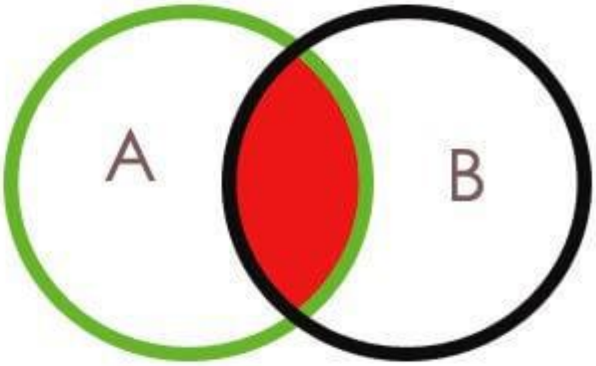
Ex: A = {1, 2, 3, 4, 5, 6}

B= {3, 4, 5, 6, 7, 8}

A|B = {1, 2, 3, 4, 5, 6, 7, 8}

Intersections:

Intersection of A and B returns the common elements in the sets. And the operation is performed using the & operator tokens.



Ex: A = {1,2,3,4,5,6}

B= {3,4,5,6 ,7,8}

A &B = {3,4,5,6}

Difference:

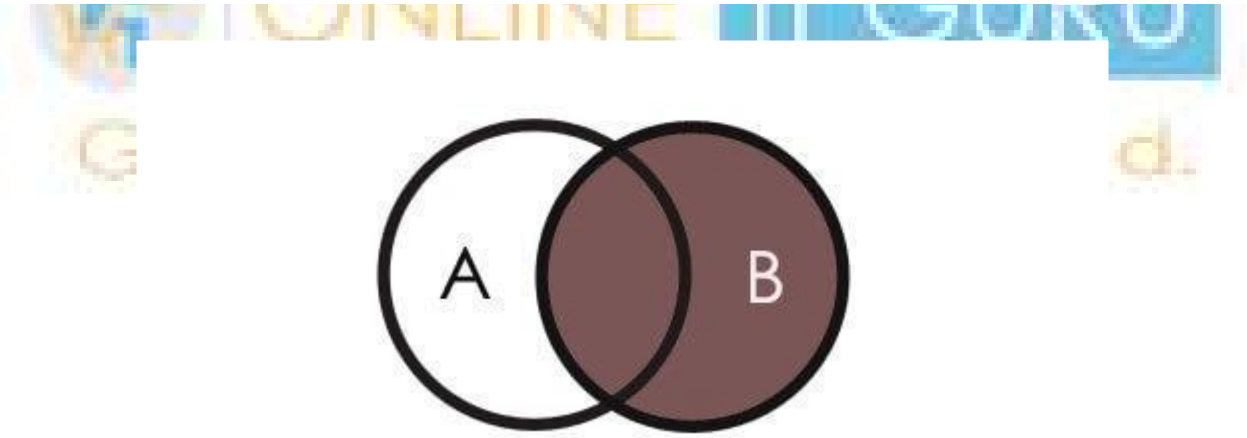
Difference of (A-B) returns the elements that are only in A but not in B. Similarly B-A returns only the elements that are only in B but not in A tokens.

Ex: A = {1, 2, 3, 4, 5, 6}

B= {3, 4, 5, 6 ,7,8}

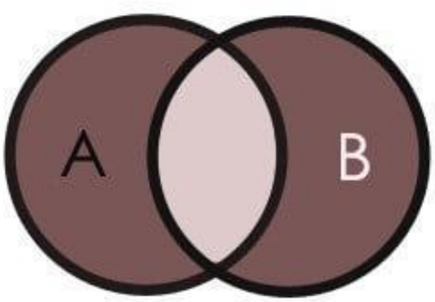
A –B = {1,2}

B-A = {7,8}



Symmetric difference:

It returns the set of elements that are both in A and B except the common elements tokens.



Ex: A = {1, 2, 3, 4, 5, 6}

B={3,4,5,6 ,7,8}

A^B ={1,2,7,8}

Dictionaries:

Python dictionaries are the key value pairs that are enclosed in curly braces. Dictionaries are separated by the ":"

Dict = {'name': 'Onlineitguru', age: 20}

And these elements accessed as Dict ['name']

Output: Onlineitguru

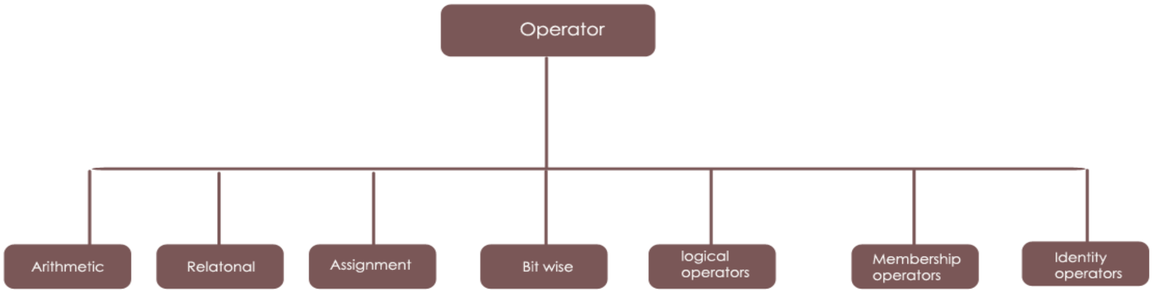
Appending the elements in Dictionaries:
Dict ['address'] =Ameer pet

Output:

'name'='onlineitguru','age'=20, address'=ameerpet.

7. Python operators

Operators are functions that perform some logical calculations. In other words, these are the constructs that manipulate the value of the operands. **Python** supports several kinds of operators. Let us discuss one –by –one in detailed.



Arithmetic operators:

Arithmetic operators are used for Addition, subtraction, multiplication and division and so on. Let us discuss them with an example.

Let us consider the operator with A =100, B = 200

Operator	Description	Example
Addition(+)	Add values on either side of the operator	A+B =300
Subtraction(-)	Subtracts A from B	B-A =100

Multiplication(*)	Multiplies A with B	A*B = 2000
Division(/)	Divides the denominator with numerator	B/A =2
Modulo (%)	Performs the division operator and returns the remainder	B%A=0
Exponent(**)	Performs exponential calculation on operators	A**B= 100 power 200

Relational / comparison operator:

These operator compares the operands and displays the result.

Let us assume a = 3, b =5

Operator	**Description**	**Example**
==	This operation returns true if two values are equal	A ==B returns **False**
!=	If the two operations are not equal this returns true	A!=B returns **true**
	If the value of the left operand is greater than right this returns true	A>B returns **false**
<	This operator true if the left operand is less than the	A< B returns **true**

	right operand	
>=	This operator returns true if the left operand is greater than or equal to the right operand	A>=B returns **false**
<=	This operator returns true if the left operand is less than or equal to the right operand	A<=B returns **True**

Visit **online courses** for more live examples

Assignment operator:

An assignment operator is used to assign a new value to the variable.

Let us assume a =5, B = 10

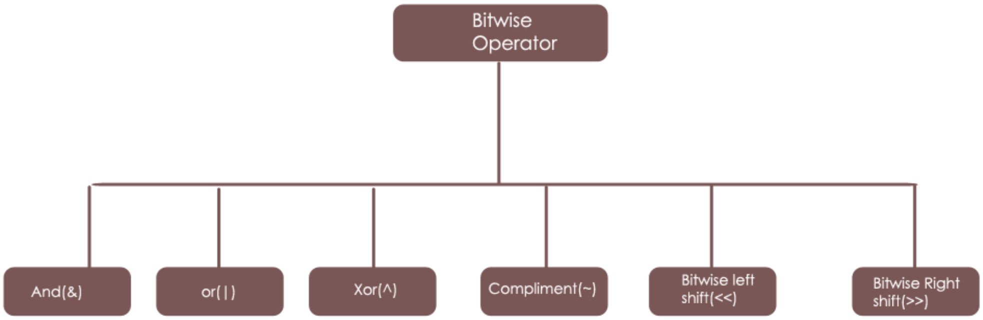
Operator	Description	Example
=	Assigns a value from right operand to left operand	C= a+b assigns the value of the sum of A and B to c
+= ADD and	It adds right operand to the left operand and assigns the result to the left operand	**C+=a** is equivalent to **c= c+a**
-=Sub and	It subtracts right operand from the left	**C-=a** is equivalent

	operand and assigns the results to the left operand	to **c = c –a**
*=Multiply and	It multiplies right operand with left operand and assigns the result to the left operand	**C*=a** is equivalent to **c = c*a**
/= division and	It divides the left operand with the right operand and assigns the result to the left operand	**C/=a** is equivalent to **c = c/a**
%= Modulo and	It performs modulus of the two operands and assigns the result to left operand	**%=a** is equivalent to **c= c%a**
**=Exponent And	It performs an exponential calculation on two operands and assigns the value to the left operand	**c**=a** is equivalent to **c = c**a**

Visit **Python Training** for more code examples

Bit-wise operators:

Every operation that needs to perform must divide into bits. And performs the operation and displays the result in the decimal format.



Let us discuss with an example A =30, B= 23

These two numbers written in binary format as

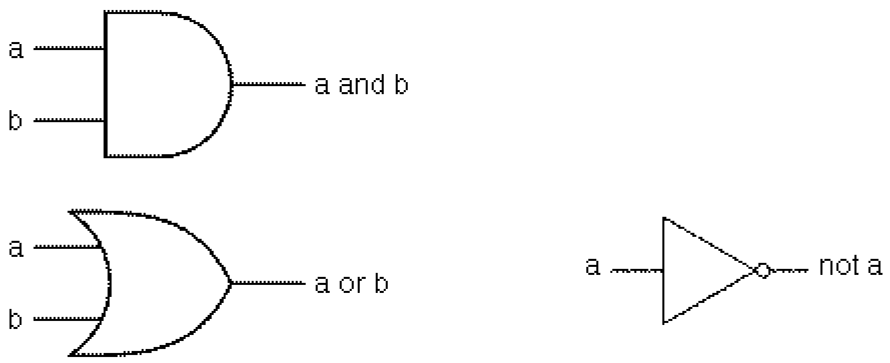
A = 30 = 0001 1110, B = 23 = 0001 0111.

Operator	Description	Example
And (&)	If both the inputs are 1 's the result is 1 else the result is zero	A&B =22 (0001 0110)
Or(|)	If both the inputs are 0's the result is zero else the result is 1	A|B= 31(0001 1111)
Xor(^)	If both the bits are same the result is 0 else the	A^B =9(0000

	result is 1	1001)
Compliment(~)	If the input is one the output is zero and vice-versa	A~B =-31(1000 1111)
Binary left shift (<<)	The left operands are move left by the number of bits specified by the right operand	A<<2 = 120(0111 1000)
Binary right shift(>>)	The right operands are moved right but the number of bits specified by the right operand	A>>2 =(0000 0111)

Logical operators:

Logical operators are used the logic condition (true (or) false). We have three types of logical operators.



Let a = 5 , b=10, c =20

Operator	**Description**	**Example**
And	True only if both the inputs are one	a<b and a< c Returns **True** a<b and a >c returns **false**
Or	True if any one of the input is one	a<b and a>c returns **true** a>b and a>c returns **false**
Not	It negates the input	a<b **false**

Will Python replace java or not let's check here.

Membership operator:

The membership operators are classified into two types.

A={1 ,2,3,4,5}

Operator	Description	Example
Is	It returns true if the value is found in the sequence	5 **in** A returns **True** 10 **in** A returns **False**

Not in	True if the variable is not found in the sequence	5 **not in** A returns **False** 10 **not in** A returns **True**

Identity operators:

It compares the objects memory address

A =100, B =100

Operator	**Description**	**Example**
Is	Returns true if the operands identity is same	**A is B** returns **True**
Is not	Returns true if the operand identity is not the same	A **is not** B returns **false**

Operator precedence:

When there is several operations to be performed, operator precedence determines which operator would be given importance over others.

**

~+/~-

*/%//

+-

>><<

&

^|

<=< >>=

<>==!=

= -= +=//=%=/=*=*==

Is isnot

In innot

Not or and

Note: we can expect a question in Operator precedence in **online certificate courses**

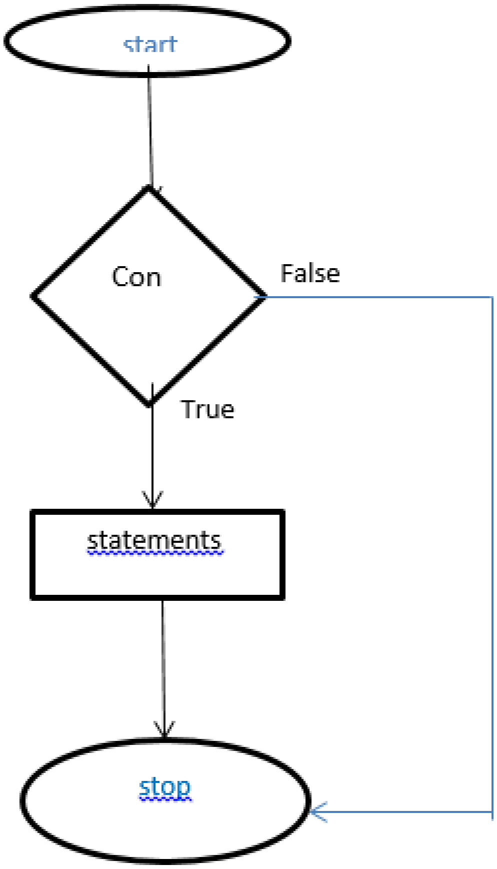
8. Conditional statements in Python

Conditional statements in python performs actions (or) computations that depends on a certain condition (true (or) false).

Python supports different types of control statements. So, let us discuss one by one in detailed.

If statement:

Python If is a statement (or) a group of statements executes when a particular condition satisfies.



Syntax:
If (condition = true)

Statement

Ex: i=1

If (i==1)

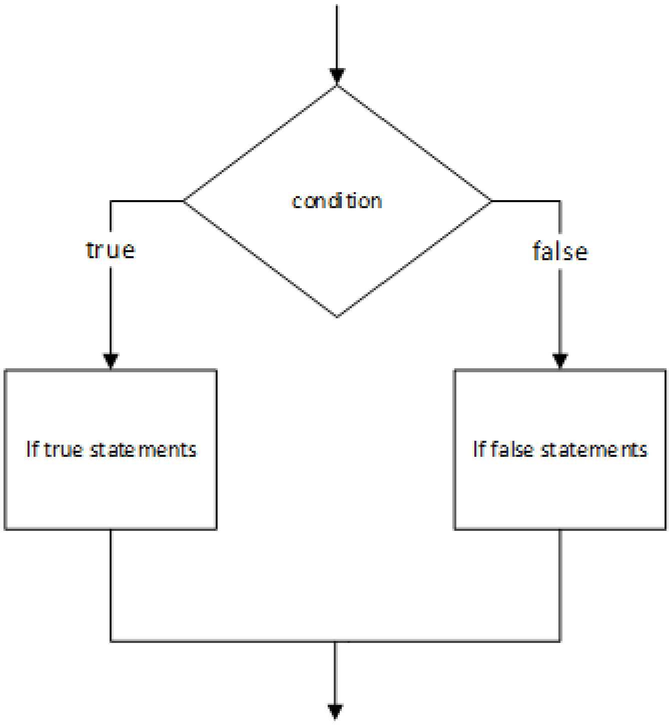
Print (“condition satisfies (true)”)

Would you like to know how people use python for Data Science?

If else statement:

In the above statement, there is no logic to execute if the **IF** condition fails. So this **python if –else** statement satisfies the condition.

As said above **IF** block executes if the condition satisfies. Otherwise **Else** block executes.



Syntax:

If (Condition =true)

Statement 1

Else

Statement 2

Ex: i=1

If (i==1)

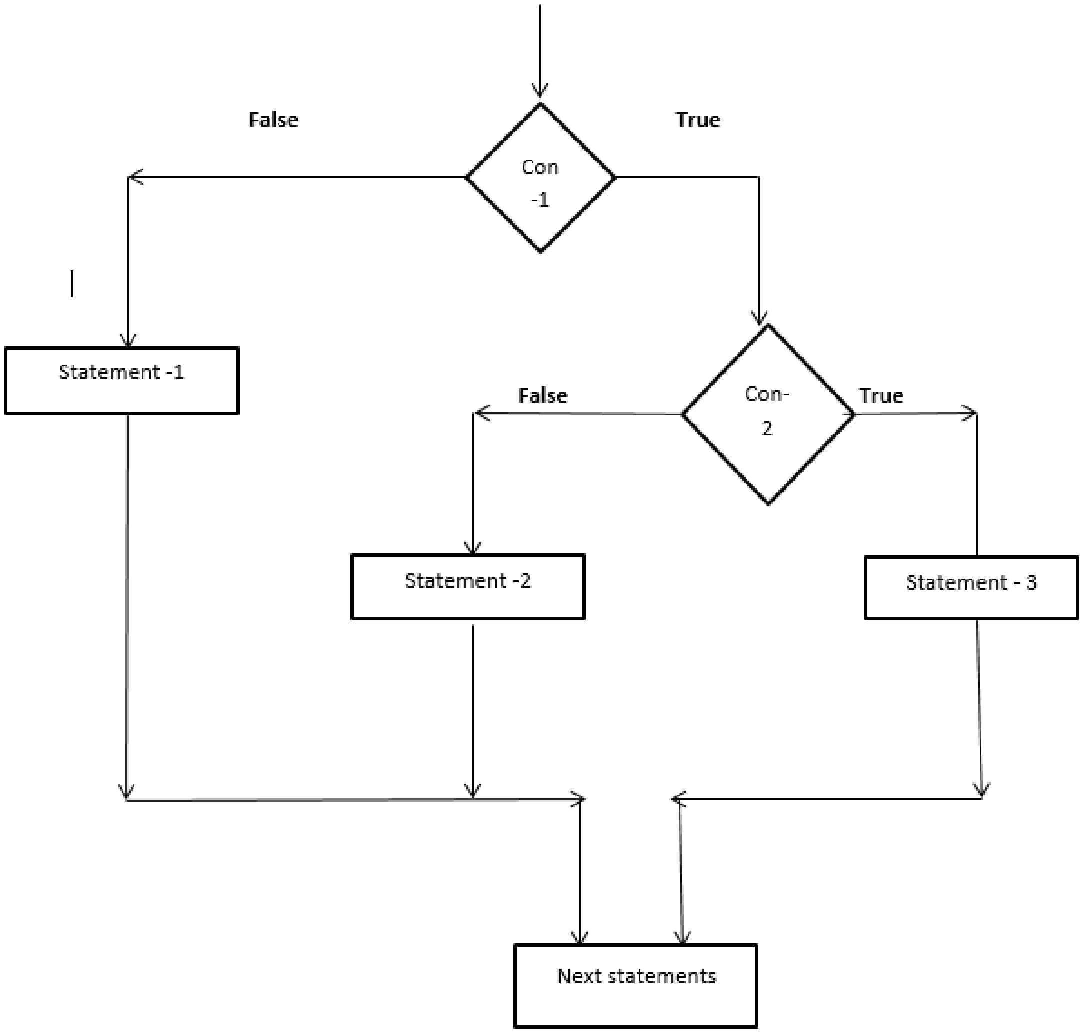
Print (Since i=1 true condition executes)

Else

Print (Since I not equal to 1 false condition executes)

Nested if:

There are some situations, where an operation needs to satisfy several conditions. Then we use Nested if condition. This is also called the **Multiple If statements.**



Syntax:

If (condition = true)

Statement 1

If (condition=true)

Statement 2

elif (condition =true)

Statement 3

Else

Statement 4

Ex:

I=10

If (i<100)

Print(’10 is less than 100’)

If (i==20)

Print (’10 equals to 20’)

Elif

Print (‘we cannot equate 10 with 20’)

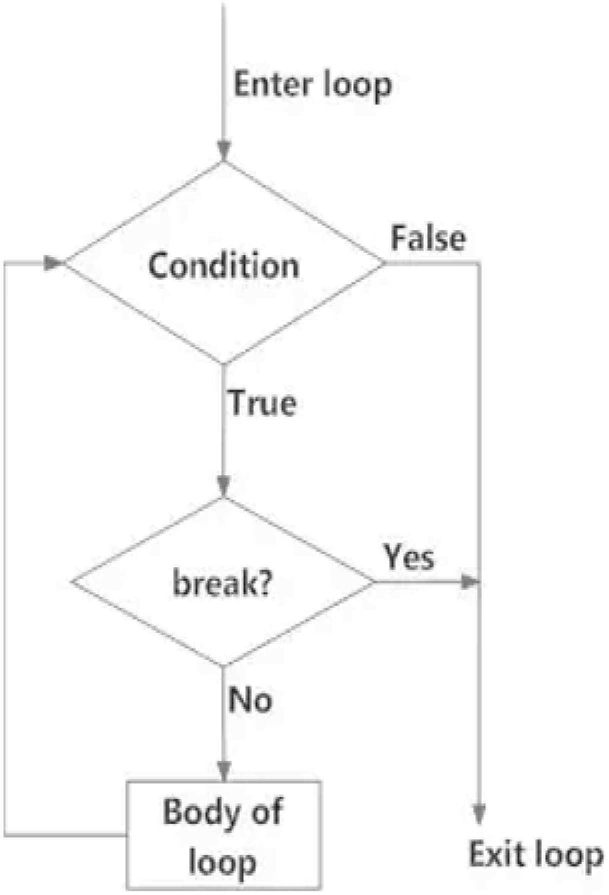
Else

Print (‘there is no true condition in the statements. So **else** block is executed’).

Since these concepts were common in many programming languages like C, JAVA .So I advise you to practice more code on this loop. Feel free to clarify the doubts at **Software training institutes**

Break statement

As like above, we cannot expect only one / two conditions. In some, there may be a 100 (or) even maybe a 1000 conditions. Moreover, there is only a need for one block of statements to execute if the condition satisfies. So in those cases, we uses **break** statement.



Syntax:

```
If (condition 1)

Statement 1

Break

If (condition 1)

Statement 2

Break

Else

Statement 3
```

Ex:

```
I = 20

If (1==10)

Print (‘I not equal to 20’)

Break

If (i==20)

Print (‘Condition satisfies’)

Break

Else

Print (‘No condition satisfies’)
```

Continue:

This is somewhat similar to the Break statement. Hence the condition satisfies it skip the current statement and forces to process the next iteration.

Since these were common in every programming languages like C, JAVA, So, I would like to leave the code for you as an exercise. And if you were struck up anywhere python online training will guide you

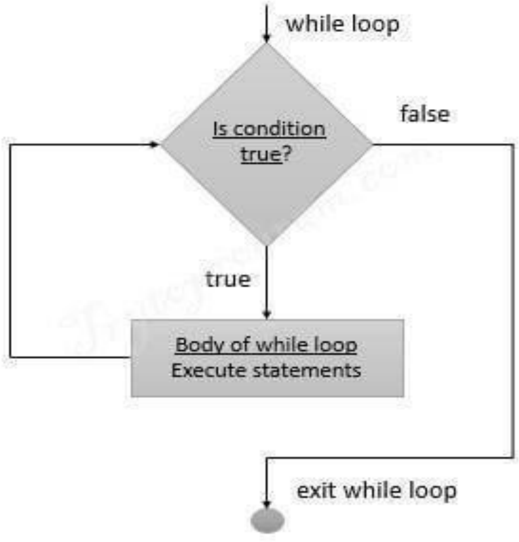
9. Python for Loop

There are some conditions where we need to repeat the logic for some times (finite (or) infinite) then in such cases, we need to use loops. We continues the execution of loops until the condition becomes false. Python supports three types of python for loop.

Would you like to know why python is important for Android apps, Read More?

While loop:

It repeats the statement (or) group of statements until the **python while** condition is true. It initially checks the conditions before the execution of the loop. If the condition satisfies then it moves into the body of the loop. We use this loop when we are not sure how many types the loop must repeat.



Syntax:

```
Initialization
While (condition)
{
Statements
}
```

Ex:

```
Count = 1
While (count < 5 ):
   Print (count)
   Count = count * 2
Print (“Good bye!”)
```

Output:

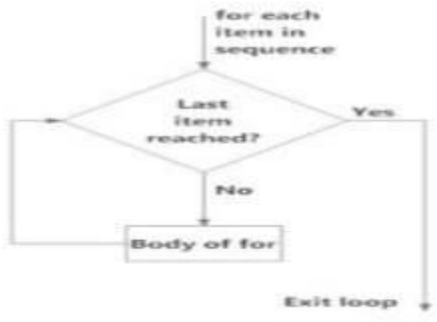
```
1
2
4
Good bye!
```

Visit **Python online training** for more coding examples on While loop

For loop:

Like while loop for loop allows the block of code to be repeated certain number of time. The major difference between **while** and **for** loop is that, in **python for loop** we know the number of iteration required to break the loop.

Flowchart of for Loop



Syntax:

```
For variable in sequence:

Statements
```

Ex:

```
Flowers = ['jasmine', 'Rose', 'Lilly']

For index in range(len(flowers)):

  Print (flowers[index])
```

Output:

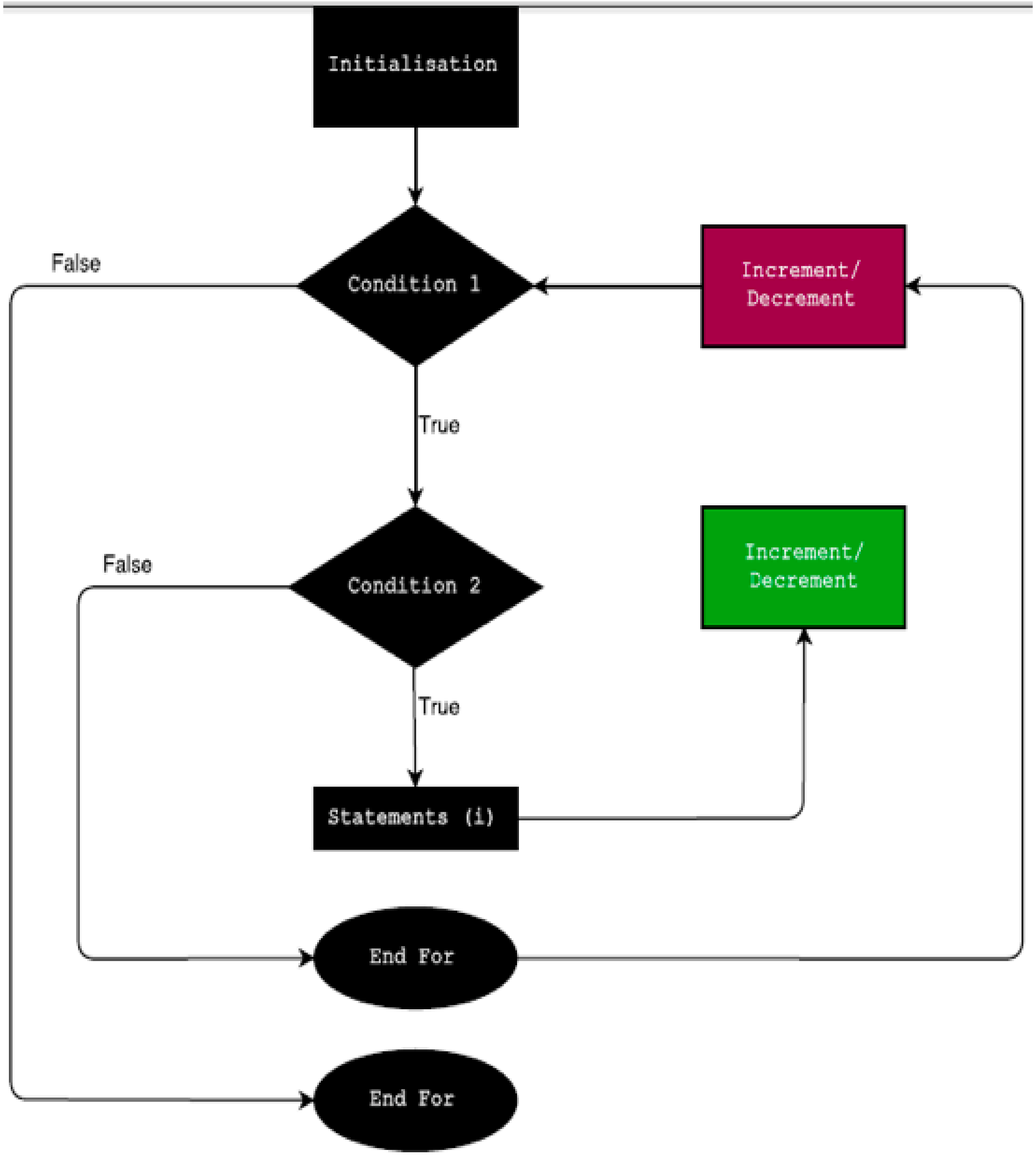
```
Jasmine

Rose

Lilly
```

Nested loop:

If there is loop with in a loop then we call it as nested loops. It can be a while loop with in a for loop and vice versa.



EX:

```
Count = 1

For i in range (5):

  Print (str(i) * i)

  For j in range (0, i):

     Count = count + 1
```

Visit **best** python course for more coding examples

10. Python Functions

There are some cases where a certain piece of code needs to perform an action several times. But writing the code at all times where the action is necessary increased the code complexity. So we need an alternative to these problems. **Python** function provides good solutions to this. So let first have a look over

What is a python function?

The function is a block of code which provides the re-usability. These functions allow the proper modularity for the application in a single action. Python offers built-in functions like Print (). Besides, it allows the users to create the function of our own. These functions are knows as the User**-defined functions.**

So my readers till now we have what is a function in **learn to code python,** So now let us have a look over how to define a function.

Rules for defining a function:

- The function name should start with a def keyword followed by a function name and parenthesis.
- All the arguments and input parameters should be placed within the parenthesis
- The functional code within the function should start with the colon :

- To exit the function, we use the **return** (this is also used to send object return to the caller)

Syntax:

```
def. func_name():

Statements ()

Return
```

Ex:

I would like to explain this with an addition example

```
def sum(x,y):

z=x+y

Return z

Sum (20, 30)
```

Output:

```
50
```

So in **learn python for beginners,** we will discuss how to call (or) how to invoke a function.

Function calling (or) function invoking:

The definition of the function would offer the name and specify the values/ parameter that should be included in a function. Once the function has given the basic structures, it can be executed by calling from the other functions. Besides this can also be called from the python prompt. As discussed above in Function calling, we can pass value by reference (or) pass by value.

Syn:

```
Def function_name ():

Print ()

Function_name ()
```

Example:

```
Def first ():

Print (‘this is the first function’)

First ()
```

Arguments:

These are the values passed in a function. The arguments are of 5 types.

Default arguments:

These arguments provide a default value if nothing is provided in the function call. These have to be defined in the function definition.

Keyword arguments:

These arguments are related to the calls. With the help of parameter name caller identify the arguments

This allows skipping arguments.

Required arguments:

These are those arguments that are passed to the function in the correct order according to their positions.

Variable- length arguments:

In some cases, we need to write functions that accept more parameter than they defined. These arguments are otherwise called as these arguments as a variable – length arguments. Also check how can you check the quality of the python code

Scope of the variable:

The variables declare this have different scopes. Let us discuss the usage in detailed.

Local variable:

The variables that can be accessed in a function, where it is declared. It cannot be accessed outside the body. Let us discuss the code with an example.

Ex:

```
def first():

a = ‘im the local variable’

Print (a)

First ()
```

Output:

```
Im the local variable
```

EX: 2:

```
def first():

a  = ‘im the lcoal variable’

Print (a)

First ()

Print (a)
```

Output:

```
I’m the local variable

Name ‘a’ is not defined
```

Reason:

Since variable 'a' is defined in a function, it cannot accessed outside the function. So we have experienced an error.

So to overcome this feature, we use Global variables

Global Variable:

This is unlike the local variable. This can be accessed globally. The variables that are declared ones can be accessed many functions.

Ex:

```
Def. f():

s= 'I'm the local function. I override the global function'

print (s)

a= 'im the local function, i cannot  access the outside the function f'

print (a)

# Global scope

b= "Since, I am the global function, print S is called before the function, and first exected"

Print

f ()

a= 'i love python'

Print (a)
```

Output:

```
Since, I am the global function, print S is called before the function, and first exected

Im the local function. I overrided the global function

Im the local function, I cannot access the outside the function f

i love python
```

Ex: 2

```
def f():

s= 'im the local function. I overrided the global function'

Print (s)

# Global scope

b = "Since, I am the global function, print S is called before the function, and first exected"

print(s)

f ()

print (b)
```

Output:

```
Since, am the global function, print S is called before the function, and first exected?

Im the local function. I overrided the global function

Am the global function, print S is called before the function, and first exected?
```

Note:

If you have observed the code, when the control comes outside the function, the value of the global variable will be stored.

***Q:** How do we access the local variable value outside the function?*

So far, we have discussed the local variables as well as the global variables. But as shown above, if the control is a function outside, the memory/cache stores only the global variable

So to store the local variable value outside the memory we need to make use of Global function. Let us consider the working of the global function with an example.

Ex:

```
def f():

Global b

s= ‘im the local function. I overrided the global function’

print (b)

# Global scope

b = “Since, I am the global function, print S is called before the function, and first exected”

print(s)

f()

print (b)
```

Output:

```
Iam the global function, since print S is called before the function, I was exected

Im the local function. I override the global function

Im the local function. I overrided the global function
```

So now let’s move to the next topic recursion.

Recursion:

So till now, we have seen how to call the function and also how to pass the variables to it. But we cannot say exactly the function may call the other functions. In some cases, function calls the same functions too. This phenomenon is nothing but recursion.

In simple words factorial is a good example of recursion.

Ex:

```
def calc_factorial(x):

“””This is a recursive function

to find the factorial of an integer”””

if x == 1:

return 1

else:

return (x * calc_factorial(x-1))

num = 10

print(“The factorial of”, num, “is”, calc_factorial(num))
```

So far, we have discussed what is a function, how to define the function and the types of functions? Now it’s time to discuss some functions in python.

Init:

After the class creation __init__ called. It also known as constructor

Self:

Self represents the instance of the class. In python, using the **self**-keyword, we can access the python class attributes and methods.

In the **best way to learn to** learn python**,** these keywords play a prominent role. So now let us have a look over the syntax and example

Syntax:

```
Class Some Class:

variable_1 = “This is a class variable”

variable_2 = 50    #this is some other variable

def __init__(self, param1, param2):

self.instance_var1 = param1

#instance_var1 is a instance variable

self.instance_var2 = param2

#instance_var2 is a instance variable
```

Ex:

```
class Rectangle:

def __init__(self, length, breadth, unit_cost=0):
```

```
self.length = length

self.breadth = breadth

self.unit_cost = unit_cost

def get perimeter(self):

return 2 * (self.length + self.breadth)

def get_area(self):

Return self.length * self.breadth

def calculate_cost(self):

Area = self.get_area()

Return area * self.unit_cost

# breadth = 50 cm, length = 70 cm, 1 cm^2 = Rs 1500

r = Rectangle(50, 70, 1500)

print(“Area of Rectangle: %s cm^2” % (r.get_area()))

print(“Cost of rectangular field: Rs. %s ” % (r.calculate_cost()))
```

Here in the above example, all the values that were the length, breadth, and unit_cost were assigned to self.length, self.breadth, and self.unit_cost respectively. And all the operations were done using the self.variable_name (variable _name may be length, breadth and unit_cost).

Hope you got a better idea regarding **_init_** and **self.**

Doc strings:

It provides a convenient way of associating documentation, with python modules, classes, functions, and methods. doc_ strings can be accessed by __doc__ attribute.

How to define a doc string?

Usually, the doc string line should start with a capital letter.

The first line should be a short description

And we should not write the object name.

Ex:

```
def my_function ():

“””Demonstrate docstrings and does nothing really.”””

return None

print (“Using __doc__:”)

print (my_function.__doc__)
```

Out Put:

```
Using __doc__:

Demonstrate docstrings and does nothing really.
```

I suggest you to practice the one line Doc string, multi-line doc strings and Doc string in classes. If you fin difficulty clarify your doubts at **best online** python course**.**

Date and Time:

The other function that we need to discuss is Date and time. Let us discuss briefly it.

A date in python is a data type of its own. But we can import a module, named **date time** to work with dates as well as date objects.

Ex:

```
Import date time

x = datetime.datetime.now ()

print(x)
```

Output:

```
2019-02-13 13:12:25.389359
```

It means it displays the result in a year, month, day, hour, minute, second and microsecond.

How to get the calendar?

Python allows you to print the calendar of the month. The following code allows you to print the calendar of the month.

Ex :

```
From datetime import datetime

import calendar

cal = calendar.month(2019, 2)

print (‘Here is the calendar:’)

print (cal)
```

Output :

How to display the year and weekday of the week

```
import datetime

x = datetime.datetime.now()

print(x.year)

print(x.strftime("%A"))
```

Output:

```
2019

Wednesday
```

How to display the name of the month?

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

Output:

```
June
```

Now let us have a look over some **Date and time** legal formats.

Directive	**Description**	**Example**
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday

%w	Weekday as a number(0-6). Weekday starts from Sunday	3
%d	Day of the month (1-31)	13
%b	Month name(short version)	Feb
%B	Month name(full version)	February
%m	Month as a number(1-12)	2
%y	Year, short version	19
%Y	Year, Full version	2019
%H	Hour (00-23)	15
%I	Hour(00-12)	3
%p	AM/PM	PM
%M	Minute(00-59)	27
%S	Second(0-59)	27
%f	Microsecond(000000-999999)	270893

%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of the year	44
%U	Week number of the year. Sunday as the first day of the week	22
%W	Week number of the year. Monday as the first day of the year	52
%c	The local version of date and time	Wed Feb 13 15:10:54 2019
%x	The local version of date	02/03/2019
%X	The Local version of time	15:14:17

In the previous topics, we have discussed the python list, python tuples, and python dictionaries**.** And these perform the Slicing operation. Basically, these tuples/dictionaries contain a group of elements. But in all the cases, we do not require to display the complete list of elements. (It is enough to display only some elements). In those cases, we use a concept of slicing.

Slicing:

A slice object specifies how to slice the sequence. Here, you can specify where to start and where to end the sequence. The slice function returns a slice object.

Syntax:

Slice (start , end ,step)

Parameter	Description
start	An integer specifying the position to start slicing. Its default value is zero. It is optional
End	An integer specifying the end of the slicing
Step	An integer specifying the step of slicing. Its default value is one

Ex:

```
temp= (“Online”, “IT”, “Guru”, “Training”, “Education”, “and”, “placement”, “centre”)

x = slice(3, 5)

print(temp[x])
```

Output:

```
(‘D’, ‘E’)
```

11. Python File

The next topic that we need to discuss is about python. Like C, Python File operations. Let us discuss one by one in detailed.

Before going to discuss the various File I/O operations. So, let us first discuss

What is a file?

A file is some information (or) data that stays in the computer storage devices. Python supports two types of files namely text file and binary file. Text files are the simple files whereas the binary files contain the binary data. Both humans and machines can read the text files. Whereas the binary files can be read by the only computer.

So in **learn python** code**,** we have seen what is meant by a file? Now let us have a look over python file operations

How to create a file?

As said above, python allows you to create the file. With open () we can create a files. This function takes three arguments. The file that you want to open and the kind of operation (or) permission, you want to do it on the file and buffer.

Buffering:

When the buffering value set to zero, no buffering takes place. But if it set to one, while accessing the file, buffering the file. Usually buffering action takes place if the buffering value greater than one. (With the indicated buffer size). If it is negative , the buffer size is system default.

It can be done through the following syntax:

f = open (file_name, access mode, buffering)

Ex :

```
f = open (itguru.txt, w+)
```

Here, in the above example, ‘W’ stands for write mode and ‘+’ is used to create the file, if it does not exist in the library.

The other operations that can be done with the file are ‘r’ for reading and ‘a’ for append. And we use the ‘+’ sign to create the file if it does not exist in the directory.

So now we have created the file. Now it’s time to know

How to write data into the file?

As shown in the example, above since we have created the file, it’s time to add the data into the file. So as of now, I would the add first 10 number (1 to 10) in a file. So to use this concept, we use a loop.

Ex:

```
f = open (“balajee.txt”, “w+”)

for i in range(10):

f.write(” %d\r\n” % (i+1))
```

Output:

This would store the file as

```
1

2

3
```

4

5

6

7

8

9

10

So now we have opened and written the data into the file right, now its time to close the file.

How to close the file?

We can close the file with the following syntax

Syn:
f.close()

And other this, we can perform some more operations on the file.

Attribute	Description
file. Mode	It returns the file of the mode in which it is opened
file.name	It returns the file name
file.softspace	It returns false if space is explicitly required with print. Otherwise it true.
flush()	It helps to write the buffer of the file system

read()	To read the ‘n’ characters from the file.
readline(n== -1)	To read and return one line from the file (if specified, it returns the at most ‘n’ bytes
readlines(n=-1)	To read and return the list of lines from the file. (if specified, it returns atmost ‘n’ bytes (or) characters.
Writable	If the file system can be written , it returns true
write(s)	To write string s to the file and returns the number of characters written.
Write lines(lines)	To write the list of lines to the file

So till now, we have discussed how to create, insert the data and close the file. But rather than this python supports some more file operations .let us discuss one –by –one in detailed.

To get in-depth knowledge on Python you can enroll for live **Python Online Training** by OnlineITGuru with 24/7 support and lifetime access

File Mode operations

Mode	**Description**
r	Opens the file in reading mode
r+	Opens the for reading as well as writing. Here the file pointer is placed at the beginning

rb	Opens the file in reading mode but in binary format
rb+	Open the file for both readings and writing the file in binary format. The file is placed at the beginning of the file
W	Opens the file for writing. It overwrites the file if the file exists
W+	Opens the file for both reading as well as writing. If the file already exists, it opens the file. And it creates a file for reading and writing
Wb	Opens for writing the file. But in binary format only
Wb+	Opens the file for both reading and writing in the binary format. If the file already exists it overwrites the file. And if the file does not exist it creates a new file for reading and writing.
a	Opens the file for appending. It does not overwrite the file, but just add the data in the file. And if there is no file, it just creates the file
a+	Opens the file for both reading as well as appending. It the file exists, the file pointer is at the end of the file. And if the file does not exist, it creates the file for editing
ab	It opens the file for appending, but in binary format
ab+	It opens the file for both reading and appending in a binary format. If the file exists, at the end the file pointer is placed. And if the file does not exist, it creates the new file for editing

File operations

Other than reading, writing, and closing, we can perform some file operations.

Functions	Operation	Description
File position	tell()	Informs the current position of the file.
File position changing	Seek(offset[,from])	This method changes the current file position. Besides the offset here indicates the number of bytes to the move. By default it is set to zero From – 0- the beginning of the file From -1 – current position of the file From-2 – end of the file
renaming	os.rename(current_file_name, new_file_name)	Renames the current file name with the new file name
removing	os.remove(filename)	Removes the desired file
Directory creation	os.mkdir(new_file)	This command creates the directory

Working dir	os.getcwd()	This method displays the current working directory
Removing directory	os.rmdir('dirname')	This method removes the directory. It takes the file name as an argument
Change directory	os.chdir('new directory')	This method is used to change the current directory

Suggestion

Hope you got a little idea regarding the file operation. And if you practice those commands of your own, you will get an idea. Feel free to clarify your doubts at python online education

12. Python modules

Since python supports code re usability, many people today love to **learn** python programming**.** One way to achieve code re usability is FUNCTION (). But with function, we can achieve code re usability for a certain line of code. But there may be a case, where we need to include code in terms of hundreds of lines. In such cases, this function does not work well (Increases the length of the code). So in this case, we will place this code in a separate file and we can use this file at the time of requirement. And this can be achieved through the concept of python modules.

What is a module?

A module is a python file containing Python definitions and statements. The file name should have an extension of .py. In python modules, we can group similar data. This makes the programmer to easily understand the code. And within the module, the module name is available the global variable __name__. This module contains the executable statements as well as the function definitions. Modules can import the other modules too. Usually, modules are imported as follows:

From – This is used to indicate the file names that should be imported from.

Import – this is used to import the entire module

Reload – It permits to reload the python modules. When the module is imported into the script, the code in the top level portion executed only once. And if you wish re-execute this top-level code, you can use the reload ().Moreover, it reloads and imports the function again

Reload (module_name)

Note:

But while importing these files, the developers do not require to support .py extension.

Python cannot import the statements started with an underscore (_).

In **learn python**, now it's time to know,

How to create a module?

AS said above, this python module is a separate python file. So, for example, create a python file, with the name primary**.** This contains the following code.

Import primary

Primary. Modules ('hero')

Now create another python and import this primary.py

```
def modules(name):

 print(“Hi, ” + name)
```

let us save this file as a second.py. So now, we will be getting the output as

Output:

```
Hi, hero
```

Note:

You can use any python import statement in some other python file. This has the following syntax:

Ex: import module1 [, module2 [,….module N]

Import * statement:

Python interpreter allows you to import all the names from the module into the current namespace. Usually * is used to import all the items into the namespace.

So now you people will be thinking of

How does the python search the file in the directory?

When the python modules are imported, the interpreter is imported in the following sequence:

1. Current directory
2. If the module is not found, then it searches each directory in the shell variable PYTHONPATH
3. And if fails then the python interpreter checks for the default path. In Unix based system, this path is usually found at /usr/local/lib/python.
   The module search path is stored in the system module as the **sys.path** variable. The **syspth** contains the current directory, the python path, and the installation path.

What is a python path variable?

The python path is an environment variable consists of a list of directories. And the **python path** syntax is the same as **shell variable path**

Usually, in the Windows system, the python path system is located at:

set PYTHONPATH = c:\python20\lib;

And in the Unix system, the python path is located at :

set PYTHONPATH = /usr/local/lib/python

In the best way to learn python, the next topic that needs to discuss is namespaces and scoping

Namespaces:

Variables are the names that map to the objects. A namespace is a variable name dictionary and their corresponding objects.

A python statement can access the variables in the local namespace as well as the global namespace. And if the local variable and the global variable have the same name, the local variable shadows the global variables.

Each function has its own local namespace. As ordinary functions, the class methods follow the same scoping rules.

Usually, python makes an educated guess on whether variables are local (or) global. It assumes any value assigned is local.

So, within a function, initially, we must use the global statement to define a global variable.

If the statement contains the global varname, then the python interpreter confirms that is a global variable. And it won’t search the variable name in the local namespace.

So as said above, this python module consists of several variables, modules, and functions. And the interested contains several built-in packages. And we cannot which module contains which files. So, have you ever thought?

How to get the directories and file names in the package?

This can get with the dir(). This function returns all the files and directory names in the package.

```
import math

content = dir(math)

print (content)
```

Output:

```
[‘__doc__’, ‘__loader__’, ‘__name__’, ‘__package__’, ‘__spec__’, ‘acos’, ‘acosh’,
‘asin’, ‘asinh’, ‘atan’, ‘atan2’, ‘atanh’, ‘ceil’, ‘copysign’, ‘cos’, ‘cosh’, ‘degrees’, ‘e’,
‘erf’, ‘erfc’, ‘exp’, ‘expm1’, ‘fabs’, ‘factorial’, ‘floor’, ‘fmod’, ‘frexp’, ‘fsum’,
‘gamma’, ‘gcd’, ‘hypot’, ‘inf’, ‘isclose’, ‘isfinite’, ‘isinf’, ‘isnan’, ‘ldexp’, ‘lgamma’,
‘log’, ‘log10’, ‘log1p’, ‘log2’, ‘modf’, ‘nan’, ‘pi’, ‘pow’, ‘radians’, ‘remainder’, ‘sin’,
‘sinh’, ‘sqrt’, ‘tan’, ‘tanh’, ‘tau’, ‘trunc’]
```

Besides the dir() , the python interpreter contains several buit- in modules . So let have a look over those built- in modules.

Function	Description
ceil(n)	Returns the given number next integer
floor(n)	Returns the previous number of the given number
sqrt(n)	Returns the square root of the number
exp(n)	Returns the natural logarithm ‘e’ raised to the power ‘n’
log(n.baseto)	Return s the natural logarithm of the number
power(base to, exp)	Returns base to raise to the exp power
tan(n)	Returns the tangent of the given radian
sin(n)	Returns the sine of the given number
cos(n)	Returns the cosine of the given number

So now let’s move on to the topic

13. Python packages

A package is a hierarchical file directory structure that defines the single python application environment. This package consists of a module, sub packages, and sub-sub packages and so on. Usually packages are the way of structuring the python module namespace using dotted module names. This python packages allows the uniform handling of files and data.

For example, **A.B** indicates, Package **A** contains a sub-module **B.**In order to treat the python directories as packages, **__init__.py** files are required. And there is no restriction for the __init__.py file w.r.t to the number of code line. (i.e it can be empty files (or) a file that contains some code). Besides this python interpreter allows the user to execute the individual modules of the package.

So, are you looking at the example code? If so don't waste your time by scrolling your mouse up and down. Since this concept is common in other programming languages like Java, I would like to leave for you as an assignment. Visit Python **online education** for support.

In the definition of the package, we have seen the package is a combination of several directories, functions, module so on. In some cases, we require only a few components of the package. And in some other cases, we require all the modules of the package. And there is no limit regarding the number of modules, files in the package. It's good to use **import** statement, if the package contains less than 5 modules, function. But what if the package contains 100's of modules/ directories. It's a bad way of programming to use **import** statement for those 100 files/directories inclusion. In such cases, we need to use an alternative to the problem. ***** provides a good solution to the problem. Usually * is used to import all the package modules.

Also check the essential packages in Python and IOT

Note:

If the package imports the file name from the current module, usually dot notation is used. And the leading dot indicates the packages involved in the relative import.

Beside this Python interpreter also supports the special attribute __path__. Usually, this contains the directory name that contains the __init__.py.

So in python tutorial**,** we will next about

Python Regular expressions:

A regular expression is a character sequence that forms a search pattern. In a variety of ways, these expressions permit you to match various string values. These are essentially a tiny highly specialized programming language available inside the python and made available through re-module. With this language, you can specify the rules for the set of possible strings to match. And this possible string may be English sentence, e-mail address (or) any other Tex commands.

Additionally, this module **re** provides full support for **Perl** like the regular expression in python. At the compilation time, if an error occurs, the re-module raises **re.error** exception.

Without invoking the special meaning, regular expressions use the backslash (\) to indicate the special forms (or) special meaning.

Regular expression syntax:

It is a set of strings that match the required expression. Additionally, this regular expression allows the concatenation to form new strings. If A and B are the regular expression, then AB is also a regular expression.

Ex: if the string **P** matches a and the other string **q** matches B then the string pq will match **AB.** Now let us have a look over the python regular expression characters

Regular Expression characters:

Symbol	Meaning
.	Except new line character matches any character
^	It matches any string starting
$	It matches any string ending
*	In the given regular expression, it searches zero (or)more repetition
\	Either a special character (or) a regular expression character (or) special regular expression sequences.
?	It Matches one (or) more previous regular expressions
+	In the One(or) more occurrences
|	It is used in the cases of Either (or)
()	Capture and gr
{}	It is used to specify the exact number of sequences
.	Uses for any character (except newline character)
^	It is used for starting of the regular expression
[]	It is used for the character set
[…]	Matches any single line character in brackets
[^…]	Matches any single character but not in brackets

Get the real-time examples on regular expression characters at **python course**

Besides this regular expression match, Python interpreter also supports the match expressions.

Match expressions:

So now, let us have a look over the match expressions:

Pattern	Descriptions
re*	It matches for the zero (or) more occurrences of the preceding expression
re+	It matches one (or) more occurrence of the preceding expression
re?	It matches for the one (or) more occurrence of the preceding expression
re{ n}	It matches for the exactly ‘n’occurances of the preceding expression
re{n, }	It matches for n (or) more occurrences of the preceding expression
re{n,m}	It matches for at least ‘n’ of the’ occurrences of the preceding expression
a|b	It matches for either a (or) b
(re)	It groups the regular expressions and remembers matches text

Like match expression, it supports the special sequences. Now let us have a look over those special sequences.

Special Sequences:

Character	Description
\A	It returns the match if the specified characters are the string beginning
\b	It returns the match where the specified characters are at the beginning (or) end of the word
\B	Returns a match where the special characters are present. But not at the beginning (or) end of the word.
\d	Returns the string that contains numbers 0-9
\D	Returns the string that does not contain digits
\s	It returns the string that contains the special character
\S	It returns the string that does not contain the special character
\w	It Returns a match, where the string contains any words
\W	It Returns a match, but string does not contain any word characters
\Z	It Returns the match, if the specified characters are the string end

Besides this, it additionally supports some built-in functions. Now let us have a look over those.

Functions:

Function _name	**Description**
search()	It searches the string for the match. If there is a match, it returns the match object
split()	The split () returns a list. Here the string has been split at each match.
sub()	It replaces the text match with the text of your choice
group()	It returns the string that was matched by the regular expression
start()	It returns the starting position of the match
end()	It returns the ending position of the match
span()	It returns the tuple that contains the staring position of the match

Match ():

A match object is an object containing the information about the search and result

So likewise python supports regular expressions. Since I have said the working of these functions I would like to leave the example code for you as an assignment. If you struck up anywhere feel contact the experts at **Python training in Hyderabad**

14. Classes and Objects in python

Since as discussed in the python definition, **Python** is an Object-oriented programming language. So, now it's time to discuss Classes and objects.

Class:

Everything in python is considered as a class. It is a user-defined prototype for the object. It defined the set of attributes defines the attributes of the class. The attributes are the data members. We can access these methods using dot (.) notation

We can define the class with **Class** followed by the class name. The class name should be followed by colon (:)

```
Class Class_name():

    Statements

Print(Class_name)
```

Ex:

```
class A:

 x = 5

 print (x)

print(A)
```

Output:

```
5

<Class '__main__.A'>
```

Object:

It is a unique instance of the data structure. Usually class defines the objects. An object comprises both data methods and members.

Method:

It is a special kind of function that is defined in the class definition.

Data member:

These can be class variables (or) the instance variables. They hold the data that is associated with the class and objects.

Instance:

An instance is an individual object of a certain class. For example, an object belongs to the class circle, would be an instance on the class circle.

Class variables:

It is a variable that can be shared via class instances. These variables are declared within the class but outside the methods. But these are not frequently used as instance variables.

Instance Variables:

Inside method, we can define variables. These variables belong to the current class instance.

Python With oop:

Before we were going to discuss python with oop, let us first have a look on

Why object-oriented programming (oop)?

Programming languages were basically classified into two types. Procedural (or) structural programming language and object-oriented programming. In object-oriented programming, a program has several mini-programs. Here each object

Represents a different part of the application. And each part of the application has its own logic. This helps to communicate among themselves.

Now, let's have a look over python object oriented features.

Abstraction:

To simplify the code complexity abstraction is used. But we cannot instantiate the abstract class. It means for these classes, we cannot create the objects (or) classes. We can abstraction to inherit certain functionalities. Moreover, here we can inherit the functionalities, but at the same time, we cannot instance of the particular class.

Since this property is common in OOP like JAVA, here I would like to leave the code for you as an assignment. If you need any help feel free to contact python **online education**

Inheritance:

This is basically the property of the class. Through this concept, we can transfer one class properties to the other. This is useful at the time of subclass creation. These sub classes can easily get the properties from the parent classes. Moreover, as per the client requirement, without affecting the parent functionality, here we can override (or) add the new functionalities.

So, to use this **inheritance**, we must have two classes.

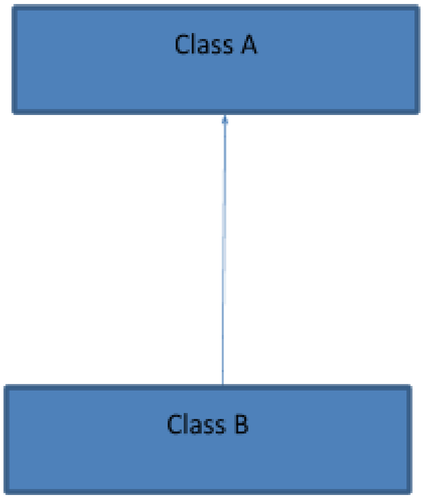
a) Parent class (Base (or) superclass)

b) Child class (Subclass (or) derived class)

Usually, the child class inherits the parent class properties. This Inheritance is further divided into types. Those are single, multilevel, hierarchical and multiple inheritances. Let us discuss one by one in detailed.

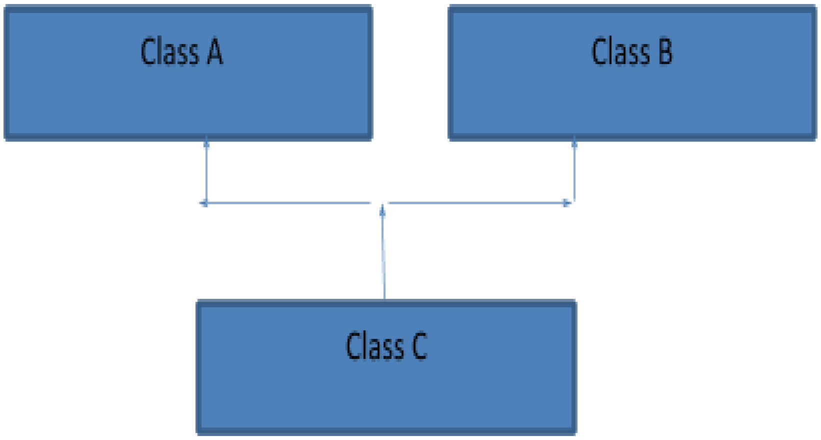
Single Inheritance:

A class that is inherited from another single parent class itself is called single inheritance



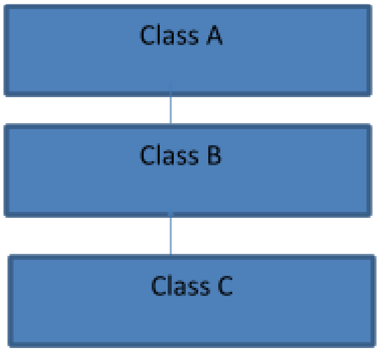
Multiple inheritances:

It means you are inheriting the properties of multiple classes into one. In this, two classes A and B inherits the properties from Class C.



Multilevel inheritance:

In this level of inheritance, various properties were acquired at various levels.



Hierarchical Inheritance:

Functional overloading:

It means assigning more than a single behavior to a particular behavior. The operation here performed varied by the argument types and the objects involved.

Operator overloading:

It means assigning more than one function to the particular operator.

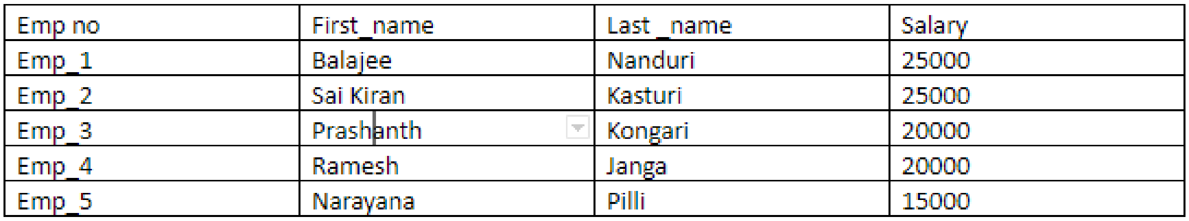
Attributes in Python:

It defines the property of the object. Element (or) file. Python supports two types of attributes.

Built-in class attributes:

These are the built-in attributes present inside the python classes. For example, _dict_, _doc_, _name_ etc. Let me consider the attributes with an example.

Let us assume, we have the below table



Emp no	First_name	Last_name	Salary
Emp_1	Balajee	Nanduri	25000
Emp_2	Sai Kiran	Kasturi	25000
Emp_3	Prashanth	Kongari	20000
Emp_4	Ramesh	Janga	20000
Emp_5	Narayana	Pilli	15000

Now, I would like to display the employee_1 details. All the details that belong to employee_ 1 are known as **Attributes** of the employee.

We can achieve this as follows

This would display the following output.

‘First_name’:’Balajee’,’ Last_name’: ‘Nanduri’, ‘salary’:25000

User-defined attributes:

We can create these attributes inside the class definition. Additionally, Python allows you to dynamically create the new classes for the existing class instances. Like C and JAVA, this python attributes also have the variable scope. Let us have a look over those in detailed.

Scope of the attributes:

Public:

We can use these attributes class definition inside (or) outside

Protected:

We can use these attributes inside the subclass definition, but not outside the class definition.

Private:

This kind of attribute is inaccessible and invisible. Except inside the class definition, it is neither possible to read nor possible to write those definitions.

Polymorphism:

It is the ability to present the same interface in different underlying forms. Technically speaking, if the child class inheriting the parents it does not necessarily mean to inherit about the parent class. In other words, unlike parent class, it may do some things in a different manner. Basically, polymorphism is applied using inheritance. Moreover, IT training says python is implicitly polymorphic. It means python has the ability to override the standard operators.

Let the get the more clarity on this with the example below:

```
class cow(object):

  def color(self):

      print(“white”)

class buffalo(object):

  def color(self):

      print(“black”)

def animalcolor(animal Type):

  animalType.color()

cowObj = cow()

bufObj = buffalo()

animalcolor(cowObj)

animal color(bufObj)
```

Output :

```
white

black
```

15. Errors and exceptions in python

And the next topic that I would like to discuss is Errors and exceptions in python?

Errors and exceptions in python:

Even this concept is familiar in other programming languages like **JAVA**, it’s the best thing to recall those terms once.

What is an error?

An error is a condition that occurs when the programme does not follow the proper structure.

Basically, errors are classified into two types :

Syntax errors and

Logical errors

Syntax errors:

These errors occur if the code violates the compiler (or) interpreter rules

For example:

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

for i in the test:

print (‘This is an error’)#here we have an indentation error
```

Output:

```
Indentation Error: expected an indented block.
```

So the next thing that we need to discuss is about the exceptions

What is an exception?

These are nothing but the logical errors that were found during run time. In other words, if the program is compiled (or) interpreted the operation (Compilation/interpretation) goes well. But soon after, at run time we may found expectations. These are nothing but exceptions.

Division with zero is an example of an exception.

Get the best real time examples of exceptions at **online certificate courses**

Python interpreter supports some built-in exception. Let us have look over those.

Exception Class	**Event**
I/O errors	It arises when the input/ output operation fails
Arithmetic error	It arises when the numeric calculation fails
Floating point error	It arises when the floating point calculation fails
Zero division error	It arises when the division (or) modulo by zero occurs
Assertion error	It arises when the assert statement fails
Overflow error	It arises when the arithmetic operation is too large
Import error	It arises when the imported module is not found
Keyboard interrupt error	It arises when the user interrupts the process execution. It is usually was done by pressing Ctrl+c
Stop iteration error	It raises when the next method of the iterator does not point to any object

Indentation error	It arises when there is an incorrect indentation
Name error	It arises when the identifier is not found in the local (or) global namespace
Key error	It arises when the specified key is not found in the dictionary
Value error	It arises when the function gets arguments of correct type but improper value.
Attribute error	It arises when the attribute reference (or) assignment fails
Syntax error	It arises by the parser when the syntax error is encountered
Index error	It arises when the index is out of range
Runtime error	It arises when the generated error does not fall into any category
Type error	It arises when the function is applied to the object of incorrect type
Unbound local error	It arises when we are trying to access the local variable (or) a method in a function. But here default value is not assigned.
EOF error	It arises when there is no input either from raw _input() (or) input function

So likewise python supports many built-in functions. Visit **it certifications** to get examples on these.

In real time, we cannot expect the code in a few lines. The program should not stop executing for an unexpected (Unknowing error), the program should not

stop executing. In such a case, we need an alternative for the execution of the programs. Try – raise exception provides a solution to this kind of problems.
So let us have a look what does these mean.

Try:

In python using try. We can catch exception. But, If any code within the try causes an exception, exception of the code will stop and jump to the except statement. But when an exception occurs in the **try** block, **python** looks for the matching **except** block to handle it.

Syntax:

```
Try:

Statements

Except:

Statements

Statements
```

Example:

```
try:

x > 100

except:

print(“Something went wrong”)

print (“Even if it raised an error, the program keeps running”)
```

Output:

```
Something went wrong

Even if it raised an error, the program keeps running.
```

The one more kind of exception that we need to handle is the **try-except** block:

Here in the **try** block, we can place the suspicious code where there is a probability of error. After the try block, we can place the **except** statement. The except statement is followed by the block of code. This handles the code as efficiently as possible. The syntax is shown below:

Syn:

Try:

```
Statements

Except exceptions 1

 ———————

———————–

Except exceptions 2

…………

………….

else:

if there is no exception, this block is executed.
```

Example:

```
try:

  ab = open(“example”, “r”)

  ab.write(“This is my test file for exception handling!!”)

except IOError:

  print (“Error: can\’t find file or read data”)

else:

  print (“Written content in the file successfully”)
```

Output :

```
Error: can’t find file or read data
```

Features of try-except blocks :

A single try block has multiple except statements. This is useful when the try block contains exceptions and throws different exceptions.

Here you can provide a generic except clause which handles any exception

After the except clause, we can include an else clause. If the code in the try block does not raise an exception the code in the else block executes.

The else block is a good place for the code which does not require the try- block protection.

Can we use except class without exceptions?

Yes, we can use the except class where there is no exception defined. Usually, the try-catch statement catches all the exception that occurs. Because in the good programming practice, try- except is not considered as a best. Besides, the **try-except** statement catches all the exceptions. But it does not make the programmer identify the root cause of the problem.

Note:

We can use the same except statement to handles multiple exceptions.

Syn:

```
try:

  Operations

  ………………….

except(Exception1[, Exception2[,…ExceptionN]]]):

  This block executes If there is an exception from the given exception list,

  ………………….

else:
```

This block executes only If there is no exception

And the last one that we need to discuss here is **try – finally,** clause:

As said many times, we cannot except the code to be alike. There might be situations where the programmer need to execute the group of statements compulsorily(irrespective) In such a condition, we would use **try- finally.**

Python interpreter allows you to use **finally** along with try. Irrespective of the condition satisfies, it is a piece of code that must execute mandatory.

try:

error possibility area

…………………

This may be skipped due to any exception

finally:

This block of statements will definitely executes.

Example:

With this syntax, let us have a look over the code

```
try:
  fh = open("testfile", "r")
  try:
    fh.write("This is the example of exception handling with try and finally")
  finally:
    print ("I will definately exceutes")
    fh.close()
```

```
except IOError:
  print (“Error: can\’t find file or read data”)
```

Output:

```
I will definately exceutes

Error: can’t find file or read data I will definately exceutes
```

Does an exception allow arguments?

Yes, an exception can have an argument. This gives additional information about the problem. The contents of the arguments varies by an exception. And we can capture an exception argument by supplying a variable in the except clause.

Syntax:

```
try:

some operations will be done here

  ………………….

except ExceptionType, Argument:

here we can print the argument value.
```

So like above are you expecting an example? if yes, I would like to say write the code, compile and run it as an assignment. And if you are unable to solve then get the answer from learn python **3.**

So till now, we have seen what is an exception, what are the type of exceptions and so on? But have you ever think about

How to raise an exception?

Using the **raise** statement, we can raise an exception in several ways. The syntax of the raise statement is shown below:

Syn:

```
Raise [Exception [,args [,traceback]]]
```

Explanation:

The exception is a type of exception(for example, NameError)

The argument is the argument type value.This is an optional value.

The traceback object is used for an exception. This is also optional.

Example:

```
try:
  raise NameError('Hello')
except NameError:
  print('This is an example of raising an error')
  raise
```

Output:

```
Trace back (most recent call last):
This is an example of raising an error
 File "C:/Users/BALU/PycharmProjects/firstproject/first.py", line 2, in <module>
  raise Name Error('Hello')
Name Error: Hello
Process finished with exit code 1
```

And the last topics that we need to discuss here is the user defined exceptions

Till now, we have seen the system defined exceptions. But rather than this, a user can also define exceptions. Now let us have a look over

User-defined exceptions:

Python allows the use the create and define exception of our own. Let us have look how to create the user define exceptions.

Example:

```
# define Python user-defined exceptions

Class Error (Exception):

  “””Base class for other exceptions”””

  Pass

Class minerror (Error):

  “””Raised when the input value is too small”””

  Pass

Class maxerror (Error):

  “””Raised when the input value is too large”””

  Pass

# Our main program

# User guesses a number until he/she gets it right

# you need to guess this number

number = 20

while True:
    try:
        i_num = int(input(“Enter a number: “))
        if i_num < number:
            raise minerror
        elif i_num > number:
            raise maxerror
        break
    except minerror:
        print(“This value is less than the initialized number, try again!”)
        print()
    except maxerror:
        Print(“This value is greater than the initialized number, try again!”)
        Print ()
Print (“Congratulations! You guessed it correctly.”)
```

Output:

```
Enter a number: 5
This value is less than the initialized number, try again!
Enter a number: 100
This value is greater than the initialized number, try again!
```

16. Python Iterator

What is an iteration?

An iteration is a process of iterating through the loop. Basically, the python iterator is a type object that can be used with a for in loop. Python list, dict, tuples, and sets are the examples of the python iterators. Let us get more idea on an iteration with an example

Ex: a=[‘python ‘,’online’,’training’]
for i in a:
print(i)

Code explanation:

In the above given example, a is an array consists of a series of elements. Now our aim is to print all the elements in the elements. This can be done with the variable i. Here the variable I is called the python iterator. Here we called the variable I as an iterator since it was iterated through an array [].

In simple words, an iterator is an element, that performs the iteration process.

But in fact, if any object wants to be an iterator, it must implement the following methods:

Iter:

At the time of initialization, an _iter_ method is called. Besides this _iter_ method should return the object that contains the next (or) _next_ method.

Next:

The next method should return the next variable value. On the iterator object, when an iterator is used within a ‘’for in”, the loop implicitly calls the next().

Besides this method should raise the stop iteration (). This indicates the end of the iteration.

The syntax of the iterator object is shown below:

while (condition)

```
try:
# To get the next item
element = next(iter_obj)
# perform some op
except StopIteration:
# At this point the loop terminates
```

Ex:

```
class test:

def __iter__(self):

self.temp = 1

return self

def __next__(self):

if self.temp <= 5:

x = self.temp

self.temp+= 1

return x

else:

raise StopIteration

var = test()

myiter = iter(var)

for x in myiter:
```

```
print(x)
```

OutPut :

Note:

There is no restriction of the iterators to be finite. I.e. there can be situation, where an iterator to be infinite. So the people must be careful while using iterators.

The advantage of using iterators is that we can save resources like memory. As explained above, in iterators, we need to perform the two functions like iter () and next (). So implementing these two functions is lengthy. So we need an alternative to this concept.

To get in-depth knowledge on Python, you can enroll for live **python online training** by OnlineITGuru with 24/7 support and lifetime access

17. Generators and Decorators

And the next topic that I would like to discuss is Generators and Decorators

Python Generator:

The python generators give an easy way of creating iterators. These generators instead of returning the function from the return statement use the **"yield** "keyword. These are the generator version of the list comprehensions.

If the function contains at least one "yield" statement, it becomes a generator function. Both the **yield** and **return** will return some value from the function.

So now you people might be thinking of

What is the difference between Yield and return?

The return statement terminates the function entirely. And the yield function passes the function. This can be done by saving all the states and later continues from there on the successive calls.

And have you ever think of

What is the difference between a generator and a normal function?

A generator can contain more than one yield statement.

When the iteration object calls the generator, execution does not start immediately.

There is no necessity to call the functions. Additionally, the generator automatically calls methods like iter () and next. Usually next () will iterate these functions

When the function gets yield, the function is paused and the control is transferred to the caller.

Between the successive calls, local variables and their states were remembered.

Finally, if the function terminates, the generator automatically calls the stop iteration ()

Click here to know the **Advantages and disadvantages of python**

Why python uses generators?

Python uses the concept of generators due to several reasons. Let us discuss some of them.

1. Easy to implement:

We can implement the generators easily in a simple and clear manner. It considerably reduces the code when compared to iterators. This is because unlike iterators, generators can call the functions like iter () and next () automatically.

2. Memory efficient:

Prior to the returning of the result, a normal function creates an entire sequence of memory. But if the item number is large memory becomes an overkill. But with generators, the sequence is memory friendly. Because it produces one item at a time.

3. Infinite stream representation:

Generators are an excellent medium to represent infinite steam data. Since the generators produce only one at a time, infinite streams can be stored in the memory.

Ex:

```
def gen():

a = 10

print('This is printed first')

# Generator function contains yield statements

yield a

a += 1

print('This is printed second')

yield a

a += 1

print ('This is printed at last')

Yield a

# using for loop

For i in gen ():

Print(i)
```

And the next topic that we discuss is about the python decorators:

Decorators:

So far in the previous topics, we have seen about many but –in python functions. These functions have already their own logic. But there are some cases, where we need to change the logic of the built-in functions.

Also check more information on **why can we use python Generators?**

Do you think python allows you to change the logic of the built-in functions?

The answer is yes. We can implement the concept of decorators using built-in functions. So

What is Python Decorator?

A decorator is a python interesting features that add functionality to the existing code. This is also called Meta programming. Decorators are also known as the Meta programming. It was said because it was trying to modify another programming part at compile time. Decorators allow us to wrap up another function in order to extend the behavior of another function.

In decorators, functions are taken as the argument in another function. And inside the wrapper function, we can call the decorators. Using the decorators' concept is easy, but the implementation of these decorators is a bit complicated. We can easily implement it with the support of **python online training.**

In Python, we can implement decorator's concept in two ways:

- **Class decorators**

Function decorators

Usually, a decorator is any callable object that is used to modify the function (or) the class. A reference to the function (or) class is passed to the decorator and the decorator returns the modified function (or), class. Usually, the modified functions (or) classes contains the function (or) class call.

So I would like to leave the decorators code as an example. If you struck up anywhere, feel free to contact it online training**.**

18. PostgreSQL Overview

So till now, we have seen the various features of python. Since everybody knows python plays a major role in development. But the application of python does not stop here, additionally, there is some situation where the user needs to supply the input. So we need a storage area to store all the inputs supplied by the user.

Today we have many storage areas to store the user supplied. One of the many storage areas is PostgreSQL. So today in this tutorial, we do have look over the connection of python with PostgreSQL.

And before going into the discussion further, I would like to formally introduce

What is a database?
A database is an organized collection of data that is stored and accessed electronically from the computer system.

So let us quickly move into the topic

What is PostgreSQL?

PostgreSQL is a powerful open source relational database management systems. Today many experts in the IT industry suggest that it has proven architecture and reputation for readability, data integrity, and correctness. The PostgreSQL runs on major of the operating systems like Linux, Unix, and Windows operating systems.

So now we have a got basic idea regarding the PostgreSQL. Now its time to have the environment setup. Follow the step to install PostgreSQL in your local system.

19. Install PostgreSQL

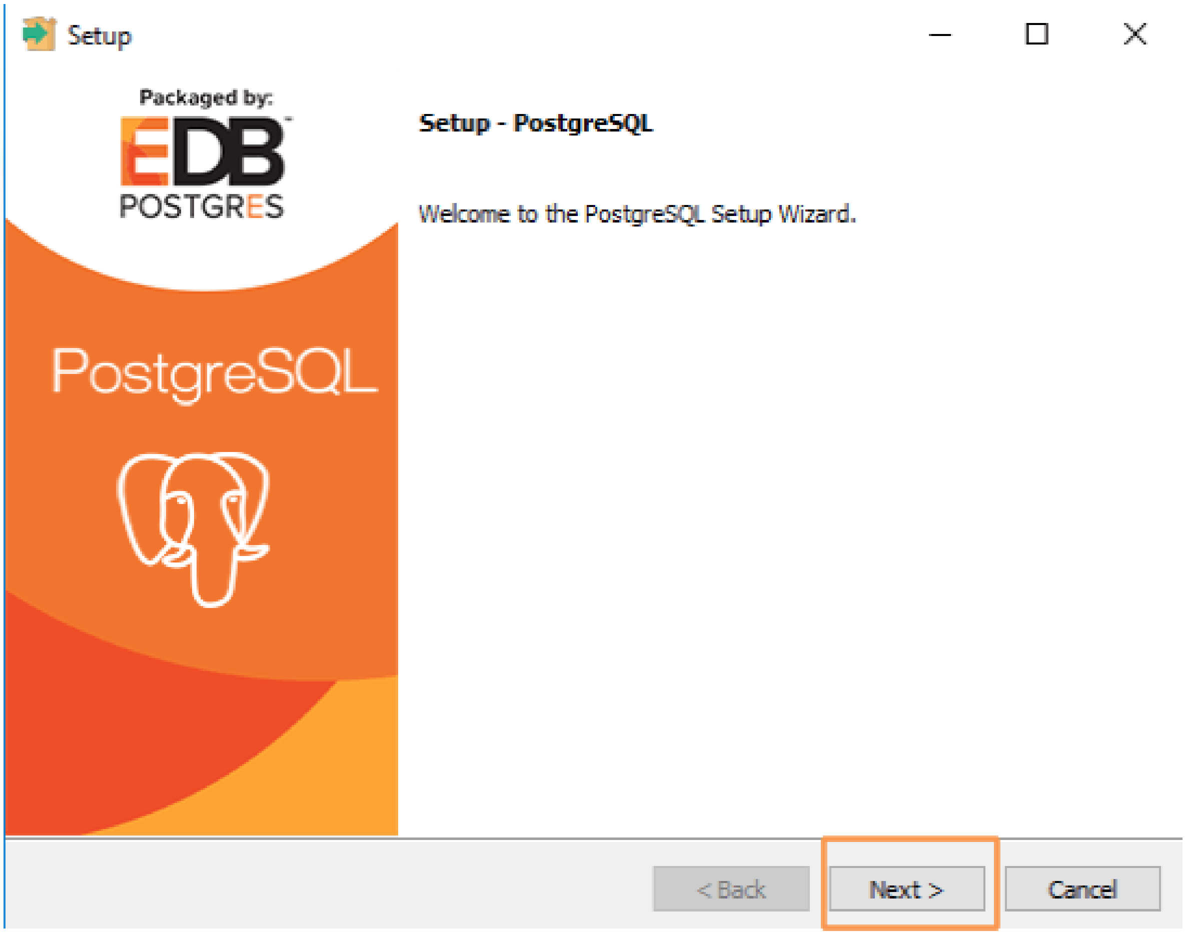
Visit https://bit.ly/2mgWlNN to down the latest versions of your platform to Install PostgreSQL.

(Now I am using the 11.2.1 version on my system).

Once you have selected the version, its time to install the environment in your system.

Install PostgreSQL

Step 1 : Double click the downloaded exe and Click on Next

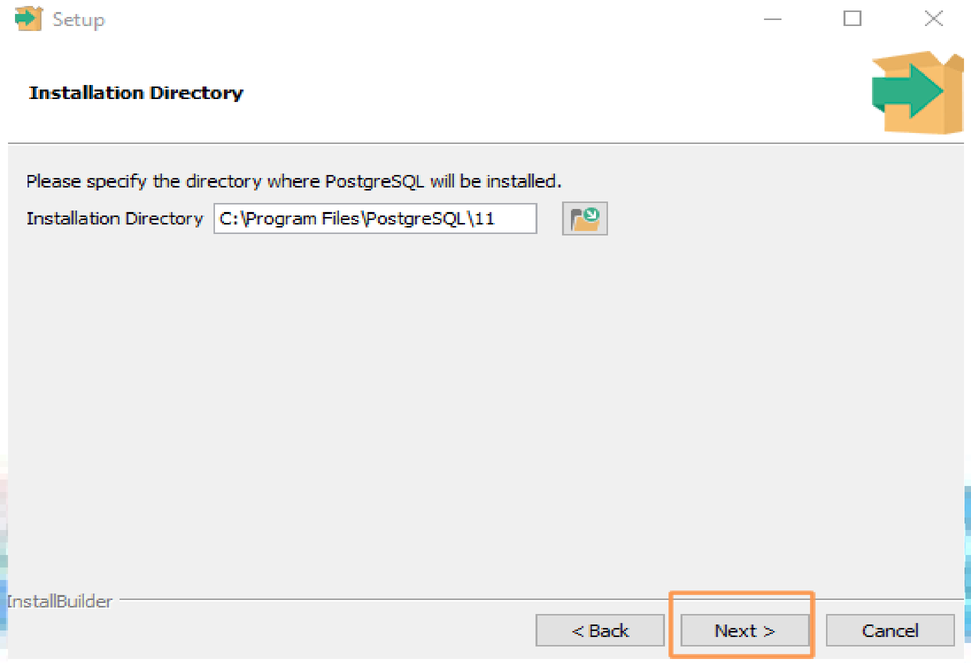


Setup - PostgreSQL

Welcome to the PostgreSQL Setup Wizard.

Step – 2 :

A default storage location is provided. If you wish you can change and click on **next**



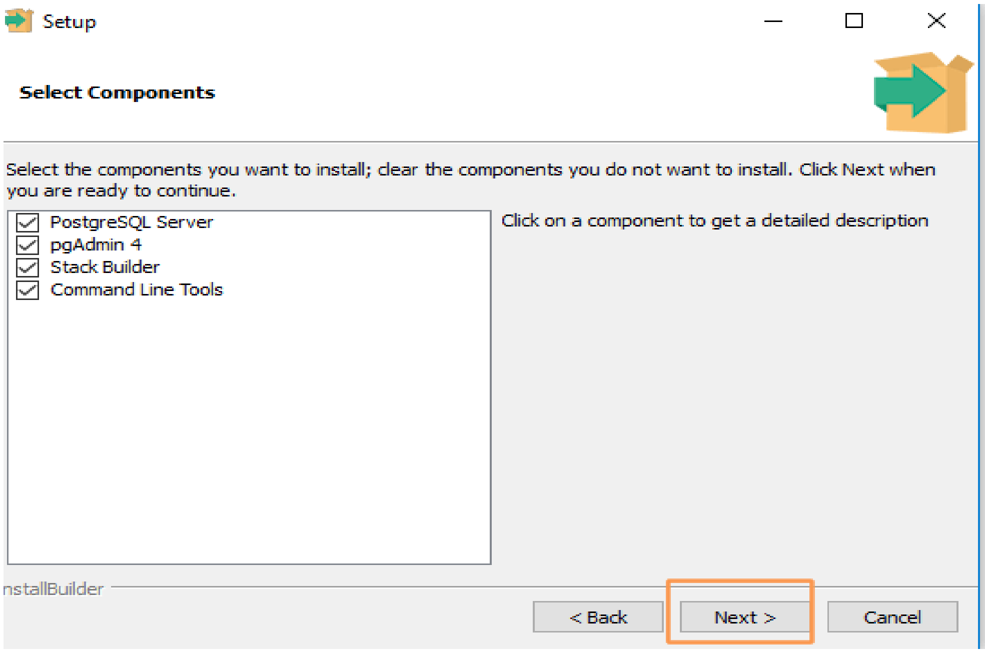
Setup

Installation Directory

Please specify the directory where PostgreSQL will be installed.

Installation Directory: C:\Program Files\PostgreSQL\11

Step -3 :



Setup

Select Components

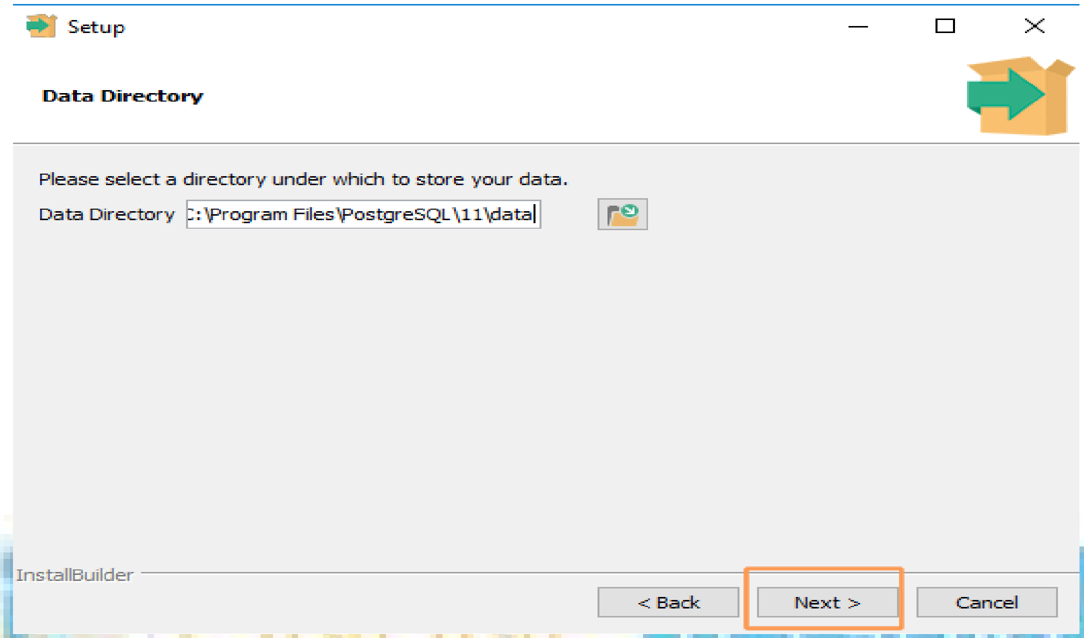
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

☑ PostgreSQL Server
☑ pgAdmin 4
☑ Stack Builder
☑ Command Line Tools

Click on a component to get a detailed description

Click on **Next**

Step – 4 :



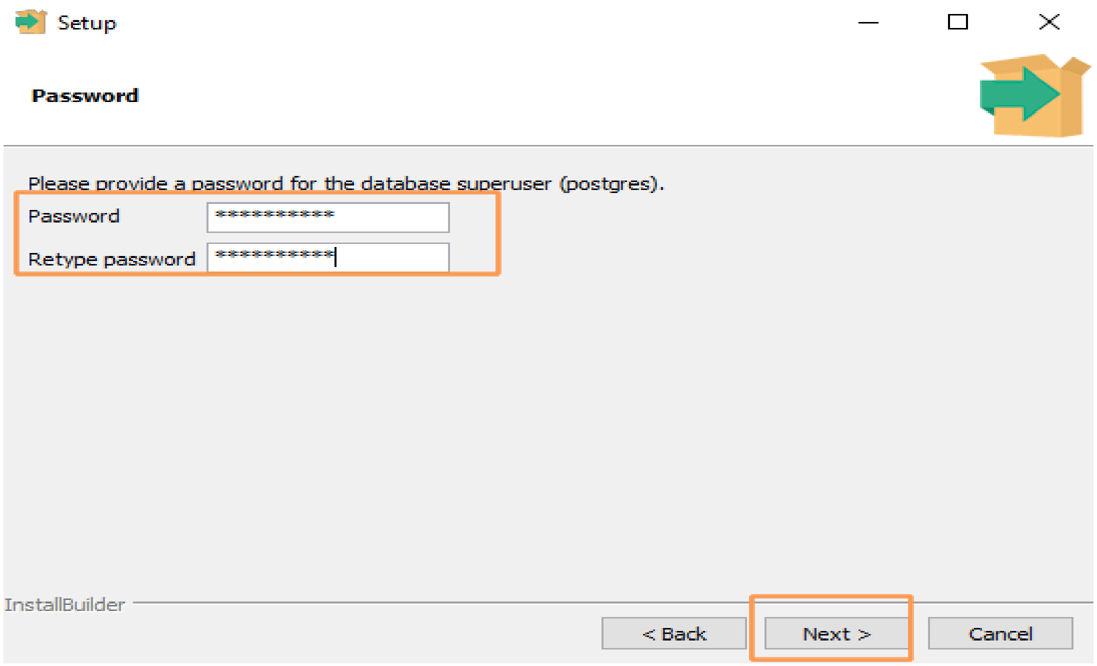
Data Directory

Please select a directory under which to store your data.

Data Directory :\Program Files\PostgreSQL\11\data

Step -5:

Its time to create a password to data base. (Choose of your own)



Password

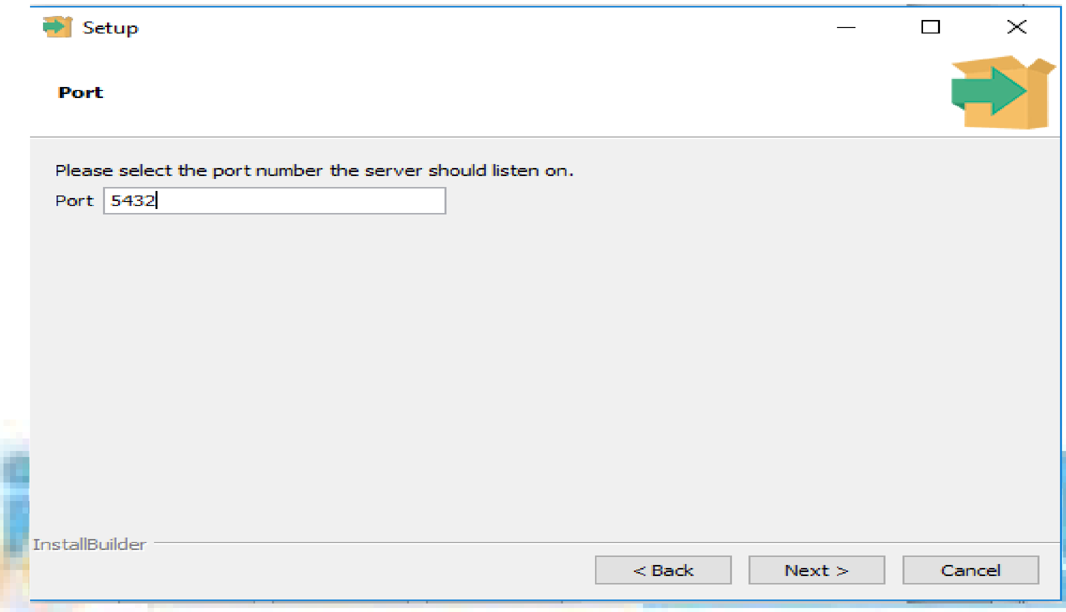
Please provide a password for the database superuser (postgres).

Password ***********

Retype password ***********

Step -6 :

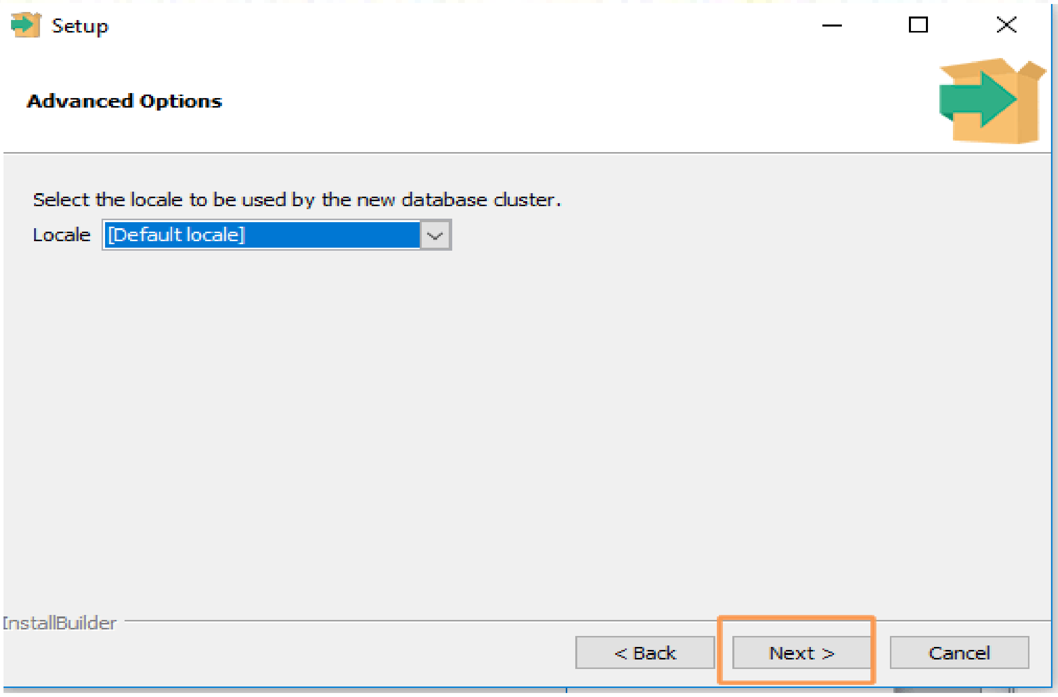
Remember the default database port is provided. If you wish you can change it. You should remember the port and this is useful at the connecting to the database.



Please select the port number the server should listen on.

Port 5432

Step -7 :

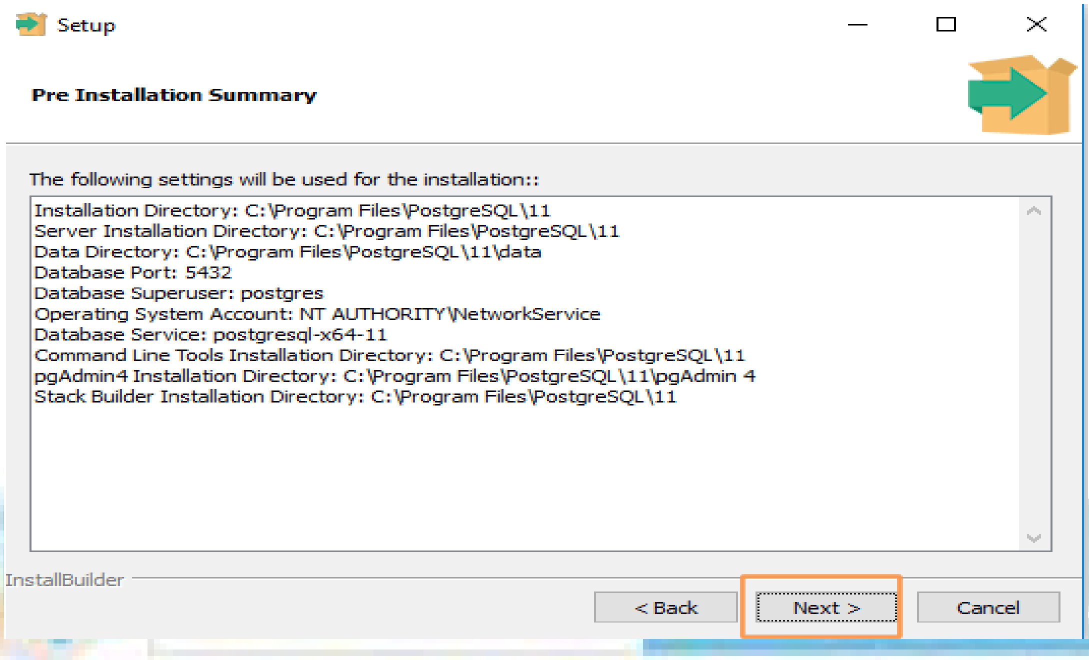


Select the locale to be used by the new database cluster.

Locale [Default locale]

Select the local of the system. A default is provided. If you wish you can change it and click on **Next**

Step -8 :

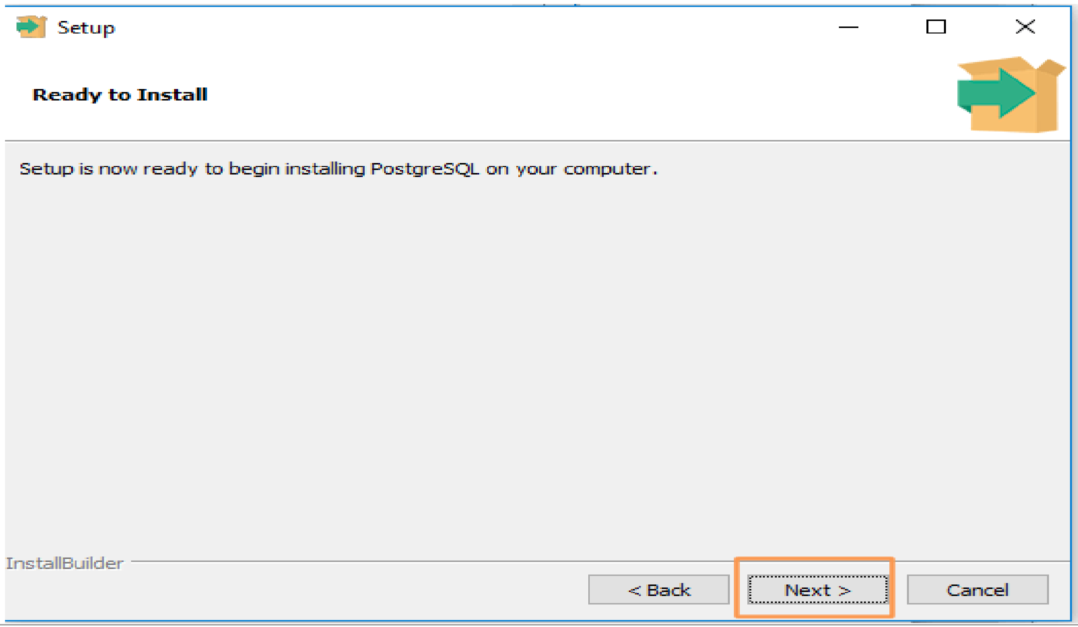


The following settings will be used for the installation::

```
Installation Directory: C:\Program Files\PostgreSQL\11
Server Installation Directory: C:\Program Files\PostgreSQL\11
Data Directory: C:\Program Files\PostgreSQL\11\data
Database Port: 5432
Database Superuser: postgres
Operating System Account: NT AUTHORITY\NetworkService
Database Service: postgresql-x64-11
Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\11
pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\11\pgAdmin 4
Stack Builder Installation Directory: C:\Program Files\PostgreSQL\11
```

Step – 9 :

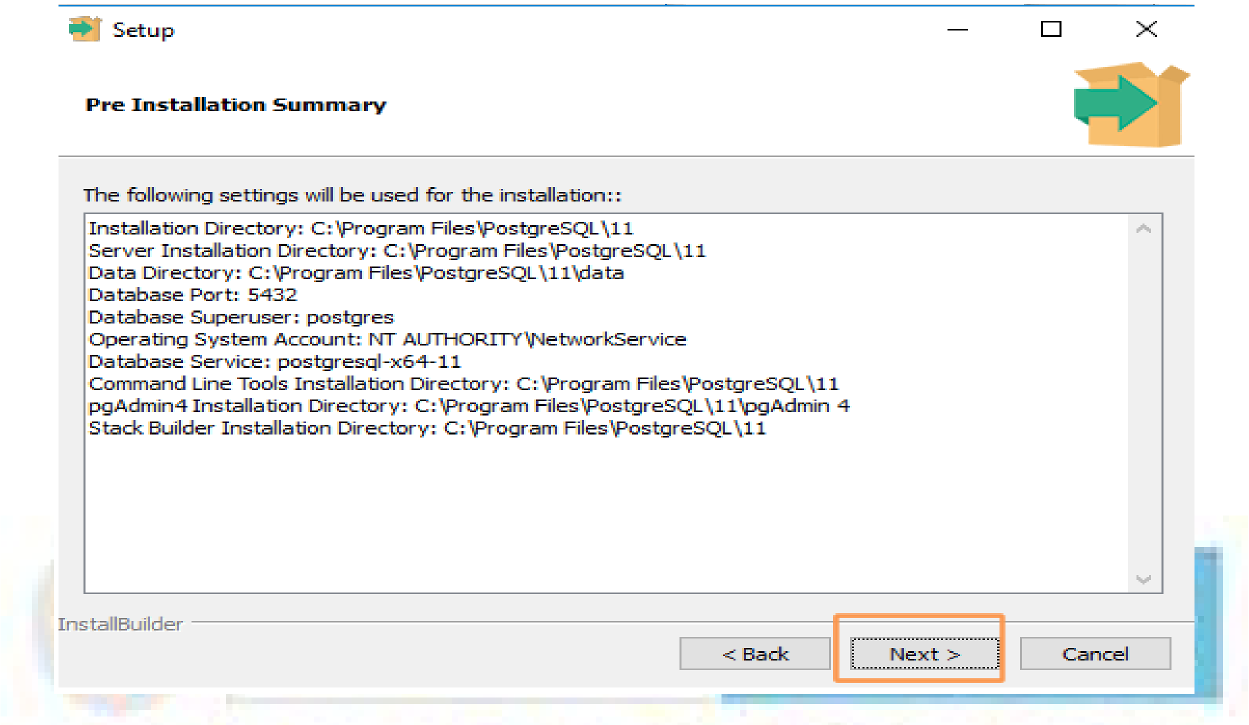
Click next



Setup is now ready to begin installing PostgreSQL on your computer.

Step -10 :

Click **Next**

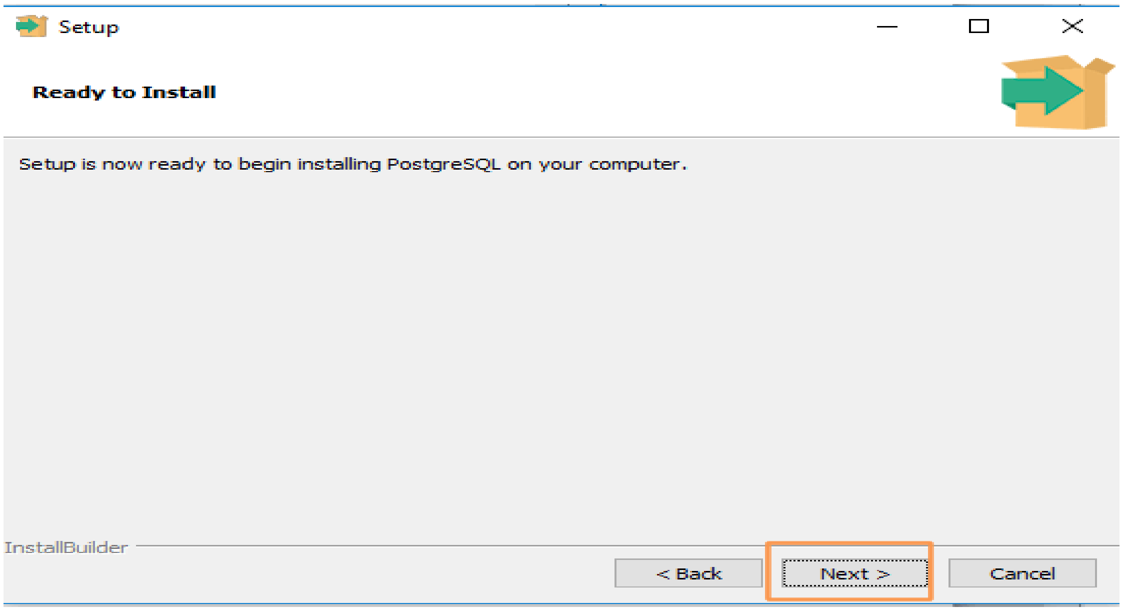


The following settings will be used for the installation::

```
Installation Directory: C:\Program Files\PostgreSQL\11
Server Installation Directory: C:\Program Files\PostgreSQL\11
Data Directory: C:\Program Files\PostgreSQL\11\data
Database Port: 5432
Database Superuser: postgres
Operating System Account: NT AUTHORITY\NetworkService
Database Service: postgresql-x64-11
Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\11
pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\11\pgAdmin 4
Stack Builder Installation Directory: C:\Program Files\PostgreSQL\11
```

Step – 11:

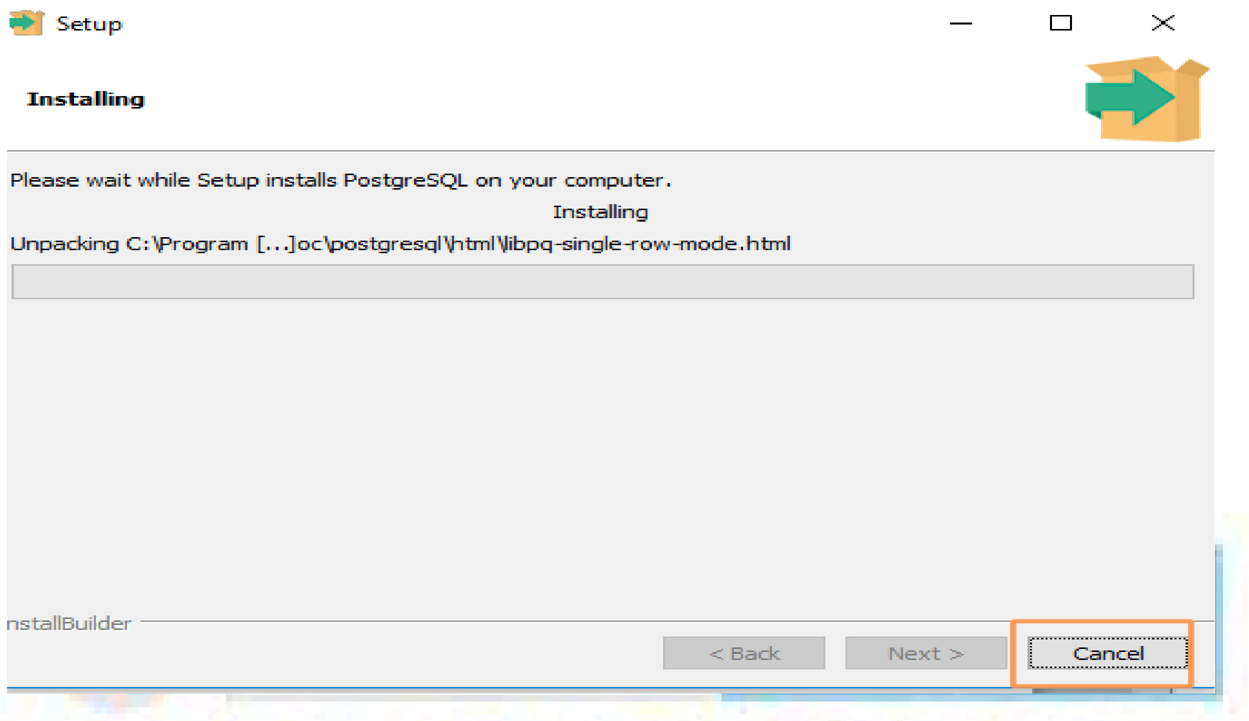
Click **Next**



Setup is now ready to begin installing PostgreSQL on your computer.

Step -12:

In this stage, you will be finding the progression bar as shown below.



Setup

Installing

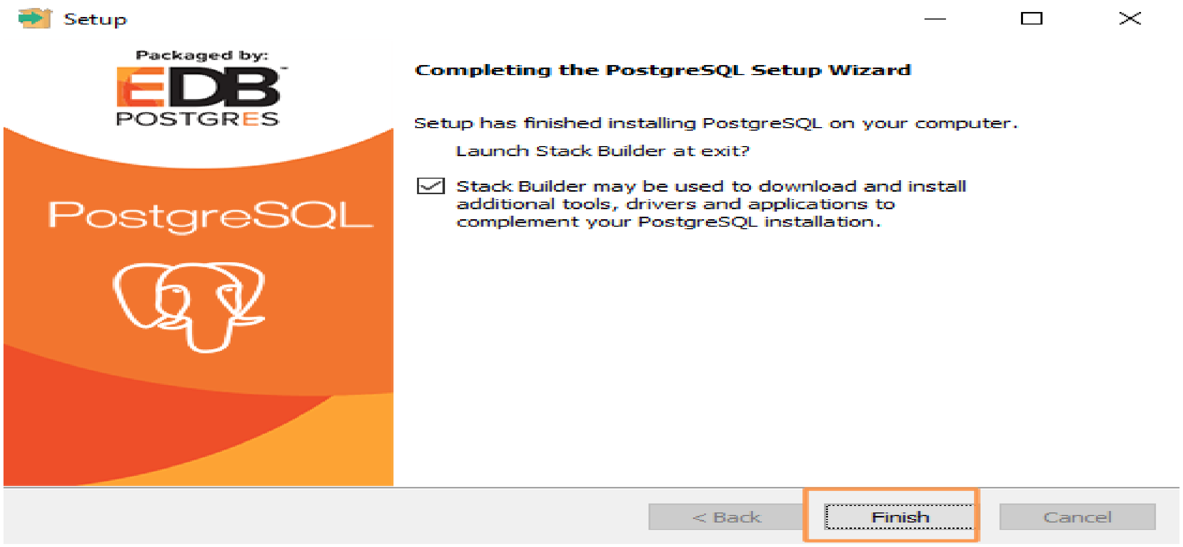
Please wait while Setup installs PostgreSQL on your computer.

Installing

Unpacking C:\Program [...]oc\postgresql\html\libpq-single-row-mode.html

Step – 13 :

Finally, you will be seeing the image as shown below and click on **Finish**to complete the installation



Setup

Completing the PostgreSQL Setup Wizard

Setup has finished installing PostgreSQL on your computer.

Launch Stack Builder at exit?

☑ Stack Builder may be used to download and install additional tools, drivers and applications to complement your PostgreSQL installation.

20. PostgreSQL Data Types

The PostgreSQL is a database that holds different values of data. So in this tutorial let have a look over the PostgreSQL data types. But before we were going to discuss the PostgreSQL supported data types, I would like to recall you once again

What is the data type?

A data type (or) simply types refers to the specific type of data that the variable holds. Moreover, this include date, timestamps, varchar, and some other formats.

Let us have a look at the PostgreSQL datatypes.

Numeric Datatypes :

PostgreSQL supports two types of Numeric datatypes namely integers and Floating point numbers. So let us now discuss the PostgreSQL supports types with range.

Name	Storage size	Range
Small	2 bytes	-32768 – +32767
Integer	4 bytes	-2147483648 to +2147483647
Bigint	8 bytes	-9223372036854775808 to 9223372036854775807
Real	4 bytes	It support 06 digits precision
Double precision	8 bytes	It supports 15digit decimal precision
Decimal	Variable	It permits up to 131072 before the decimal point, up to 16383 after the decimal point

Numeric	Variable	It permits up to 131072 before the decimal point, up to 16383 after the decimal point
Small serial	2 bytes	1 – 32767
Serial	4 bytes	1 – 2147483647
Double serial	8 bytes	1 to 9223372036854775807

Monetary data types :

This is the special data that the database supports. Here in this data type, values of numeric, int and bigint can be converted into money. So today many people say that we can use Float datatype. But this is not recommended to handle money due to the potential for rounding errors.

Name	**Storage**	**Range**
Money	8 bytes	-92233720368547758.08 to +92233720368547758.07

Character data types:

The following table contains the PostgreSQL character datatypes**.**

Name	**Description**
Char(size)/ Character(size)	Here size is the number of characters to store. It contains the fixed length strings. Here the space padded on right to equal size characters.
Varchar(size)	Here size is the number of characters to store. It contains the variable string length

Date/time data type:

The date/time datatype is used to represent the columns using the date and time values.

Name	size	Range	Resolution
Timestamp (with/ without timezone)	8 bytes	4713 BC to 294276 AD	1 microsecond / 14 digits
Date	4bytes	4713 BC to 294276 AD	One day
Time without time zone	8 bytes	00:00:00 to 24:00:00	1 microsecond /14 digits
Timewith timezone	12 bytes	00:00:00 + 1459 to 24:00:00-1459	I microsecond/ 14 digits
Interval	12bytes	-178000000 to 178000000 years	I microsecond/14 digits

Geometric data types :

The geometric data types represent two-dimensional data objects. Moreover these data types help to perform a various operation like rotation, scaling and translation etc.

Name	Storage	Representation	Description
Point	16 bytes	Point on a plane	(x,y)
Line	32 bytes	Iinfinte line	((x1.y1).(x2.y2))
Line segment	32 bytes	Finite line segment	((x1.y1).(x2.y2))
Box	32 bytes	Rectangular box	((x1.y1).(x2.y2))
Path	16 n + 16 n bytes	Close and open path	((x1.y1),…..)
Polygon	40 + 16 n bytes	Polygon	[(x1.y1)…]
Circle	24 bytes	Circle	<(x.y).r>

Network Address type:

Today we do have many network address types like IPV4, IP V6, and MAC address. Moreover, this PostgreSQL offers different data types to store in these address.

To store in network address, instead of plain text, it is preferable to store these data types.

Name	Storage size	Description
Cidr	7 (or) 19 bytes	IPV4 and IPV6 networks
Inet	7 (or) 19 bytes	IPV4 and IPV 6 hosts and networks
Macaddr	6 bytes	Mac address

Enumerated data types:

In PostgreSQL, enumerated data types are useful for representing the rarely changing information such as country code (or) branch id. so to ensure data integrity, the enumerated data type is represented with a table with a foreign key.

Range type :

These represent data that uses the data range. These range types can be a discrete range (or) continuous ranges. Postgre SQL contains the following built-in data types.

Name	Description
Int4range	Range of the integer
Int8range	Range of the big int
Numrange	Range of the numeric
tsrange	Without timezone, time stamp range
tstzrange	With timezone, time stamp range
Date range	It represents the range of the date

Pseudo types :

PostgreSQL contains a number of special purpose entries that are collectively called as pseudo types. Moreover , a pseudo data type cannot be used as a column data type that can be used as a column data type. Additionally, this can be used to declare a function argument (or) a result type.

Name	Description
Any	It indicates the function accepts any input data type
Any element	It indicates the function accepts any data type
Any array	It indicates the function accepts any array data type
Anynonarray	It indicates the function accepts any non-array data type
Anyenum	It indicates the function accepts any enum data type
Anyrange	It indicates the function accepts any range data type.
Internal	It indicates the function accepts (or) returns an internal server data.
Record	It indicates the function returning an unspecified row type
Void	It indicates the function that has no value
CString	It indicates the function accepts (or) returns the Null-terminated C string

21. Crud Operations

So till now, we have how to connect the database in your systems. Now its time to connect with the Pycharm, let’s discuss about crud operations in below.

Till now, we have used the Community version of the PyCharm. This works best for the programs till we have used. But if we go into the advanced step like the connection with databases this won’t work. So, to perform this kind of operation, we need to use the commercial version of the PyCharm. And this commercial version has a trial period for some days. So to explain to you the examples, I have used the trial version of Professional Pycharm.

Note :

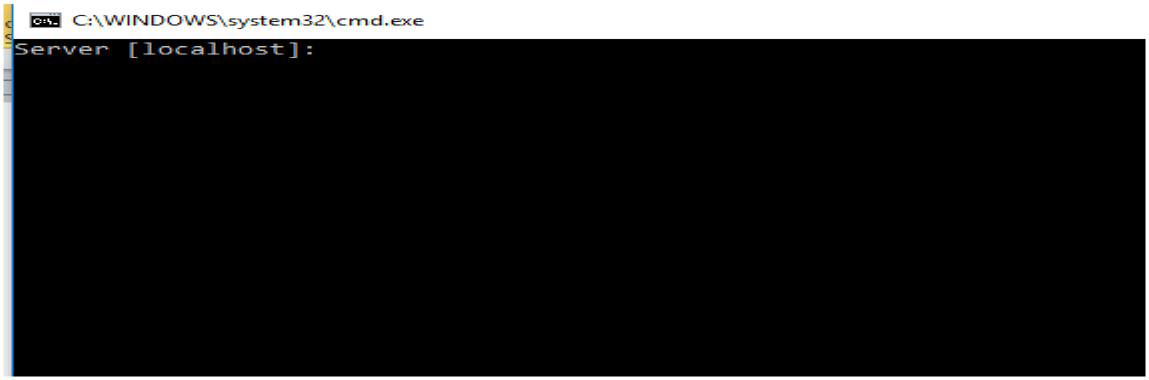
The Professional version of Pycharm is the same as the commercial Version of Python. If you are still unable to install in your systems, feel free to contact python online training**.**

Hope you have successfully created the database in your system, now its time to create Databases and perform **crud** operations.

So now initially look at how to connect database. Since you have installed the database, now, I would like to know how to open the database. And now open the script at the following location.

C:\Program Files\PostgreSQL\11\scripts\runpsql

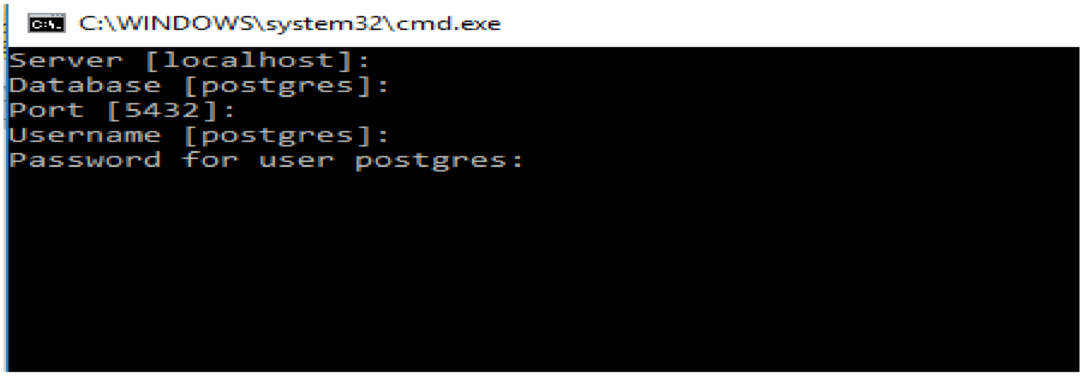
Once you went through the location and have opened the exe, then you would have seen the following screen.



```
Server [localhost]:
```

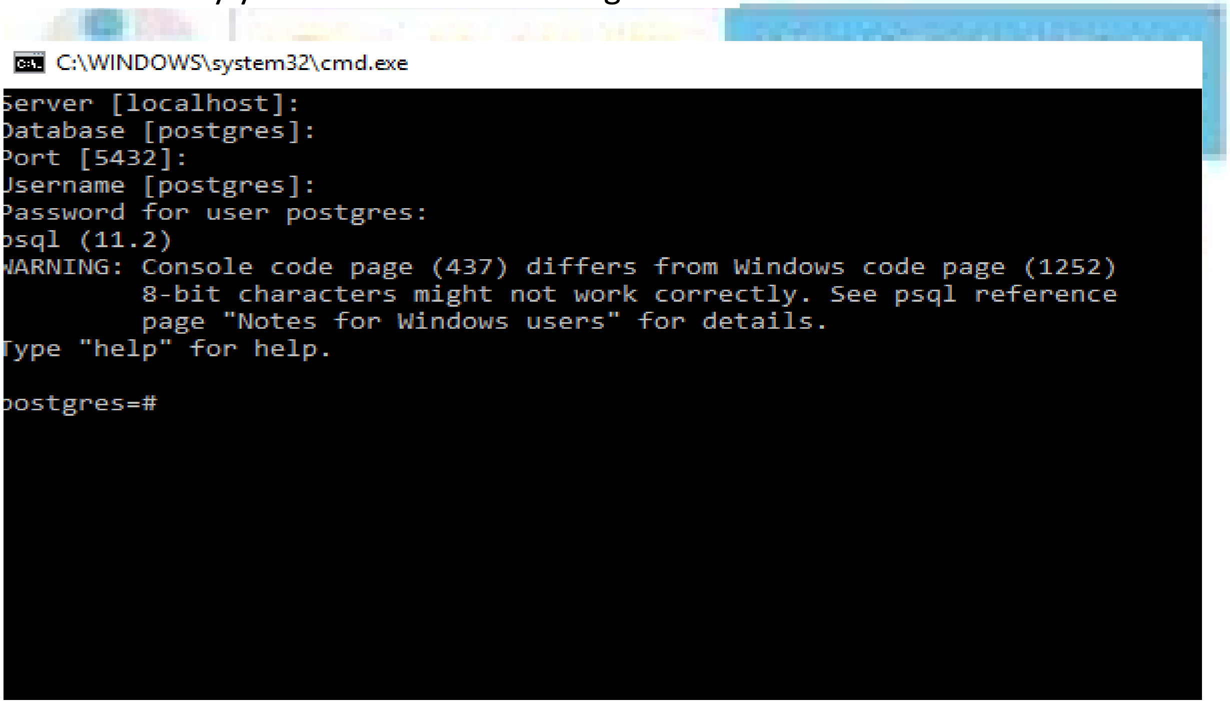
So after seeing this, there is nothing to give at this moment. Just only press **Enter.** for 4 times.

Soon after that, you will see the following screen.



```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
```

So its time to provide the password that you have provided at the installation time. So finally you will see the following screen.



```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (11.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

We will create the database now.

Let us first create the function

CREATE DATABASE database_name[description];

Ex: CREATE DATABASE ONLINEITGURU;

The create Database consists of the following parameters.

S.no	Dbname	Parameter
1	Dbname	The database name to create
2	Description	It specifies a comment to be associated with the newly created database.
3	Options	There are the command line arguments that accepts the database

The create DB consists of the following commands:

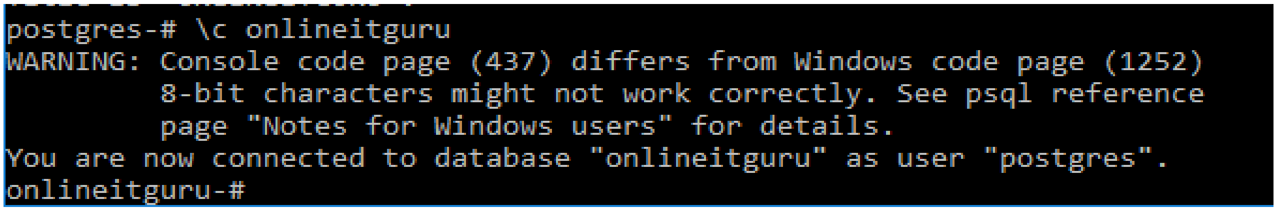
Option	Description
-d tablespace	It specifies the default tablespace for the database
-e	It echos the command created generates and sends to the server
-E encoding	It specifies the character encoding scheme in the database
-U username	It specifies the username to connected
-w	It never issues a password prompt
-W	It forces the database creation to prompt for the password prior to the database connection
-p port	It specifies the TCP port (or) local Unix domain socket extension. This is used for listening to the server connections
-h host	It specifies the machine name on which the server is running
-l locale	It specifies the locale used in the database

-T template	It specifies the template database
-h help	It helps the create db arguments and exit

Since we have created the database, it time to use this database. We can easily move to the database through the following command.

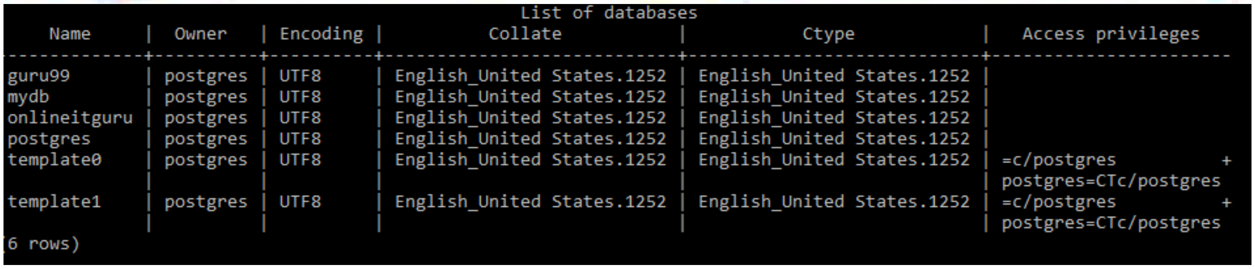
Syn : \c DATABASE_NAME

Ex: \c onlineitguru



```
postgres-# \c onlineitguru
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
You are now connected to database "onlineitguru" as user "postgres".
onlineitguru-#
```

So Since we have moved to the required database, its time to display the database list. This can be done using the **\l** command. Here I'm showing my database list tables



```
                                        List of databases
    Name      |  Owner   | Encoding |          Collate           |           Ctype            |   Access privileges
--------------+----------+----------+----------------------------+----------------------------+-----------------------
 guru99       | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 mydb         | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 onlineitguru | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 postgres     | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 template0    | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
              |          |          |                            |                            | postgres=CTc/postgres
 template1    | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
              |          |          |                            |                            | postgres=CTc/postgres
(6 rows)
```

Once again give the command \c database_ name(Ex: \c onlineitguru) to move into the desired database. (if required)

Hope you are in the desired database. Now its time to create a table in the desired database. (In my case, I have gone through OnlineITGuru). Now you can easily create a table using the following name**.**

Syn :

Create table table_name(var_name1 datatype constraint, var_name2 datatype constraint ——————-);

Ex:

Create table emp_list(id INT, name text, address char(50), salary REAL);

Since we have created the table in the database, it time to insert data into the database. We can create the database using the following command.

Syntax:

Insert into table_name(‘variable_name1, variable_name1 ,variable_name1, variable_name1)

Note : Use single cotes (‘’)for the string values. But this is not necessary for the integers.

Ex:

Insert into emp_list values (10,’balajee’,’kphb’ ,25000);

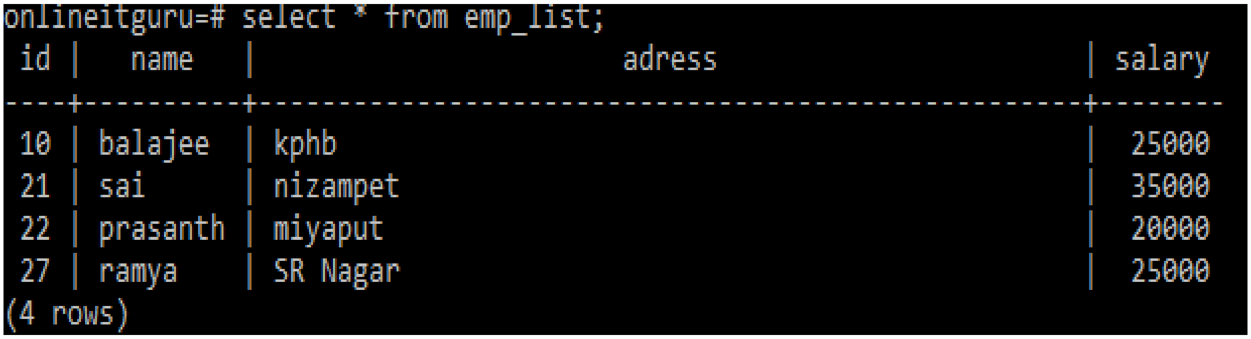
Insert into emp_list values(21,’sai’,’nizampet’,35000);

Insertinto emp_list values(22,’prasanth’,’miyapur’,20000);

Insert intoemp_list values(27,’ramya’,’sr nagar’,25000);

So now the table consists of 4 rows. Now you can get the output of 4 rows as below

Query: select * from emp_list;

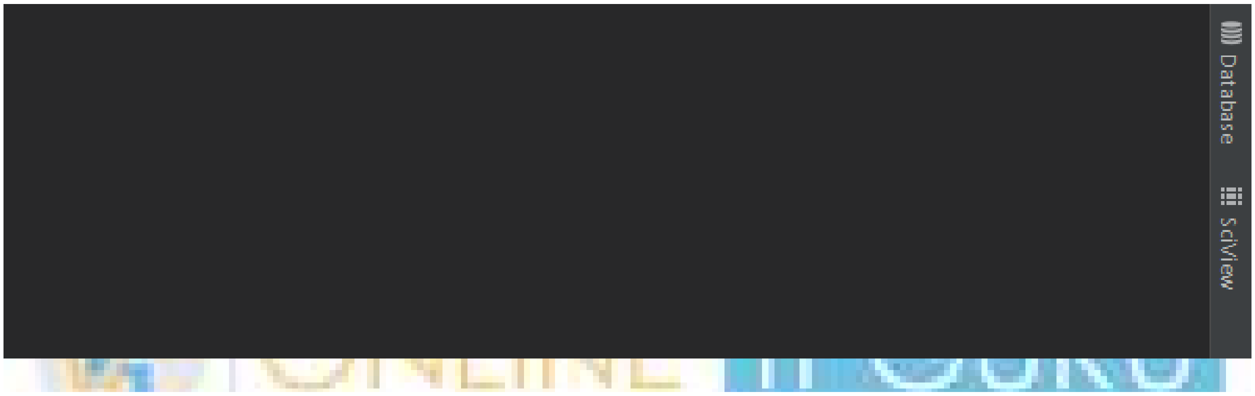


```
onlineitguru=# select * from emp_list;
 id |   name   |                       adress                       | salary
----+----------+----------------------------------------------------+--------
 10 | balajee  | kphb                                               |  25000
 21 | sai      | nizampet                                           |  35000
 22 | prasanth | miyaput                                            |  20000
 27 | ramya    | SR Nagar                                           |  25000
(4 rows)
```

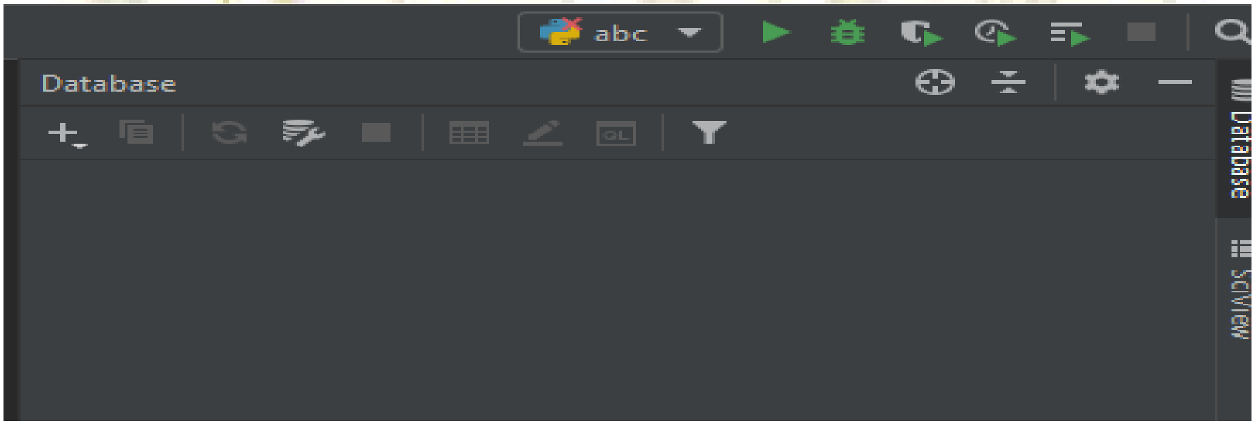
Since rows were created now manually using database , now its time to create more using python code

Since we have downloaded the professional version of Pycharm, now its time to connect to the database. The database connection in pycharm consists of the following steps :

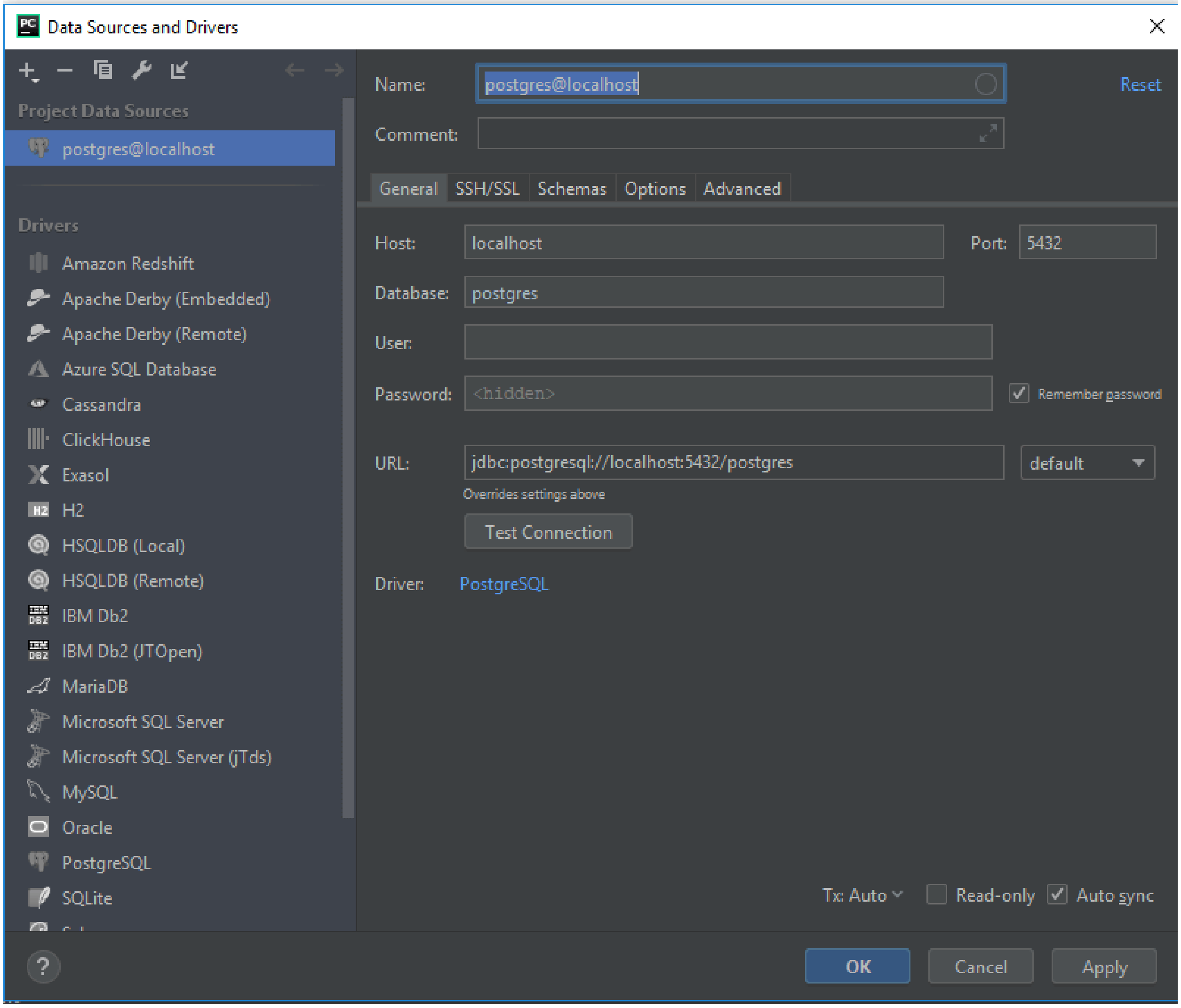
When you have opened the pycharm, on the right corner, you will be finding the **database**. Just click over it.



Once you clicked over the database, you will be finding the **plus symbol**over it.



When you have mouse over the '+' you will be finding database over it and select your desired database. In my case, I'm selecting the PostgreSQL database. When selected, you will be taken to the following screen.



Name: postgres@localhost

Comment:

Host: localhost
Port: 5432

Database: postgres

User:

Password: <hidden> ☑ Remember password

URL: jdbc:postgresql://localhost:5432/postgres default

Overrides settings above

Driver: PostgreSQL

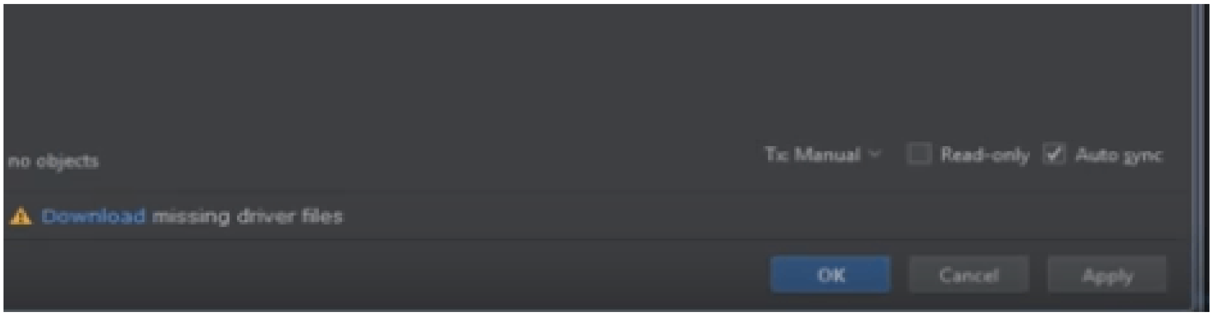
Tx: Auto ☐ Read-only ☑ Auto sync

So here you need to provide your database name, **username** as well as the **password**. (these names were given by the user at the time of database installation.

When provided, you will be asking for the Pycharm to test the connection. (We can proceed further when the test is successful).

Note :

In prior to the test connection, you will be seeing the option like Download the missing driver files. Just click on it to **download missing driver files**. (We cannot see this step in all systems. It may (or) may not be).

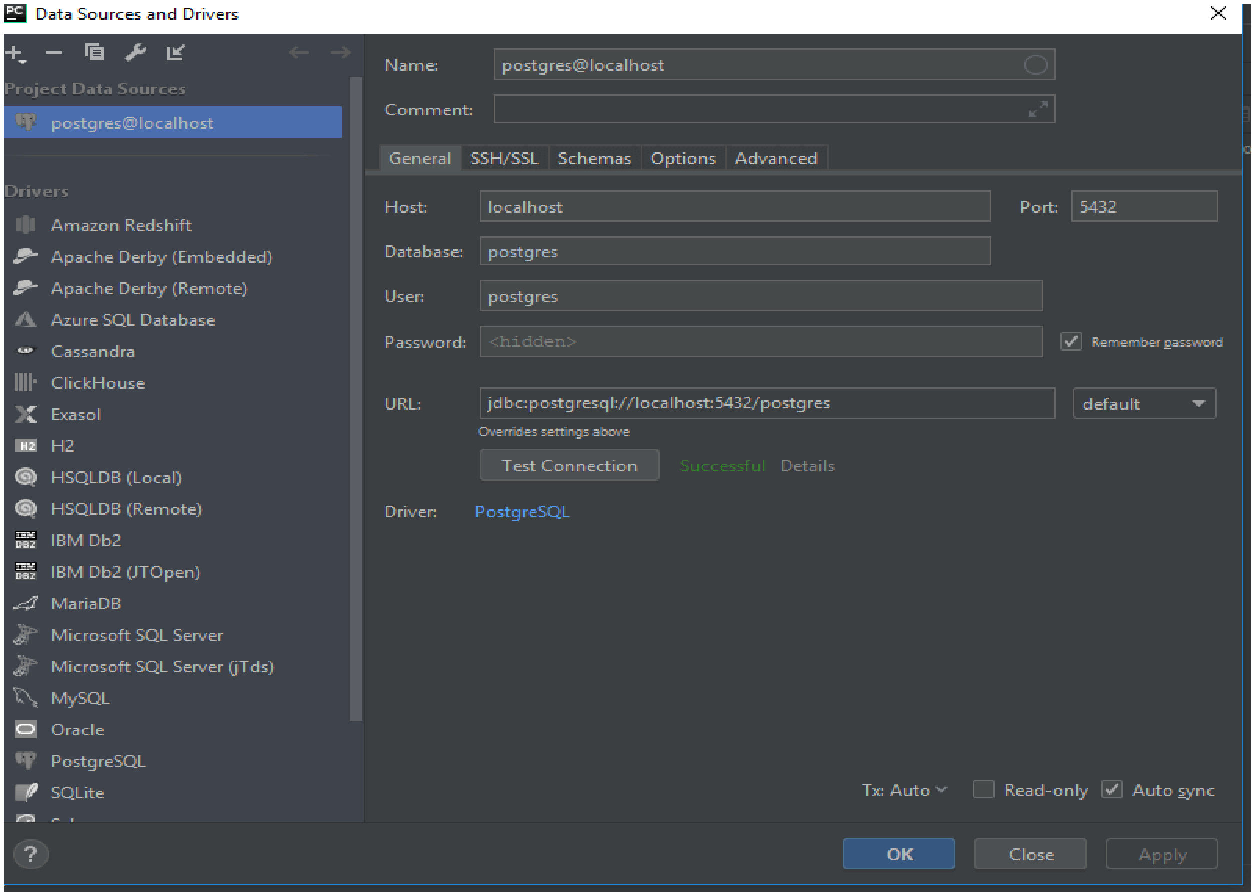


no objects

Tx: Manual ☐ Read-only ☑ Auto sync

Download missing driver files

So, I hope you people have given all the necessary details. Now its time to **test connection**. You can do this by just clicking the Test button. So soon after the **TEST** completion, you will be seen as **Test is successful**as shown below.



Name: postgres@localhost

Comment:

Host: localhost Port: 5432

Database: postgres

User: postgres

Password: <hidden> ☑ Remember password

URL: jdbc:postgresql://localhost:5432/postgres default

Overrides settings above

Test Connection Successful Details

Driver: PostgreSQL

Tx: Auto ☐ Read-only ☑ Auto sync

So till now, we have configured the data using pycharm. Now its time run the sample program

But before we were going to run the sample program, we need to configure the psycopg2 in our system. You can do this in the Command prompt (CMD).

Go to your windows search bar and type CMD your system. So soon after opening your cmd prompt, you must give the command as shown below.

Cmd :

```
Pip install psycopg2
```

So once you have given the command, the required package would be installed in your system and its time and its tun the py file. So lets have a look at the code .

```
import psycopg2

try:

  connection = psycopg2.connect(user=”postgres”,

                              password=”mydatabase”,

                              host=”127.0.0.1”,

                              port=”5432”,

                              database=”onlineitguru”)

  cursor = connection.cursor()

  postgres_insert_query = “”” INSERT INTO emp (id,name,adress,salary) VALUES (%s,%s,%s)”””

  record_to_insert = (5, ‘sampu’, ‘one town’, 95000)

  cursor.execute(postgres_insert_query, record_to_insert)

  connection.commit()

  count = cursor.rowcount
```

```
print (count, “Record inserted successfully into emp table”)
except (Exception, psycopg2.Error) as error :
  if(connection):
     print(“Failed to insert record into mobile table”, error)
finally:
  #closing database connection.
  if(connection):
     cursor.close()
     connection. Close()
     print(“PostgreSQL connection is closed”)
```

So soon after the execution, you can see the effect of databases in the table.

Here I have given only the basic connection of the database through python. Likewise, by changing the commands, you can do any crud operations.

But I have given all the database operation that PostgreSQL and you can perform the same using Python. (from .py file)

Now lets move to other database operations

So in the above example, for emp id “22”, we have given the address as miyapur which used to Miyapur. So we need to update the address of the prasanth.

This can be done using the following command.

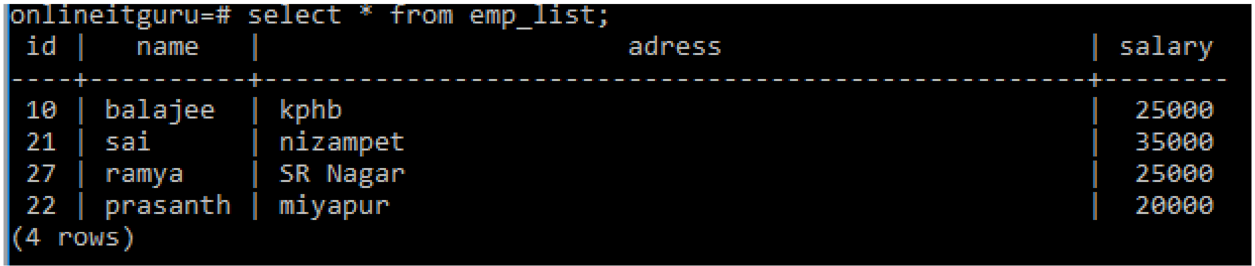
Syntax:

update table_name set column =value where condition;

Ex:

update emp_1 set address='miyapur' where id =22;

So now if you see the table we can update the address of prasanth. So now have a look over the table once again.



```
onlineitguru=# select * from emp_list;
 id |   name   |                      adress                       | salary
----+----------+---------------------------------------------------+--------
 10 | balajee  | kphb                                              |  25000
 21 | sai      | nizampet                                          |  35000
 27 | ramya    | SR Nagar                                          |  25000
 22 | prasanth | miyapur                                           |  20000
(4 rows)
```

Deleting the Query:

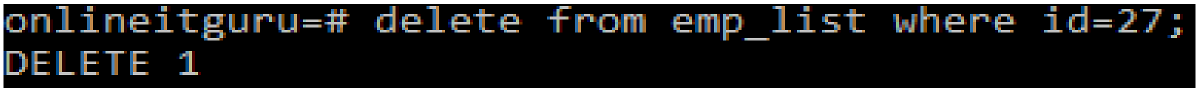
Now its time to delete the row from the table. Now I would like to delete Ramya from the table

Syn :

delete from table_name where condition;

ex:

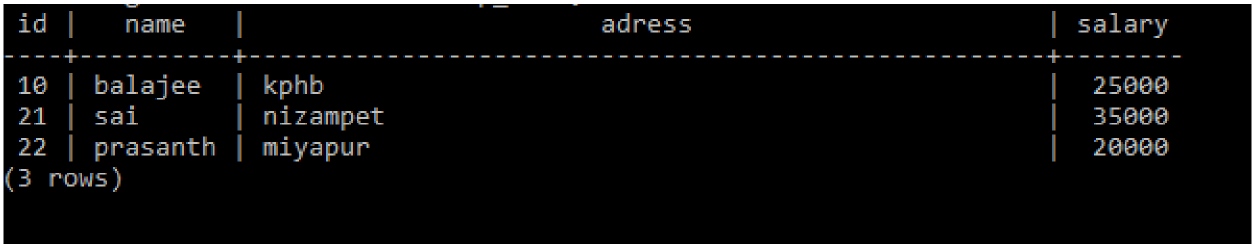
delete from emp_list where id=27;



```
onlineitguru=# delete from emp_list where id=27;
DELETE 1
```

Now we will verify the table

Select * from emp_list;



```
 id |   name   |                    adress                    | salary
----+----------+----------------------------------------------+--------
 10 | balajee  | kphb                                         |  25000
 21 | sai      | nizampet                                     |  35000
 22 | prasanth | miyapur                                      |  20000
(3 rows)
```

Great! The row successfully deleted!!!

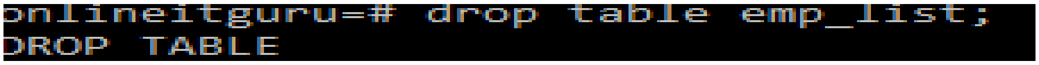
Deleting the table:

Since we have deleted one row from the table, now I would like to delete the table;

Syntax:

drop table table_name;

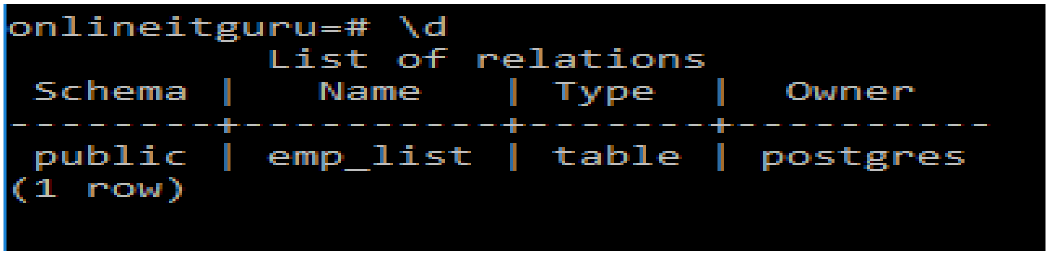
Ex: drop table emp_list;



```
onlineitguru=# drop table emp_list;
DROP TABLE
```

Now we will select the tables in our database;

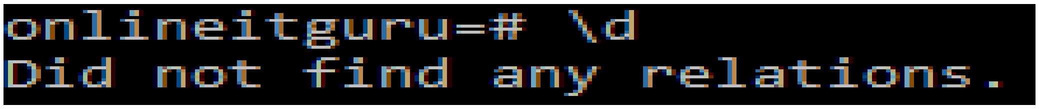
Syn: \d



```
onlineitguru=# \d
          List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | emp_list | table | postgres
(1 row)
```

Now its time to check the tables in the database;

Ex: \d

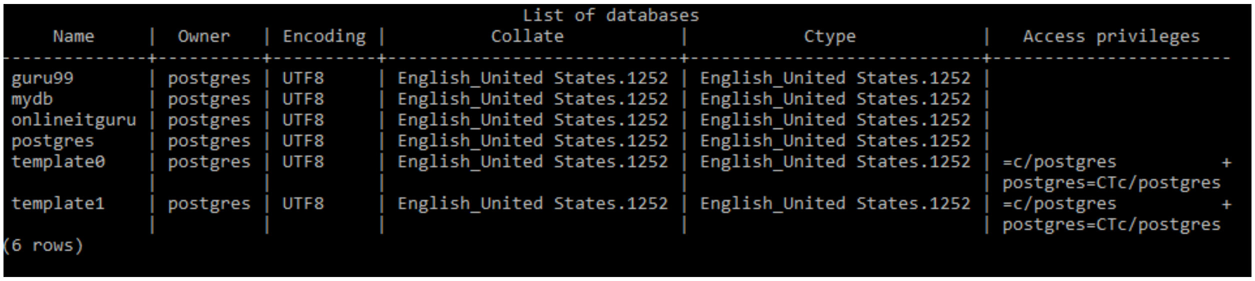


```
onlineitguru=# \d
Did not find any relations.
```

And the last operation that we need to discuss is about dropping the database.

Dropping the database:

Dropping the database is nothing but deleting the database. Now, before deleting the database, let us first have a look over the databases we have. This can be done using the **\l** command



```
                                                  List of databases
    Name      |  Owner   | Encoding |          Collate           |           Ctype            |   Access privileges
--------------+----------+----------+----------------------------+----------------------------+-----------------------
 guru99       | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 mydb         | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 onlineitguru | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 postgres     | postgres | UTF8     | English_United States.1252 | English_United States.1252 |
 template0    | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
              |          |          |                            |                            | postgres=CTc/postgres
 template1    | postgres | UTF8     | English_United States.1252 | English_United States.1252 | =c/postgres          +
              |          |          |                            |                            | postgres=CTc/postgres
(6 rows)
```

Now I would like to drop the database named onlineitguru.

Syn: drop database database_name;

drop database guru99;

Now its time to chek all the databases

Ex: \l

Recommended audience:

This Python tutorial is recommended for both software professional and Graduates, who were interested to learn python programming and master in python coding

Prerequisites:

People who have Python installation set (Either on Windows, Mac, Linux) can start learning and Practice python programming.

THANK YOU……