**WEEK 7**

**Task 1: Mathematical Operators**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 1: Mathematical Operators](https://dashboard.stige.in/index.php/lessons/task-1-mathematical-operators/)

In Progress

**Arithmetic operators** are used to perform mathematical operations like addition, subtraction, multiplication and division.

There are 7 arithmetic operators in Python :

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Modulus
6. Exponentiation
7. Floor division

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Syntax** |
| + | Addition: adds two operands | x + y |
| – | Subtraction: subtracts two operands | x – y |
| \* | Multiplication: multiplies two operands | x \* y |
| / | Division (float): divides the first operand by the second | x / y |
| // | Division (floor): divides the first operand by the second | x // y |
| % | Modulus: returns the remainder when first operand is divided by the second | x % y |
| \*\* | Power : Returns first raised to power second | x \*\* y |

# Task 2: Data Structures

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/)

In Progress

This chapter describes some things you’ve learned about already in more detail, and adds some new things as well.

# Lists

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Lists](https://dashboard.stige.in/index.php/topic/lists/)

In Progress

The list data type has some more methods. Here are all of the methods of list objects:

#### list.**append**(x)

Add an item to the end of the list. Equivalent to a[len(a):] = [x].

#### list.**extend**(iterable)

Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.

#### list.**insert**(i, x)

Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

#### list.**remove**(x)

Remove the first item from the list whose value is equal to x. It raises a [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError) if there is no such item.

#### list.**pop**([i])

Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

#### list.**clear**()

Remove all items from the list. Equivalent to del a[:].

#### list.**index**(x[, start[, end]])

Return zero-based index in the list of the first item whose value is equal to x. Raises a [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError) if there is no such item.

The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the start argument.

#### list.**count**(x)

Return the number of times x appears in the list.

#### list.**sort**(\*, key=None, reverse=False)

Sort the items of the list in place (the arguments can be used for sort customization, see [sorted()](https://docs.python.org/3/library/functions.html#sorted) for their explanation).

#### list.**reverse**()

Reverse the elements of the list in place.list.copy()

Return a shallow copy of the list. Equivalent to a[:].

An example that uses most of the list methods:

**>>>** fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

**>>>** fruits.count('apple')

2

**>>>** fruits.count('tangerine')

0

**>>>** fruits.index('banana')

3

**>>>** fruits.index('banana', 4) # Find next banana starting a position 4

6

**>>>** fruits.reverse()

**>>>** fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']

**>>>** fruits.append('grape')

**>>>** fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']

**>>>** fruits.sort()

**>>>** fruits

['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']

**>>>** fruits.pop()

'pear'

You might have noticed that methods like insert, remove or sort that only modify the list have no return value printed – they return the default None. [[1]](https://docs.python.org/3/tutorial/datastructures.html#id2) This is a design principle for all mutable data structures in Python.

Another thing you might notice is that not all data can be sorted or compared. For instance, [None, 'hello', 10] doesn’t sort because integers can’t be compared to strings and None can’t be compared to other types. Also, there are some types that don’t have a defined ordering relation. For example, 3+4j < 5+7j isn’t a valid comparison.

# Using Lists as Stacks

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Using Lists as Stacks](https://dashboard.stige.in/index.php/topic/using-lists-as-stacks/)

In Progress

The list methods make it very easy to use a list as a stack, where the last element added is the first element retrieved (“last-in, first-out”). To add an item to the top of the stack, use append(). To retrieve an item from the top of the stack, use pop() without an explicit index. For example:

**>>>** stack = [3, 4, 5]

**>>>** stack.append(6)

**>>>** stack.append(7)

**>>>** stack

[3, 4, 5, 6, 7]

**>>>** stack.pop()

7

**>>>** stack

[3, 4, 5, 6]

**>>>** stack.pop()

6

**>>>** stack.pop()

5

**>>>** stack

[3, 4]

# Using Lists as Queues

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Using Lists as Queues](https://dashboard.stige.in/index.php/topic/using-lists-as-queues/)

In Progress

It is also possible to use a list as a queue, where the first element added is the first element retrieved (“first-in, first-out”); however, lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow (because all of the other elements have to be shifted by one).

To implement a queue, use [collections.deque](https://docs.python.org/3/library/collections.html" \l "collections.deque) which was designed to have fast appends and pops from both ends. For example:

**>>> from** **collections** **import** deque

**>>>** queue = deque(["Eric", "John", "Michael"])

**>>>** queue.append("Terry")

# Terry arrives

**>>>** queue.append("Graham")

# Graham arrives

**>>>** queue.popleft()

# The first to arrive now leaves

'Eric'

**>>>** queue.popleft()

# The second to arrive now leaves

'John'

**>>>** queue

# Remaining queue in order of arrival

deque(['Michael', 'Terry', 'Graham'])

# List Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [List Comprehensions](https://dashboard.stige.in/index.php/topic/list-comprehensions/)

In Progress

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

For example, assume we want to create a list of squares, like:

**>>>** squares = []

**>>> for** x **in** range(10):

**...**  squares.append(x\*\*2)

**...**

**>>>** squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Note that this creates (or overwrites) a variable named x that still exists after the loop completes. We can calculate the list of squares without any side effects using:

squares = list(map(**lambda** x: x\*\*2, range(10)))

or, equivalently:

squares = [x\*\*2 **for** x **in** range(10)]

which is more concise and readable.

A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it. For example, this listcomp combines the elements of two lists if they are not equal:

**>>>** [(x, y) **for** x **in** [1,2,3] **for** y **in** [3,1,4] **if** x != y]

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

and it’s equivalent to:

**>>>** combs = []

**>>> for** x **in** [1,2,3]:

**...**  **for** y **in** [3,1,4]:

**...**  **if** x != y:

**...**  combs.append((x, y))

**...**

**>>>** combs

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

Note how the order of the [for](https://docs.python.org/3/reference/compound_stmts.html#for) and [if](https://docs.python.org/3/reference/compound_stmts.html#if) statements is the same in both these snippets.

If the expression is a tuple (e.g. the (x, y) in the previous example), it must be parenthesized.

**>>>** vec = [-4, -2, 0, 2, 4]

**>>>** # create a new list with the values doubled

**>>>** [x\*2 **for** x **in** vec]

[-8, -4, 0, 4, 8]

**>>>** # filter the list to exclude negative numbers

**>>>** [x **for** x **in** vec **if** x >= 0]

[0, 2, 4]

**>>>** # apply a function to all the elements

**>>>** [abs(x) **for** x **in** vec]

[4, 2, 0, 2, 4]

**>>>** # call a method on each element

**>>>** freshfruit = [' banana', ' loganberry ', 'passion fruit ']

**>>>** [weapon.strip() **for** weapon **in** freshfruit]

['banana', 'loganberry', 'passion fruit']

**>>>** # create a list of 2-tuples like (number, square)

**>>>** [(x, x\*\*2) **for** x **in** range(6)]

[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

**>>>** # the tuple must be parenthesized, otherwise an error is raised

**>>>** [x, x\*\*2 **for** x **in** range(6)]

File "<stdin>", line 1, in <module>

[x, x\*\*2 for x in range(6)]

^

SyntaxError: invalid syntax

**>>>** # flatten a list using a listcomp with two 'for'

**>>>** vec = [[1,2,3], [4,5,6], [7,8,9]]

**>>>** [num **for** elem **in** vec **for** num **in** elem]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

List comprehensions can contain complex expressions and nested functions:

**>>> from** **math** **import** pi

**>>>** [str(round(pi, i)) **for** i **in** range(1, 6)]

['3.1', '3.14', '3.142', '3.1416', '3.14159']

# Nested List Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Nested List Comprehensions](https://dashboard.stige.in/index.php/topic/nested-list-comprehensions/)

In Progress

The initial expression in a list comprehension can be any arbitrary expression, including another list comprehension.

Consider the following example of a 3×4 matrix implemented as a list of 3 lists of length 4:

**>>>** matrix = [

**...**  [1, 2, 3, 4],

**...**  [5, 6, 7, 8],

**...**  [9, 10, 11, 12],

**...** ]

The following list comprehension will transpose rows and columns:>>>

**>>>** [[row[i] **for** row **in** matrix] **for** i **in** range(4)]

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

As we saw in the previous section, the nested listcomp is evaluated in the context of the [for](https://docs.python.org/3/reference/compound_stmts.html#for) that follows it, so this example is equivalent to:>>>

**>>>** transposed = []

**>>> for** i **in** range(4):

**...**  transposed.append([row[i] **for** row **in** matrix])

**...**

**>>>** transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

which, in turn, is the same as:>>>

**>>>** transposed = []

**>>> for** i **in** range(4):

**...**  # the following 3 lines implement the nested listcomp

**...**  transposed\_row = []

**...**  **for** row **in** matrix:

**...**  transposed\_row.append(row[i])

**...**  transposed.append(transposed\_row)

**...**

**>>>** transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

In the real world, you should prefer built-in functions to complex flow statements. The [zip()](https://docs.python.org/3/library/functions.html#zip) function would do a great job for this use case:>>>

**>>>** list(zip(\*matrix))

[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]

See [Unpacking Argument Lists](https://docs.python.org/3/tutorial/controlflow.html#tut-unpacking-arguments) for details on the asterisk in this line.

# The del Statement

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [The del Statement](https://dashboard.stige.in/index.php/topic/the-del-statement/)

In Progress

There is a way to remove an item from a list given its index instead of its value: the [del](https://docs.python.org/3/reference/simple_stmts.html#del) statement. This differs from the pop() method which returns a value. The del statement can also be used to remove slices from a list or clear the entire list (which we did earlier by assignment of an empty list to the slice). For example:

>>> a = [-1, 1, 66.25, 333, 333, 1234.5]

>>> del a[0]

>>> a

[1, 66.25, 333, 333, 1234.5]

>>> del a[2:4]

>>> a

[1, 66.25, 1234.5]

>>> del a[:]

>>> a

[]

[del](https://docs.python.org/3/reference/simple_stmts.html#del) can also be used to delete entire variables:

>>> del a

Referencing the name a hereafter is an error (at least until another value is assigned to it). We’ll find other uses for [del](https://docs.python.org/3/reference/simple_stmts.html#del) later.

# Tuples and Sequences

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Tuples and Sequences](https://dashboard.stige.in/index.php/topic/tuples-and-sequences/)

In Progress

We saw that lists and strings have many common properties, such as indexing and slicing operations. They are two examples of sequence data types (see [Sequence Types — list, tuple, range](https://docs.python.org/3/library/stdtypes.html#typesseq)). Since Python is an evolving language, other sequence data types may be added. There is also another standard sequence data type: the tuple.

A tuple consists of a number of values separated by commas, for instance:

**>>>** t = 12345, 54321, 'hello!'

**>>>** t[0]

12345

**>>>** t

(12345, 54321, 'hello!')

**>>>** # Tuples may be nested:

**...** u = t, (1, 2, 3, 4, 5)

**>>>** u

((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))

**>>>** # Tuples are immutable:

**...** t[0] = 88888

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

**>>>** # but they can contain mutable objects:

**...** v = ([1, 2, 3], [3, 2, 1])

**>>>** v

([1, 2, 3], [3, 2, 1])

As you see, on output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses, although often parentheses are necessary anyway (if the tuple is part of a larger expression). It is not possible to assign to the individual items of a tuple, however it is possible to create tuples which contain mutable objects, such as lists.

Though tuples may seem similar to lists, they are often used in different situations and for different purposes. Tuples are [immutable](https://docs.python.org/3/glossary.html#term-immutable), and usually contain a heterogeneous sequence of elements that are accessed via unpacking (see later in this section) or indexing (or even by attribute in the case of [namedtuples](https://docs.python.org/3/library/collections.html" \l "collections.namedtuple)). Lists are [mutable](https://docs.python.org/3/glossary.html#term-mutable), and their elements are usually homogeneous and are accessed by iterating over the list.

A special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these. Empty tuples are constructed by an empty pair of parentheses; a tuple with one item is constructed by following a value with a comma (it is not sufficient to enclose a single value in parentheses). Ugly, but effective. For example:

**>>>** empty = ()

**>>>** singleton = 'hello', # <-- note trailing comma

**>>>** len(empty)

0

**>>>** len(singleton)

1

**>>>** singleton

('hello',)

The statement t = 12345, 54321, 'hello!' is an example of tuple packing: the values 12345, 54321 and 'hello!' are packed together in a tuple. The reverse operation is also possible:>>>

**>>>** x, y, z = t

This is called, appropriately enough, sequence unpacking and works for any sequence on the right-hand side. Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence. Note that multiple assignment is really just a combination of tuple packing and sequence unpacking.

# Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Sets](https://dashboard.stige.in/index.php/topic/sets/)

In Progress

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the [set()](https://docs.python.org/3/library/stdtypes.html#set) function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary, a data structure that we discuss in the next section.

Here is a brief demonstration:

**>>>** basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

**>>>** print(basket)

# show that duplicates have been removed

{'orange', 'banana', 'pear', 'apple'}

**>>>** 'orange' **in** basket

# fast membership testing

True

**>>>** 'crabgrass' **in** basket

False

**>>>** # Demonstrate set operations on unique letters from two words

**...**

**>>>** a = set('abracadabra')

**>>>** b = set('alacazam')

**>>>** a

# unique letters in a

{'a', 'r', 'b', 'c', 'd'}

**>>>** a - b

# letters in a but not in b

{'r', 'd', 'b'}

**>>>** a | b

# letters in a or b or both

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

**>>>** a & b

# letters in both a and b

{'a', 'c'}

**>>>** a ^ b

# letters in a or b but not both

{'r', 'd', 'b', 'm', 'z', 'l'}

Similarly to [list comprehensions](https://docs.python.org/3/tutorial/datastructures.html#tut-listcomps), set comprehensions are also supported:>>>

**>>>** a = {x **for** x **in** 'abracadabra' **if** x **not** **in** 'abc'}

**>>>** a

{'r', 'd'}

# Dictionaries

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Dictionaries](https://dashboard.stige.in/index.php/topic/dictionaries/)

In Progress

Another useful data type built into Python is the dictionary (see [Mapping Types — dict](https://docs.python.org/3/library/stdtypes.html#typesmapping)). Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Performing list(d) on a dictionary returns a list of all the keys used in the dictionary, in insertion order (if you want it sorted, just use sorted(d) instead). To check whether a single key is in the dictionary, use the [in](https://docs.python.org/3/reference/expressions.html#in) keyword.

Here is a small example using a dictionary:

**>>>** tel = {'jack': 4098, 'sape': 4139}

**>>>** tel['guido'] = 4127

**>>>** tel

{'jack': 4098, 'sape': 4139, 'guido': 4127}

**>>>** tel['jack']

4098

**>>> del** tel['sape']

**>>>** tel['irv'] = 4127

**>>>** tel

{'jack': 4098, 'guido': 4127, 'irv': 4127}

**>>>** list(tel)

['jack', 'guido', 'irv']

**>>>** sorted(tel)

['guido', 'irv', 'jack']

**>>>** 'guido' **in** tel

True

**>>>** 'jack' **not** **in** tel

False

The [dict()](https://docs.python.org/3/library/stdtypes.html" \l "dict) constructor builds dictionaries directly from sequences of key-value pairs:

**>>>** dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])

{'sape': 4139, 'guido': 4127, 'jack': 4098}

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:

**>>>** {x: x\*\*2 **for** x **in** (2, 4, 6)}

{2: 4, 4: 16, 6: 36}

When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments:

**>>>** dict(sape=4139, guido=4127, jack=4098)

{'sape': 4139, 'guido': 4127, 'jack': 4098}

# Looping Techniques

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Looping Techniques](https://dashboard.stige.in/index.php/topic/looping-techniques/)

In Progress

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the items() method.

**>>>** knights = {'gallahad': 'the pure', 'robin': 'the brave'}

**>>> for** k, v **in** knights.items():

**...**  print(k, v)

**...**

gallahad the pure

robin the brave

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the [enumerate()](https://docs.python.org/3/library/functions.html#enumerate) function.

**>>> for** i, v **in** enumerate(['tic', 'tac', 'toe']):

**...**  print(i, v)

**...**

0 tic

1 tac

2 toe

To loop over two or more sequences at the same time, the entries can be paired with the [zip()](https://docs.python.org/3/library/functions.html#zip) function.

**>>>** questions = ['name', 'quest', 'favorite color']

**>>>** answers = ['lancelot', 'the holy grail', 'blue']

**>>> for** q, a **in** zip(questions, answers):

**...**  print('What is your {0}? It is {1}.'.format(q, a))

**...**

What is your name? It is lancelot.

What is your quest? It is the holy grail.

What is your favorite color? It is blue.

To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the [reversed()](https://docs.python.org/3/library/functions.html#reversed) function.

**>>> for** i **in** reversed(range(1, 10, 2)):

**...**  print(i)

**...**

9

7

5

3

1

To loop over a sequence in sorted order, use the [sorted()](https://docs.python.org/3/library/functions.html#sorted) function which returns a new sorted list while leaving the source unaltered.

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** i **in** sorted(basket):

**...**  print(i)

**...**

apple

apple

banana

orange

orange

pear

Using [set()](https://docs.python.org/3/library/stdtypes.html#set) on a sequence eliminates duplicate elements. The use of [sorted()](https://docs.python.org/3/library/functions.html#sorted) in combination with [set()](https://docs.python.org/3/library/stdtypes.html#set) over a sequence is an idiomatic way to loop over unique elements of the sequence in sorted order.

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** f **in** sorted(set(basket)):

**...**  print(f)

**...**

apple

banana

orange

pear

It is sometimes tempting to change a list while you are looping over it; however, it is often simpler and safer to create a new list instead.

**>>> import** **math**

**>>>** raw\_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]

**>>>** filtered\_data = []

**>>> for** value **in** raw\_data:

**...**  **if** **not** math.isnan(value):

**...**  filtered\_data.append(value)

**...**

**>>>** filtered\_data

[56.2, 51.7, 55.3, 52.5, 47.8]

# More on Conditions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [More on Conditions](https://dashboard.stige.in/index.php/topic/more-on-conditions/)

In Progress

The conditions used in while and if statements can contain any operators, not just comparisons.

The comparison operators in and not in check whether a value occurs (does not occur) in a sequence. The operators is and is not compare whether two objects are really the same object. All comparison operators have the same priority, which is lower than that of all numerical operators.

Comparisons can be chained. For example, a < b == c tests whether a is less than b and moreover b equals c.

Comparisons may be combined using the Boolean operators and and or, and the outcome of a comparison (or of any other Boolean expression) may be negated with not. These have lower priorities than comparison operators; between them, not has the highest priority and or the lowest, so that A and not B or C is equivalent to (A and (not B)) or C. As always, parentheses can be used to express the desired composition.

The Boolean operators and and or are so-called short-circuit operators: their arguments are evaluated from left to right, and evaluation stops as soon as the outcome is determined. For example, if A and C are true but B is false, A and B and C does not evaluate the expression C. When used as a general value and not as a Boolean, the return value of a short-circuit operator is the last evaluated argument.

It is possible to assign the result of a comparison or other Boolean expression to a variable. For example,

**>>>** string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'

**>>>** non\_null = string1 **or** string2 **or** string3

**>>>** non\_null

'Trondheim'

Note that in Python, unlike C, assignment inside expressions must be done explicitly with the [walrus operator](https://docs.python.org/3/faq/design.html#why-can-t-i-use-an-assignment-in-an-expression) :=. This avoids a common class of problems encountered in C programs: typing = in an expression when == was intended.

# Comparing Sequences and Other Types

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Data Structures](https://dashboard.stige.in/index.php/lessons/task-2-data-structures/) [Comparing Sequences and Other Types](https://dashboard.stige.in/index.php/topic/comparing-sequences-and-other-types/)

In Progress

Sequence objects typically may be compared to other objects with the same sequence type. The comparison uses **lexicographical ordering**: first the first two items are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two items are compared, and so on, until either sequence is exhausted. If two items to be compared are themselves sequences of the same type, the lexicographical comparison is carried out recursively. If all items of two sequences compare equal, the sequences are considered equal. If one sequence is an initial sub-sequence of the other, the shorter sequence is the smaller (lesser) one. Lexicographical ordering for strings uses the Unicode code point number to order individual characters. Some examples of comparisons between sequences of the same type:

(1, 2, 3) < (1, 2, 4)

[1, 2, 3] < [1, 2, 4]

'ABC' < 'C' < 'Pascal' < 'Python'

(1, 2, 3, 4) < (1, 2, 4)

(1, 2) < (1, 2, -1)

(1, 2, 3) == (1.0, 2.0, 3.0)

(1, 2, ('aa', 'ab')) < (1, 2, ('abc', 'a'), 4)

Note that comparing objects of different types with < or > is legal provided that the objects have appropriate comparison methods. For example, mixed numeric types are compared according to their numeric value, so 0 equals 0.0, etc. Otherwise, rather than providing an arbitrary ordering, the interpreter will raise a [TypeError](https://docs.python.org/3/library/exceptions.html" \l "TypeError) exception.

Footnotes[[1]](https://docs.python.org/3/tutorial/datastructures.html#id1)

Other languages may return the mutated object, which allows method chaining, such as d->insert("a")->remove("b")->sort();.

# Task 3: Lists

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/)

In Progress

**Lists**are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it a most powerful tool in [Python](https://www.geeksforgeeks.org/python-programming-language/). A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

List in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

**Note-** Lists are a useful tool for preserving a sequence of data and further iterating over it.

# Creating a List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Creating a List](https://dashboard.stige.in/index.php/topic/creating-a-list/)

In Progress

Lists in Python can be created by just placing the sequence inside the square brackets[]. Unlike [Sets](https://www.geeksforgeeks.org/python-sets/), list doesn’t need a built-in function for creation of list.

**Note –** Unlike Sets, list may contain mutable elements.

# Python program to demonstrate

# Creation of List

# Creating a List

List = []

print("Blank List: ")

print(List)

# Creating a List of numbers

List = [10, 20, 14]

print("\nList of numbers: ")

print(List)

# Creating a List of strings and accessing

# using index

List = ["Geeks", "For", "Geeks"]

print("\nList Items: ")

print(List[0])

print(List[2])

# Creating a Multi-Dimensional List

# (By Nesting a list inside a List)

List = [['Geeks', 'For'] , ['Geeks']]

print("\nMulti-Dimensional List: ")

print(List)

**Output:**

Blank List:

[]

List of numbers:

[10, 20, 14]

List Items

Geeks

Geeks

Multi-Dimensional List:

[['Geeks', 'For'], ['Geeks']]

### Creating a list with multiple distinct or duplicate elements

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

|  |
| --- |
| # Creating a List with  # the use of Numbers # (Having duplicate values)  List = [1, 2, 4, 4, 3, 3, 3, 6, 5] print("\nList with the use of Numbers: ") print(List)   # Creating a List with # mixed type of values # (Having numbers and strings)  List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks'] print("\nList with the use of Mixed Values: ") print(List) |

**Output:**

List with the use of Numbers:

[1, 2, 4, 4, 3, 3, 3, 6, 5]

List with the use of Mixed Values:

[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']

# Knowing the size of List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Knowing the size of List](https://dashboard.stige.in/index.php/topic/knowing-the-size-of-list/)

In Progress

|  |
| --- |
| # Creating a List  List1 = [] print(len(List1))   # Creating a List of numbers  List2 = [10, 20, 14] print(len(List2)) |

**Output:**

0

3

# Adding Elements to a List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Adding Elements to a List](https://dashboard.stige.in/index.php/topic/adding-elements-to-a-list/)

In Progress

### Using append() method

Elements can be added to the List by using built-in [**append()**](https://www.geeksforgeeks.org/list-methods-python/) function. Only one element at a time can be added to the list by using append() method, for addition of multiple elements with the append() method, loops are used. Tuples can also be added to the List with the use of append method because tuples are immutable. Unlike Sets, Lists can also be added to the existing list with the use of append() method.

|  |
| --- |
| # Python program to demonstrate  # Addition of elements in a List  # Creating a List  List = [] print("Initial blank List: ") print(List)   # Addition of Elements # in the List  List.append(1) List.append(2) List.append(4) print("\nList after Addition of Three elements: ") print(List)   # Adding elements to the List # using Iterator  for i in range(1, 4): List.append(i) print("\nList after Addition of elements from 1-3: ") print(List)    # Adding Tuples to the List  List.append((5, 6)) print("\nList after Addition of a Tuple: ") print(List)   # Addition of List to a List  List2 = ['For', 'Geeks'] List.append(List2) print("\nList after Addition of a List: ") print(List) |

**Output:**

Initial blank List:

[]

List after Addition of Three elements:

[1, 2, 4]

List after Addition of elements from 1-3:

[1, 2, 4, 1, 2, 3]

List after Addition of a Tuple:

[1, 2, 4, 1, 2, 3, (5, 6)]

List after Addition of a List:

[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]

### Using insert() method

append() method only works for addition of elements at the end of the List, for addition of element at the desired position, insert() method is used. Unlike append() which takes only one argument, insert() method requires two arguments(position, value).

|  |
| --- |
| # Python program to demonstrate  # Addition of elements in a List   # Creating a List  List = [1,2,3,4] print("Initial List: ") print(List)   # Addition of Element at  # specific Position # (using Insert Method)   List.insert(3, 12) List.insert(0, 'Geeks') print("\nList after performing Insert Operation: ") print(List) |

**Output:**

Initial List:

[1, 2, 3, 4]

List after performing Insert Operation:

['Geeks', 1, 2, 3, 12, 4]

### Using extend() method

Other than append() and insert() methods, there’s one more method for Addition of elements, [**extend()**](https://www.geeksforgeeks.org/append-extend-python/), this method is used to add multiple elements at the same time at the end of the list.

**Note –**[append() and extend()](https://www.geeksforgeeks.org/append-extend-python/) methods can only add elements at the end.

|  |
| --- |
| # Python program to demonstrate  # Addition of elements in a List   # Creating a List  List = [1,2,3,4] print("Initial List: ") print(List)   # Addition of multiple elements # to the List at the end # (using Extend Method)  List.extend([8, 'Geeks', 'Always']) print("\nList after performing Extend Operation: ") print(List) |

**Output:**

Initial List:

[1, 2, 3, 4]

List after performing Extend Operation:

[1, 2, 3, 4, 8, 'Geeks', 'Always']

# Accessing elements from the List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Accessing elements from the List](https://dashboard.stige.in/index.php/topic/accessing-elements-from-the-list/)

In Progress

In order to access the list items refer to the index number.Use the index operator [ ] to access an item in a list.The index must be an integer.Nested list are accessed using nested indexing.

|  |
| --- |
| # Python program to demonstrate  # accessing of element from list   # Creating a List with # the use of multiple values List = ["Geeks", "For", "Geeks"]   # accessing a element from the  # list using index number print("Accessing a element from the list") print(List[0])  print(List[2])   # Creating a Multi-Dimensional List # (By Nesting a list inside a List) List = [['Geeks', 'For'] , ['Geeks']]   # accessing an element from the  # Multi-Dimensional List using # index number print("Acessing a element from a Multi-Dimensional list") print(List[0][1]) print(List[1][0]) |

**Output:**

Accessing a element from the list

Geeks

Geeks

Acessing a element from a Multi-Dimensional list

For

Geeks

### Negative indexing

In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

|  |
| --- |
| List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']   # accessing a element using # negative indexing print("Accessing element using negative indexing")   # print the last element of list print(List[-1])   # print the third last element of list  print(List[-3]) |

**Output:**

Accessing element using negative indexing

Geeks

For

# Removing Elements from the List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Removing Elements from the List](https://dashboard.stige.in/index.php/topic/removing-elements-from-the-list/)

In Progress

### Using remove() method

Elements can be removed from the List by using built-in [**remove()**](https://www.geeksforgeeks.org/python-list-remove/) function but an Error arises if element doesn’t exist in the set. [Remove()](https://www.geeksforgeeks.org/python-list-remove/) method only removes one element at a time, to remove range of elements, iterator is used. The remove() method removes the specified item.

**Note –** Remove method in List will only remove the first occurrence of the searched element.

# Python program to demonstrate

# Removal of elements in a List

# Creating a List

List = [1, 2, 3, 4, 5, 6,

7, 8, 9, 10, 11, 12]

print("Intial List: ")

print(List)

# Removing elements from List

# using Remove() method

List.remove(5)

List.remove(6)

print("\nList after Removal of two elements: ")

print(List)

# Removing elements from List

# using iterator method

for i in range(1, 5):

List.remove(i)

print("\nList after Removing a range of elements: ")

print(List)

**Output:**

Intial List:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

List after Removal of two elements:

[1, 2, 3, 4, 7, 8, 9, 10, 11, 12]

List after Removing a range of elements:

[7, 8, 9, 10, 11, 12]

### Using pop() method

[Pop()](https://www.geeksforgeeks.org/python-list-pop/) function can also be used to remove and return an element from the set, but by default it removes only the last element of the set, to remove element from a specific position of the List, index of the element is passed as an argument to the pop() method.

List = [1,2,3,4,5]

# Removing element from the

# Set using the pop() method

List.pop()

print("\nList after popping an element: ")

print(List)

# Removing element at a

# specific location from the

# Set using the pop() method

List.pop(2)

print("\nList after popping a specific element: ")

print(List)

**Output:**

List after popping an element:

[1, 2, 3, 4]

List after popping a specific element:

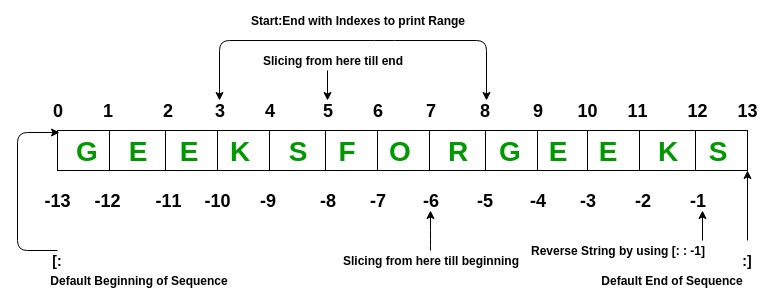
[1, 2, 4]

# Slicing of a List

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [Slicing of a List](https://dashboard.stige.in/index.php/topic/slicing-of-a-list/)

In Progress

In Python List, there are multiple ways to print the whole List with all the elements, but to print a specific range of elements from the list, we use [Slice operation](https://www.geeksforgeeks.org/python-list-comprehension-and-slicing/). Slice operation is performed on Lists with the use of a colon(:). To print elements from beginning to a range use [: Index], to print elements from end-use [:-Index], to print elements from specific Index till the end use [Index:], to print elements within a range, use [Start Index:End Index] and to print the whole List with the use of slicing operation, use [:]. Further, to print the whole List in reverse order, use [::-1].



**Note –** To print elements of List from rear end, use Negative Indexes.

|  |
| --- |
| # Python program to demonstrate  # Removal of elements in a List   # Creating a List  List = ['G','E','E','K','S','F', 'O','R','G','E','E','K','S'] print("Intial List: ") print(List)   # Print elements of a range # using Slice operation  Sliced\_List = List[3:8] print("\nSlicing elements in a range 3-8: ") print(Sliced\_List)   # Print elements from a  # pre-defined point to end  Sliced\_List = List[5:] print("\nElements sliced from 5th " "element till the end: ") print(Sliced\_List)   # Printing elements from # beginning till end  Sliced\_List = List[:] print("\nPrinting all elements using slice operation: ") print(Sliced\_List) |

**Output:**

Intial List:

['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

Slicing elements in a range 3-8:

['K', 'S', 'F', 'O', 'R']

Elements sliced from 5th element till the end:

['F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

Printing all elements using slice operation:

['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

### Negative index List slicing

|  |
| --- |
| # Creating a List  List = ['G','E','E','K','S','F', 'O','R','G','E','E','K','S'] print("Initial List: ") print(List)   # Print elements from beginning # to a pre-defined point using Slice Sliced\_List = List[:-6] print("\nElements sliced till 6th element from last: ") print(Sliced\_List)   # Print elements of a range # using negative index List slicing Sliced\_List = List[-6:-1] print("\nElements sliced from index -6 to -1") print(Sliced\_List)   # Printing elements in reverse # using Slice operation Sliced\_List = List[::-1] print("\nPrinting List in reverse: ") print(Sliced\_List) |

**Output:**

Initial List:

['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

Elements sliced till 6th element from last:

['G', 'E', 'E', 'K', 'S', 'F', 'O']

Elements sliced from index -6 to -1

['R', 'G', 'E', 'E', 'K']

Printing List in reverse:

['S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E', 'G']

# List Comprehension

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [List Comprehension](https://dashboard.stige.in/index.php/topic/list-comprehension/)

In Progress

[**List comprehensions**](https://www.geeksforgeeks.org/python-list-comprehension/) are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.

A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

**Syntax:**

newList = [ expression(element) for element in oldList if condition ]

**Example:**

# Python program to demonstrate list

# comprehension in Python

# below list contains square of all

# odd numbers from range 1 to 10

odd\_square = [x \*\* 2 for x in range(1, 11) if x % 2 == 1]

print (odd\_square)

**Output:**

[1, 9, 25, 49, 81]

For better understanding the above code is similar to –

|  |
| --- |
| # for understanding, above generation is same as,  odd\_square = []    for x in range(1, 11):  if x % 2 == 1:  odd\_square.append(x\*\*2)    print (odd\_square) |

**Output:**

[1, 9, 25, 49, 81]

# List Methods

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Lists](https://dashboard.stige.in/index.php/lessons/task-3-lists/) [List Methods](https://dashboard.stige.in/index.php/topic/list-methods/)

In Progress

|  |  |
| --- | --- |
| **Function** | **Description** |
| [Append()](https://www.geeksforgeeks.org/append-extend-python/) | Add an element to the end of the list |
| [Extend()](https://www.geeksforgeeks.org/append-extend-python/) | Add all elements of a list to the another list |
| [Insert()](https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/) | Insert an item at the defined index |
| [Remove()](https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/) | Removes an item from the list |
| [Pop()](https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/) | Removes and returns an element at the given index |
| [Clear()](https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/) | Removes all items from the list |
| [Index()](https://www.geeksforgeeks.org/python-list-index/) | Returns the index of the first matched item |
| [Count()](https://www.geeksforgeeks.org/python-list-function-count/) | Returns the count of number of items passed as an argument |
| [Sort()](https://www.geeksforgeeks.org/sort-in-python/) | Sort items in a list in ascending order |
| [Reverse()](https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/) | Reverse the order of items in the list |
| [copy()](https://www.geeksforgeeks.org/python-list-copy-method/) | Returns a copy of the list |

# Task 4: Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/)

In Progress

A Tuple is a collection of Python objects separated by commas. In some ways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists which are mutable.

# Creating Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Creating Tuples](https://dashboard.stige.in/index.php/topic/creating-tuples/)

In Progress

|  |
| --- |
| # An empty tuple empty\_tuple = () print (empty\_tuple) |

Output:

()

# Creating non-empty tuples

# One way of creation

tup = 'python', 'geeks'

print(tup)

# Another for doing the same

tup = ('python', 'geeks')

print(tup)

Output

('python', 'geeks')

('python', 'geeks')

***Note:*** In case your generating a tuple with a single element, make sure to add a comma after the element.

# Concatenation of Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Concatenation of Tuples](https://dashboard.stige.in/index.php/topic/concatenation-of-tuples/)

In Progress

# Code for concatenating 2 tuples

tuple1 = (0, 1, 2, 3)

tuple2 = ('python', 'geek')

# Concatenating above two

print(tuple1 + tuple2)

Output:

(0, 1, 2, 3, 'python', 'geek')

# Nesting of Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Nesting of Tuples](https://dashboard.stige.in/index.php/topic/nesting-of-tuples/)

In Progress

# Code for creating nested tuples

tuple1 = (0, 1, 2, 3)

tuple2 = ('python', 'geek')

tuple3 = (tuple1, tuple2)

print(tuple3)

Output :

((0, 1, 2, 3), ('python', 'geek'))

# Repetition in Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Repetition in Tuples](https://dashboard.stige.in/index.php/topic/repetition-in-tuples/)

In Progress

# Code to create a tuple with repetition

tuple3 = ('python',)\*3

print(tuple3)

Output

('python', 'python', 'python')

Try the above without a comma and check. You will get tuple3 as a string ‘pythonpythonpython’.

# Immutable Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Immutable Tuples](https://dashboard.stige.in/index.php/topic/immutable-tuples/)

In Progress

#code to test that tuples are immutable

tuple1 = (0, 1, 2, 3)

tuple1[0] = 4

print(tuple1)

Output

Traceback (most recent call last):

File "e0eaddff843a8695575daec34506f126.py", line 3, in

tuple1[0]=4

TypeError: 'tuple' object does not support item assignment

# Slicing in Tuples

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Slicing in Tuples](https://dashboard.stige.in/index.php/topic/slicing-in-tuples/)

In Progress

# code to test slicing

tuple1 = (0 ,1, 2, 3)

print(tuple1[1:])

print(tuple1[::-1])

print(tuple1[2:4])

Output

(1, 2, 3)

(3, 2, 1, 0)

(2, 3)

# Deleting a Tuple

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Deleting a Tuple](https://dashboard.stige.in/index.php/topic/deleting-a-tuple/)

In Progress

# Code for deleting a tuple

tuple3 = ( 0, 1)

del tuple3

print(tuple3)

Error:

Traceback (most recent call last):

File "d92694727db1dc9118a5250bf04dafbd.py", line 6, in <module>

print(tuple3)

NameError: name 'tuple3' is not defined

Output:

(0, 1)

# Finding Length of a Tuple

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Finding Length of a Tuple](https://dashboard.stige.in/index.php/topic/finding-length-of-a-tuple/)

In Progress

# Code for printing the length of a tuple

tuple2 = ('python', 'geek')

print(len(tuple2))

Output

2

# Converting list to a Tuple

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Converting list to a Tuple](https://dashboard.stige.in/index.php/topic/converting-list-to-a-tuple/)

In Progress

# Code for converting a list and a string into a tuple

list1 = [0, 1, 2]

print(tuple(list1))

print(tuple('python')) # string 'python'

Output

(0, 1, 2)

('p', 'y', 't', 'h', 'o', 'n')

Takes a single parameter which may be a list,string,set or even a dictionary( only keys are taken as elements) and converts them to a tuple.

# Tuples in a loop

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Tuples](https://dashboard.stige.in/index.php/lessons/task-4-tuples/) [Tuples in a loop](https://dashboard.stige.in/index.php/topic/tuples-in-a-loop/)

In Progress

#python code for creating tuples in a loop

tup = ('geek',)

n = 5 #Number of time loop runs

for i in range(int(n)):

tup = (tup,)

print(tup)

Output :

(('geek',),)

((('geek',),),)

(((('geek',),),),)

((((('geek',),),),),)

(((((('geek',),),),),),)

# Task 5: Dictionaries in Python

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/)

In Progress

**Dictionary**in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds **key:value** pair. **Key value** is provided in the dictionary to make it more optimized.

**Note –** Keys in a dictionary doesn’t allows Polymorphism.

# Creating a Dictionary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Creating a Dictionary](https://dashboard.stige.in/index.php/topic/creating-a-dictionary/)

In Progress

In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.

**Note –** Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

# Creating a Dictionary

# with Integer Keys

Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print("\nDictionary with the use of Integer Keys: ")

print(Dict)

# Creating a Dictionary

# with Mixed keys

Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}

print("\nDictionary with the use of Mixed Keys: ")

print(Dict)

**Output:**

Dictionary with the use of Integer Keys:

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with the use of Mixed Keys:

{1: [1, 2, 3, 4], 'Name': 'Geeks'}

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)

# Creating a Dictionary

# with dict() method

Dict = dict({1: 'Geeks', 2: 'For', 3:'Geeks'})

print("\nDictionary with the use of dict(): ")

print(Dict)

# Creating a Dictionary

# with each item as a Pair

Dict = dict([(1, 'Geeks'), (2, 'For')])

print("\nDictionary with each item as a pair: ")

print(Dict)# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)

# Creating a Dictionary

# with dict() method

Dict = dict({1: 'Geeks', 2: 'For', 3:'Geeks'})

print("\nDictionary with the use of dict(): ")

print(Dict)

# Creating a Dictionary

# with each item as a Pair

Dict = dict([(1, 'Geeks'), (2, 'For')])

print("\nDictionary with each item as a pair: ")

print(Dict)

**Output:**

Empty Dictionary:

{}

Dictionary with the use of dict():

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

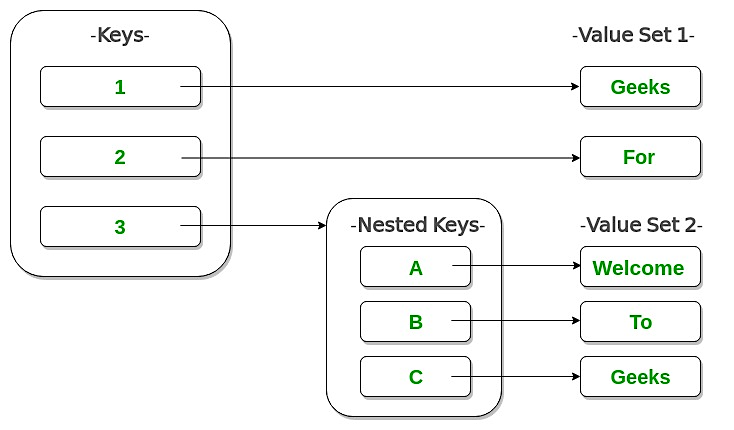
Dictionary with each item as a pair:

{1: 'Geeks', 2: 'For'}

# Nested Dictionary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Nested Dictionary](https://dashboard.stige.in/index.php/topic/nested-dictionary/)

In Progress



# Creating a Nested Dictionary

# as shown in the below image

Dict = {1: 'Geeks', 2: 'For',

3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

print(Dict)

**Output:**

{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

# Adding elements to a Dictionary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Adding elements to a Dictionary](https://dashboard.stige.in/index.php/topic/adding-elements-to-a-dictionary/)

In Progress

In Python Dictionary, Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = ‘Value’. Updating an existing value in a Dictionary can be done by using the built-in **update()** method. Nested key values can also be added to an existing Dictionary.

**Note-** While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)

# Adding elements one at a time

Dict[0] = 'Geeks'

Dict[2] = 'For'

Dict[3] = 1

print("\nDictionary after adding 3 elements: ")

print(Dict)

# Adding set of values

# to a single Key

Dict['Value\_set'] = 2, 3, 4

print("\nDictionary after adding 3 elements: ")

print(Dict)

# Updating existing Key's Value

Dict[2] = 'Welcome'

print("\nUpdated key value: ")

print(Dict)

# Adding Nested Key value to Dictionary

Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}

print("\nAdding a Nested Key: ")

print(Dict)

**Output:**

Empty Dictionary:

{}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1, 'Value\_set': (2, 3, 4)}

Updated key value:

{0: 'Geeks', 2: 'Welcome', 3: 1, 'Value\_set': (2, 3, 4)}

Adding a Nested Key:

{0: 'Geeks', 2: 'Welcome', 3: 1, 5: {'Nested': {'1': 'Life', '2': 'Geeks'}}, 'Value\_set': (2, 3, 4)}

# Accessing elements from a Dictionary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Accessing elements from a Dictionary](https://dashboard.stige.in/index.php/topic/accessing-elements-from-a-dictionary/)

In Progress

In order to access the items of a dictionary refer to its key name.Key can be used inside square brackets.

# Python program to demonstrate

# accessing a element from a Dictionary

# Creating a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key

print("Accessing a element using key:")

print(Dict['name'])

# accessing a element using key

print("Accessing a element using key:")

print(Dict[1])

**Output:**

Accessing a element using key:

For

Accessing a element using key:

Geeks

There is also a method called [**get()**](https://www.geeksforgeeks.org/get-method-dictionaries-python/) that will also help in acessing the element from a dictionary.

# Creating a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using get()

# method

print("Accessing a element using get:")

print(Dict.get(3))

**Output:**

Accessing a element using get:

Geeks

#### Accessing element of a nested dictionary

In order to access the value of any key in nested dictionary, use indexing [] syntax.

# Creating a Dictionary

Dict = {'Dict1': {1: 'Geeks'},

'Dict2': {'Name': 'For'}}

# Accessing element using key

print(Dict['Dict1'])

print(Dict['Dict1'][1])

print(Dict['Dict2']['Name'])

**Output:**

{1: 'Geeks'}

Geeks

For

# Removing Elements from Dictionary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Removing Elements from Dictionary](https://dashboard.stige.in/index.php/topic/removing-elements-from-dictionary/)

In Progress

#### Using del keyword

In Python Dictionary, deletion of keys can be done by using the **del**keyword. Using del keyword, specific values from a dictionary as well as whole dictionary can be deleted. Items in a Nested dictionary can also be deleted by using del keyword and providing specific nested key and particular key to be deleted from that nested Dictionary.

**Note-** **del Dict** will delete the entire dictionary and hence printing it after deletion will raise an Error.

# Initial Dictionary

Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Geeks',

'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},

'B' : {1 : 'Geeks', 2 : 'Life'}}

print("Initial Dictionary: ")

print(Dict)

# Deleting a Key value

del Dict[6]

print("\nDeleting a specific key: ")

print(Dict)

# Deleting a Key from

# Nested Dictionary

del Dict['A'][2]

print("\nDeleting a key from Nested Dictionary: ")

print(Dict)

**Output:**

Initial Dictionary:

{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 6: 'To', 7: 'Geeks'}

Deleting a specific key:

{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 7: 'Geeks'}

Deleting a key from Nested Dictionary:

{'A': {1: 'Geeks', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 7: 'Geeks'}

#### Using pop() method

[Pop(](https://www.geeksforgeeks.org/python-dictionary-pop-method/)) method is used to return and delete the value of the key specified.

# Creating a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Deleting a key

# using pop() method

pop\_ele = Dict.pop(1)

print('\nDictionary after deletion: ' + str(Dict))

print('Value associated to poped key is: ' + str(pop\_ele))

**Output:**

Dictionary after deletion: {3: 'Geeks', 'name': 'For'}

Value associated to poped key is: Geeks

#### Using popitem() method

The popitem() returns and removes an arbitrary element (key, value) pair from the dictionary.

# Creating Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Deleting an arbitrary key

# using popitem() function

pop\_ele = Dict.popitem()

print("\nDictionary after deletion: " + str(Dict))

print("The arbitrary pair returned is: " + str(pop\_ele))

**Output:**

Dictionary after deletion: {3: 'Geeks', 'name': 'For'}

The arbitrary pair returned is: (1, 'Geeks')

#### Using clear() method

All the items from a dictionary can be deleted at once by using **clear()** method.

# Creating a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Deleting entire Dictionary

Dict.clear()

print("\nDeleting Entire Dictionary: ")

print(Dict)

**Output:**

Deleting Entire Dictionary:

{}

# Dictionary Methods

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Dictionaries in Python](https://dashboard.stige.in/index.php/lessons/task-5-dictionaries-in-python/) [Dictionary Methods](https://dashboard.stige.in/index.php/topic/dictionary-methods/)

In Progress

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [copy()](https://www.geeksforgeeks.org/python-dictionary-copy/) | They copy() method returns a shallow copy of the dictionary. |
| [clear()](https://www.geeksforgeeks.org/python-dictionary-clear/) | The clear() method removes all items from the dictionary. | [pop()](https://www.geeksforgeeks.org/python-dictionary-pop-method/) | Removes and returns an element from a dictionary having the given key. | [popitem()](https://www.geeksforgeeks.org/python-dictionary-popitem-method/) | Removes the arbitrary key-value pair from the dictionary and returns it as tuple. |
| [get()](https://www.geeksforgeeks.org/get-method-dictionaries-python/) | It is a conventional method to access a value for a key. |  |  |  |  |
| [dictionary\_name.values()](https://www.geeksforgeeks.org/python-dictionary-values/) | returns a list of all the values available in a given dictionary. |  |  |  |  |
| str() | Produces a printable string representation of a dictionary. |  |  |  |  |
| [update()](https://www.geeksforgeeks.org/python-dictionary-update-method/) | Adds dictionary dict2’s key-values pairs to dict |  |  |  |  |
| [setdefault()](https://www.geeksforgeeks.org/python-dictionary-setdefault-method/) | Set dict[key]=default if key is not already in dict |  |  |  |  |
| [keys()](https://www.geeksforgeeks.org/python-dictionary-keys-method/) | Returns list of dictionary dict’s keys |  |  |  |  |
| [items()](https://www.geeksforgeeks.org/python-dictionary-items-method/) | Returns a list of dict’s (key, value) tuple pairs |  |  |  |  |
| [has\_key()](https://www.geeksforgeeks.org/python-dictionary-has_key/) | Returns true if key in dictionary dict, false otherwise |  |  |  |  |
| [fromkeys()](https://www.geeksforgeeks.org/python-dictionary-fromkeys-method/) | Create a new dictionary with keys from seq and values set to value. |  |  |  |  |
| [type()](https://www.geeksforgeeks.org/python-type-function/) | Returns the type of the passed variable. |  |  |  |  |
| [cmp()](https://www.geeksforgeeks.org/dictionary-methods-in-python-set-1-cmp-len-items/) | Compares elements of both dict. |  |  |  |  |

# Task 6: Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/)

In Progress

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements. Python’s set class represents the mathematical notion of a set. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a [hash table](https://www.geeksforgeeks.org/hashing-set-1-introduction/). Since sets are unordered, we cannot access items using indexes like we do in [lists](https://www.geeksforgeeks.org/python-list/).

# Python program to

# demonstrate sets

# Same as {"a", "b", "c"}

myset = set(["a", "b", "c"])

print(myset)

# Adding element to the set

myset.add("d")

print(myset)

**Output:**

{'c', 'b', 'a'}

{'d', 'c', 'b', 'a'}

# Frozen Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/) [Frozen Sets](https://dashboard.stige.in/index.php/topic/frozen-sets/)

In Progress

**Frozen sets** in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.  
If no parameters are passed, it returns an empty frozenset.

# Python program to demonstrate differences

# between normal and frozen set

# Same as {"a", "b","c"}

normal\_set = set(["a", "b","c"])

print("Normal Set")

print(normal\_set)

# A frozen set

frozen\_set = frozenset(["e", "f", "g"])

print("\nFrozen Set")

print(frozen\_set)

# Uncommenting below line would cause error as

# we are trying to add element to a frozen set

# frozen\_set.add("h")

**Output:**

Normal Set

set(['a', 'c', 'b'])

Frozen Set

frozenset(['e', 'g', 'f'])

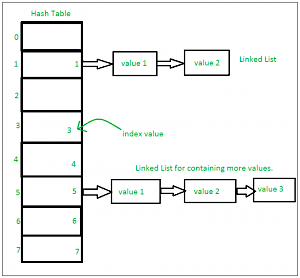
# Internal working of Set

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/) [Internal working of Set](https://dashboard.stige.in/index.php/topic/internal-working-of-set/)

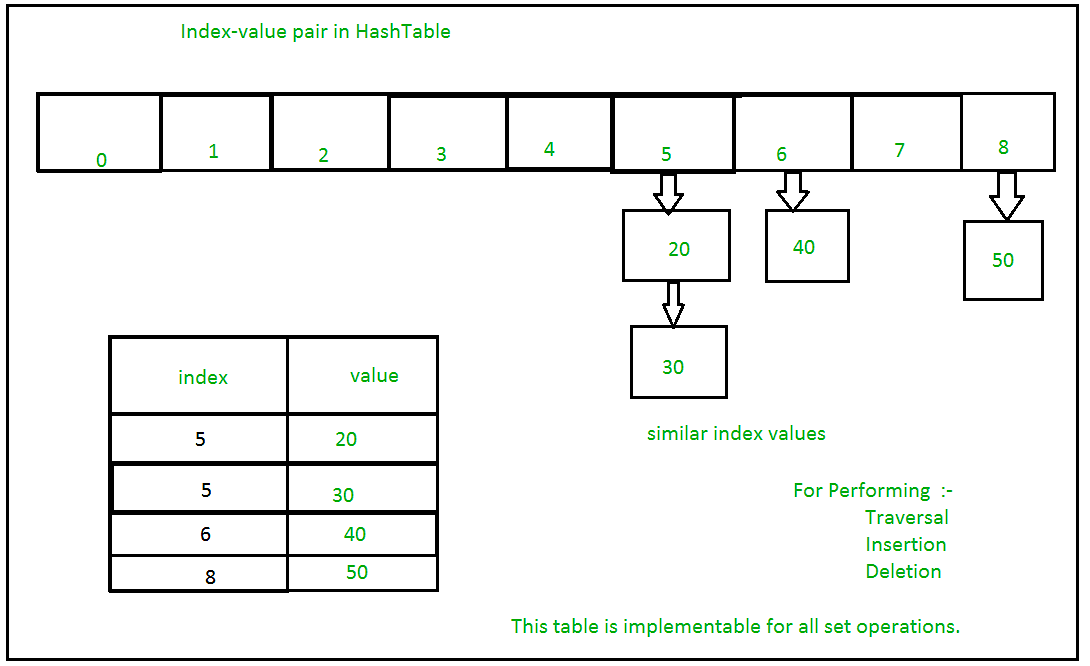
In Progress

This is based on a data structure known as a [hash table](https://www.geeksforgeeks.org/hashing-set-1-introduction/).  
If Multiple values are present at the same index position, then the value is appended to that index position, to form a Linked List. In, Python Sets are implemented using dictionary with dummy variables, where key beings the members set with greater optimizations to the time complexity.

**Set Implementation:-**



**Sets with Numerous operations on a single HashTable:-**



# Methods for Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/) [Methods for Sets](https://dashboard.stige.in/index.php/topic/methods-for-sets/)

In Progress

Insertion in set is done through set.add() function, where an appropriate record value is created to store in the hash table. Same as checking for an item, i.e., O(1) on average. However, in worst case it can become O(n).

# A Python program to

# demonstrate adding elements

# in a set

# Creating a Set

people = {"Jay", "Idrish", "Archi"}

print("People:", end = " ")

print(people)

# This will add Daxit

# in the set

people.add("Daxit")

# Adding elements to the

# set using iterator

for i in range(1, 6):

people.add(i)

print("\nSet after adding element:", end = " ")

print(people)

**Output:**

People: {'Idrish', 'Archi', 'Jay'}

Set after adding element: {1, 2, 3, 4, 5, 'Idrish', 'Archi', 'Jay', 'Daxit'}

#### Union

Two sets can be merged using union() function or | operator. Both Hash Table values are accessed and traversed with merge operation perform on them to combine the elements, at the same time duplicates are removed. Time Complexity of this is O(len(s1) + len(s2)) where s1 and s2 are two sets whose union needs to be done.

# Python Program to

# demonstrate union of

# two sets

people = {"Jay", "Idrish", "Archil"}

vampires = {"Karan", "Arjun"}

dracula = {"Deepanshu", "Raju"}

# Union using union()

# function

population = people.union(vampires)

print("Union using union() function")

print(population)

# Union using "|"

# operator

population = people|dracula

print("\nUnion using '|' operator")

print(population)

**Output:**

Union using union() function

{'Karan', 'Idrish', 'Jay', 'Arjun', 'Archil'}

Union using '|' operator

{'Deepanshu', 'Idrish', 'Jay', 'Raju', 'Archil'}

#### Intersection

This can be done through intersection() or & operator. Common Elements are selected. They are similar to iteration over the Hash lists and combining the same values on both the Table. Time Complexity of this is O(min(len(s1), len(s2)) where s1 and s2 are two sets whose union needs to be done.

# Python program to

# demonstrate intersection

# of two sets

set1 = set()

set2 = set()

for i in range(5):

set1.add(i)

for i in range(3,9):

set2.add(i)

# Intersection using

# intersection() function

set3 = set1.intersection(set2)

print("Intersection using intersection() function")

print(set3)

# Intersection using

# "&" operator

set3 = set1 & set2

print("\nIntersection using '&' operator")

print(set3)

**Output:**

Intersection using intersection() function

{3, 4}

Intersection using '&' operator

{3, 4}

#### Difference

To find difference in between sets. Similar to find difference in linked list. This is done through difference() or – operator. Time complexity of finding difference s1 – s2 is O(len(s1))

# Python program to

# demonstrate difference

# of two sets

set1 = set()

set2 = set()

for i in range(5):

set1.add(i)

for i in range(3,9):

set2.add(i)

# Difference of two sets

# using difference() function

set3 = set1.difference(set2)

print(" Difference of two sets using difference() function")

print(set3)

# Difference of two sets

# using '-' operator

set3 = set1 - set2

print("\nDifference of two sets using '-' operator")

print(set3)

**Output:**

Difference of two sets using difference() function

{0, 1, 2}

Difference of two sets using '-' operator

{0, 1, 2}

#### Clearing sets

Clear() method empties the whole set.

# Python program to

# demonstrate clearing

# of set

set1 = {1,2,3,4,5,6}

print("Initial set")

print(set1)

# This method will remove

# all the elements of the set

set1.clear()

print("\nSet after using clear() function")

print(set1)

**Output:**

Initial set

{1, 2, 3, 4, 5, 6}

Set after using clear() function

set()

However, there are two major pitfalls in Python sets:

1. The set doesn’t maintain elements in any particular order.
2. Only instances of immutable types can be added to a Python set.

# Time complexity of Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/) [Time complexity of Sets](https://dashboard.stige.in/index.php/topic/time-complexity-of-sets/)

In Progress

| **Operation** | **Average case** | **Worst Case** | **notes** |
| --- | --- | --- | --- |
|  |  |  |  |
| x in s | O(1) | O(n) |  |
| Union s|t | O(len(s)+len(t)) |  |  |
| Intersection s&t | O(min(len(s), len(t)) | O(len(s) \* len(t)) | replace “min” with “max” if t is not a set |
| Multiple intersection s1&s2&..&sn |  | (n-1)\*O(l) where l is max(len(s1),..,len(sn)) |  |
| Difference s-t | O(len(s)) |  |  |

# Operators for Sets

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Sets](https://dashboard.stige.in/index.php/lessons/task-6-sets/) [Operators for Sets](https://dashboard.stige.in/index.php/topic/operators-for-sets/)

In Progress

Sets and frozen sets support the following operators:

| **Operators** | **Notes** |
| --- | --- |
| key in s | containment check |
| key not in s | non-containment check |
| s1 == s2 | s1 is equivalent to s2 |
| s1 != s2 | s1 is not equivalent to s2 |
| s1 <= s2 | s1 is subset of s2 |
| s1 < s2 | s1 is proper subset of s2 |
| s1 >= s2 | s1 is superset of s2 |
| s1 > s2 | s1 is proper superset of s2 |
| s1 | s2 | the union of s1 and s2 |
| s1 & s2 | the intersection of s1 and s2 |
| s1 – s2 | the set of elements in s1 but not s2 |
| s1 ˆ s2 | the set of elements in precisely one of s1 or s2 |

#### Code Snippet to illustrate all Set operations in Python

# Python program to demonstrate working# of

# Set in Python

# Creating two sets

set1 = set()

set2 = set()

# Adding elements to set1

for i in range(1, 6):

set1.add(i)

# Adding elements to set2

for i in range(3, 8):

set2.add(i)

print("Set1 = ", set1)

print("Set2 = ", set2)

print("\n")

# Union of set1 and set2

set3 = set1 | set2# set1.union(set2)

print("Union of Set1 & Set2: Set3 = ", set3)

# Intersection of set1 and set2

set4 = set1 & set2# set1.intersection(set2)

print("Intersection of Set1 & Set2: Set4 = ", set4)

print("\n")

# Checking relation between set3 and set4

if set3 > set4: # set3.issuperset(set4)

print("Set3 is superset of Set4")

elif set3 < set4: # set3.issubset(set4)

print("Set3 is subset of Set4")

else : # set3 == set4

print("Set3 is same as Set4")

# displaying relation between set4 and set3

if set4 < set3: # set4.issubset(set3)

print("Set4 is subset of Set3")

print("\n")

# difference between set3 and set4

set5 = set3 - set4

print("Elements in Set3 and not in Set4: Set5 = ", set5)

print("\n")

# checkv if set4 and set5 are disjoint sets

if set4.isdisjoint(set5):

print("Set4 and Set5 have nothing in common\n")

# Removing all the values of set5

set5.clear()

print("After applying clear on sets Set5: ")

print("Set5 = ", set5)

**Output:**

('Set1 = ', set([1, 2, 3, 4, 5]))

('Set2 = ', set([3, 4, 5, 6, 7]))

('Union of Set1 & Set2: Set3 = ', set([1, 2, 3, 4, 5, 6, 7]))

('Intersection of Set1 & Set2: Set4 = ', set([3, 4, 5]))

Set3 is superset of Set4

Set4 is subset of Set3

('Elements in Set3 and not in Set4: Set5 = ', set([1, 2, 6, 7]))

Set4 and Set5 have nothing in common

After applying clear on sets Set5:

('Set5 = ', set([]))

# Task 7: Control Structures

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Control Structures](https://dashboard.stige.in/index.php/lessons/task-7-control-structures/)

In Progress

Python programming language provides following types of loops to handle looping requirements.

# While Loop

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Control Structures](https://dashboard.stige.in/index.php/lessons/task-7-control-structures/) [While Loop](https://dashboard.stige.in/index.php/topic/while-loop/)

In Progress

Syntax :

while expression:

statement(s)

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

# prints Hello Geek 3 Times

count = 0

while (count < 3):

count = count+1

print("Hello Geek")

**Output:**

Hello Geek

Hello Geek

Hello Geek

See [this](https://www.geeksforgeeks.org/using-iterations-in-python-effectively/) for an example where while loop is used for iterators. As mentioned in the article, it is not recommended to use while loop for iterators in python.

# For in Loop

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Control Structures](https://dashboard.stige.in/index.php/lessons/task-7-control-structures/) [For in Loop](https://dashboard.stige.in/index.php/topic/for-in-loop/)

In Progress

In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to [for each](https://www.geeksforgeeks.org/g-fact-40-foreach-in-c-and-java/) loop in other languages.

Syntax:

for iterator\_var in sequence:

statements(s)

It can be used to iterate over iterators and a range.

# Iterating over a list

print("List Iteration")

l = ["geeks", "for", "geeks"]

for i in l:

print(i)

# Iterating over a tuple (immutable)

print("\nTuple Iteration")

t = ("geeks", "for", "geeks")

for i in t:

print(i)

# Iterating over a String

print("\nString Iteration")

s = "Geeks"

for i in s :

print(i)

# Iterating over dictionary

print("\nDictionary Iteration")

d = dict()

d['xyz'] = 123

d['abc'] = 345

for i in d :

print("%s %d" %(i, d[i]))

Output:

List Iteration

geeks

for

geeks

Tuple Iteration

geeks

for

geeks

String Iteration

G

e

e

k

s

Dictionary Iteration

xyz 123

abc 345

We can use for in loop for user defined iterators. See [this](https://www.geeksforgeeks.org/iterators-in-python/) for example.

# Nested Loops

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Control Structures](https://dashboard.stige.in/index.php/lessons/task-7-control-structures/) [Nested Loops](https://dashboard.stige.in/index.php/topic/nested-loops/)

In Progress

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.  
Syntax:  
https://5bda5a6ee8ddd59a62f310f0631fe36c.safeframe.googlesyndication.com/safeframe/1-0-38/html/container.html

for iterator\_var in sequence:

for iterator\_var in sequence:

statements(s)

statements(s)

The syntax for a nested while loop statement in Python programming language is as follows:

while expression:

while expression:

statement(s)

statement(s)

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

from \_\_future\_\_ import print\_function

for i in range(1, 5):

for j in range(i):

print(i, end=' ')

print()

**Output:**

1

2 2

3 3 3

4 4 4 4

# Loop Control Statements

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Control Structures](https://dashboard.stige.in/index.php/lessons/task-7-control-structures/) [Loop Control Statements](https://dashboard.stige.in/index.php/topic/loop-control-statements/)

In Progress

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.

#### **Continue Statement**

It returns the control to the beginning of the loop.

# Prints all letters except 'e' and 's'

for letter in 'geeksforgeeks':

if letter == 'e' or letter == 's':

continue

print 'Current Letter :', letter

var = 10

**Output:**

Current Letter : g

Current Letter : k

Current Letter : f

Current Letter : o

Current Letter : r

Current Letter : g

Current Letter : k

#### **Break Statement**

It brings control out of the loop.

for letter in 'geeksforgeeks':

# break the loop as soon it sees 'e'

# or 's'

if letter == 'e' or letter == 's':

break

print 'Current Letter :', letter

**Output:**

Current Letter : e

#### **Pass Statement**

We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

# An empty loop

for letter in 'geeksforgeeks':

pass

print 'Last Letter :', letter

Output:

Last Letter : s

# Task 8: Loops

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/)

In Progress

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

# While Loop

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/) [While Loop](https://dashboard.stige.in/index.php/topic/while-loop-2/)

In Progress

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

**Syntax** :

while expression:

statement(s)

All the statements indented by the same number of character spaces after a programming construct are considered to be  part of a single block of code. Python uses indentation as its method of grouping statements.

**Example:**

# Python program to illustrate

# while loop

count = 0

while (count < 3):

count = count + 1

print("Hello Geek")

**Output:**

Hello Geek

Hello Geek

Hello Geek

**Using else statement with while loops:** As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.   
The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won’t be executed.   
**If else like this:**

if condition:

# execute these statements

else:

# execute these statements

**and while loop like this are similar**

while condition:

# execute these statements

else:

# execute these statements

#Python program to illustrate

# combining else with while

count = 0

while (count < 3):

count = count + 1

print("Hello Geek")

else:

print("In Else Block")

**Output:**

Hello Geek

Hello Geek

Hello Geek

In Else Block

**Single statement while block:**Just like the if block, if the while block consists of a single statement the we can declare the entire loop in a single line as shown below:

# Python program to illustrate

# Single statement while block

count = 0

while (count == 0):

print("Hello Geek")

**Note**: It is suggested **not to use** this type of loops as it is a never ending infinite loop where the condition is always true and you have to forcefully terminate the compiler.

**For in Loop**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/) [For in Loop](https://dashboard.stige.in/index.php/topic/for-in-loop-2/)

In Progress

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

**Syntax:**

for iterator\_var in sequence:

statements(s)

It can be used to iterate over a range and iterators.

# Python program to illustrate

# Iterating over range 0 to n-1

n = 4

for i in range(0, n):

print(i)

**Output :**

0

1

2

3

# Python program to illustrate

# Iterating over a list

print("List Iteration")

l = ["geeks", "for", "geeks"]

for i in l:

print(i)

# Iterating over a tuple (immutable)

print("\nTuple Iteration")

t = ("geeks", "for", "geeks")

for i in t:

print(i)

# Iterating over a String

print("\nString Iteration")

s = "Geeks"

for i in s :

print(i)

# Iterating over dictionary

print("\nDictionary Iteration")

d = dict()

d['xyz'] = 123

d['abc'] = 345

for i in d :

print("%s %d" %(i, d[i]))

**Output:**

List Iteration

geeks

for

geeks

Tuple Iteration

geeks

for

geeks

String Iteration

G

e

e

k

s

Dictionary Iteration

xyz 123

abc 345

**Iterating by index of sequences**: We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and in iterate over the sequence within the range of this length.   
See the below example:

# Python program to illustrate

# Iterating by index

list = ["geeks", "for", "geeks"]

for index in range(len(list)):

print list[index]

**Output:**

geeks

for

geeks

**Using else statement with for loops:**We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.   
Below example explains how to do this:

# Python program to illustrate

# combining else with for

list = ["geeks", "for", "geeks"]

for index in range(len(list)):

print list[index]

else:

print "Inside Else Block"

**Output:**

geeks

for

geeks

Inside Else Block

# Nested Loops

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/) [Nested Loops](https://dashboard.stige.in/index.php/topic/nested-loops-2/)

In Progress

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

for iterator\_var in sequence:

for iterator\_var in sequence:

statements(s)

statements(s)

The syntax for a nested while loop statement in Python programming language is as follows:

while expression:

while expression:

statement(s)

statement(s)

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

# Python program to illustrate

# nested for loops in Python

from \_\_future\_\_ import print\_function

for i in range(1, 5):

for j in range(i):

print(i, end=' ')

print()

**Output:**

1

2 2

3 3 3

4 4 4 4

**Loop Control Statements**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/) [Loop Control Statements](https://dashboard.stige.in/index.php/topic/loop-control-statements-2/)

In Progress

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.

* **Continue Statement:**It returns the control to the beginning of the loop.

# Prints all letters except 'e' and 's'

for letter in 'geeksforgeeks':

if letter == 'e' or letter == 's':

continue

print 'Current Letter :', letter

var = 10

**Output:**

Current Letter : g

Current Letter : k

Current Letter : f

Current Letter : o

Current Letter : r

Current Letter : g

Current Letter : k

* **Break Statement:** It brings control out of the loop

for letter in 'geeksforgeeks':

# break the loop as soon it sees 'e'

# or 's'

if letter == 'e' or letter == 's':

break

print 'Current Letter :', letter

**Output:**

Current Letter : e

* **Pass Statement:**We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

# An empty loop

for letter in 'geeksforgeeks':

pass

print 'Last Letter :', letter

**Output:**

Last Letter : s

**How for loop in Python works internally?**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: Loops](https://dashboard.stige.in/index.php/lessons/task-8-loops/) [How for loop in Python works internally?](https://dashboard.stige.in/index.php/topic/how-for-loop-in-python-works-internally/)

In Progress

Before proceeding to this section, you should have a prior understanding of Python Iterators.

Firstly, lets see how a simple for loop looks like.

# A simple for loop example

fruits = ["apple", "orange", "kiwi"]

for fruit in fruits:

print(fruit)

**Output**

apple

orange

kiwi

Here we can see the for loops iterates over a iterable object fruits which is a list. Lists, sets, dictionary these are few iterable objects while an integer object is not an iterable object.

For loops can iterate over any iterable object (example: List, Set, Dictionary, Tuple or String).

Now with the help of above example lets dive deep and see what happens internally here.

1. Make the list (iterable) an iterable object with help of iter() function.
2. Run a infinite while loop and break only if the StopIteration is raised.
3. In the try block we fetch the next element of fruits with next() function.
4. After fetching the element we did the operation to be performed in with the element. (i.e print(fruit))

fruits = ["apple", "orange", "kiwi"]

# Creating an iterator object

# from that iterable i.e fruits

iter\_obj = iter(fruits)

# Infinite while loop

while True:

try:

# getting the next item

fruit = next(iter\_obj)

print(fruit)

except StopIteration:

# if StopIteration is raised,

# break from loop

break

**Output**

apple

orange

kiwi

We can see that under the hood we are calling iter() and next() method.

**Exercise:**How to print a list in reverse order (from last to first item) using while and for in loops.

**Task 9: Comprehension**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 9: Comprehension](https://dashboard.stige.in/index.php/lessons/task-9-comprehension/)

In Progress

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:

* List Comprehensions
* Dictionary Comprehensions
* Set Comprehensions
* Generator Comprehensions

# List Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 9: Comprehension](https://dashboard.stige.in/index.php/lessons/task-9-comprehension/) [List Comprehensions](https://dashboard.stige.in/index.php/topic/list-comprehensions-2/)

In Progress

**List Comprehensions** provide an elegant way to create new lists. The following is the basic structure of a list comprehension:

output\_list = [output\_exp for var in input\_list if (var satisfies this condition)]

Note that list comprehension may or may not contain an if condition. List comprehensions can contain multiple **for** (nested list comprehensions).

**Example #1:** Suppose we want to create an output list which contains only the even numbers which are present in the input list. Let’s see how to do this using for loops and list comprehension and decide which method suits better.

# Constructing output list WITHOUT

# Using List comprehensions

input\_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

output\_list = []

# Using loop for constructing output list

for var in input\_list:

if var % 2 == 0:

output\_list.append(var)

print("Output List using for loop:", output\_list)

**Output:**

Output List using for loop: [2, 4, 4, 6]

# Using List comprehensions

# for constructing output list

input\_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

list\_using\_comp = [var for var in input\_list if var % 2 == 0]

print("Output List using list comprehensions:",

list\_using\_comp)

**Output:**

Output List using list comprehensions: [2, 4, 4, 6]

**Example #2:** Suppose we want to create an output list which contains squares of all the numbers from 1 to 9. Let’s see how to do this using for loops and list comprehension.

# Constructing output list using for loop

output\_list = []

for var in range(1, 10):

output\_list.append(var \*\* 2)

print("Output List using for loop:", output\_list)

**Output:**

Output List using for loop: [1, 4, 9, 16, 25, 36, 49, 64, 81]

# Constructing output list

# using list comprehension

list\_using\_comp = [var\*\*2 for var in range(1, 10)]

print("Output List using list comprehension:",

list\_using\_comp)

**Output:**

Output List using list comprehension: [1, 4, 9, 16, 25, 36, 49, 64, 81]

# Dictionary Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 9: Comprehension](https://dashboard.stige.in/index.php/lessons/task-9-comprehension/) [Dictionary Comprehensions](https://dashboard.stige.in/index.php/topic/dictionary-comprehensions/)

In Progress

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

output\_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

**Example #1:** Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values. Let’s see how to do this using for loops and dictionary comprehension.

input\_list = [1, 2, 3, 4, 5, 6, 7]

output\_dict = {}

# Using loop for constructing output dictionary

for var in input\_list:

if var % 2 != 0:

output\_dict[var] = var\*\*3

print("Output Dictionary using for loop:",

output\_dict )

**Output:**

Output Dictionary using for loop: {1: 1, 3: 27, 5: 125, 7: 343}

# Using Dictionary comprehensions

# for constructing output dictionary

input\_list = [1,2,3,4,5,6,7]

dict\_using\_comp = {var:var \*\* 3 for var in input\_list if var % 2 != 0}

print("Output Dictionary using dictionary comprehensions:",

dict\_using\_comp)

**Output:**

Output Dictionary using dictionary comprehensions: {1: 1, 3: 27, 5: 125, 7: 343}

**Example #2:** Given two lists containing the names of states and their corresponding capitals, construct a dictionary which maps the states with their respective capitals. Let’s see how to do this using for loops and dictionary comprehension.

state = ['Gujarat', 'Maharashtra', 'Rajasthan']

capital = ['Gandhinagar', 'Mumbai', 'Jaipur']

output\_dict = {}

# Using loop for constructing output dictionary

for (key, value) in zip(state, capital):

output\_dict[key] = value

print("Output Dictionary using for loop:",

output\_dict)

**Output:**

Output Dictionary using for loop: {'Gujarat': 'Gandhinagar',

'Maharashtra': 'Mumbai',

'Rajasthan': 'Jaipur'}

# Using Dictionary comprehensions

# for constructing output dictionary

state = ['Gujarat', 'Maharashtra', 'Rajasthan']

capital = ['Gandhinagar', 'Mumbai', 'Jaipur']

dict\_using\_comp = {key:value for (key, value) in zip(state, capital)}

print("Output Dictionary using dictionary comprehensions:",

dict\_using\_comp)

**Output:**

Output Dictionary using dictionary comprehensions: {'Rajasthan': 'Jaipur',

'Maharashtra': 'Mumbai',

'Gujarat': 'Gandhinagar'}

# Set Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 9: Comprehension](https://dashboard.stige.in/index.php/lessons/task-9-comprehension/) [Set Comprehensions](https://dashboard.stige.in/index.php/topic/set-comprehensions/)

In Progress

**Set comprehensions** are pretty similar to list comprehensions. The only difference between them is that set comprehensions use **curly brackets** { }. Let’s look at the following example to understand set comprehensions.

**Example #1 :** Suppose we want to create an output set which contains only the even numbers that are present in the input list. Note that set will discard all the duplicate values. Let’s see how we can do this using for loops and set comprehension.

input\_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]

output\_set = set()

# Using loop for constructing output set

for var in input\_list:

if var % 2 == 0:

output\_set.add(var)

print("Output Set using for loop:", output\_set)

**Output:**

Output Set using for loop: {2, 4, 6}

# Using Set comprehensions

# for constructing output set

input\_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]

set\_using\_comp = {var for var in input\_list if var % 2 == 0}

print("Output Set using set comprehensions:",

set\_using\_comp)

**Output:**

Output Set using set comprehensions: {2, 4, 6}

# Generator Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 9: Comprehension](https://dashboard.stige.in/index.php/lessons/task-9-comprehension/) [Generator Comprehensions](https://dashboard.stige.in/index.php/topic/generator-comprehensions/)

In Progress

**Generator Comprehensions** are very similar to list comprehensions. One difference between them is that generator comprehensions **use circular brackets** whereas list comprehensions use **square brackets**. The major difference between them is that generators don’t allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient. Let’s look at the following example to understand generator comprehension:

input\_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

output\_gen = (var for var in input\_list if var % 2 == 0)

print("Output values using generator comprehensions:", end = ' ')

for var in output\_gen:

print(var, end = ' ')

**Output:**

Output values using generator comprehensions: 2 4 4 6

# Task 10: Comprehension Visually Explained

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/)

In Progress

Sometimes a programming design pattern becomes common enough to warrant its own special syntax. Python’s [list comprehensions](https://docs.python.org/3/tutorial/datastructures.html#tut-listcomps) are a prime example of such a **syntactic sugar.**

List comprehensions in Python are great, but mastering them can be tricky because they don’t solve a new problem: they just provide a **new syntax** to solve **an existing problem**.

Let’s learn what list comprehensions are and how to identify when to use them.

# What are list comprehensions?

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [What are list comprehensions?](https://dashboard.stige.in/index.php/topic/what-are-list-comprehensions/)

In Progress

List comprehensions are a tool for transforming **one list** (any [iterable](https://docs.python.org/3/glossary.html" \l "term-iterable) actually) **into another list**. During this transformation, elements can be conditionally included in the new list and each element can be transformed as needed.

If you’re familiar with functional programming, you can think of list comprehensions as syntactic sugar for a **filter** followed by a**map**:

>>> doubled\_odds = map(lambda n: n \* 2, filter(lambda n: n % 2 == 1, numbers))

>>> doubled\_odds = [n \* 2 for n in numbers if n % 2 == 1]

# From loops to comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [From loops to comprehensions](https://dashboard.stige.in/index.php/topic/from-loops-to-comprehensions/)

In Progress

Every **list comprehension** can be rewritten as a**for** loop but not every **for**loop can be rewritten as a list comprehension.

The key to understanding when to use list comprehensions is to practice **identifying problems** that **smell**like list comprehensions.

If you can rewrite your code to look just like this***for*** loop, you can also rewrite it as a list comprehension:

new\_things = []

for ITEM in old\_things:

if condition\_based\_on(ITEM):

new\_things.append("something with " + ITEM)

You can rewrite the above for loop as a list comprehension like this:

new\_things = ["something with " + ITEM for ITEM in old\_things if condition\_based\_on(ITEM)]

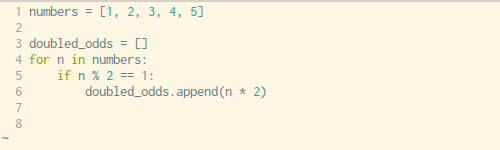
**List Comprehensions: The Animated Movie**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [List Comprehensions: The Animated Movie](https://dashboard.stige.in/index.php/topic/list-comprehensions-the-animated-movie/)

In Progress

That’s great, but how did we do that?

We **copy-pasted** our way from a for loop to a list comprehension.



Here’s the order we copy-paste in:

1. **Copy** the **variable** assignment for our new empty list (line 3)
2. **Copy the expression** that we’ve been **append–**ing into this new list (line 6)
3. **Copy the for loop** line, excluding the final : (line 4)
4. **Copy the if statement** line, also without the : (line 5)

We’ve now copied our way from this:

numbers = [1, 2, 3, 4, 5]

doubled\_odds = []

for n in numbers:

if n % 2 == 1:

doubled\_odds.append(n \* 2)

To this:

numbers = [1, 2, 3, 4, 5] doubled\_odds = [n \* 2 for n in numbers if n % 2 == 1]

**List Comprehensions: Now in Color**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [List Comprehensions: Now in Color](https://dashboard.stige.in/index.php/topic/list-comprehensions-now-in-color/)

In Progress

Let’s use colors to highlight what’s going on.

doubled\_odds = []

for n in numbers:

if n % 2 == 1:

doubled\_odds.append(n \* 2)

doubled\_odds = [n \* 2 for n in numbers if n % 2 == 1]

We copy-paste from a **for** loop into a list comprehension by:

1. Copying the variable assignmentfor our new empty list
2. Copying the expression that we’ve been**append**-ing into this new list
3. Copying the **for** loop line, excluding the final :
4. Copying the **if** statement line, also without the :

**Unconditional Comprehensions**

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [Unconditional Comprehensions](https://dashboard.stige.in/index.php/topic/unconditional-comprehensions/)

In Progress

But what about comprehensions that don’t have a conditional clause (that if SOMETHING part at the end)? These loop-and-append for loops are even simpler than the loop-and-conditionally-append ones we’ve already covered.

A for loop that doesn’t have an if statement:

doubled\_numbers = []

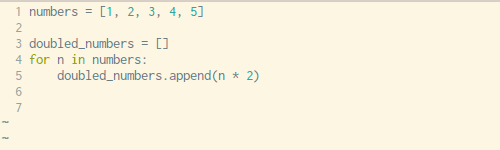
for n in numbers:

doubled\_numbers.append(n \* 2)

That same code written as a comprehension:

doubled\_numbers = [n \* 2 for n in numbers]

Here’s the transformation animated:



We can copy-paste our way from a simple loop-and-append for loop by:

1. Copying the variable assignment for our new empty list (line 3)
2. Copying the expression that we’ve been **append**-ing into this new list (line 5)
3. Copying the **for** loop line, excluding the final : (line 4)

# Nested Loops

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [Nested Loops](https://dashboard.stige.in/index.php/topic/nested-loops-3/)

In Progress

What about list comprehensions with nested looping?…

Here’s a**for** loop that flattens a matrix (a list of lists):

flattened = []

for row in matrix:

for n in row:

flattened.append(n)

Here’s a list comprehension that does the same thing:

flattened = [n for row in matrix for n in row]

Nested loops in list comprehensions do not read like English prose.

**Note:** My brain wants to write this list comprehension as:

flattened = [n for n in row for row in matrix]

**But that’s not right!** I’ve mistakenly flipped the for loops here. The correct version is the one above.

When working with nested loops in list comprehensions remember that **the for clauses remain in the same order** as in our original for loops.

# Other Comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [Other Comprehensions](https://dashboard.stige.in/index.php/topic/other-comprehensions/)

In Progress

This same principle applies to [set comprehensions](https://docs.python.org/3/tutorial/datastructures.html#sets) and [dictionary comprehensions](https://docs.python.org/3/tutorial/datastructures.html#dictionaries).

Code that creates a set of all the first letters in a sequence of words:

first\_letters = set()

for w in words:

first\_letters.add(w[0])

That same code written as a set comprehension:

first\_letters = {w[0] for w in words}

Code that makes a new dictionary by swapping the keys and values of the original one:

flipped = {}

for key, value in original.items():

flipped[value] = key

That same code written as a dictionary comprehension:

flipped = {value: key for key, value in original.items()}

# Readability Counts

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [Readability Counts](https://dashboard.stige.in/index.php/topic/readability-counts/)

In Progress

Did you find the above list comprehensions hard to read? I often find longer list comprehensions very difficult to read when they’re written on one line.

Remember that [Python allows line breaks](https://docs.python.org/3/reference/lexical_analysis.html#implicit-line-joining) between brackets and braces.

### List comprehension

Before

doubled\_odds = [n \* 2 for n in numbers if n % 2 == 1]

After

doubled\_odds = [

n \* 2

for n in numbers

if n % 2 == 1

]

### Nested loops in list comprehension

Before

|  |  |
| --- | --- |
|  | flattened **=** [n for row **in** matrix for n **in** row] |

After

|  |  |
| --- | --- |
|  | flattened = [  n  for row in matrix for n in row  ] |

### Dictionary comprehension

Before

|  |  |
| --- | --- |
|  | flipped **=** {value: key for key, value **in** original**.**items()} |

After

|  |  |
| --- | --- |
|  | flipped **=** {  value: key  for key, value **in** original**.**items()  } |

**Note** – that we are not adding line breaks arbitrarily: we’re breaking between each of the lines of code we copy-pasted to make these comprehension. Our line breaks occur where color changes occur in the colorized versions.

# Copy-paste into comprehensions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 10: Comprehension Visually Explained](https://dashboard.stige.in/index.php/lessons/task-10-comprehension-visually-explained/) [Copy-paste into comprehensions](https://dashboard.stige.in/index.php/topic/copy-paste-into-comprehensions/)

In Progress

When struggling to write a comprehension, don’t panic. Start with a **for** loop first and copy-paste your way into a comprehension.

Any **for** loop that looks like this:

new\_things = []

for ITEM in old\_things:

if condition\_based\_on(ITEM):

new\_things.append("something with " + ITEM)

Can be rewritten into a list comprehension like this:

new\_things = ["something with " + ITEM for ITEM in old\_things if condition\_based\_on(ITEM)]

If you can nudge a **for** loop until it looks like the ones above, you can rewrite it as a list comprehension.

# Task 11: Functions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/)

In Progress

A function in Python is an **aggregation** of related statements designed to perform a **computational**, **logical,** or **evaluative task**. The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function to reuse code contained in it over and over again.

Functions can be **both built-in** or **user-defined.** It helps the program to be concise, non-repetitive, and organized.

**Syntax:**

def function\_name(parameters):

"""docstring"""

statement(s)

**Example:**

# A simple Python function to check

# whether x is even or odd

def evenOdd(x):

if (x % 2 == 0):

print "even"

else:

print "odd"

# Driver code to call the function

evenOdd(2)

evenOdd(3)

**Output**

even

odd

# Docstring

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/) [Docstring](https://dashboard.stige.in/index.php/topic/docstring/)

In Progress

The first string after the function is called the **Document string** or [Docstring](https://www.geeksforgeeks.org/python-docstrings/) in short. This is used to describe the **functionality** of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function:

**Syntax:** print(function\_name.\_\_doc\_\_)

**Example:**

def say\_Hi():

"Hello! geeks!"

print(say\_Hi.\_\_doc\_\_)

**Output:**

Hello! geeks!

# The return statement

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/) [The return statement](https://dashboard.stige.in/index.php/topic/the-return-statement/)

In Progress

The return statement is used to **exit** from a function and go back to the function **caller** and **return** the specified value or data item to the caller.

**Syntax:** return [expression\_list]

The return statement can consist of **a variable**, **an expression**, or a **constant** which is returned to the end of the function execution. If none of the above is present with the return statement a **None** object is returned.

**Example:**

def square\_value(num):

"""This function returns the square

value of the entered number"""

return num\*\*2

print(square\_value(2))

print(square\_value(-4))

**Output:**

4

16

# Pass by Reference or pass by value?

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/) [Pass by Reference or pass by value?](https://dashboard.stige.in/index.php/topic/pass-by-reference-or-pass-by-value/)

In Progress

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

**Example:**

# Here x is a new reference to same list lst

def myFun(x):

x[0] = 20

# Driver Code (Note that lst is modified

# after function call.

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

**Output**

[20, 11, 12, 13, 14, 15]

When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken. For example, consider the below program.

def myFun(x):

# After below line link of x with previous

# object gets broken. A new object is assigned

# to x.

x = [20, 30, 40]

# Driver Code (Note that lst is not modified

# after function call.

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

**Output**

[10, 11, 12, 13, 14, 15]

Another example to demonstrate that the reference link is broken if we assign a new value (inside the function).

def myFun(x):

# After below line link of x with previous

# object gets broken. A new object is assigned

# to x.

x = 20

# Driver Code (Note that lst is not modified

# after function call.

x = 10

myFun(x)

print(x)

**Output**

10

**Exercise:** Try to guess the output of the following code.

def swap(x, y):

temp = x

x = y

y = temp

# Driver code

x = 2

y = 3

swap(x, y)

print(x)

print(y)

**Output**

2

3

# Default arguments

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/) [Default arguments](https://dashboard.stige.in/index.php/topic/default-arguments/)

In Progress

A **default argument** is a parameter that assumes a **default value** if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

# Python program to demonstrate

# default arguments

def myFun(x, y=50):

print("x: ", x)

print("y: ", y)

# Driver code (We call myFun() with only

# argument)

myFun(10)

**Output**

('x: ', 10)

('y: ', 50)

Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have **default values**.

# Keyword arguments

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 11: Functions](https://dashboard.stige.in/index.php/lessons/task-11-functions/) [Keyword arguments](https://dashboard.stige.in/index.php/topic/keyword-arguments/)

In Progress

The idea is to allow the **caller to specify** the argument name with values so that caller does not need to remember the order of parameters.

# Python program to demonstrate Keyword Arguments

def student(firstname, lastname):

print(firstname, lastname)

# Keyword arguments

student(firstname='Geeks', lastname='Practice')

student(lastname='Practice', firstname='Geeks')

**Output**

('Geeks', 'Practice')

('Geeks', 'Practice')

# Anonymous functions

In Python, **an anonymous function** means that a function is **without a name.** As we already know the def keyword is used to define the **normal functions** and the lambda keyword is used to create anonymous functions. Please see [this](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) for details.

# Python code to illustrate the cube of a number

# using lambda function

def cube(x): return x\*x\*x

cube\_v2 = lambda x : x\*x\*x

print(cube(7))

print(cube\_v2(7))

**Output**

343

# Task 12: Map, Filter and Reduce

Video - <https://youtu.be/rRgD1yVwIvE>