# Brief Python

First Python Course for Beginners

Chapter 4: String, Tuple and Lists

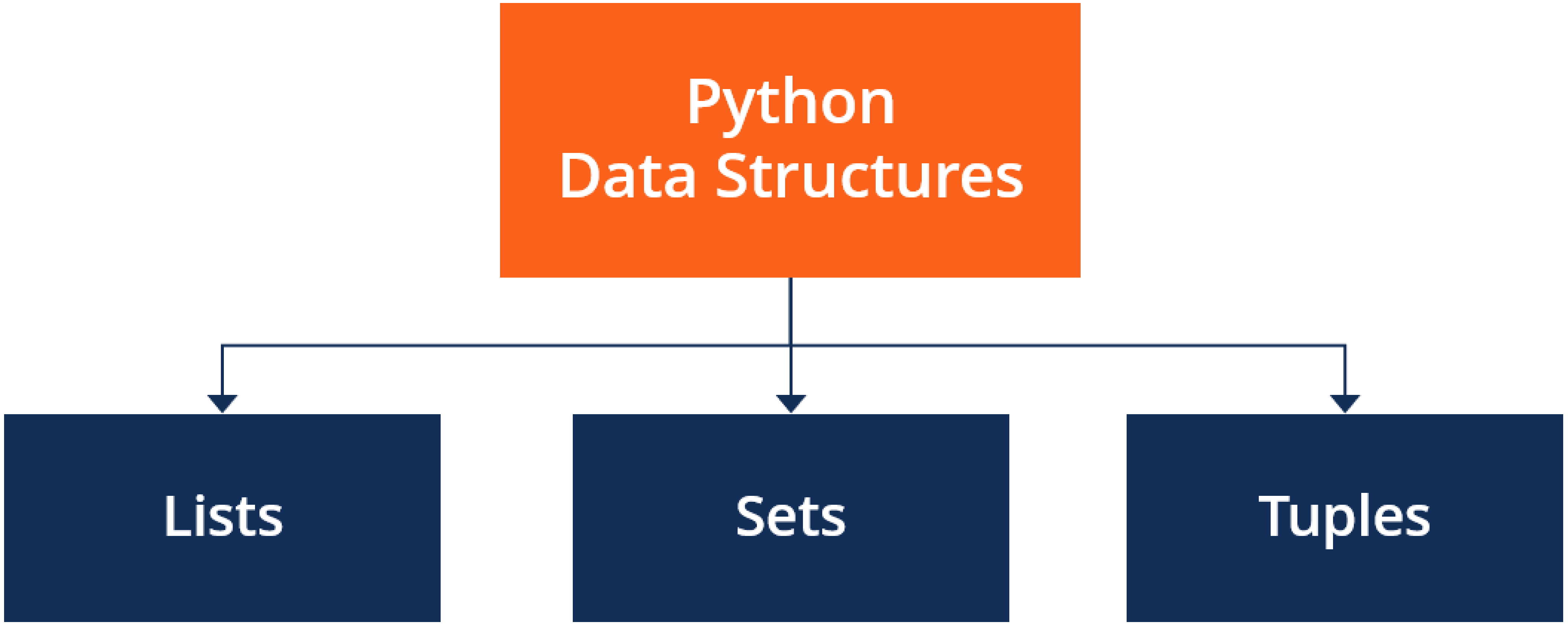
Dr. Eric Chou IEEE Senior Member

## Objectives

- Study the built-in Python Basic Data Structure
    - Tuple
    - List
    - Set

# Python Data Collections

LECTURE 1


Python
Data Structures
Lists
Sets
Tuples

# Tuple

LECTURE 2

## Tuples

Same as lists but

- Immutable
- Enclosed in parentheses
- A tuple  with a single element ***must*** have a comma inside the parentheses:
  - **a = (11,)**

## Examples

```
>>> mytuple = (11, 22, 33)

>>> mytuple[0]
11

>>> mytuple[-1]
33

>>> mytuple[0:1]
(11,)
```

The comma is required!

## Why?

No confusion possible between **[11]** and **11**

**(11)** is a perfectly acceptable expression

- **(11) without the comma** is the integer 11
- **(11, ) with the comma** is  a list containing the integer 11

Sole dirty trick played on us by tuples!

## Tuples are immutable

```
>>> mytuple = (11, 22, 33)
>>> saved = mytuple
>>> mytuple += (44,)
>>> mytuple
(11, 22, 33, 44)
>>> saved
(11, 22, 33)
```

## Things that do not work

```
mytuple += 55
Traceback (most recent call last):Z
…
TypeError:
 can only concatenate tuple (not "int")
to tuple
```

- Can understand that!

## Sorting tuples

```
>>> atuple = (33, 22, 11)
>>> atuple.sort()
Traceback (most recent call last):
…
AttributeError:
'tuple' object has no attribute 'sort'
>>> atuple = sorted(atuple)
>>> atuple
[11, 22, 33]
```

**Tuples are immutable!**

**sorted( ) returns a list!**

## Most other things work!

```
>>> atuple = (11, 22, 33)

>>> len(atuple)
3

>>> 44 in atuple
False

>>> [ i for i in atuple]
[11, 22, 33]
```

## The reverse does not work

```
>>> alist = [11, 22, 33]

>>> (i for i in alist)
<generator object <genexpr> at 0x02855DA0>
```

Does not work!

## Converting sequences into tuples

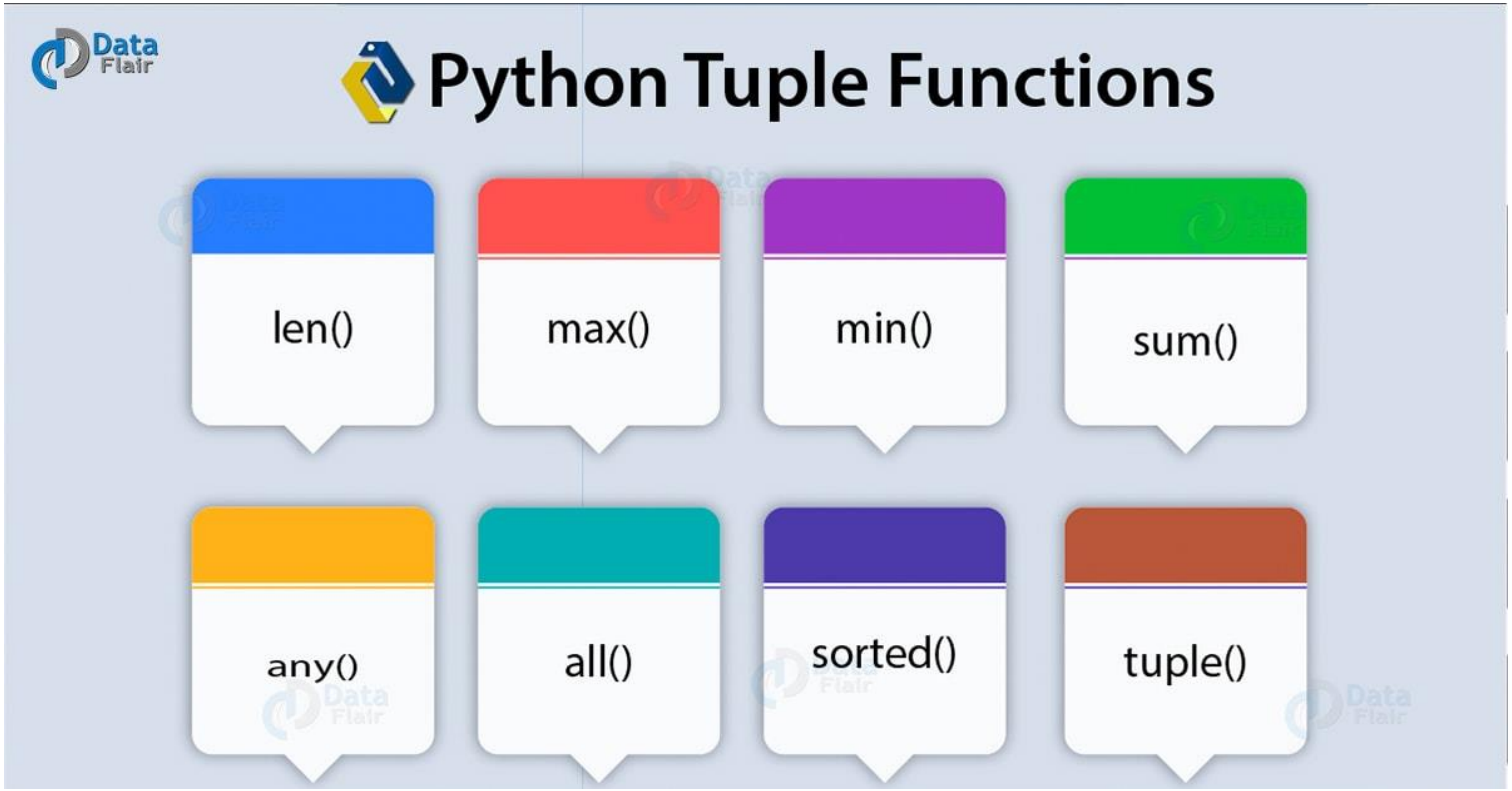
```
>>> alist = [11, 22, 33]

>>> atuple = tuple(alist)

>>> atuple
(11, 22, 33)

>>> newtuple = tuple('Hello World!')

>>> newtuple
('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!')
```


Data Flair
Python Tuple Functions
len()
max()
min()
sum()
any()
all()
sorted()
tuple()

# List

LECTURE 3

## A List is a Kind of Collection

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

## List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

## We Already Use Lists!

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

## Lists and Definite Loops - Best Pals

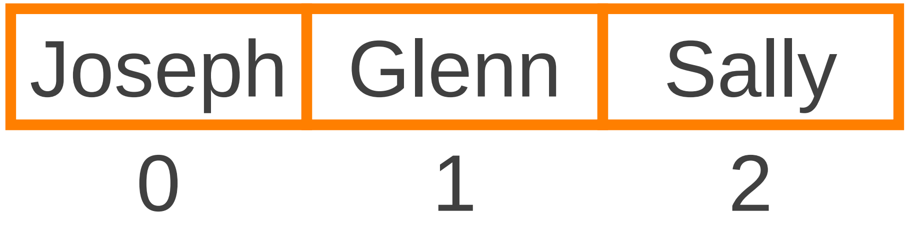
```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:',  friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

```
z = ['Joseph', 'Glenn', 'Sally']
for x in z:
    print('Happy New Year:',  x)
print('Done!')
```

## Looking Inside Lists

- Just like strings, we can get at any single element in a list using an index specified in square brackets



```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
>>>
```

## Lists are Mutable

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change
- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

## How Long is a List?

- The len() function takes a list as a parameter and returns the number of elements in the list
- Actually len() tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

| List Functions | Meanings |
|---|---|
| list.append(x) | Appends object x to list |
| list.count(x) | Returns count of how many times x occurs in list |
| list.remove(x) | Removes xect x from list |
| list.reverse() | Reverses objects of list in place |
| list.extend(seq) | Appends the contents of seq to list |
| list.index(x) | Returns the lowest index in list that x appears |
| list.insert(index, x) | Inserts xect x into list at offset index |
| list.pop(x=list[-1]) | Removes and returns last object or x from list |

## Using the range Function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

## A Tale of Two Loops...

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
    print('Happy New Year:',  friend)

for i in range(len(friends)) :
    friend = friends[i]
    print('Happy New Year:',  friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
```

## Concatenating Lists Using +

- We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

## Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is “up to but not including”

## List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

http://docs.python.org/tutorial/datastructures.html

# Building a List from Scratch

- We can create an empty list and then add elements using the append method

- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

## Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return True or False
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

# Lists are in Order

- A list can hold many items and keeps those items in the order until we do something to change the order

- A list can be sorted
  (i.e., change its order)

- The sort method (unlike in strings) means “sort yourself”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[1])
Joseph
>>>
```

## Built-in Functions and Lists

- There are a number of functions built into Python that take lists as parameters
- Remember the loops we built? These are much simpler.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

**Algorithm 1:**

```
total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print('Average:', average)
```

**Algorithm 2:**

```
numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)
```

```
Enter a number: 3
Enter a number: 9
Enter a number: 5
Enter a number: done
Average: 5.66666666667
```

# Best Friends: Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
```

```
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

Split breaks a string into parts and produces a list of strings.  We think of these as words.  We can access a particular word or loop through all the words.

## Example:

- When you do not specify a delimiter, multiple spaces are treated like one delimiter
- You can specify what delimiter character to use in the splitting

```
>>> line = 'A lot        of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

## Example:

From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

```
Sat
Fri
Fri
Fri
  ...
```

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

## The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

words = line.split()
email = words[1]
print pieces[1]
```

## The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]                    stephen.marquard@uct.ac.za
print pieces[1]
```

## The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print pieces[1]
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
```

## The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

# Set

LECTURE 3

## Sets

Indentified by ***curly braces***

- **{'Alice', 'Bob', 'Carol'}**
- **{'Dean'}** is a ***singleton***

Can only contain ***unique elements***

- ***Duplicates are eliminated***

***Immutable*** like tuples and strings

## Sets do not contain duplicates

```
>>> cset = {11, 11, 22}

>>> cset
{11, 22}
```

# Sets are immutable

```
>>> aset = {11, 22, 33}

>>> bset = aset

>>> aset = aset | {55}

>>> aset
{33, 11, 22, 55}

>>> bset
{33, 11, 22}
```

Union of two sets

## Sets have no order

```
>>> {1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}

>>> {11, 22, 33}
{33, 11, 22}
```

## Sets do not support indexing

```
>>> myset = {'Apples', 'Bananas', 'Oranges'}

>>> myset
{'Bananas', 'Oranges', 'Apples'}

>>> myset[0]
Traceback (most recent call last):
   File "<pyshell#2>", line 1, in <module>
      myset[0]
TypeError: 'set' object does not support
indexing
```

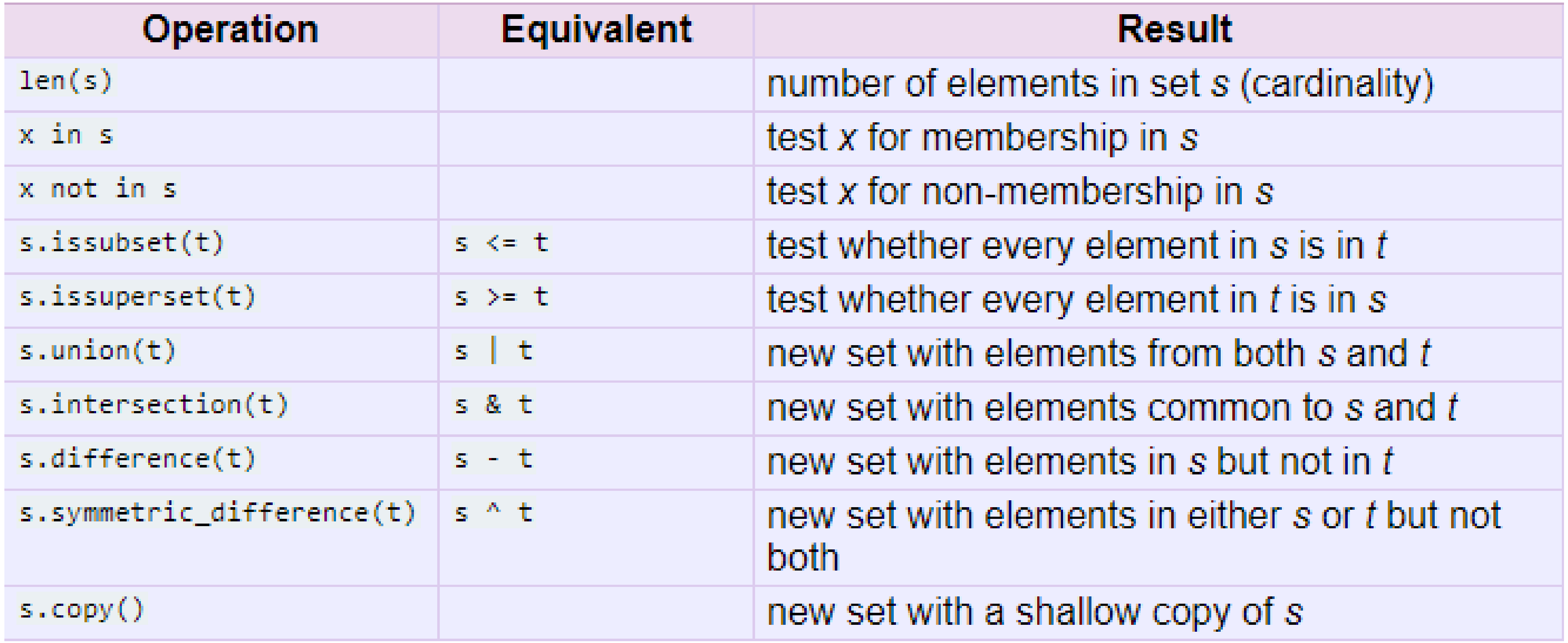
## Examples

```
>>> alist = [11, 22, 33, 22, 44]

>>> aset = set(alist)

>>> aset
{33, 11, 44, 22}

>>> aset = aset + {55}
SyntaxError: invalid syntax
```

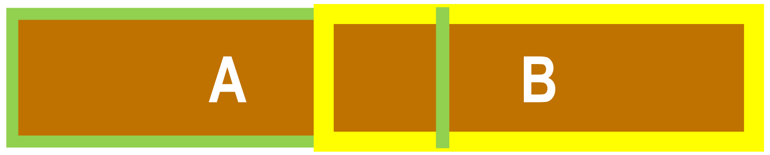


| Operation | Equivalent | Result |
|---|---|---|
| `len(s)` | | number of elements in set *s* (cardinality) |
| `x in s` | | test *x* for membership in *s* |
| `x not in s` | | test *x* for non-membership in *s* |
| `s.issubset(t)` | `s <= t` | test whether every element in *s* is in *t* |
| `s.issuperset(t)` | `s >= t` | test whether every element in *t* is in *s* |
| `s.union(t)` | `s \| t` | new set with elements from both *s* and *t* |
| `s.intersection(t)` | `s & t` | new set with elements common to *s* and *t* |
| `s.difference(t)` | `s - t` | new set with elements in *s* but not in *t* |
| `s.symmetric_difference(t)` | `s ^ t` | new set with elements in either *s* or *t* but not both |
| `s.copy()` | | new set with a shallow copy of *s* |

# Boolean operations on sets (I)

**Union of two sets**

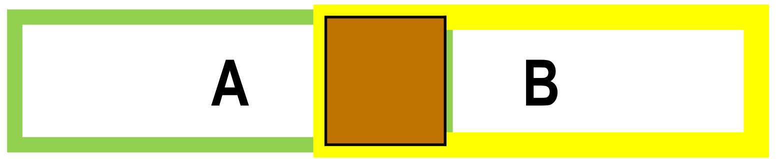
Contains all elements that are in set **A** **or** in set **B**



# Boolean operations on sets (II)

**Intersection of two sets**

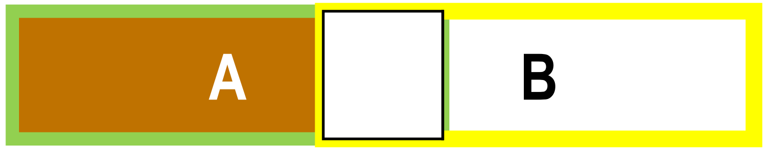
Contains all elements that are in both sets **A and B**



## Boolean operations on sets (III)

**Difference of two sets**

Contains all elements that are **in** **A** but **not in** **B**

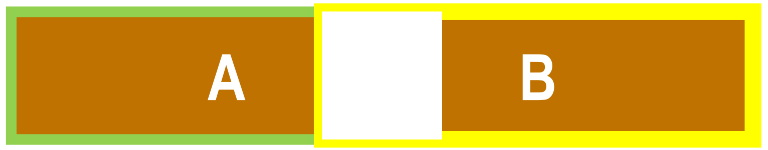


## Boolean operations on sets (IV)

**Symmetric difference  of two sets**

Contains all elements that are **either**

- **in** set **A**  but **not  in** set **B or**
- **in** set **B**  but **not in** set **A**



## Boolean operations on sets (V)

```
>>> aset = {11, 22, 33}
```

```
>>> bset = {12, 23, 33}
```

<u>Union of two sets</u>

-
  ```
  >>> aset | bset
  {33, 22, 23, 11, 12}
  ```

<u>Intersection of two sets:</u>

-
  ```
  >>> aset & bset
  {33}
  ```

## Boolean operations on sets (VI)

```
>>> aset = {11, 22, 33}
```

```
>>> bset = {12, 23, 33}
```

Difference:

-
  ```
  >>> aset – bset
  {11, 22}
  ```

Symmetric difference:

-
  ```
  >>> aset ^ bset
  {11, 12, 22, 23}
  ```

| JSON | Python |
|---|---|
| object | dict |
| array | list |
| string | unicode |
| number (int) | int, long |
| number (real) | float |
| TRUE | TRUE |
| FALSE | FALSE |
| null | None |