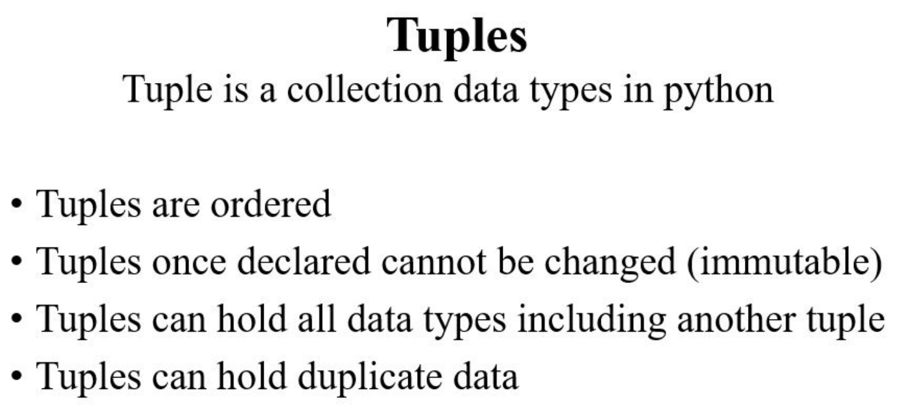
# Session 8

1. Tuples
2. Built-in Tuple methods
3. Dictionary
4. Built-in Dictionary methods

---

## Introduction to Tuples



**Tuples**

Tuple is a collection data types in python

- Tuples are ordered
- Tuples once declared cannot be changed (immutable)
- Tuples can hold all data types including another tuple
- Tuples can hold duplicate data

Tuples are constructed with brackets `( )` and commas separating every element in the tuple.

In [5]:
```python
# Assign a tuple to an variable named my_tup
my_tup = (1,2,3)
print(my_tup)
```

```
(1, 2, 3)
```

**NOTE** : Tuples are similar to list in every aspect except for the fact that tuples are immutable while list are mutable.

We just created a tuple of integers, but tuples can actually hold different object types. They can also hold duplicate elemetns hust like lists
For example:

In [4]:
```python
my_tup = ('A string',23,100.232,'o')
print(my_tup)
```

```
('A string', 23, 100.232, 'o')
```

In [2]:
```python
# Check len just like a list
my_tup = (1,2,3)
```

```
len(my_tup)
```

Out[2]:
```
3
```

Tuple indexing and slicing also works similar to the list and string slicing

In [6]:
```
# Use indexing just like we did in lists
my_tup[0]
```

Out[6]:
```
1
```

In [7]:
```
# Slicing just like a list
my_tup[-1]
```

Out[7]:
```
3
```

It can't be stressed enough that tuples are immutable.
Which means we cannot do individual item assignment on them

In [10]:
```
my_tup[0]= 'change'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-378d31028a15> in <module>
----> 1 my_tup[0]= 'change'

TypeError: 'tuple' object does not support item assignment
```

## Built-in Tuple Methods

Tuples have built-in methods, but not as many as lists do. Let's look at two of them:

In [1]:
```
# Use .index to enter a value and return the index
my_tup = ('one',2)
my_tup.index('one')
```

Out[1]:
```
0
```

In [2]:
```
# Use .count to count the number of times a value appears
my_tup.count('one')
```

Out[2]:
```
1
```

## Introduction to Dictionary

# Dictionary

Dictionary is a collection of key and value pair

- Dictionaries are unordered
- Dictionary values changed
- Dictionaries values can hold all data types including another Dictionary
- Dictionaries cannot hold duplicate data for keys
- Dictionary keys can only be a string, integer or float

Dictionaries are constructed with brackets `{ }` and commas separating every pair in the Dictionary and `:` separating every key and value

```
In [4]:
# Make a dictionary with {} and : to signify a key and a value
my_dict = {'key1':'value1','key2':'value2'}
```

A key and value pair encompases one element inside a dictionary

```
In [2]:
# Call values by their key
my_dict['key2']
```

```
Out[2]: 'value2'
```

**Values of the dictionary elements can be any datatype (list,strings,int,float etc)**

```
In [1]:
my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
```

**The keys can only have datatype int,float and string.**

```
In [4]:
my_dict = {1:123,1.1:[12,23,33],'a':['item0','item1','item2']}
```

We can access individual elements from the dictionary using the key name as mentioned earlier.

```
In [2]:
my_dict['key3']
```

```
Out[2]: ['item0', 'item1', 'item2']
```

```
In [3]:
# Can call an index on that value
my_dict['key3'][0]
```

```
Out[3]: 'item0'
```

```
In [4]:
# Can then even call methods on that value
my_dict['key3'][0].upper()
```

```
Out[4]: 'ITEM0'
```

We can also create keys by assignment. For instance if we started off with an empty dictionary, we could continually add to it:

```
In [11]: # Create a new dictionary
         d = {}
```

```
In [12]: # Create a new key through assignment
         d['animal'] = 'Dog'
```

```
In [13]: # Can do this with any object
         d['answer'] = 42
```

```
In [14]: #Show
         d
```

```
Out[14]: {'animal': 'Dog', 'answer': 42}
```

we can also have nested (multi-dimensional) dictionaries just like lists
we can use indexing (but with key names) to grab the elements

```
In [15]: # Dictionary nested inside a dictionary nested inside a dictionary
         d = {'key1':{'nestkey':{'subnestkey':'value'}}}
```

Let's see how we can grab that value:

```
In [16]: # Keep calling the keys
         d['key1']['nestkey']['subnestkey']
```

```
Out[16]: 'value'
```

## Built-in Dictionary methods

There are a few methods we can call on a dictionary. Let's get a quick introduction to a few of them:

```
In [5]: # Create a typical dictionary
        d = {'key1':1,'key2':2,'key3':3}
```

```
In [9]: # Method to return a list of all keys
        d.keys()
```

```
Out[9]: dict_keys(['key1', 'key2', 'key3'])
```

```
In [19]: # Method to grab all values
         d.values()
```

Out[19]:
```
dict_values([1, 2, 3])
```

In [20]:
```python
# Method to return tuples of all items  (we'll learn about tuples soon)
d.items()
```

Out[20]:
```
dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

## Iterating over dictionary

We can use for loop to iterate over dictionary elements

In [6]:
```python
my_dict = {'key1':'value1','key2':'value2','key3':'value3','key4':'value4'}

for i in my_dict:
    print(i)
```

```
key1
key2
key3
key4
```

Iterating over dictionary only gives out the keys and not the corresponding vlaues.
We can use the methods discussed above to access the values or the both the keys and values together from a dictionary

**using `values()` method to iterate over values**

In [7]:
```python
my_dict = {'key1':'value1','key2':'value2','key3':'value3','key4':'value4'}

for i in my_dict.values():
    print(i)
```

```
value1
value2
value3
value4
```

**using `items()` method to iterate and access both keys and values at the same time.**

In [8]:
```python
my_dict = {'key1':'value1','key2':'value2','key3':'value3','key4':'value4'}

for i in my_dict.items():
    print(i)
```

```
('key1', 'value1')
('key2', 'value2')
('key3', 'value3')
('key4', 'value4')
```

---

In [ ]:
```python

```

## HOMEWORK

## 1. Print the following pattern using nested for loop

```
Size: 7 x 21
---------.|.---------
------.|..|..|.------
---.|..|..|..|..|.---
-------WELCOME-------
---.|..|..|..|..|.---
------.|..|..|.------
---------.|.---------
```

(A simple google search will give you the solution. But where is the fun in it!!!!)

# HOMEWORK SOLUTION

In [1]:

```python
#TASK 1:
#TASK 2:
rows = int(input("enter number of rows: "))

for x in range(rows):
    for i in range(x,rows):
        print("-",end = '')
    print()
```

```
enter number of rows: 5
-----
----
---
--
-
```

In [ ]:

```python

```

1. Draw pattern

P
Py
Pyt
Pyth
Pytho
Python

In [ ]:

```python

```