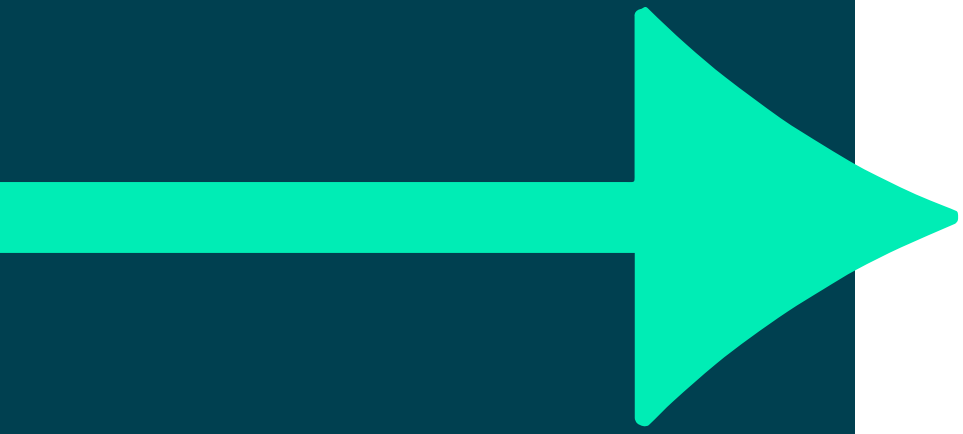
QA
POWERING
POTENTIAL

Collections – Tuples, Dictionaries, Sets

LESSON OBJECTIVES



In this chapter, you'll learn about:

- Tuples
- Dictionaries
- Sets

TUPLES

A tuple is a fixed (*immutable*) sequence of python objects

```
>>> x = 'fred'
>>> y = 'jim'
>>> z = 'toby'
>>> t = x,y,z
>>> t
('fred', 'jim', 'toby')
```

```
>>> person = ()
>>> person = tuple()
>>> person = ('James', 'Bond', 7, True, 10.99)
>>> person = tuple(('James', 'Bond', 7, True,
10.99)) # note the double brackets
```

Tuples may be indexed and sliced in the same way as all other sequences, including strings. Like strings, **tuples are immutable**.

TUPLES PACKING AND UNPACKING

When we create a tuple, we normally assign values to it. This is called **packing** a tuple.

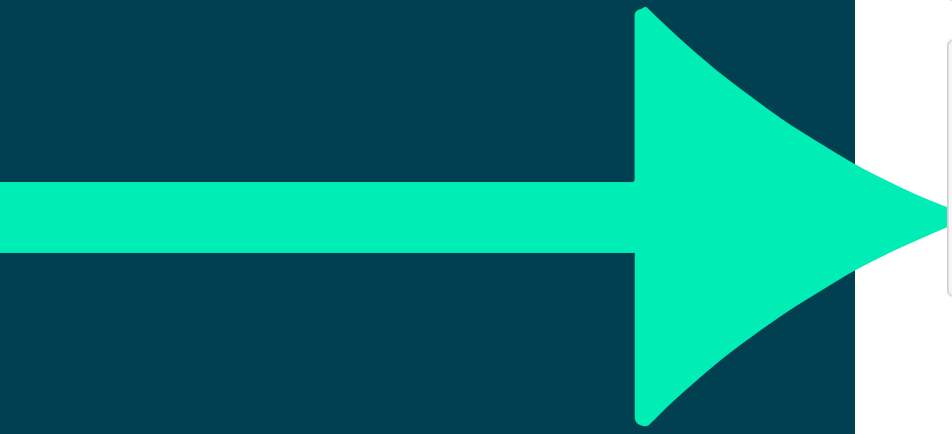
```
person = ('James', 'Bond', 7)
```

Extracting the values back into variables is called **unpacking**.

```
(Fname, Lname, ID) = person
print(Fname)
print(Lname)
print(ID)
```

```
James
Bond
7
```

TUPLES PACKING AND UNPACKING



If the number of variables is less than the number of values, you can add an asterisk ***** to the variable name and the values will be assigned to the variable as a list:

```
(Fname, *data) = person
print(Fname)
print(data)
```

```
James
['Bond', 7]
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

```
person = ('James', 'Bond', 7, 'M')
```

```
(Fname, *data, boss) = person
print(Fname)
print(data)
print(boss)
```

```
James
['Bond', 7]
M
```

TUPLE OPERATIONS

Tuples can be sliced in the same way as lists.

Remember the indexing: in Python we count from zero on the left, from -1 on the right.

```
breakfast=('coffee','croissant','jam','butter','fruit')
print(breakfast[2:4])
('jam', 'butter')
print(breakfast[-4])
croissant
```

Tuples can be concatenated:

```
breakfast = breakfast + ('bacon', 'eggs')
print(breakfast)
('coffee','croissant','jam','butter','fruit', 'bacon',
'eggs')
```

TUPLE OPERATIONS

Join tuples

```
tuple_1 = (‘a’, ‘b’, ‘c’)
tuple_2 = (‘x’, ‘y’, ‘z’)
tuple_3 = tuple_1 + tuple_2
```

```
(‘a’ ‘b’ ‘c’ ‘x’ ‘y’ ‘z’)
```

Multiply tuples

```
mytuple = 'a', 'b', 'c'
another = mytuple * 4
```

```
('a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c')
```

Be careful of single values and the trailing comma

```
thing = ('Hello')
print(type(thing))

thing = ('Hello',)
print(type(thing))
```

```
<class 'str'>

<class 'tuple'>
```

DICTIONARIES

A dictionary is a collection of unique keys, each referring to a value

- The key can be any immutable python object – often a string
- The position of the value in memory is determined by the key
- Dictionaries are ***unordered*** – they are not sequences
- .Dictionaries are similar to sets but are accessed by keys

✓**Constructed from { }**

varname **= {***key1***:***object1***,***key2***:***object2***,***key3***:***object3***,...}**

✓**Or using dict()**

varname **= dict(***key1***=***object1***,***key2***=***object2***,***key3***=***object3***,...)**

✓**Accessed by key**

A key is usually a text string, or anything that yields a text string

*varname***[***key***] =** *object*

These produce identical dictionaries

```
mydict = dict(Make='Audi', Model='A4',
Colour='Red', Doors = 4, New = True)

mydict = {'Make':'Audi', 'Model':'A4',
'Colour':'Red', 'Doors':4, 'New':True}
```

DICTIONARY VALUES

Objects stored can be of any type

- Lists, tuples, other dictionaries, etc...
- Can be accessed using multiple indexes or keys in `[ ]`
- Add a new value just by assigning to it

```
mydict = {'UK':['London', 'Wigan', 'Macclesfield'],
          'US':['Miami', 'Boston', 'New York']}
print(mydict['UK'][2])

homer = 1
print(mydict['US'][homer])

mydict['FR'] = ['Paris', 'Lyon', 'Bordeaux', 'Lille']
for country in mydict.keys():
    print(country, ': ', mydict[country])
```

```
FR :  ['Paris', 'Lyon', 'Bordeaux', 'Lille']
US :  ['Miami', 'Boston', 'New York']
UK :  ['London', 'Wigan', 'Macclesfield']
```

DICTIONARY VALUES

To add a key – assign using the new key

To delete a key – use the `del` statement

```
mydict['Mileage'] = '80k'
mydict['BHP'] = 160
del mydict[‘New’]
print(mydict)
```

To access a single element from a dictionary

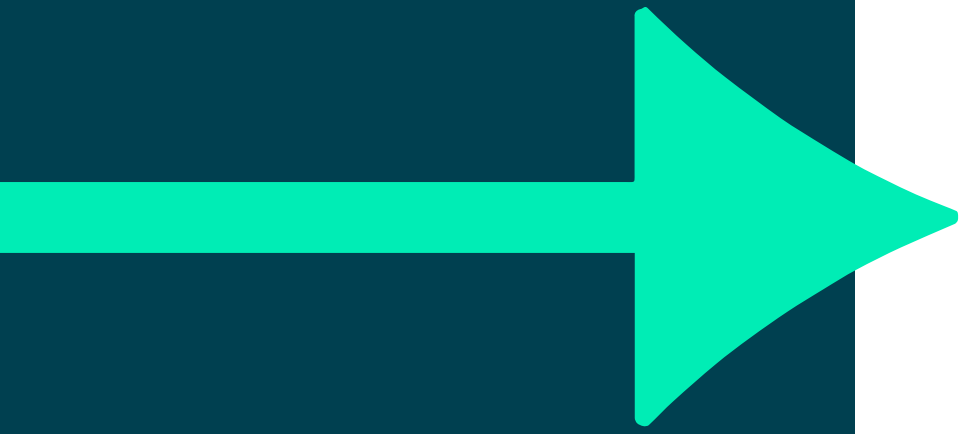
```
print('Colour is', mydict['Colour'])
```

There are several methods used for iterating over dictionaries

- dict.keys(), dict.values(), dict.items()
- Remember that dictionaries are not ordered

```
for key,value in mydict.items():
    print key, value
```

REMOVING ITEMS FROM A DICTIONARY



To remove a single key/value pair:

- `del` *dict*`[`*key*`]`
- Raises a KeyError exception if the key does not exist
- *dict*`.pop(`*key*`[,`*default*`])`
- Returns *default* if the key does not exist

```
>>> fred={}
>>> del fred['dob']
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    del fred['dob']
KeyError: 'dob'
>>> fred.pop('dob', False)
False
```

- Also:
- *dict*`.popitem()` removes the next key/value pair used in iteration
- *dict*`.clear()` removes all key/value pairs from the dictionary

DICTIONARY METHODS

dict.clear()	Remove all items from *dict*
dict.copy()	Return a copy of *dict*
dict.fromkeys(*seq*[,*value*])	Create a new dictionary from *seq*
dict.get(*key*[,*def*ault])	Return the value for *key*, or *default* if it does not exist
dict.items()	Return a view of the key-value pairs
dict.keys()	Return a view of the keys
dict.pop(*key*[,*default*])	Remove and return *key's* value, else return *default*
dict.popitem()	Remove the next item from the dictionary
dict.setdefault(*key*[,*default*])	Add *key* if it does not already exist
dict.update(*dictionary*)	Merge another dictionary into *dict*.
dict.values()	Return a view of the values

NOTE: Return values from `keys()`, `values()`, and `items()` are *view objects*.

DICTIONARIES: VIEW OBJECTS

View objects, as returned by **items()**, **keys()**, and **values()** methods, can be used in iteration and as objects to construct a list.

- **in iteration**

```
nebula = {'M42':'Orion',
          'C33':'Veil',
          'M8' :'Lagoon',
          'M17':'Swan'
}

for kv in nebula.items():
    print(kv)
```

```
('M42', 'Orion')
('M17', 'Swan')
('M8', 'Lagoon')
('C33', 'Veil')
```

- **To store as a list**

```
lkeys = list(nebula.keys())
print(lkeys)
```

```
['M42', 'M17', 'M8', 'C33']
```

- **In set operations**

```
jelly = nebula.keys() | {'M37', 'M5'}
print(jelly)
```

```
{'M5', 'M37', 'M17', 'M42', 'M8', 'C33'}
```

SETS

A **set** is an **unordered** collection of **unique** python objects.

A set is **mutable.**

NOTE: Once a set is created, we cannot change its items, but we can add and remove items.

Creating a set:

```
s1 = {5, 6, 7, 8, 5}
print(s1)
```

```
{8, 5, 6, 7}
```

```
s2 = set([9, 10, 11, 12, 9])
print(s2)
```

```
{9, 10, 11, 12}
```

Iterating over a set:

```
for i in s1:
    print(s1)
```

```
{8, 5, 6, 7}
```

SET METHODS

Add using the add() method, remove using remove()

```
s4 = {23, 42, 66, 123}
s5 = {56, 27, 42}

s4.remove(123)
s5.add(123)
```

```
{66, 123, 42, 23}     {56, 42, 27}
{66, 42, 23}          {56, 123, 42, 27}
```

Other set methods:

- `Len` Return the number of elements in the set
- `discard` Remove element *if present*
- `pop` Remove and return the next element from the set
- `clear` Remove all elements

USING SETS

Removing duplicates from a list

→ But we lose the original order

```
cheese = ['Cheddar', 'Stilton', 'Camembert',
          'Brie', 'Stilton', 'Cheshire']
cheese = list(set(cheese))
```

list() is required, otherwise 'cheese' would now refer to a set

```
['Camembert', 'Cheshire', 'Cheddar', 'Stilton', 'Brie']
```

Removing several items from a list

```
cheese = ['Cheddar', 'Stilton', 'Camembert',
          'Brie', 'Stilton', 'Cheshire']
cheese = list(set(cheese) - {'Stilton', 'Brie'})
```

```
['Camembert', 'Cheshire', 'Cheddar']
```

SET OPERATORS

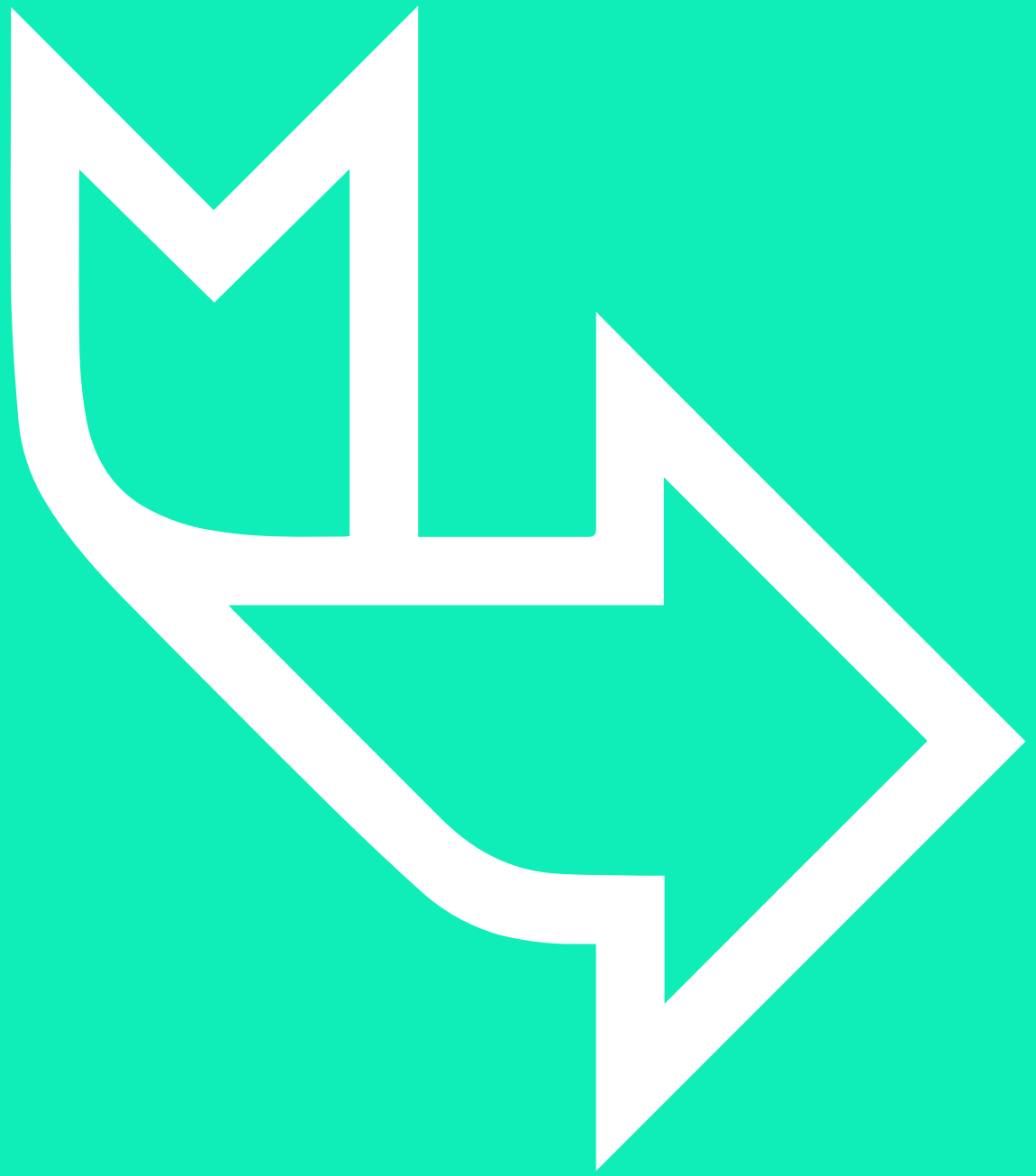
Set operators and method calls

Operator	Method	Returns a new set containing
&	s6.intersection(s7)	Each item that is in both sets
|	s6.union(s7)	All items in both sets
-	s6.difference(s7)	Items in s6 not in s7
^	s6.symmetric_difference(s7)	Items that occur in one set only

```
s6 = {23, 42, 66, 123}
s7 = {123, 56, 27, 42}

print(s6 & s7)
print(s6 | s7)
print(s6 - s7)
print(s6 ^ s7)
```

```
{42, 123}
{66, 27, 42, 23, 56, 123}
{66, 23}
{66, 23, 56, 27}
```



Further Reading

https://www.python.org/