**PYTHON**

1. What are lists and tuples? What is the key difference between the two?
2. What is break, continue and pass in Python?
3. How is memory managed in Python?
4. What are decorators in Python?
5. How do you identify a string from a book that matches a pattern using Python?
6. Identify items that are uncommon/common between two series?
7. How do you perform unit testing in Python?
8. What is a dataframe and explain few operations using Python?
9. **LISTS AND TUPLES**

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets

thislist = ["apple", "banana", "cherry"]

List items are ordered, changeable, and allow duplicate values.  
List items are indexed, the first item has index [0], the second item has index [1] etc.

Similar to Lists,

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

Key Differences:   
Lists are Changeable, whereas Tuples are Unchangeale

Lists have Square brackets, Tuples have Round brackets

Lists have11 methods, whereas Tuples only have 2

LIST METHODS

append() Adds an element at the end of the list

clear() Removes all the elements from the list

copy() Returns a copy of the list

count() Returns the number of elements with the specified value

extend() Add the elements of a list (or any iterable), to the end of the current list

index() Returns the index of the first element with the specified value

insert() Adds an element at the specified position

pop() Removes the element at the specified position

remove() Removes the item with the specified value

reverse() Reverses the order of the list

sort() Sorts the list

TUPLE METHODS

count() Returns the number of elements with the specified value

index() Returns the index of the first element with the specified value

1. **break, continue and pass**

Using loops in Python automates and repeats the tasks in an efficient manner. But sometimes, there may arise a condition where you want to exit the loop completely, skip an iteration or ignore that condition. These can be done by **loop control statements**. Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

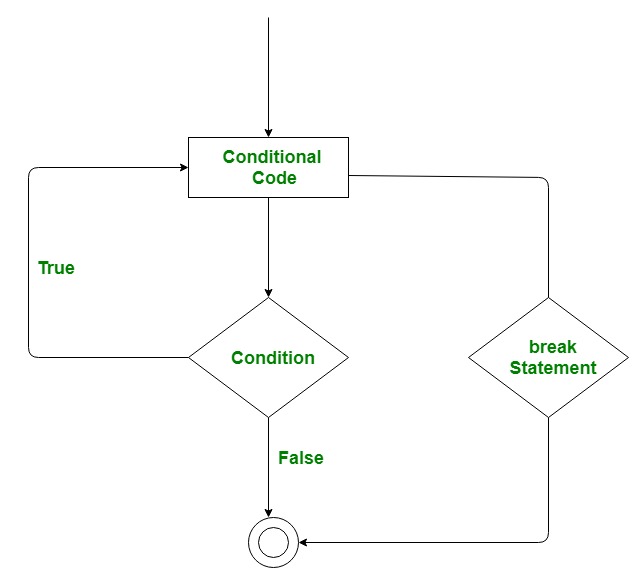
* [Break statement](https://www.geeksforgeeks.org/break-continue-and-pass-in-python/#break)
* [Continue statement](https://www.geeksforgeeks.org/break-continue-and-pass-in-python/#continue)
* [Pass statement](https://www.geeksforgeeks.org/break-continue-and-pass-in-python/#pass)

## Break statement

The break statement is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available. If the break statement is present in the nested loop, then it terminates only those loops which contains break statement.

**Syntax:**

break



**Example:**  
Consider a situation where you want to iterate over a string and want to print all the characters until a letter ‘e’ or ‘s’ is encountered. It is specified that you have to do this using loop and only one loop is allowed to use.  
Here comes the usage of break statement. What we can do is iterate over a string using either a while loop or for loop and every time we have to compare the value of iterator with ‘e’ or ‘s’. If it is ‘e’ or ‘s’ we will use the break statement to exit the loop.

Below is the implementation.

|  |
| --- |
| # Python program to demonstrate  # break statement    # Python program to  # demonstrate break statement    s = 'geeksforgeeks'  # Using for loop  for letter in s:        print(letter)      # break the loop as soon it sees 'e'      # or 's'      if letter == 'e' or letter == 's':          break    print("Out of for loop")  print()    i = 0    # Using while loop  while True:      print(s[i])        # break the loop as soon it sees 'e'      # or 's'      if s[i] == 'e' or s[i] == 's':          break      i += 1    print("Out of while loop") |

**Output:**

g

e

Out of for loop

g

e

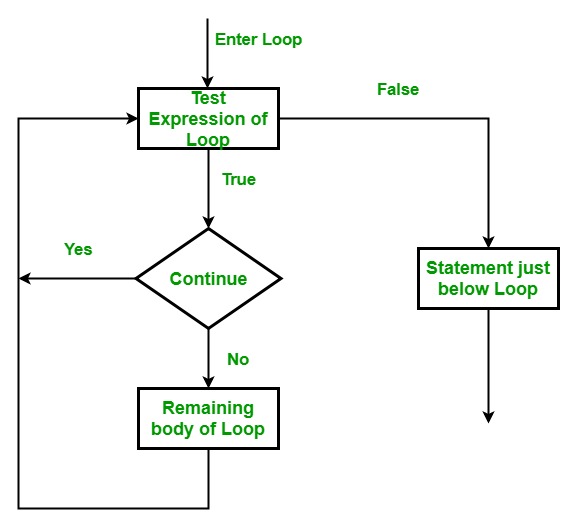
Out of while loop

## Continue statement

Continue is also a loop control statement just like the break statement. continue statement is opposite to that of break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.  
As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

**Syntax:**

continue



**Example:**  
Consider the situation when you need to write a program which prints the number from 1 to 10 and but not 6. It is specified that you have to do this using loop and only one loop is allowed to use.  
Here comes the usage of continue statement. What we can do here is we can run a loop from 1 to 10 and every time we have to compare the value of iterator with 6. If it is equal to 6 we will use the continue statement to continue to next iteration without printing anything otherwise we will print the value.

Below is the implementation of the above idea:

|  |
| --- |
| # Python program to  # demonstrate continue  # statement    # loop from 1 to 10  for i in range(1, 11):        # If i is equals to 6,      # continue to next iteration      # without printing      if i == 6:          continue      else:          # otherwise print the value          # of i          print(i, end = " ") |

**Output:**

1 2 3 4 5 7 8 9 10

## Pass statement

As the name suggests pass statement simply does nothing. The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like null operation, as nothing will happen is it is executed. Pass statement can also be used for writing empty loops. Pass is also used for empty control statement, function and classes.

**Syntax:**

pass

**Example:**

|  |
| --- |
| # Python program to demonstrate  # pass statement    s = "geeks"    # Empty loop  for i in s:      # No error will be raised      pass    # Empty function  def fun():      pass    # No error will be raised  fun()    # Pass statement  for i in s:      if i == 'k':          print('Pass executed')          pass      print(i) |

**Output:**

g

e

e

Pass executed

k

s

In the above example, when the value of i becomes equal to ‘k’, the pass statement did nothing and hence the letter ‘k’ is also printed.

1. **MEMORY MANAGEMENT**

Understanding Memory allocation is important to any software developer as writing efficient code means writing a memory-efficient code. Memory allocation can be defined as allocating a block of space in the computer memory to a program. In Python memory allocation and deallocation method is automatic as the Python developers created a garbage collector for Python so that the user does not have to do manual garbage collection.

**Garbage Collection**

Garbage collection is a process in which the interpreter frees up the memory when not in use to make it available for other objects.

Assume a case where no reference is pointing to an object in memory i.e. it is not in use so, the virtual machine has a garbage collector that automatically deletes that object from the heap memory

Note: For more information, refer to Garbage Collection in Python

**Reference Counting**

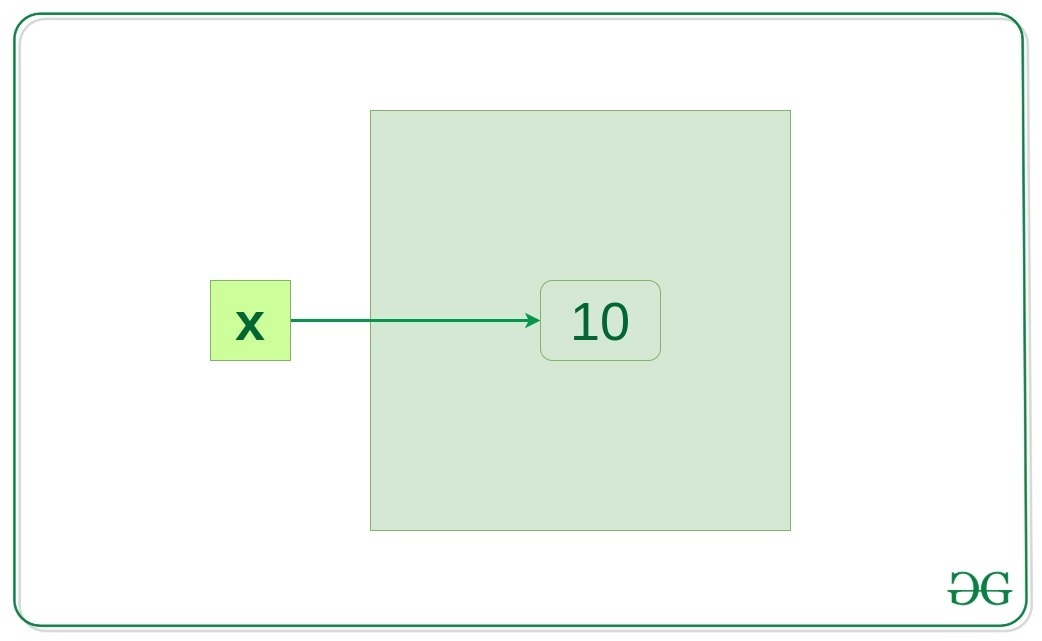
Reference counting works by counting the number of times an object is referenced by other objects in the system. When references to an object are removed, the reference count for an object is decremented. When the reference count becomes zero, the object is deallocated.

For example, Let’s suppose there are two or more variables that have the same value, so, what Python virtual machine does is, rather than creating another object of the same value in the private heap, it actually makes the second variable point to that originally existing value in the private heap. Therefore, in the case of classes, having a number of references may occupy a large amount of space in the memory, in such a case referencing counting is highly beneficial to preserve the memory to be available for other objects

Example:

x = 10

When x = 10 is executed an integer object 10 is created in memory and its reference is assigned to variable x, this is because everything is object in Python.



Let’s verify if it’s true

x = 10

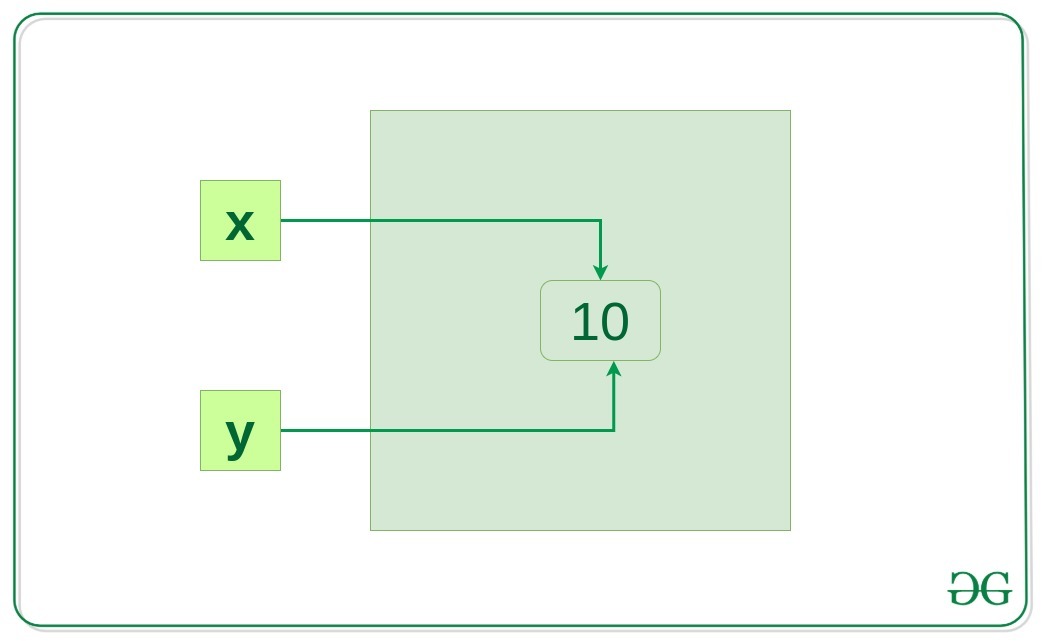
y = x

if id(x) == id(y):

print("x and y refer to the same object")

Output:  
x and y refer to the same object

In the above example, y = x will create another reference variable y which will refer to the same object because Python optimizes memory utilization by allocation the same object reference to a new variable if the object already exists with the same value.



Now, let’s change the value of x and see what happens.

x = 10

y = x

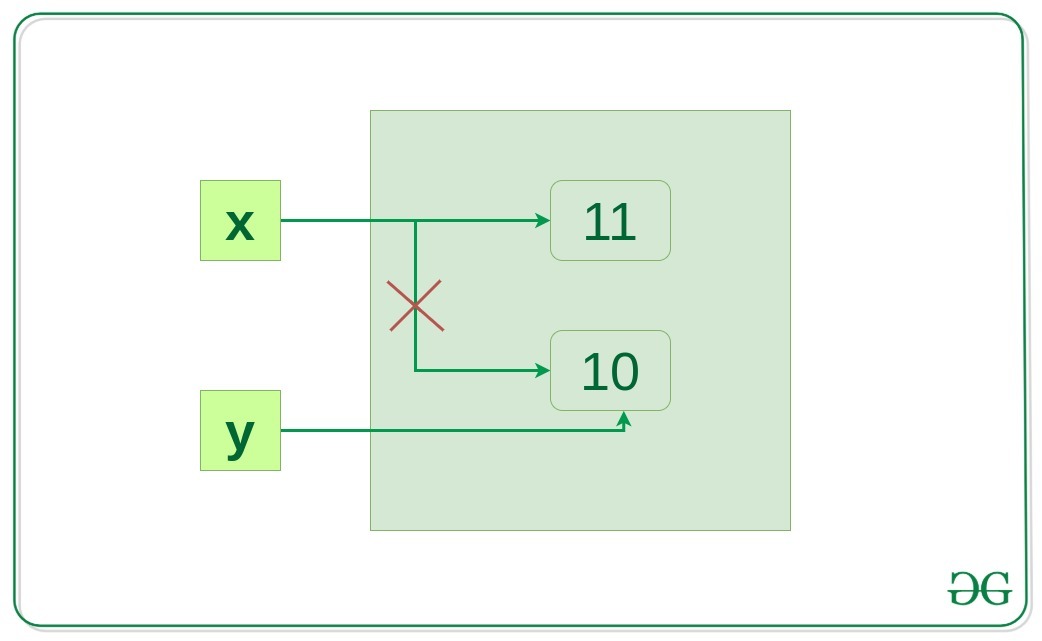
x += 1

if id(x) != id(y):

print("x and y do not refer to the same object")

Output:  
x and y do not refer to the same object

So now x refer to a new object x and the link between x and 10 disconnected but y still refer to 10.

  
 MEMORY ALLOCATION IN PYTHON

There are two parts of memory:

* stack memory
* heap memory

The methods/method calls and the references are stored in stack memory and all the values objects are stored in a private heap.

**Work of Stack Memory**

The allocation happens on contiguous blocks of memory. We call it stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack.

It is the memory that is only needed inside a particular function or method call. When a function is called, it is added onto the program’s call stack. Any local memory assignments such as variable initializations inside the particular functions are stored temporarily on the function call stack, where it is deleted once the function returns, and the call stack moves on to the next task. This allocation onto a contiguous block of memory is handled by the compiler using predefined routines, and developers do not need to worry about it.

Example:

def func():

# All these variables get memory

# allocated on stack

a = 20

b = []

c = ""

**Work of Heap Memory**

The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate. The variables are needed outside of method or function calls or are shared within multiple functions globally are stored in Heap memory.

Example:

# This memory for 10 integers

# is allocated on heap.

a = [0]\*10

1. **DECORATORS**

 the decorators are used to modify the behaviour of function or class. In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

**Syntax for Decorator:**

@gfg\_decorator

def hello\_decorator():

print("Gfg")

'''Above code is equivalent to -

def hello\_decorator():

print("Gfg")

hello\_decorator = gfg\_decorator(hello\_decorator)'''

In the above code, gfg\_decorator is a callable function, that will add some code on the top of some another callable function, hello\_decorator function and return the wrapper function.

Decorator can modify thebehaviour**:**

# defining a decorator

def hello\_decorator(func):

# inner1 is a Wrapper function in

# which the argument is called

# inner function can access the outer local

# functions like in this case "func"

def inner1():

print("Hello, this is before function execution")

# calling the actual function now

# inside the wrapper function.

func()

print("This is after function execution")

return inner1

# defining a function, to be called inside wrapper

def function\_to\_be\_used():

print("This is inside the function !!")

# passing 'function\_to\_be\_used' inside the

# decorator to control its behaviour

function\_to\_be\_used = hello\_decorator(function\_to\_be\_used)

# calling the function

function\_to\_be\_used()

**OUTPUT**

Hello, this is before function execution

This is inside the function !!

This is after function execution

**CHAINED DECORATORS**

# code for testing decorator chaining

def decor1(func):

def inner():

x = func()

return x \* x

return inner

def decor(func):

def inner():

x = func()

return 2 \* x

return inner

@decor1

@decor

def num():

return 10

print(num())

**OUTPUT**

400

1. **IDENTIFYING A STRING FROM A BOOK**

Pattern matching in Python with Regex:

You may be familiar with searching for text by pressing ctrl-F and typing in the words you’re looking for. Regular expressions go one step further: They allow you to specify a pattern of text to search for.  
Regular expressions, called regexes for short, are descriptions for a pattern of text. For example, a \d in a regex stands for a digit character — that is, any single numeral 0 to 9.

* Following regex is used in Python to match a string of three numbers, a hyphen, three more numbers, another hyphen, and four numbers.
* Any other string would not match the pattern.

\d\d\d-\d\d\d-\d\d\d\d

* Regular expressions can be much more sophisticated. For example, adding a 3 in curly brackets ({3}) after a pattern is like saying, “ Match this pattern three times.” So the slightly shorter regex

\d{3}-\d{3}-\d{4}

(It matches the correct phone number format.)

**Creating Regex object**

All the regex functions in Python are in the re module

import re

To create a Regex object that matches the phone number pattern, enter the following into the interactive shell.

phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

Now the phoneNumRegex variable contains a Regex object.

**Matching regex objects**

A Regex object’s search() method searches the string it is passed for any matches to the regex. Match objects have a group() method that will return the actual matched text from the searched string.

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  import re  phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')  mo = phoneNumRegex.search('My number is 415-555-4242.')  print('Phone number found: ' + mo.group()) |

Output:

Phone number found: 415-555-4242

**Steps of Regular Expression Matching**

While there are several steps to using regular expressions in Python, each step is fairly simple.

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)
3. Pass the string you want to search into the Regex object’s search() method. This returns a Match object.
4. Call the Match object’s group() method to return a string of the actual matched text.

**Grouping with parentheses**

**Matching objects:**Say you want to separate the area code from the rest of the phone number. Adding parentheses will create groups in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text from just one group.

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  # with grouping  import re  phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')  mo = phoneNumRegex.search('My number is 415-555-4242.')  print(mo.group(1)) |

OUTPUT:

'415'

**Retrieve all the groups at once :**If you would like to retrieve all the groups at once, use the groups(), method—note the plural form for the name.

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  # with groups  import re  phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')  mo = phoneNumRegex.search('My number is 415-555-4242.')  print(mo.groups()) |

OUTPUT:

('415', '555-4242')

**Using mo.groups :** mo.groups() will return a tuple of multiple values, you can use the multiple-assignment trick to assign each value to a separate variable, as in the following areaCode, mainNumber = mo.groups() line.

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  # with mo.groups()  import re  phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')  mo = phoneNumRegex.search('My number is 415-555-4242.')  areaCode, mainNumber = mo.groups()  print(mainNumber) |

OUTPUT:

'555-4242'

**Match a parenthesis :** Parentheses have a special meaning in regular expressions, but what do you do if you need to match a parenthesis in your text. For instance, maybe the phone numbers you are trying to match have the area code set in parentheses. In this case, you need to escape the ( and ) characters with a backslash. Enter the following into the interactive shell:

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  # with grouping  import re  phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')  mo = phoneNumRegex.search('My phone number is (415) 555-4242.')  print(mo.group(1)) |

OUTPUT:

'(415)'

The \( and \) escape characters in the raw string passed to re.compile() will match actual parenthesis characters.

**Matching Multiple Groups with the Pipe**

The | character is called a pipe. You can use it anywhere you want to match one of many expressions. For example, the regular expression r’Batman|Tina Fey’ will match either ‘Batman’ or ‘Tina Fey’.

When both Batman and Tina Fey occur in the searched string, the first occurrence of matching text will be returned as the Match object. Enter the following into the interactive shell:

|  |
| --- |
| # Python program to illustrate  # Matching regex objects  # with multiple Groups with the Pipe  import re  heroRegex = re.compile (r'Batman|Tina Fey')  mo1 = heroRegex.search('Batman and Tina Fey.')  print(mo1.group()) |

OUTPUT:

'Batman'

**Matching Specific Repetitions with Curly Brackets**

If you have a group that you want to repeat a specific number of times, follow the group in your regex with a number in curly brackets. For example, the regex (Ha){3} will match the string ‘HaHaHa’, but it will not match ‘HaHa’, since the latter has only two repeats of the (Ha) group.

Instead of one number, you can specify a range by writing a minimum, a comma, and a maximum in between the curly brackets. For example, the regex (Ha){3, 5} will match ‘HaHaHa’, ‘HaHaHaHa’, and ‘HaHaHaHaHa’.

You can also leave out the first or second number in the curly brackets to leave the minimum or maximum unbounded. For example, (Ha){3, } will match three or more instances of the (Ha) group, while (Ha){, 5} will match zero to five instances. Curly brackets can help make your regular expressions shorter. These two regular expressions match identical patterns:

(Ha){3}

(Ha)(Ha)(Ha)

And these two regular expressions also match identical patterns:

(Ha){3, 5}

((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))

Enter the following into the interactive shell:

|  |
| --- |
| # Python program to illustrate  # Matching Specific Repetitions  # with Curly Brackets  import re  haRegex = re.compile(r'(Ha){3}')  mo1 = haRegex.search('HaHaHa')  print(mo1.group()) |

OUTPUT:

'HaHaHa'

|  |
| --- |
| # Python program to illustrate  # Matching Specific Repetitions  # with Curly Brackets  import re  haRegex = re.compile(r'(Ha){3}')  mo2 = haRegex.search('Ha')== None  print(mo2) |

OUTPUT:

True

Here, (Ha){3} matches ‘HaHaHa’ but not ‘Ha’. Since it doesn’t match ‘Ha’, search() returns None.

**Optional Matching with the Question Mark**

Sometimes there is a pattern that you want to match only optionally. That is, the regex should find a match whether or not that bit of text is there. The **?**character flags the group that precedes it as an optional part of the pattern. For example, enter the following into the interactive shell:

|  |
| --- |
| # Python program to illustrate  # optional matching  # with question mark(?)  import re  batRegex = re.compile(r'Bat(wo)?man')  mo1 = batRegex.search('The Adventures of Batman')  print(mo1.group()) |

OUTPUT:

'Batman'

|  |
| --- |
| # Python program to illustrate  # optional matching  # with question mark(?)  import re  batRegex = re.compile(r'Bat(wo)?man')  mo2 = batRegex.search('The Adventures of Batwoman')  print(mo2.group()) |

OUTPUT:

'Batwoman'

The (wo)? part of the regular expression means that the pattern wo is an optional group. The regex will match text that has zero instances or one instance of wo in it. This is why the regex matches both ‘Batwoman’ and ‘Batman’.  
You can think of the ? as saying,**“Match zero or one of the group preceding this question mark.”**  
If you need to match an actual question mark character, escape it with \?.

**Matching Zero or More with the Star**

The \* (called the star or asterisk) means “match zero or more”—the group that precedes the star can occur any number of times in the text. It can be completely absent or repeated over and over again. Let’s look at the Batman example again.

|  |
| --- |
| # Python program to illustrate  # matching a regular expression  # with asterisk(\*)  import re  batRegex = re.compile(r'Bat(wo)\*man')  mo1 = batRegex.search('The Adventures of Batman')  print(mo1.group()) |

OUTPUT:

'Batman'

|  |
| --- |
| #python program to illustrate  #matching a regular expression  #with asterisk(\*)  import re  batRegex = re.compile(r'Bat(wo)\*man')  mo2 = batRegex.search('The Adventures of Batwoman')  print(mo2.group()) |

OUTPUT:

'Batwoman'

|  |
| --- |
| # Python program to illustrate  # matching a regular expression  # with asterisk(\*)  import re  batRegex = re.compile(r'Bat(wo)\*man')  mo3 = batRegex.search('The Adventures of Batwowowowoman')  print(mo3.group()) |

OUTPUT:

'Batwowowowoman'

For ‘Batman’, the (wo)\* part of the regex matches zero instances of wo in the string; for ‘Batwoman’, the (wo)\* matches one instance of wo; and for ‘Batwowowowoman’, (wo)\* matches four instances of wo.

If you need to match an actual star character, prefix the star in the regular expression with a backslash, \\*.

**Matching One or More with the Plus**

While \* means “match zero or more, ” the + (or plus) means **“match one or more.”** Unlike the star, which does not require its group to appear in the matched string, the group preceding a plus must appear at least once. It is not optional. Enter the following into the interactive shell, and compare it with the star regexes in the previous section:

|  |
| --- |
| # Python program to illustrate  # matching a regular expression  # with plus(+)  import re  batRegex = re.compile(r'Bat(wo)+man')  mo1 = batRegex.search('The Adventures of Batwoman')  print(mo1.group()) |

OUTPUT:

'Batwoman'

|  |
| --- |
| # Python program to illustrate  # matching a regular expression  # with plus(+)  import re  batRegex = re.compile(r'Bat(wo)+man')  mo2 = batRegex.search('The Adventures of Batwowowowoman')  print(mo2.group()) |

OUTPUT:

'Batwowowowoman'

batRegex = re.compile(r’Bat(wo)+man’)

|  |
| --- |
| # Python program to illustrate  # matching a regular expression  # with plus(+)  import re  batRegex = re.compile(r'Bat(wo)+man')  mo3 = batRegex.search('The Adventures of Batman')== None  print(mo3) |

OUTPUT:

True

The regex Bat(wo)+man will not match the string ‘The Adventures of Batman’ because at least one wo is required by the plus sign.

If you need to match an actual plus sign character, prefix the plus sign with a backslash to escape it: \+.

1. **COMMON AND UNCOMMON BETWEEN SERIES**

Pandas is used in implementing series in Python

Pandas does not support specific methods to perform set operations. However, we can use the following formula to get unique items from both the sets :

**Algorithm :**

1. Import the Pandas and NumPy modules.
2. Create 2 Pandas Series.
3. Find the union of the series using the union1d() method.
4. Find the intersection of the series using the intersect1d() method.
5. Find the difference between the union and the intersection elements. Use the isin() method to get the boolean list of items present in both ‘union’ and ‘intersect’.
6. Print the result

|  |
| --- |
| # import the modules  import pandas as pd  import numpy as np    # create the series  ser1 = pd.Series([1, 2, 3, 4, 5])  ser2 = pd.Series([3, 4, 5, 6, 7])    # union of the series  union = pd.Series(np.union1d(ser1, ser2))    # intersection of the series  intersect = pd.Series(np.intersect1d(ser1, ser2))  **HERE WE OBTAINED THE COMMONS NOW TO FIND THE UNCOMMONS**  # uncommon elements in both the series  notcommonseries = union[~union.isin(intersect)]    # displaying the result  print(notcommonseries) |

**Output :**

1, 2, 6, 7

1. **UNIT TESTING IN PYTHON**

Unit testing is a technique in which particular module is tested to check by developer himself whether there are any errors. The primary focus of unit testing is test an individual unit of system to analyze, detect, and fix the errors.

Python provides the unittest module to test the unit of source code. The unittest plays an essential role when we are writing the huge code, and it provides the facility to check whether the output is correct or not.

Normally, we print the value and match it with the reference output or check the output manually.

This process takes lots of time. To overcome this problem, Python introduces the unittest module. We can also check the application's performance by using it.

We will learn how to create a basic test, finds the bugs, and execute it before the code delivers to the users.

**Unittest module**

The unittest is built into the Python standard library since 2.1. The best thing about the unittest, it comes with both a test framework and a test runner. There are few requirements of the unittest to write and execute the code.

The code must be written using the classes and functions.

The sequence of distinct assertion methods in the TestCase class apart from the built-in asserts statements.

Let's implement the above example using the unittest case:

import unittest

class TestingSum(unittest.TestCase):

def test\_sum(self):

self.assertEqual(sum([2, 3, 5]), 10, "It should be 10")

def test\_sum\_tuple(self):

self.assertEqual(sum((1, 3, 5)), 10, "It should be 10")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**OUTPUT**

.F

-

FAIL: test\_sum\_tuple (\_\_main\_\_.TestingSum)

--

Traceback (most recent call last):

File "<string>", line 11, in test\_sum\_tuple

AssertionError: 9 != 10 : It should be 10

----------------------------------------------------------------------

Ran 2 tests in 0.001s

FAILED (failures=1)

Traceback (most recent call last):

File "<string>", line 14, in <module>

File "/usr/lib/python3.8/unittest/main.py", line 101, in \_\_init\_\_

self.runTests()

File "/usr/lib/python3.8/unittest/main.py", line 273, in runTests

sys.exit(not self.result.wasSuccessful())

SystemExit: True

…………….

**Method:**  
White Box Testing method is used for Unit testing.

**OOP concepts supported by unittest framework:**

* **test fixture:**  
  A test fixture is used as a baseline for running tests to ensure that there is a fixed environment in which tests are run so that results are repeatable.  
  Examples :
  + creating temporary databases.
  + starting a server process.
* **test case:**  
  A test case is a set of conditions which is used to determine whether a system under test works correctly.
* **test suite:**  
  Test suite is a collection of testcases that are used to test a software program to show that it has some specified set of behaviours by executing the aggregated tests together.
* **test runner:**  
  A test runner is a component which set up the execution of tests and provides the outcome to the user.

**Basic Test Structure :**  
unittest defines tests by the following two ways :

* Manage test “fixtures” using code.
* test itself.

|  |
| --- |
| import unittest    class SimpleTest(unittest.TestCase):        # Returns True or False.      def test(self):          self.assertTrue(True)    if \_\_name\_\_ == '\_\_main\_\_':      unittest.main() |

This is the basic test code using unittest framework, which is having a single test. This test() method will fail if TRUE is ever FALSE.

**Running Tests**

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

The last block helps to run the test by running the file through the command line.

.

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK

Here, in the output the “.” on the first line of output means that a test passed.  
“-v” option is added in the command line while running the tests to obtain more detailed test results.

test (\_\_main\_\_.SimpleTest) ... ok

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK

**Outcomes Possible :**  
There are three types of possible test outcomes :

* OK – This means that all the tests are passed.
* FAIL – This means that the test did not pass and an AssertionError exception is raised.
* ERROR – This means that the test raises an exception other than AssertionError.

Let’s walk through an example to understand the implementation of unittest framework.

**Implementation :**

|  |
| --- |
| # Python code to demonstrate working of unittest  import unittest    class TestStringMethods(unittest.TestCase):        def setUp(self):          pass        # Returns True if the string contains 4 a.      def test\_strings\_a(self):          self.assertEqual( 'a'\*4, 'aaaa')        # Returns True if the string is in upper case.      def test\_upper(self):          self.assertEqual('foo'.upper(), 'FOO')        # Returns TRUE if the string is in uppercase      # else returns False.      def test\_isupper(self):          self.assertTrue('FOO'.isupper())          self.assertFalse('Foo'.isupper())        # Returns true if the string is stripped and      # matches the given output.      def test\_strip(self):          s = 'geeksforgeeks'          self.assertEqual(s.strip('geek'), 'sforgeeks')        # Returns true if the string splits and matches      # the given output.      def test\_split(self):          s = 'hello world'          self.assertEqual(s.split(), ['hello', 'world'])          with self.assertRaises(TypeError):              s.split(2)    if \_\_name\_\_ == '\_\_main\_\_':      unittest.main() |

The above code is a short script to test 5 string methods. **unittest.TestCase** is used to create test cases by subclassing it. The last block of the code at the bottom allows us to run all the tests just by running the file.

Basic terms used in the code :

* **assertEqual() –** This statement is used to check if the result obtained is equal to the expected result.
* **assertTrue() / assertFalse() –**This statement is used to verify if a given statement is true or false.
* **assertRaises() –**This statement is used to raise a specific exception.

Description of tests :

* **test\_strings\_a**  
  This test is used to test the property of string in which a character say ‘a’ multiplied by a number say ‘x’ gives the output as x times ‘a’. The assertEqual() statement returns true in this case if the result matches the given output.
* **test\_upper**  
  This test is used to check if the given string is converted to uppercase or not. The assertEqual() statement returns true if the string returned is in uppercase.
* **test\_isupper**  
  This test is used to test the property of string which returns TRUE if the string is in uppercase else returns False. The assertTrue() / assertFalse() statement is used for this verification.
* **test\_strip**  
  This test is used to check if all chars passed in the function have been stripped from the string. The assertEqual() statement returns true if the string is stripped and matches the given output.
* **test\_split**  
  This test is used to check the split function of the string which splits the string through the argument passed in the function and returns the result as list. The assertEqual() statement returns true in this case if the result matches the given output.

**unittest.main()** provides a command-line interface to the test script.On running the above script from the command line, following output is produced :

.....

----------------------------------------------------------------------

Ran 5 tests in 0.000s

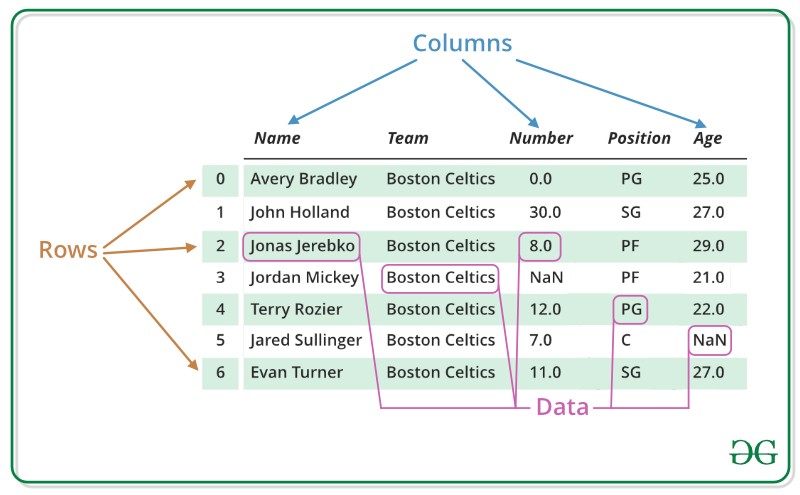
OK

1. **DATAFRAME AND ITS OPERATIONS IN PYTHON**

# DataFrame

* Last Updated : 10 Jan, 2019

**Pandas DataFrame** is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



We will get a brief insight on all these basic operation which can be performed on Pandas DataFrame :

* [Creating a DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics)
* [Dealing with Rows and Columns](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics1)
* [Indexing and Selecting Data](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics2)
* [Working with Missing Data](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics3)
* [Iterating over rows and columns](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics4)

#### Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

[**Creating a dataframe using List**](https://www.geeksforgeeks.org/create-a-pandas-dataframe-from-lists/)**:** DataFrame can be created using a single list or a list of lists.

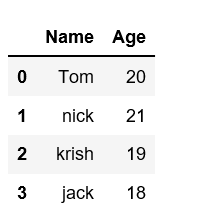
|  |
| --- |
| # import pandas as pd  import pandas as pd    # list of strings  lst = ['Geeks', 'For', 'Geeks', 'is',              'portal', 'for', 'Geeks']    # Calling DataFrame constructor on list  df = pd.DataFrame(lst)  print(df) |

Run on IDE

**Output:**  
  
   
[**Creating DataFrame from dict of ndarray/lists**](https://www.geeksforgeeks.org/python-create-a-pandas-dataframe-from-a-dict-of-equal-length-lists/)**:** To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

|  |
| --- |
| # Python code demonstrate creating  # DataFrame from dict narray / lists  # By default addresses.    import pandas as pd    # intialise data of lists.  data = {'Name':['Tom', 'nick', 'krish', 'jack'],          'Age':[20, 21, 19, 18]}    # Create DataFrame  df = pd.DataFrame(data)    # Print the output.  print(df) |

Run on IDE

**Output:**  


For more details refer to [Creating a Pandas DataFrame](https://www.geeksforgeeks.org/creating-a-pandas-dataframe/)

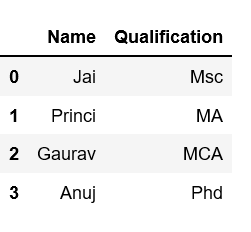
#### Dealing with Rows and Columns

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

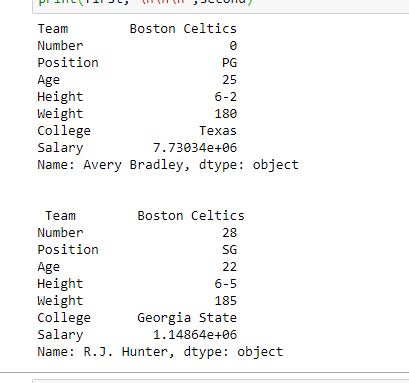
[**Column Selection**](https://www.geeksforgeeks.org/how-to-select-multiple-columns-in-a-pandas-dataframe/)**:** In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

|  |
| --- |
| # Import pandas package  import pandas as pd    # Define a dictionary containing employee data  data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],          'Age':[27, 24, 22, 32],          'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],          'Qualification':['Msc', 'MA', 'MCA', 'Phd']}    # Convert the dictionary into DataFrame  df = pd.DataFrame(data)    # select two columns  print(df[['Name', 'Qualification']]) |

Run on IDE

**Output:**  
  
   
[**Row Selection**](https://www.geeksforgeeks.org/python-pandas-extracting-rows-using-loc/)**:** Pandas provide a unique method to retrieve rows from a Data frame. [DataFrame.loc[]](https://www.geeksforgeeks.org/python-pandas-extracting-rows-using-loc/" \t "_blank) method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an [iloc[]](https://www.geeksforgeeks.org/python-extracting-rows-using-pandas-iloc/" \t "_blank) function.  
  
**Note:** We’ll be using [nba.csv](https://media.geeksforgeeks.org/wp-content/uploads/nba.csv) file in below examples.

|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("nba.csv", index\_col ="Name")    # retrieving row by loc method  first = data.loc["Avery Bradley"]  second = data.loc["R.J. Hunter"]      print(first, "\n\n\n", second) |

**Output:**  
As shown in the output image, two series were returned since there was only one parameter both of the times.  
  
For more Details refer to [Dealing with Rows and Columns](https://www.geeksforgeeks.org/dealing-with-rows-and-columns-in-pandas-dataframe/)

#### Indexing and Selecting Data

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as **Subset Selection**.

**Indexing a Dataframe using indexing operator [] :**  
Indexing operator is used to refer to the square brackets following an object. The [.loc](https://www.geeksforgeeks.org/python-pandas-extracting-rows-using-loc/) and [.iloc](https://www.geeksforgeeks.org/python-extracting-rows-using-pandas-iloc/) indexers also use the indexing operator to make selections. In this indexing operator to refer to df[].

#### Selecting a single columns

In order to select a single column, we simply put the name of the column in-between the brackets

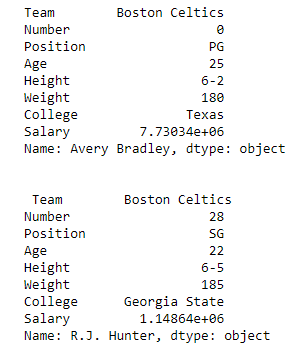
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("nba.csv", index\_col ="Name")    # retrieving columns by indexing operator  first = data["Age"]        print(first) |

**Output:**  
  
   
**Indexing a DataFrame using**[**.loc[ ]**](https://www.geeksforgeeks.org/python-pandas-extracting-rows-using-loc/)**:**  
This function selects data by the **label** of the rows and columns. The df.loc indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

#### Selecting a single row

In order to select a single row using .loc[], we put a single row label in a .loc function.

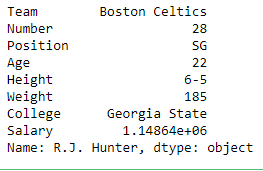
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("nba.csv", index\_col ="Name")    # retrieving row by loc method  first = data.loc["Avery Bradley"]  second = data.loc["R.J. Hunter"]      print(first, "\n\n\n", second) |

**Output:**  
As shown in the output image, two series were returned since there was only one parameter both of the times.  
  
   
**Indexing a DataFrame using**[**.iloc[ ]**](https://www.geeksforgeeks.org/python-extracting-rows-using-pandas-iloc/)**:**  
This function allows us to retrieve rows and columns by position. In order to do that, we’ll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

#### Selecting a single row

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

|  |
| --- |
| import pandas as pd    # making data frame from csv file  data = pd.read\_csv("nba.csv", index\_col ="Name")      # retrieving rows by iloc method  row2 = data.iloc[3]        print(row2) |

**Output:**  
  
   
For more Details refer

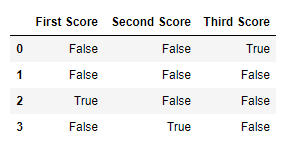
* [Indexing and Selecting Data with Pandas](https://www.geeksforgeeks.org/indexing-and-selecting-data-with-pandas/)
* [Boolean Indexing in Pandas](https://www.geeksforgeeks.org/boolean-indexing-in-pandas/)

#### Working with Missing Data

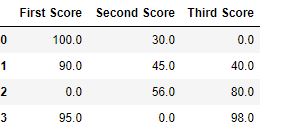
Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA(Not Available) values in pandas.

**Checking for missing values using isnull() and notnull() :**  
In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

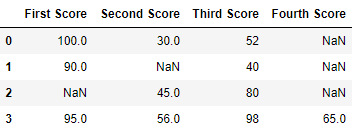
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from list  df = pd.DataFrame(dict)    # using isnull() function  df.isnull() |

**Output:**  
  
   
**Filling missing values using fillna(), replace() and interpolate() :**  
In order to fill null values in a datasets, we use fillna(), replace() and interpolate() function these function replace NaN values with some value of their own. All these function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.

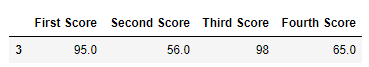
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # filling missing value using fillna()  df.fillna(0) |

**Output:**  
  
   
**Dropping missing values using dropna() :**  
In order to drop a null values from a dataframe, we used dropna() function this fuction drop Rows/Columns of datasets with Null values in different ways.

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, 40, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    df |

  
Now we drop rows with at least one Nan value (Null value)

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, 40, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # using dropna() function  df.dropna() |

**Output:**  
  
For more Details refer to [Working with Missing Data in Pandas](https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/)

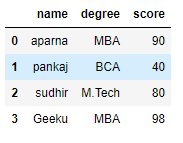
#### Iterating over rows and columns

Iteration is a general term for taking each item of something, one after another. Pandas DataFrame consists of rows and columns so, in order to iterate over dataframe, we have to iterate a dataframe like a dictionary.

**Iterating over rows :**  
In order to iterate over rows, we can use three function iteritems(), iterrows(), itertuples() . These three function will help in iteration over rows.

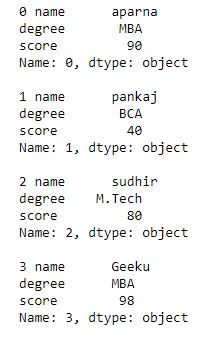
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # dictionary of lists  dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],          'degree': ["MBA", "BCA", "M.Tech", "MBA"],          'score':[90, 40, 80, 98]}    # creating a dataframe from a dictionary  df = pd.DataFrame(dict)    print(df) |

Run on IDE

  
Now we apply iterrows() function in order to get a each element of rows.

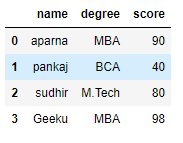
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # dictionary of lists  dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],          'degree': ["MBA", "BCA", "M.Tech", "MBA"],          'score':[90, 40, 80, 98]}    # creating a dataframe from a dictionary  df = pd.DataFrame(dict)    # iterating over rows using iterrows() function  for i, j in df.iterrows():      print(i, j)      print() |

Run on IDE

**Output:**  
  
   
**Iterating over Columns :**  
In order to iterate over columns, we need to create a list of dataframe columns and then iterating through that list to pull out the dataframe columns.

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # dictionary of lists  dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],          'degree': ["MBA", "BCA", "M.Tech", "MBA"],          'score':[90, 40, 80, 98]}    # creating a dataframe from a dictionary  df = pd.DataFrame(dict)    print(df) |

Run on IDE

  
Now we iterate through columns in order to iterate through columns we first create a list of dataframe columns and then iterate through list.

|  |
| --- |
| # creating a list of dataframe columns  columns = list(df)    for i in columns:        # printing the third element of the column      print (df[i][2]) |

**Output:**  
https://media.geeksforgeeks.org/wp-content/uploads/Capture55.jpg  
   
For more Details refer to [Iterating over rows and columns in Pandas DataFrame](https://www.geeksforgeeks.org/iterating-over-rows-and-columns-in-pandas-dataframe/)

**DataFrame Methods:**

FUNCTION DESCRIPTION

index() Method returns index (row labels) of the DataFrame

insert() Method inserts a column into a DataFrame

add() Method returns addition of dataframe and other, element-wise (binary operator add)

sub() Method returns subtraction of dataframe and other, element-wise (binary operator sub)

mul() Method returns multiplication of dataframe and other, element-wise (binary operator mul)

div() Method returns floating division of dataframe and other, element-wise (binary operator truediv)

unique() Method extracts the unique values in the dataframe

nunique() Method returns count of the unique values in the dataframe

value\_counts() Method counts the number of times each unique value occurs within the Series

columns() Method returns the column labels of the DataFrame

axes() Method returns a list representing the axes of the DataFrame

isnull() Method creates a Boolean Series for extracting rows with null values

notnull() Method creates a Boolean Series for extracting rows with non-null values

between() Method extracts rows where a column value falls in between a predefined range

isin() Method extracts rows from a DataFrame where a column value exists in a predefined collection

dtypes() Method returns a Series with the data type of each column. The result’s index is the original DataFrame’s columns

astype() Method converts the data types in a Series

values() Method returns a Numpy representation of the DataFrame i.e. only the values in the DataFrame will be returned, the axes labels will be removed

sort\_values()- Set1, Set2 Method sorts a data frame in Ascending or Descending order of passed Column

sort\_index() Method sorts the values in a DataFrame based on their index positions or labels instead of their values but sometimes a data frame is made out of two or more data frames and hence later index can be changed using this method

loc[] Method retrieves rows based on index label

iloc[] Method retrieves rows based on index position

ix[] Method retrieves DataFrame rows based on either index label or index position. This method combines the best features of the .loc[] and .iloc[] methods

rename() Method is called on a DataFrame to change the names of the index labels or column names

columns() Method is an alternative attribute to change the coloumn name

drop() Method is used to delete rows or columns from a DataFrame

pop() Method is used to delete rows or columns from a DataFrame

sample() Method pulls out a random sample of rows or columns from a DataFrame

nsmallest() Method pulls out the rows with the smallest values in a column

nlargest() Method pulls out the rows with the largest values in a column

shape() Method returns a tuple representing the dimensionality of the DataFrame

ndim() Method returns an ‘int’ representing the number of axes / array dimensions.

Returns 1 if Series, otherwise returns 2 if DataFrame

dropna() Method allows the user to analyze and drop Rows/Columns with Null values in different ways

fillna() Method manages and let the user replace NaN values with some value of their own

rank() Values in a Series can be ranked in order with this method

query() Method is an alternate string-based syntax for extracting a subset from a DataFrame

copy() Method creates an independent copy of a pandas object

duplicated() Method creates a Boolean Series and uses it to extract rows that have duplicate values

drop\_duplicates() Method is an alternative option to identifying duplicate rows and removing them through filtering

set\_index() Method sets the DataFrame index (row labels) using one or more existing columns

reset\_index() Method resets index of a Data Frame. This method sets a list of integer ranging from 0 to length of data as index

where() Method is used to check a Data Frame for one or more condition and return the result accordingly. By default, the rows not satisfying the condition are filled with NaN value

**DATABASE**

1. Difference between RDBMS and Datalake?
2. Different types of modelling and their benefits?
3. What is normalization vs Denormalization?
4. What are primary and foreign keys?
5. What are cross joins and does it have any disadvantages?
6. What is Indexing and where is it used?
7. Difference between Truncate and Delete?
8. What is the difference between Union and Join and when do we use them?
9. **DIFFERENCE BETWEEN RDBMS AND DATALAKE?**

**RDBMS**

A database is used to store, search and report on structured data from a single source. They are the simplest to create and SQL can be used to query and report on the data. There are both open source and proprietary databases, making it widely accessible to install and start using on premium or on the cloud.

**relational database** require schemas and are not a fit for unstructured or semi-structured data. Because of this rigid schema, they are not suited to be the centralized place to store data from multiple sources where the raw data varies in format and structure. However, they are popular for data analysis and monolithic applications

**DATALAKE**

A data lake stores structured, semi-structured and unstructured data, supporting the ability to store raw data from all sources without the need to process or transform it at that time.

Only when the data needs to be retrieved, will some structure need to be applied, which is ideal in the hands of data scientists and data analysis developers who can create new data models on the fly but does not provide the same reporting capabilities and ease of use for business users. Storing data in data lakes is much cheaper than in a data warehouse. Data lakes are very popular in the modern stack because of its flexibility and costs but they are not a replacement for data warehouses or relational databases.

1. **DIFFERENT TYPES OF MODELLING AND THEIR BENEFITS**

**What is Data Modelling?**

**Data modeling (data modelling)** is the process of creating a data model for the data to be stored in a database. This data model is a conceptual representation of Data objects, the associations between different data objects, and the rules.  
  
Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

**Data Model**

The **Data Model** is defined as an abstract model that organizes data description, data semantics, and consistency constraints of data. The data model emphasizes on what data is needed and how it should be organized instead of what operations will be performed on data. Data Model is like an architect’s building plan, which helps to build conceptual models and set a relationship between data items.

The two types of Data Modeling Techniques are

1. Entity Relationship (E-R) Model
2. UML (Unified Modelling Language)

We will discuss them in detail later.

This Data Modeling Tutorial is best suited for freshers, beginners as well as experienced professionals. In this data model tutorial, data modeling concepts in detail-

* [Why use Data Model?](https://www.guru99.com/data-modelling-conceptual-logical.html#2)
* [Types of Data Models](https://www.guru99.com/data-modelling-conceptual-logical.html#3)
* [Conceptual Data Model](https://www.guru99.com/data-modelling-conceptual-logical.html#4)
* [Logical Data Model](https://www.guru99.com/data-modelling-conceptual-logical.html#5)
* [Physical Data Model](https://www.guru99.com/data-modelling-conceptual-logical.html#6)
* [Advantages and Disadvantages of Data Model](https://www.guru99.com/data-modelling-conceptual-logical.html#8)

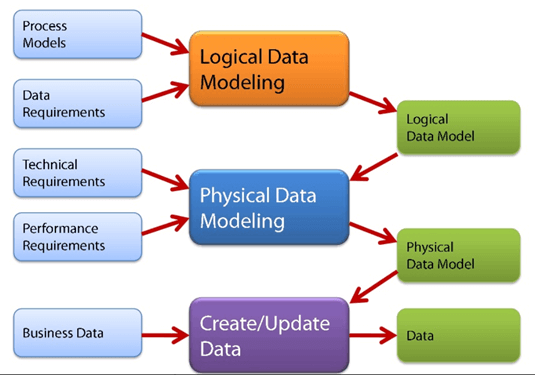
**Why use Data Model?**

The primary goal of using data model are:

* Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
* A data model helps design the database at the conceptual, physical and logical levels.
* Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
* It provides a clear picture of the base data and can be used by database developers to create a physical database.
* It is also helpful to identify missing and redundant data.
* Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

**Types of Data Models**: There are mainly three different types of data models: conceptual data models, logical data models, and physical data models, and each one has a specific purpose. The data models are used to represent the data and how it is stored in the database and to set the relationship between data items.

1. **Conceptual Data Model:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.
2. **Logical Data Model:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical Data Model**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.



Types of Data Model

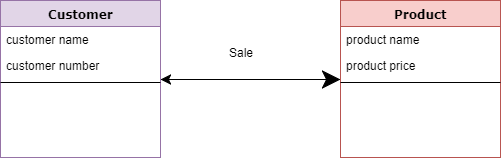
A **Conceptual Data Model** is an organized view of database concepts and their relationships. The purpose of creating a conceptual data model is to establish entities, their attributes, and relationships. In this data modeling level, there is hardly any detail available on the actual database structure. Business stakeholders and data architects typically create a conceptual data model.

The 3 basic tenants of Conceptual Data Model are

* **Entity**: A real-world thing
* **Attribute**: Characteristics or properties of an entity
* **Relationship**: Dependency or association between two entities

Data model example:

* Customer and Product are two entities. Customer number and name are attributes of the Customer entity
* Product name and price are attributes of product entity
* Sale is the relationship between the customer and product

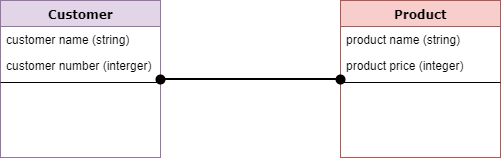
Conceptual Data Model

**Characteristics of a conceptual data model**

* Offers Organisation-wide coverage of the business concepts.
* This type of Data Models are designed and developed for a business audience.
* The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the “real world.”

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

The **Logical Data Model** is used to define the structure of data elements and to set relationships between them. The logical data model adds further information to the conceptual data model elements. The advantage of using a Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.

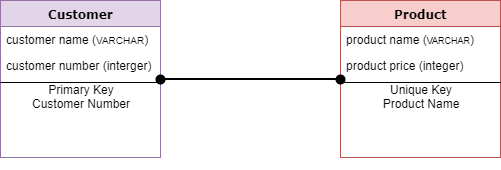
Logical Data Model

At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

**Characteristics of a Logical data model**

* Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
* Designed and developed independently from the DBMS.
* Data attributes will have datatypes with exact precisions and length.
* Normalization processes to the model is applied typically till 3NF.

A **Physical Data Model** describes a database-specific implementation of the data model. It offers database abstraction and helps generate the schema. This is because of the richness of meta-data offered by a Physical Data Model. The physical data model also helps in visualizing database structure by replicating database column keys, constraints, indexes, triggers, and other RDBMS features.

Physical Data Model

**Characteristics of a physical data model:**

* The physical data model describes data need for a single project or application though it maybe integrated with other physical data models based on project scope.
* Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
* Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
* Columns should have exact datatypes, lengths assigned and default values.
* Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

**Advantages and Disadvantages of Data Model:**

**Advantages of Data model:**

* The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
* The data model should be detailed enough to be used for building the physical database.
* The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
* Data Model helps business to communicate the within and across organizations.
* Data model helps to documents data mappings in ETL process
* Help to recognize correct sources of data to populate the model

**Disadvantages of Data model:**

* To develop Data model one should know physical data stored characteristics.
* This is a navigational system produces complex application development, management. Thus, it requires a knowledge of the biographical truth.
* Even smaller change made in structure require modification in the entire application.
* There is no set data manipulation language in DBMS.

1. **WHAT IS NORMALIZATION AND DENORMALIZATION**

[Normalization](https://www.geeksforgeeks.org/database-normalization-normal-forms/)**:**  
Normalization is the method used in a database to reduce the data redundancy and data inconsistency from the table. It is the technique in which Non-redundancy and consistency data are stored in the set schema. By using normalization the number of tables is increased instead of decreased.

Denormalization:  
Denormalization is also the method which is used in a database. It is used to add the redundancy to execute the query quickly. It is a technique in which data are combined to execute the query quickly. By using denormalization the number of tables is decreased which oppose to the normalization.

**Difference between Normalization and Denormalization:**

|  |  |  |
| --- | --- | --- |
| S.NO | Normalization | Denormalization |
| 1. | In normalization, Non-redundancy and consistency data are stored in set schema. | In denormalization, data are combined to execute the query quickly. |
| 2. | In normalization, Data redundancy and inconsistency is reduced. | In denormalization, redundancy is added for quick execution of queries. |
| 3. | Data integrity is maintained in normalization. | Data integrity is not maintained in denormalization. |
| 4. | In normalization, redundancy is reduced or eliminated. | In denormalization redundancy is added instead of reduction or elimination of redundancy. |
| 5. | Number of tables in normalization is increased. | Denormalization, Number of tables in decreased. |
| 6. | Normalization optimize the uses of disk spaces. | Denormalization do not optimize the disk spaces. |

1. **WHAT IS PRIMARY KEY AND FOREIGN KEY?**

[Primary Key](https://www.geeksforgeeks.org/types-of-keys-in-relational-model-candidate-super-primary-alternate-and-foreign/)**:**  
A primary key is used to ensure data in the specific column is unique. It is a column cannot have NULL values. It is either an existing table column or a column that is specifically generated by the database according to a defined sequence.

**Example:** Refer the figure –  
STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_NO can be chosen as the primary key (only one out of many candidate keys).

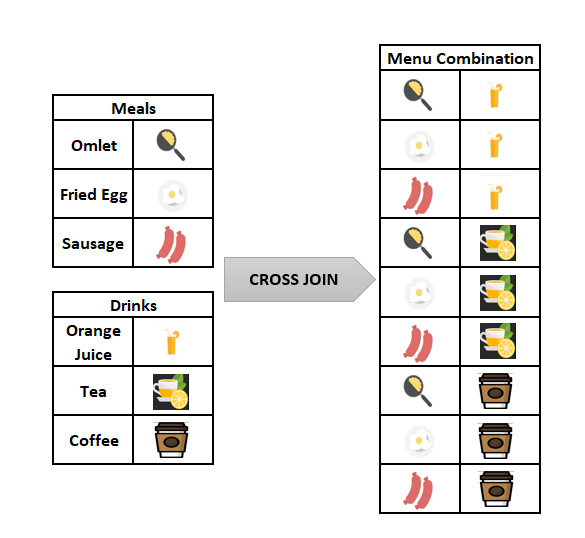
[Foreign Key](https://www.geeksforgeeks.org/types-of-keys-in-relational-model-candidate-super-primary-alternate-and-foreign/)**:**  
A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It is a column (or columns) that references a column (most often the primary key) of another table.

Let’s see the difference between Primary Key and Foreign Key:

| S.NO. | PRIMARY KEY | FOREIGN KEY |
| --- | --- | --- |
| 1 | A primary key is used to ensure data in the specific column is unique. | A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. |
| 2 | It uniquely identifies a record in the relational database table. | It refers to the field in a table which is the primary key of another table. |
| 3 | Only one primary key is allowed in a table. | Whereas more than one foreign key are allowed in a table. |
| 4 | It is a combination of UNIQUE and Not Null constraints. | It can contain duplicate values and a table in a relational database. |
| 5 | It does not allow NULL values. | It can also contain NULL values. |
| 6 | Its value cannot be deleted from the parent table. | Its value can be deleted from the child table. |
| 7 | It constraint can be implicitly defined on the temporary tables. | It constraint cannot be defined on the local or global temporary tables. |

1. **What are cross joins and does it have any disadvantages?**

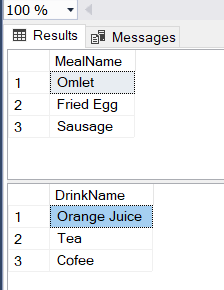
In SQL, the CROSS JOIN is **used to combine each row of the first table with each row of the second table**. It is also known as the Cartesian join since it returns the Cartesian product of the sets of rows from the joined tables.



we will consider the breakfast menu example again, which we mentioned in the earlier part of the article. Firstly, we will create the two-sample tables which contain the drink and meal names. After then, we will populate them with some sample data.

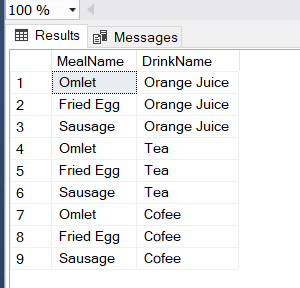
Through the following query, we will perform these two-steps:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE TABLE Meals(MealName VARCHAR(100))  CREATE TABLE Drinks(DrinkName VARCHAR(100))  INSERT INTO Drinks  VALUES('Orange Juice'), ('Tea'), ('Cofee')  INSERT INTO Meals  VALUES('Omlet'), ('Fried Egg'), ('Sausage')  SELECT \*  FROM Meals;  SELECT \*  FROM Drinks |

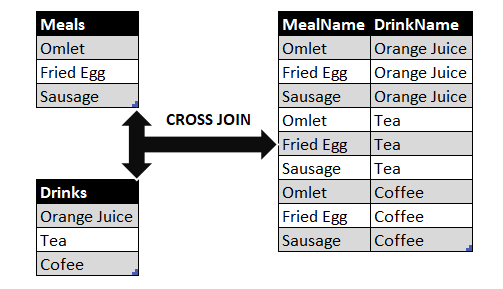


The following query will join the Meals and Drinks table with the CROSS JOINkeyword and we will obtain all of the paired combinations of the meal and drink names.

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Meals  CROSS JOIN Drinks |

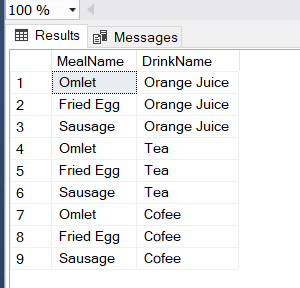


The below image illustrates the working principle of the CROSS JOIN.



At the same time, we can use the following query in order to obtain the same result set with an alternative syntax without CROSS JOIN.

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Meals  ,Drinks |

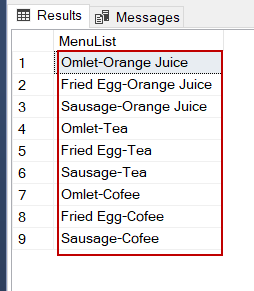


***Tip:****The resultset row count will equal to multiplication of tables row counts that will be joined. For the breakfast menu example, the Meals table row count is 3 and the Drinks table row count is 3, so the resultset row count can find with the following calculation.*

***3****(Meals table row count) x****3****(Drinks table row count) =****9****(Resultset row count)*

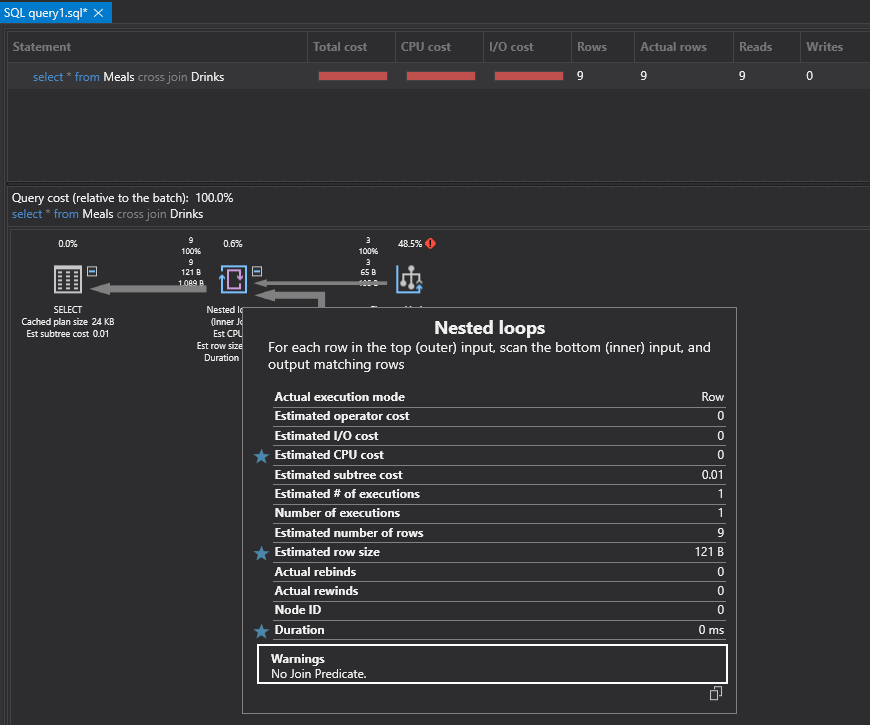
[CONCAT\_WS](https://docs.microsoft.com/en-us/sql/t-sql/functions/concat-ws-transact-sql?view=sql-server-ver15) function will help to concatenate the column expressions. Thus, we can create a more meaningful breakfast menu resultset.

|  |  |
| --- | --- |
| 1  2  3 | SELECT CONCAT\_WS('-',MealName,DrinkName) AS MenuList  FROM Meals CROSS JOIN  Drinks |



SQL CROSS JOIN and Performance Considerations

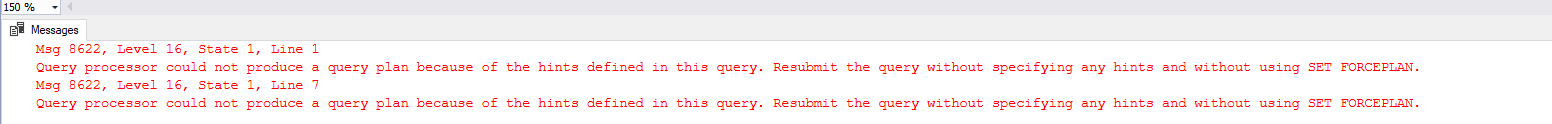
The SQL queries which contain the CROSS JOIN keyword can be very costly. We try to say that these queries have a high potential to consume more resources and can cause performance issues. For the following query, we will analyze the execution plan with [ApexSQL Plan](https://www.apexsql.com/sql-tools-plan.aspx). In the generated actual execution plan, we will see a Nested loops operator and when we hover over the mouse on this operator, the detail pop-up window will appear.



In this window, a warning message leaps to our eyes. “No Join Predicate” message specifies that this query can be faced with performance problems. For this reason, the query optimizer warns us about this potential problem. Briefly, when we decide to use the CROSS JOIN in any query, we should consider the number of the tables that will be joined. Such as, when we CROSS JOIN two tables and if the first one contains 1000 rows and the second one contains 1000 rows, the row count of the resultset will be 1.000.000 rows.

***Tip:****CROSS JOIN can only be implemented with Nested Loops, so the following queries will return an error if we force Query Optimizer to use other join types.*

|  |  |
| --- | --- |
| https://contribute.geeksforgeeks.org/wp-content/uploads/image7.png | https://contribute.geeksforgeeks.org/wp-content/uploads/image7.png |

[](https://www.sqlshack.com/wp-content/uploads/2020/02/performance-considerations-of-the-cross-join-1.png)

Disadvantages: When CROSS JOIN is used for tables that have a high number of rows, it might affect the performance negatively.

1. **What is Indexing and where is it used?**

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to create, delete, and uses the INDEX in the database.

It is used, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and is then referred to one or more specific page numbers.

Creating an Index:

Syntax:

CREATE INDEX index  
 ON TABLE column;

where the index is the name given to that index and TABLE is the name of the table on which that index is created and column is the name of that column for which it is applied.

When should indexes be created:

* A column contains a wide range of values.
* A column does not contain a large number of null values.
* One or more columns are frequently used together in a where clause or a join condition.

When should indexes be avoided:

* The table is small
* The columns are not often used as a condition in the query
* The column is updated frequently

Removing an Index:

To remove an index from the data dictionary by using the DROP INDEX command.

Syntax:

DROP INDEX index;

To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

Altering an Index:

To modify an existing table’s index by rebuilding, or reorganizing the index.

ALTER INDEX IndexName   
ON TableName REBUILD;

Confirming Indexes :

You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

Syntax:

select \* from USER\_INDEXES;

It will show you all the indexes present in the server, in which you can locate your own tables too.

Renaming an index :

You can use the system stored procedure sp\_rename to rename any index in the database.

Syntax:

EXEC sp\_rename   
 index\_name,  
 new\_index\_name,  
 N'INDEX';

1. **Difference between Truncate and Delete?**

Both the SQL commands DELETE and TRUNCATE deletes records from a table. But, the difference between DELETE and TRUNCATE is that, the DELETE command uses the WHERE clause to specify rows in a table for deletion operation and the TRUNCATE command does not use any clause; the row deletion is done at once.

TRUNCATE logs every single record in the transaction log.  
DELETE only logs the deallocation of the pages where the data is stored.

Thus TRUNCATE is slower compared to DELETE

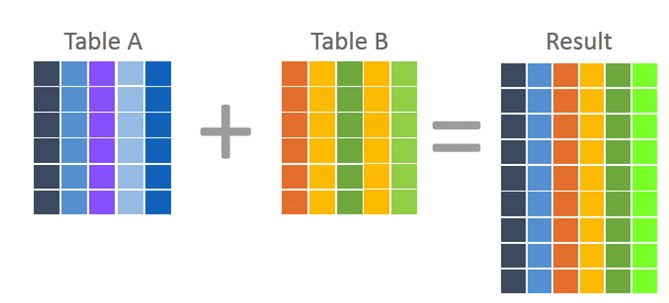
If you want to delete all records of a table: Use TRUNCATE.

If you want to delete specific records of a table: Use DELETE.

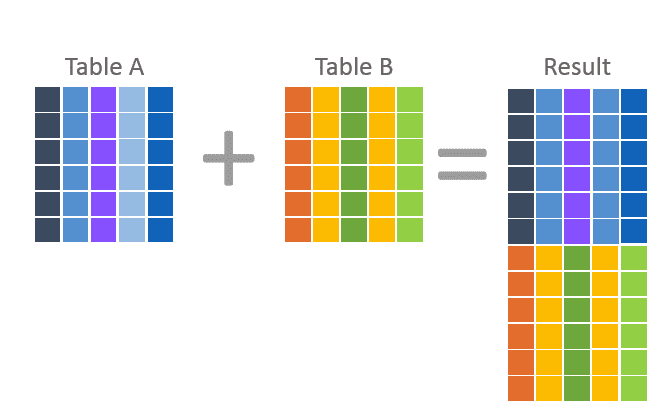
|  |  |  |
| --- | --- | --- |
| Topic | DELETE | TRUNCATE |
| Definition | DELETE is a SQL command that removes one or multiple rows from a table using conditions. | TRUNCATE is a SQL command that removes all the rows from a table without using any condition. |
| Language | It is a DML(Data Manipulation Language) command. | It is a DDL(Data Definition Language) command. |
| Commit | We have to manually COMMIT the changes in the DELETE command. | The changes are automatically COMMIT in the TRUNCATE command. |
| Process | It deletes rows one by one with conditions. | It deletes all the data at once. |
| Condition | It uses the WHERE clause as a condition. | It does not take any condition i.e. does not use WHERE clause. |
| Lock | It locks all the rows for deletion. | It uses table lock to lock the pages for deletion. |
| Log | It logs every single record in the transaction log. | It only logs the deallocation of the pages where the data is stored. |
| Transaction space | It uses more transaction space compared to the TRUNCATION command. | It uses less transaction space compared to the DELETE command. |
| Identity | It does not reset the table identity to its seed value if there is an identity column. | It resets the table identity to its seed value. |
| Permission | It needs delete permission. | It needs alter table permission. |
| Speed | It is much slower when comes to big databases. | It is faster or in other words, instant. |

1. **What is the difference between Union and Join and when do we use them?**

**Joins combine data into new columns.** If two tables are joined together, then the data from the first table is shown in one set of column alongside the second table’s column in the same row.



**Unions combine data into new rows.** If two tables are “unioned” together, then the data from the first table is in one set of rows, and the data from the second table in another set. The rows are in the same result.



­­