Tuples and its Uses

Like List and String, there is also another standard sequence data type in Python: the tuple. Basically, tuples are useful when you need to store a list of information that does not change over time. An example of that is the list of the months of the year:

```python
# months is a tuple with 12 elements, the names of the months
>>> months = ('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')
```

As you can see from the months tuple and the code example above, tuples can be created by assigning a list of values enclosed in round parenthesis to a variable (look at the `t` variable). A tuple can also be created by assigning to a variable a list of values without enclosing them in round parenthesis (see `firstYearCSProgCourses` in code example 1). On output, tuples are always represented enclosed in round parenthesis ()

Elements of a tuple can be accessed in a way similar to a list, that is you use an index. So, what is the difference between a list and a tuple? For one thing, lists are mutable while tuples are immutable just like strings: once defined, the tuples cannot be changed.

```python
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t[1] = 'z'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

In the code Example 2, we are trying to assign 'z' to the 1st index of tuple t. In other words, we are trying to change 'b' with 'z'. Since, tuples are immutable, Python throws a TypeError telling us that tuples do not support assignment.
The next question that comes up is when to use tuples rather than list. In Python, tuples are usually used when you know the elements and size of the list and you know that the list contents will not change.

**Functions that return Tuple**

In order to return more than one value using one tuple, simply insert the values in a tuple and use it in the return statement.

Example 3

```python
def myFun():
    name = 'John'
    GPA = 3.65
    return (name, GPA)

myTuple = myFun()
print (myTuple)
('John', 3.65)
print (myTuple[0])
John
print (myTuple[1])
3.65
```

**Dictionaries**

Another useful data type built into Python is the dictionary. A simple example of dictionary is a phone book. A phone book contains a number of names and, associated to each name the phone number. When you need to find the phone number of your friend, you search his name in the phone book and then you get the phone number. Hence, you can think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).

Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”. Unlike lists and tuples, which are indexed by a range of numbers, dictionaries are always indexed by keys, which can be any immutable type. Strings and numbers can always be keys.

Since tuples are immutable, tuples can be used as keys if they contain only immutable types like strings, numbers, or tuples(nested tuples); if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. The dictionaries are defined by a pair of curly brackets {}.

Code Example 4

```python
>>> telephoneDict = {"Jack": 7786668768, "Katie": 8876765433}
>>> telephoneDict["Jack"]
7786668768
>>> telephoneDict["Jack"] = 8898878765
```
As you can see in the code example above, “Jack” and “Katie” are the keys of the dictionary, and their telephone numbers are the values. Any telephone number in the dictionary can be accessed by providing the corresponding key in the dictionary.

You can delete any particular key-value pair in the dictionary using “del” keyword. You can also access the list of keys in the dictionary using keys() function and a list of values using the values() function in the dictionary object. See the example below.

Code Example 5

```python
>>> telephoneDict = {"Jack": 7786668768, "Katie": 8876765433, "Jillian", 9876543210}
>>> del telephoneDict["Jillian"]
>>> telephoneDict
{'Jack': 7786668768, 'Katie': 8876765433}
>>> telephoneDict.keys()
dict_keys(['Jack', 'Katie'])
>>> telephoneDict.values()
dict_values([7786668768, 8876765433])
```

Another useful keyword in the dictionary is dict. You can use dict keyword to create a dictionary from a list of key-value tuples.

Code Example 6

```python
>>> telephoneDict = dict([('Jack', 4139234565), ('Katie', 4127234523)])
>>> telephoneDict
{'Jack': 7786668768, 'Katie': 8876765433}
>>> numToNumSquareMap = dict((x, x**2) for x in range(5))
>>> numToNumSquareMap
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Here, the dict keyword is being used to create a key-value dictionary where every key is a number in range(5) which is 0 to 4 and its corresponding value is the square of the number.

**Scanning the contents of a dictionary**

Example 7

```python
>>> myDic = {1: 'ABC', 2: 'XYZ', 3: 'LMN'}
>>> for k, v in myDic.items():
...     print (k, v)

1 ABC
2 XYZ
3 LMN
```

In the previous example, k holds the key and v holds the value.