LEARN WEB

ale66

SETS

- Sets are constructed from a sequence.

- Sets **cannot** have duplicated values

  - they are used to build sequences of unique items, e.g., identifiers

```
1  my_list = ['Stelios', 'Tom', 'Tom', 'Stelios', 'Claudia']
2
3  my_set = set(my_list)
4
5  print(list(my_set))
6
7  # the set includes only the unique elements!
8  ['Stelios', 'Tom', 'Claudia']
```

SETS

ideal when we need to remove duplicated values

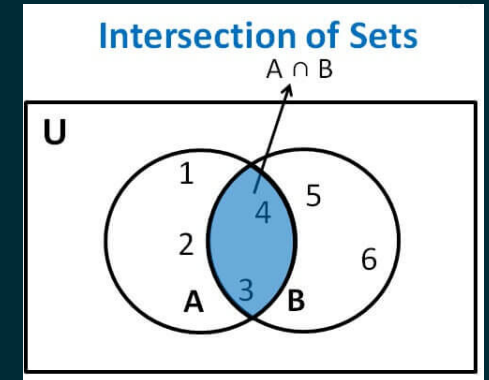also ideal for *logical* operations

duplicates are eliminated

```
1  a = set([1, 2, 3, 4])
2
3  b = set([3, 4, 5, 6])
4
5  a.intersection(b)
6  set([3, 4])
```

```
1  c = a.intersection(b)
2
3  c.issubset(a)
4  True
```

```
1  a.issubset(b)
2  False
```

```
1  a.difference(b)
2
3  # will  return
4  set([1,2,5,6])
```

Intersection of Sets
A ∩ B

SETS: BASIC OPERATIONS

Create an empty set

Iterate over a set and print elements

```python
1   #Create a new empty set
2   empty_set = set()
3
4   #Create a non empty set
5   aset = set([0, 1, 2, 3, 4])
6
7   #Create a set
8
9   num_set = set([0, 1, 2, 3, 4, 5])
10
11  for element in num_set:
12    print(element)
```

github.com/ale66/learn-web

SETS: BASIC OPERATIONS CONT'D

Add/remove data from a set

```python
1  color_set = set()
2
3  color_set.add('Red')
4
5  #Add multiple items
6  color_set.update(['Blue', 'Green'])
7
8  # remove the last element
9  color_set.pop()
```

SETS: MORE BASIC OPERATIONS

```
1  x = set(['green', 'blue'])
2
3  y = set(['blue', 'yellow'])
4
5  z  = x & y        # or x.intersection(y)
```

```
1  'green', 'blue', 'yellow'
```

```
1  x = set(['green', 'blue'])
2
3  y = set(['blue', 'yellow'])
4
5  z = x | y         # or x.union(y)
```

Cardinality

```
1  colors = set(['green', 'blue'])
2
3  print(len(colors))
```

QUIZ 2: TRUE OR FALSE?

- Sets allow duplicated values

- Sets are unordered

- Sets are not indexed

SOLUTIONS!

- Sets allow duplicated values

    - False

- Sets are unordered

    - True

- Sets are not indexed

    - True

SUMMARY

Duplicates are eliminated

no ordering

no support for indexing

QUIZ 3!

Will it
print True
or False?

:90

```
1  x = set(['Agatha Christie',
2            'J.R.R. Tolkien'])
3
4  y = set(['J.R.R. Tolkien',
5            'Virginia Woolf'])
6
7  z = set(['J.R.R. Tolkien'])
```

```
1  print(x.issubset(y))
```

```
1  print(y. issuperset(x)
```

```
1  print(z.issubset)
```

```
1  print(y.issuperset(z))
```

SOLUTION

Will it
print True
or False?


```
1  x = set(['Agatha Christie',
2            'J.R.R. Tolkien'])
3
4  y = set(['J.R.R. Tolkien',
5            'Virginia Woolf'])
6
7  z = set(['J.R.R. Tolkien'])
```

```
1  print(x.issubset(y))
2  False
```

```
1  print(y. issuperset(x)
2  False
```

```
1  print(z.issubset)
2  True
```

```
1  print(y.issuperset(z))
2  True
```

TUPLES

A data structure similar to lists

Main difference: tuples are *immutable*

Their application is faster than lists

```
1  mytuple = (10, 20, 30)
2
3  mytuple[0]
```

FUNCTIONS FOR TUPLES

There are two functions available only:

- `index` to find the occurrence of a value

- `count` to count the number of occurrences of a value

TUPLES IN DICTIONARIES

```
1  my_protected_dictionary = dict([('jan', 1), ('feb', 2), ('march', 3)])
2
3  my_protected_dictionary['feb']
4  2
```

Often tuples are used as keys to dictionaries

Tuples are useful because they are

- are faster than lists

- protect the data from change

TUPLE UNPACKING

```
1  data   =  (1,2,3)
2
3  x, y, z = data
```

Slicing is similar to lists

```
1  my_tuple = (1, 2, 3, 4, 5)
2
3  print(mytuple[2:])
4
5  (3, 4, 5)
```

WORKING WITH TUPLES

```
1  tuplex = (4, 6, 2, 8, 3, 1)
```

Since tuples are immutable we can not add new elements

tuple merge, with the + operator, can add elements and create a new tuple

```
1  # notice the extra ','
2  tuplex = tuplex + (9,)
3
4  print(tuplex)
```

Another workaround

```python
1  tuplex = (4, 6, 2, 8, 3, 1)
2
3  listx = list(tuplex)
4
5  #Add an item in the list
6  listx.append(30)
7
8  #Make the list tuplex
9  tuplex = tuple(listx)
```

THE count() METHOD

```
1  tuplex = (2, 4, 5, 6, 2, 3, 4, 4, 7)
2
3  # find the np. of times it appears in the tuple:
4  count = tuplex.count(4)
```

COMPARISON, A

	Lists	Dictionaries	Sets	Tuples
When to use?	• Collection of data that does not need random access. • When you need to iterate and modify items.	• You need a key to value data association. • Fast lookup, based on a key. • When data needs to be modified	• When you want to eliminate duplicated values. • When you need uniqueness of elements	• To ensure immutability. • Can be used in combination with other data structure, for example a tuple could represent a key in a dictionary
Duplicated values?	Yes	No duplicated keys, Yes duplicated values	**No**	Yes (faster than lists)
Mutable?	Yes	Yes	Yes	No!
Slicing?	alist[0:2] (by index) * Remember index starts from zero	Not available	Not available (you need to create a loop and extract data)	Not available
When to use?	• Collection of data that does not need random access. • When you need to iterate and modify items.	• You need a key to value data association. • Fast lookup, based on a key. • When data needs to be modified.	• When you want to eliminate duplicated values. • When you need uniqueness of elements.	• To ensure immutability. • Can be used in combination with other data structure, for example a tuple could represent a key in a dictionary.

COMPARISON, B

	Lists	Dictionaries	Sets	Tuples
Create a new	alist = [10,20,30,40]	adict ={'**name**':Tom, '**email**':'tom@msn.com'}	aset={10,20,30,40}	atuple=(10,20,30)
Create an empty	alist = list() or alist = []	adict = dict() or adict = {}	aset=set()	atuple=() *Useless, you cannot add!
Add a value	alist.append(50) This will make: alist = [10,20,30,40,50]	Add a new key/value adict['city']='London'	aset.add(50)	You cannot add! Indirectly, you need to convert to a list
Remove a value	alist.remove(40) This will make: alist = [10,20,30]	Delete key and value adict.pop('email')	Delete the first element of the set aset.pop()	You cannot delete! Indirectly, you need to convert to a list
Update a value	alist[1] = 200 This will make alist = [10,200,30,40]	adict['city']='Athens'	Best way is to convert to list, update and make it a set again	You cannot update! Indirectly, you need to convert to a list
Access an element	If you want to access element 20, you will use the element index number 0 1 2 3 alist = [10,20,30,40] **alist[1]**	If you want to access an **element by key you use the key** adict['email'] it will extract the values for key 'email'	You have to use a loop to iterate	x,y,z = atuple So x = 10, y =20, z=30 Or index: atuple[

Loops	for element in **alist**:
	    print(element)
		for **key,value** in adict.**items()**:
		    print(**key,value**)
			for element in **aset**:
			    print(element
				for element in **atuple**:
				    print(element)
				
				# We prefer to extract using variable