PRIME INTUIT
Finishing School

Python – Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.
The main difference between lists and tuples is- Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print (tuple) # Prints complete tuple
print (tuple[0]) # Prints first element of the tuple
print (tuple[1:3]) # Prints elements starting from 2nd till 3rd
print (tuple[2:]) # Prints elements starting from 3rd element
print (tinytuple * 2) # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

Python – Accessing values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain the value available at that index.

```
tup1 = ('physics', 'chemistry',
1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

Python – Updating values in Tuples

Tuples are immutable, which means you cannot update or change the values of tuple elements. You are able to take portions of the existing tuples to create new tuples as the below:

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
# Following action is not valid for tuples
# tup1[0] = 100;
# So let's create a new tuple as follows
tup3 = tup1 + tup2
print (tup3)
```

Python – Deleting Tuple elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.
To explicitly remove an entire tuple, just use the **del** statement.

```
tup = ('physics', 'chemistry', 1997, 2000);
print (tup)
del tup;
print "After deleting tup : "
print tup
```

Python –Tuples Basic Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x, end=' ')	1 2 3	Iteration

PRIME INTUIT
Finishing School

Python –Tuples Slicing & Indexing

T = (‘C++’, ‘Java’, ‘Python’)

Python Expression	Results	Description
T[2]	'Python'	Offsets start at zero
T[-2]	'Java'	Negative: count from the right
T[1:]	('Java', 'Python')	Slicing fetches sections

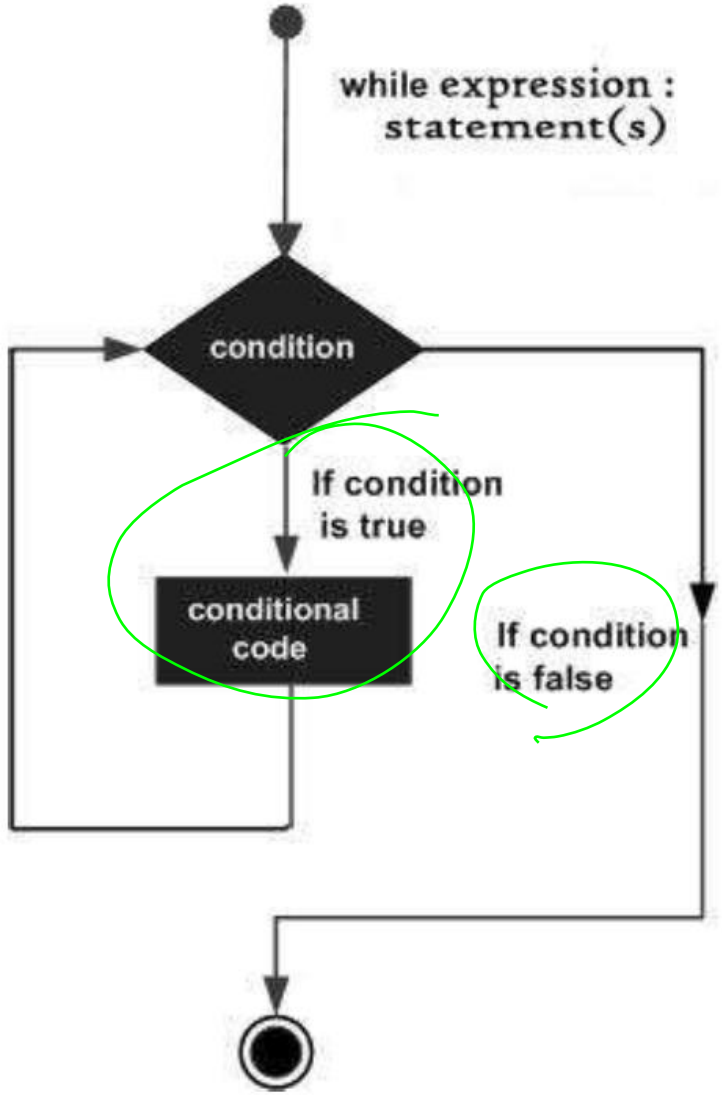
Python –Tuples –Built In Functions

SN	Function with Description
1	cmp(tuple1, tuple2) No longer available in Python 3.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

PRIME INTUIT
Finishing School

Python – While Loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.




```
While expression:
statement(s)
```

Python – List Comprehensions

List Comprehensions

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: