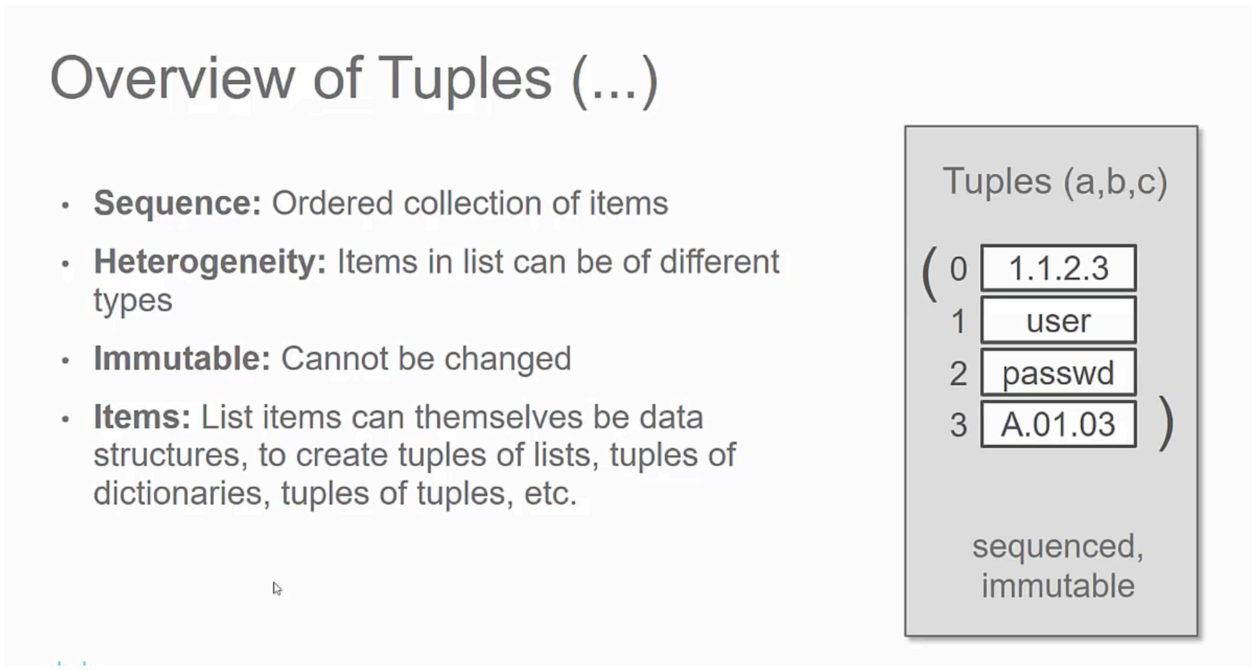
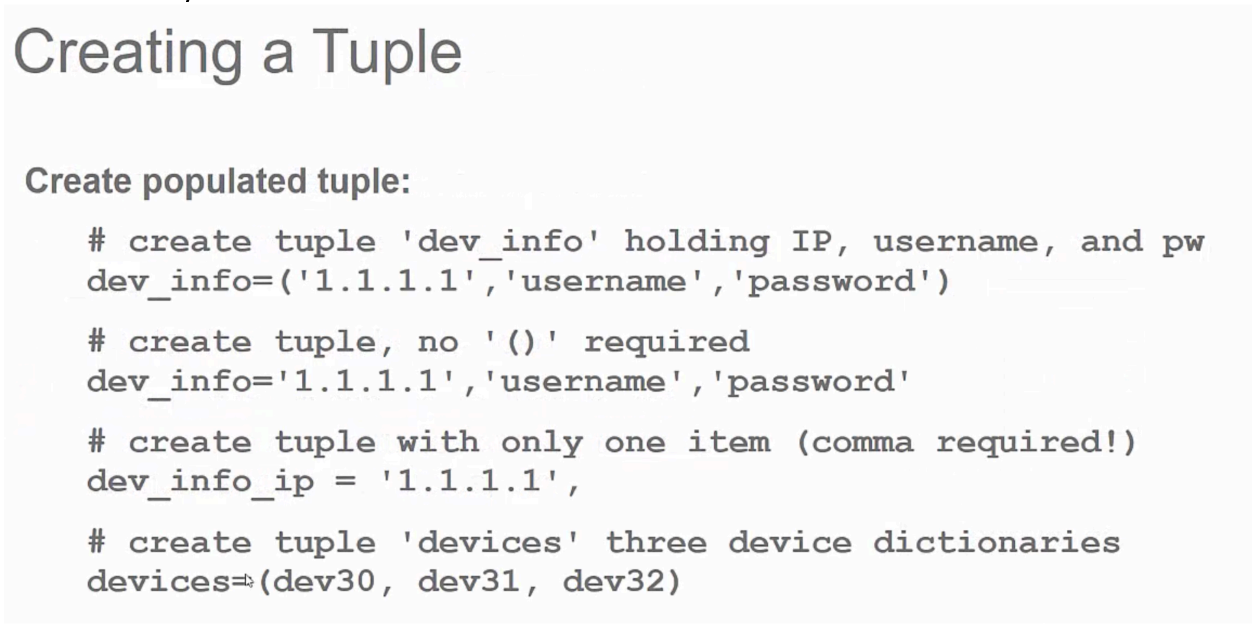


Overview of Tuples (...)

- **Sequence:** Ordered collection of items
- **Heterogeneity:** Items in list can be of different types
- **Immutable:** Cannot be changed
- **Items:** List items can themselves be data structures, to create tuples of lists, tuples of dictionaries, tuples of tuples, etc.

Surrounded by round brackets



Creating a Tuple

Create populated tuple:

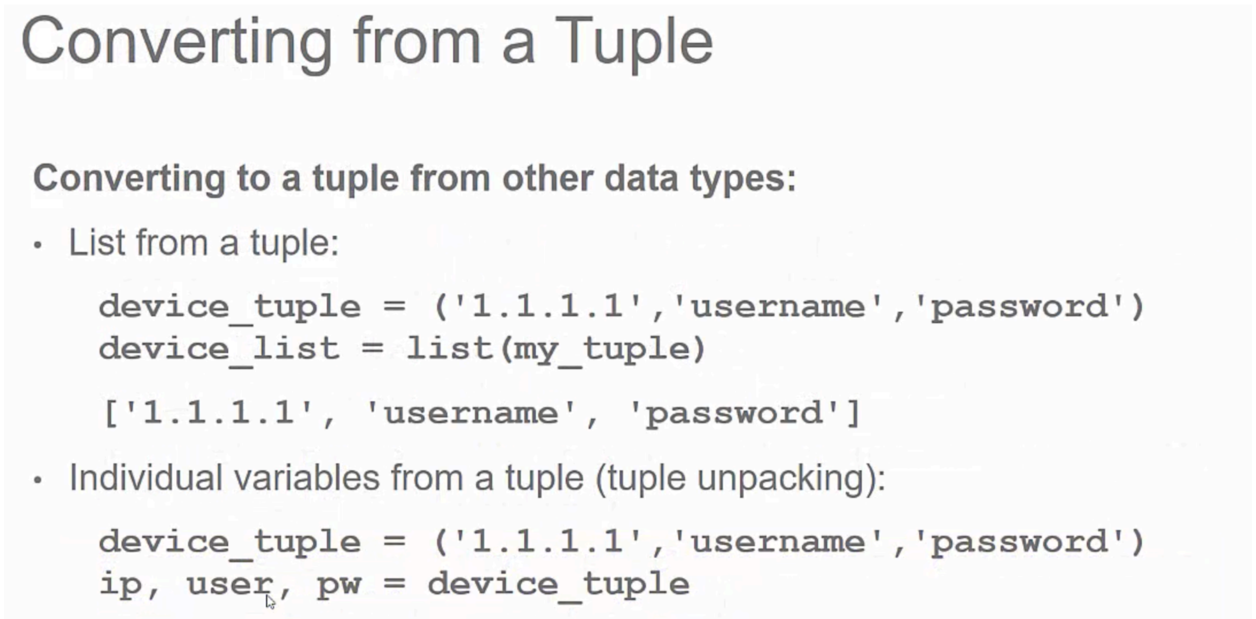
```
# create tuple 'dev_info' holding IP, username, and pw
dev_info=('1.1.1.1','username','password')

# create tuple, no '()' required
dev_info='1.1.1.1','username','password'

# create tuple with only one item (comma required!)
dev_info_ip = '1.1.1.1',

# create tuple 'devices' three device dictionaries
devices=(dev30, dev31, dev32)
```

No need to initialize or create an empty tuple as it is immutable and cannot be changed. This generally applies to any immutable data structure.



Converting from a Tuple

Converting to a tuple from other data types:

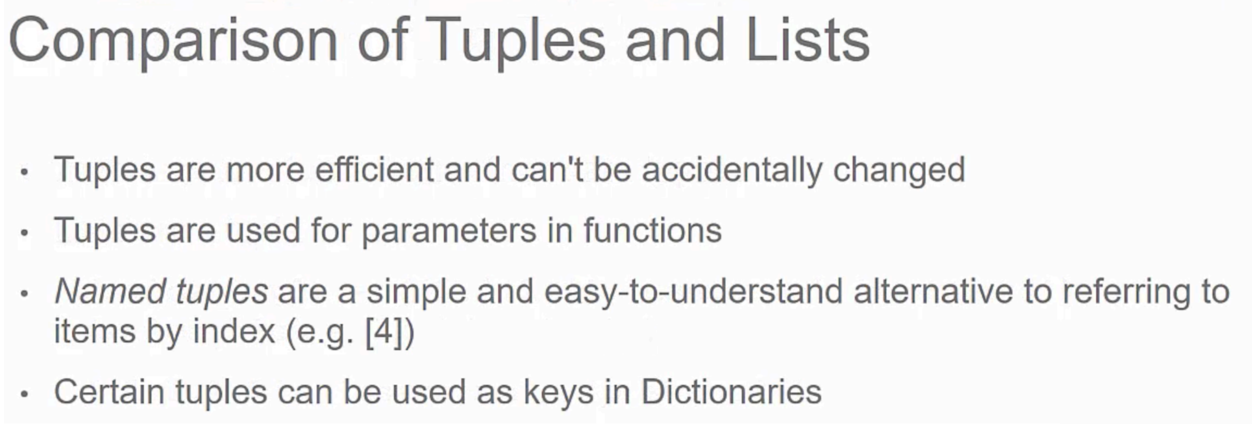
- List from a tuple:

```
device_tuple = ('1.1.1.1','username','password')
device_list = list(my_tuple)

['1.1.1.1', 'username', 'password']
```

- Individual variables from a tuple (tuple unpacking):

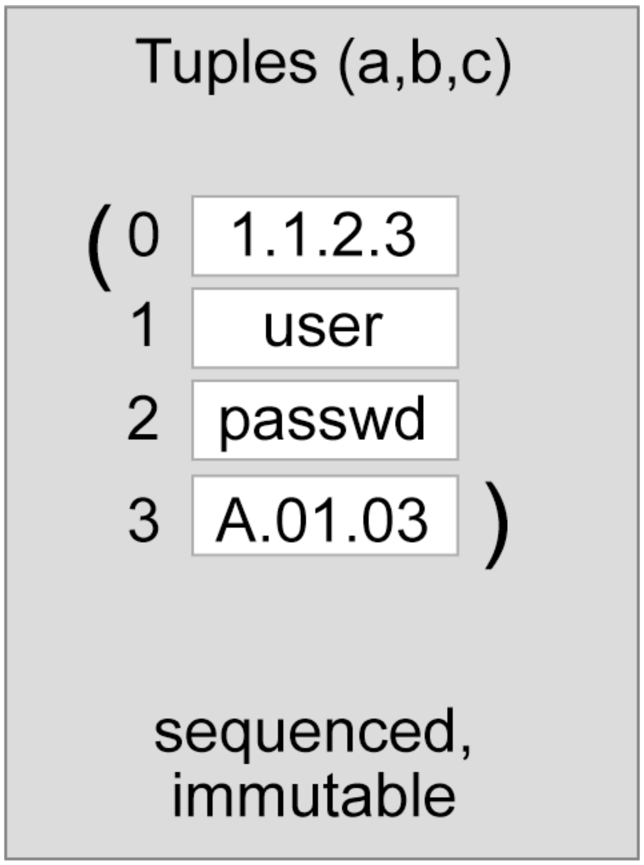
```
device_tuple = ('1.1.1.1','username','password')
ip, user, pw = device_tuple
```



Comparison of Tuples and Lists

- Tuples are more efficient and can't be accidentally changed
- Tuples are used for parameters in functions
- *Named tuples* are a simple and easy-to-understand alternative to referring to items by index (e.g. [4])
- Certain tuples can be used as keys in Dictionaries

- **Sequence:** Ordered collection of items
- **Heterogeneity:** Items can be of different types
- **Immutable:** Cannot be changed
- **Items:** Items can be strings or numbers or can be data structures such as lists or dictionaries


Tuples (a,b,c)
(
0
1.1.2.3
1
user
2
passwd
3
A.01.03
)
sequenced,
immutable

Tuples Overview

Tuples are very much like lists, with a few significant differences. The following are attributes of tuples:

- **Sequence:** Tuples, like lists, are ordered sequences of items which means that you reference items in the tuple using the numerical index.
- **Heterogeneity:** Tuples, like lists, can have items of differing types.
- **Immutable:** Unlike lists, tuples cannot be changed. Once you have created a tuple, the values set can never be changed. If your tuple needs to change for some reason, create a new tuple.
- **Items:** Items in a tuple can be simple, like strings or numbers, or they can be data structures of some type, like lists or dictionaries.

Tuples are identified in Python by parentheses, '(' and ')'.

Comparing Tuples and Lists

- Tuples are more efficient than lists
- Tuples cannot be accidentally changed
- Tuples are actually used by Python for passing parameters to functions
- Named tuples are sometimes a better way to use tuple data structures, allowing you to refer to the items in a tuple by their name, rather than by the numerical index.
- Since tuples never change, if they consist of immutable objects like strings and numbers, they can be used as dictionary keys.

To summarize, tuples are less flexible than lists, but they do have several distinct advantages, that may be useful in certain situations.

Creating a Tuple

Creating a tuple is like creating a list. Empty tuples can be created using either the tuple function, or by using round brackets:

```
my_tuple = tuple()   # creates an empty tuple 'my_tuple'
my_tuple = ()          # creates an empty tuple 'my_tuple'
```

However, creating an empty tuple has limited value, since items cannot be added to it. Of more interest is creating a populated tuple. The following code shows some examples of creating populated tuples.

The following example creates a tuple, which looks just like creating a list except for the parenthesis.

```
# create tuple 'dev_info' holding IP, username, and pw
dev_info=('1.1.1.1','username','password')
```

The following example creates the same tuple, but notice that the parenthesis are not required. Omitting the parenthesis is sometimes called 'tuple packing' – Python implicitly knows from the statement syntax that a tuple is being created.

```
# create tuple, no '()' required
dev_info='1.1.1.1','username','password'
```

You can create tuples with values that are data structures. In the following example, a tuple with three dictionary items is created:

```
# Create three dictionaries holding device information for specific devices
dev30 = {'name':'xrv-0','ip':'10.1.1.30','user':'cisco'}
dev31 = {'name':'xrv-1','ip':'10.1.1.31','user':'cisco'}
dev32 = {'name':'xrv-2','ip':'10.1.1.32','user':'cisco'}

# create tuple 'devices' with three device dictionaries
devices=(dev30, dev31, dev32)
```

Concatenating a Tuple

You can create other data structures and variables from tuples as well. The following example takes a tuple, and creates a list:

```
device_tuple = ('1.1.1.1','username','password')
device_list = list(device_tuple)
```

The inverse of tuple packing, 'tuple unpacking, is also possible. In this situation, a tuple is separated into its constituent parts. In the following example, a tuple of device information is used to set individual variables of 'ip', 'user', and 'pw':

```
device_tuple = ('1.1.1.1','username','password')
ip, user, pw = device_tuple
```