

# Python language: Core data structures

The FOSSEE Group

Department of Aerospace Engineering  
IIT Bombay

Mumbai, India

# Outline

## 1 Data structures

- Lists
- Tuples
- Dictionaries
- Sets

# Outline

1

## Data structures

- Lists
- Tuples
- Dictionaries
- Sets

# Outline

## 1 Data structures

- Lists
- Tuples
- Dictionaries
- Sets

# Lists

We already know that

```
num = [1, 2, 3, 4]
```

is a list

- Lists can contain different types
- including nested lists

```
x = ['apple', 1, 'banana', 2.5,  
     'shirt', [1, 2, 3]]
```

```
len(x)
```

# Lists

We already know that

```
num = [1, 2, 3, 4]
```

is a list

- Lists can contain different types
- including nested lists

```
x = ['apple', 1, 'banana', 2.5,  
     'shirt', [1, 2, 3]]
```

```
len(x)
```

# Lists: methods

```
In []: num = [9, 8, 2, 3, 7]
```

```
In []: num + [4, 5, 6]
```

```
Out []: [9, 8, 2, 3, 7, 4, 5, 6]
```

```
In []: num.append([4, 5, 6])
```

```
In []: num
```

```
Out []: [9, 8, 2, 3, 7, [4, 5, 6]]
```

# Lists: methods

```
In []: num = [9, 8, 2, 3, 7]
```

```
In []: num.extend([4, 5, 6])
```

```
In []: num
```

```
Out []: [9, 8, 2, 3, 7, 4, 5, 6]
```

```
In []: num.reverse()
```

```
In []: num
```

```
Out []: [6, 5, 4, 7, 3, 2, 8, 9]
```

```
In []: num.remove(6)
```

```
In []: num
```



# Lists: methods

```
In []: num = [9, 8, 2, 3, 1, 2, 3, 4]
```

```
In []: num.count(2)
```

```
Out []: 2
```

```
In []: num.index(2)
```

```
Out []: 2
```

```
In []: num.pop()
```

```
4
```

```
In []: num
```

```
[9, 8, 2, 3, 1, 2, 3]
```

# Removing elements

- Elements can be removed based on their index OR
- based on the value of the element

```
In []: del num[1]
```

```
In []: num.remove(3)
```

- When removing by value, first element is removed

# Sorting

```
In []: a = [5, 1, 6, 7, 7, 10]
```

```
In []: a.sort()
```

```
In []: a
```

- **sort** method sorts the list in-place
- Use **sorted** if you require a new list
- Pass **reverse=True** to reverse the ordering

```
In []: a = [5, 1, 6, 7, 7, 10]
```

```
In []: sorted(a, reverse=True)
```

```
In []: a
```

# List containership

Recall `num` is `[9, 8, 2, 3, 7]`

```
In []: 4 in num
```

```
Out []: False
```

```
In []: b = 8
```

```
In []: b in num
```

```
Out []: True
```

```
In []: b not in num
```

```
Out []: False
```

# Outline

## 1 Data structures

- Lists
- **Tuples**
- Dictionaries
- Sets

# Tuples: Immutable sequence

```
In []: x = 1, 2, 3
```

```
In []: x  
(1, 2, 3)
```

```
In []: t = (1, 2, 3, 4, 5, 6, 7, 8)
```

```
In []: t[0] + t[3] + t[-1]
```

```
Out []: 13
```

```
In []: t[4] = 7
```

## Note:

- Tuples are immutable - cannot be changed

# Tuples: Immutable sequence

```
In []: x = 1, 2, 3
```

```
In []: x  
(1, 2, 3)
```

```
In []: t = (1, 2, 3, 4, 5, 6, 7, 8)
```

```
In []: t[0] + t[3] + t[-1]
```

```
Out []: 13
```

```
In []: t[4] = 7
```

## Note:

- Tuples are immutable - cannot be changed

# Tuple packing and expansion

Try these:

```
In []: a, b = 1, 2
```

```
In []: a, b, c = 1, 2, 3
```

```
In []: a, b = 1, 2, 3
```

```
In []: a, = [1]
```



# Swapping values

```
In[]: a, b = 5, 7
```

```
In[]: temp = a
```

```
In[]: a = b
```

```
In[]: b = temp
```

- Here's the Pythonic way of doing it

```
In[]: a, b = b, a
```

- The variables can be of different data-types

```
In[]: a = 2.5
```

```
In[]: b = "hello"
```

```
In[]: a, b = b, a
```

# tuple, list functions

Try these:

```
In []: x = 1, 2, 3
```

```
In []: list(x)
```

```
Out []: [1, 2, 3]
```

```
In []: tuple(list(x))
```

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

```
In []: tuple('hello')
```

# Slicing

- Slicing works the same way for all sequence types
- Indices also work the same way
- Lists, tuples, and strings
- **`sequence[initial:final:step]`**

# Slicing & Striding examples

```
In[]: primes = [2, 3, 5, 7, 11, 13, 17]
```

```
In[]: primes[2:6]
```

```
In[]: primes[:4]
```

```
In[]: num = list(range(14))
```

```
In[]: num[1:10:2]
```

```
In[]: num[:10]
```

```
In[]: num[10:]
```

```
In[]: num[::2]
```

```
In[]: num[::-1]
```

# Problem - Day of the Week?

- Strings have methods to manipulate them

## Problem

Given a list, `week`, containing names of the days of the week and a string `s`, check if the string is a day of the week. We should be able to check for any of the forms like, *sat*, *Sat*, *SAT*

- Get the first 3 characters of the string
- Convert it all to lower case
- Check for existence in the list, `week`

# Solution - Day of the Week?

```
week = 'mon tue wed thu fri sat sun'.split()  
s = 'Sunday'  
s.lower()[ :3] in week
```

OR

```
s[ :3].lower() in week
```

# Outline

## 1 Data structures

- Lists
- Tuples
- **Dictionaries**
- Sets

# Dictionaries: Introduction

- Lists index using integers; recall that :

If  $p = [2, 3, 5, 7]$   
 $p[1]$  is equal to 3

- Dictionaries index not just with integers!
- Following examples are for string indices
- Will also work for integers, floats, tuples, and others
- **No ordering of keys!**



# Dictionaries ...

```
In []: mtdict = {}
```

```
In []: d = {'png' : 'image file',  
           'txt' : 'text file',  
           'py'  : 'python code',  
           'java': 'bad code',  
           'cpp' : 'complex code'}
```

```
In []: d['txt']
```

```
Out []: 'text file'
```

# Dictionaries ...

```
In []: 'py' in d
```

```
Out []: True
```

```
In []: 'jpg' in d
```

```
Out []: False
```

# Dictionaries ...

```
In []: list(d.keys())
```

```
Out []: ['cpp', 'py', 'txt', 'java', 'png']
```

```
In []: list(d.values())
```

```
Out []: ['complex code', 'python code',  
        'text file', 'bad code',  
        'image file']
```

# Inserting elements

```
d[key] = value
```

```
In []: d['bin'] = 'binary file'
```

```
In []: d
```

```
Out []:
```

```
{'bin': 'binary file',  
 'cpp': 'complex code',  
 'java': 'bad code',  
 'png': 'image file',  
 'py': 'python code',  
 'txt': 'text file'}
```

Duplicate keys are overwritten!

# Adding & Removing Elements

- Adding a new key-value pair

```
In[]: d['c++'] = 'C++ code'
```

```
In[]: d
```

- Deleting a key-value pair

```
In[]: del d['java']
```

```
In[]: d
```

- Assigning to existing key, modifies the value

```
In[]: d['cpp'] = 'C++ source code'
```

```
In[]: d
```

# Dictionaries: methods

```
In []: d.keys()
```

```
In []: d.values()
```

```
In []: d.get('bin')
```

```
Out []: 'binary file'
```

```
In []: d.get('junk')    # No error!
```

```
In []: d.update({'bin': 'Binary file',  
.....:          'exe': 'executable'})
```

```
In []: d.clear()
```

# Dictionaries: containership

```
In []: 'bin' in d
```

```
Out []: True
```

```
In []: 'hs' in d
```

```
Out []: False
```

## Note

- We can check for the containership of keys only
- Not values

# Dictionaries: iteration

```
In []: for v in d:  
.....:     print(v)  
.....:
```

```
In []: for v in d.values():  
.....:     print(v)  
.....:
```

```
In []: for k, v in d.items():  
.....:     print(k, v)  
.....:
```



# dict built-in

Like `list`, `tuple` can also create dicts like so:

```
In []: d = dict(png='image file',  
.....:          txt='text file',  
.....:          py='python code')
```

- Check more of the dictionary methods yourself

# Problem 2.1

You are given date strings of the form “29 Jul, 2009”, or “4 January 2008”. In other words a number, a string and another number, with a comma sometimes separating the items.

Write a program that takes such a string as input and prints a tuple (yyyy, mm, dd) where all three elements are ints.

# Some hints

- Simplify the problem
- Try to explain how someone can do it
- Replace ',' with space
- **`string.split()`** method is your friend
- Need a mapping from month (string) to a number:  
use a dict

# Partial solution

```
month2mm = {'jan': 1, 'feb': 2, #...  
            }  
date = input('Enter a date string: ')  
date = date.replace(',', ' ')  
day, month, year = date.split()  
dd, yyyy = int(day), int(year)  
mon = month[:3].lower()  
mm = month2mm[mon]  
print((yyyy, mm, dd))
```

# Outline

## 1 Data structures

- Lists
- Tuples
- Dictionaries
- **Sets**

# Sets

- Conceptually identical to the sets in mathematics
- Simplest container, mutable
- No ordering, no duplicates
- usual suspects: union, intersection, subset ...
- $>$ ,  $>=$ ,  $<$ ,  $<=$ ,  $\text{in}$ , ...

```
In[]: a_list = [1, 2, 1, 4, 5, 6, 2]
In[]: a = set(a_list)
In[]: a
```

# Operations on Sets

```
In []: f10 = set([1, 2, 3, 5, 8])
```

```
In []: p10 = set([2, 3, 5, 7])
```

- Mathematical operations can be performed on sets

- Union

```
In []: f10 | p10
```

- Intersection

```
In []: f10 & p10
```

- Difference

```
In []: f10 - p10
```

- Symmetric Difference

```
In []: f10 ^ p10
```

# Sub-sets

- Proper Subset

```
In[]: b = set([1, 2])
```

```
In[]: b < f10
```

- Subsets

```
In[]: f10 <= f10
```



# Elements of sets

- Containership

```
In[]: 1 in f10
```

```
In[]: 4 in f10
```

- Iterating over elements

```
In[]: for i in f10:
```

```
.....:     print(i)
```

```
.....:
```

- Subsets

```
In[]: f10 <= f10
```

# Problem 2.2

Given a dictionary of the names of students and their marks,

- 1 identify how many duplicate marks are there?
- 2 and what are these duplicate marks?

# Hints

- Use a set to find the unique marks: first part
- Second part can be done in a few different ways

# Possible solution

```
students = {'x': 60, 'y': 60, 'z': 55}
all_marks = list(students.values())
unique = set(all_marks)
n_dups = len(all_marks) - len(unique)
print(n_dups, 'duplicates')
```

*# Part 2*

```
for i in unique:
    all_marks.remove(i)
print('Duplicates are', set(all_marks))
```

# What did we learn?

- Core Python data structures:
  - Lists
  - Tuples
  - Dictionaries
  - Sets

# Homework

- Explore all the methods of the various data structures you have seen so far.
- Read the documentation for each of these methods.
- Solve all problems discussed