Overview

- Lists
- Tuples
- Dictionaries
- Assignment statement
- Data structure – parsing, analysis
- Comparing sequences

Data structure

- A data structure is a particular way of **organizing data** in a computer so that it can be used effectively.

	math	language	science
Jane	10	50	78
Tom	12	35	60
Julia	13	22	40
Amy	14	15	66
Peter	15	80	80

How can we properly/efficiently organize this data in python

Overview

- Python has two other sequence structures: *tuples* and *lists*.
- These contain zero or more elements.

```
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

- Unlike strings, the elements can be of different types.

```
weekdays = ['Monday', 1, 'Wednesday', 'Thursday', ['Friday', 'hello']]
```

- It will allow you to create structures as deep and complex as you like.

Overview

- Why does Python contain both lists and tuples?
- Lists are *mutable*, meaning you can insert and delete elements.
- Tuples are *immutable*; when you assign elements to a tuple, they're baked in the cake and can't be changed.
- String is also ...

```
>>> a = 'hello'
>>> print(a[2])
l
>>> a[2] = 'a'
```

Overview

- Why does Python contain both lists and tuples?
- Lists are *mutable*, meaning you can insert and delete elements.
- Tuples are *immutable*; when you assign elements to a tuple, they're baked in the cake and can't be changed.
- String is also *immutable.*

```
>>> a = 'hello'
>>> print(a[2])
l
>>> a[2] = 'a'
error
```

Lists

- A list is made from zero or more elements, separated by commas, and surrounded by square brackets.

```
>>> empty_list = [ ]
>>> weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
>>> big_birds = ['emu', 'ostrich', 'cassowary']
>>> first_names = ['Graham', 'John', 'Terry', 'Terry',
 'Michael']

>>> empty_list = list()
```

Convert other data type to lists

- Python's list() function converts other data types to lists.

```
>>> str1 = 'hello world' ; print(str1)
'hello world'
>>> str1.split(' ')
['hello', 'world']
>>> str2 = list(str1) ; print(str2)
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> birthday = '1/6/1952'
>>> birthday.split('/')
['1', '6', '1952']
```

Get an item by using [offset]

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
'Groucho'
>>> marxes[1]
'Chico'
>>> marxes[2]
'Harpo'
>>> marxes[-1]
'Harpo'
>>> marxes[-2]
'Chico'
>>> marxes[-3]
'Groucho'
```

Get an item by using [offset]

- As with strings, you can extract a single value from a list by specifying its offset

```
>>> squares = [1, 4, 9, 12, 25]
>>> squares[3] = 16
>>> print(squares)
[1, 4, 9, 16, 25]
```

```
>>> squares[-1]
25
>>> squares[-3:]
9, 16, 25
```

Lists

- **Concatenation via +**
  - **Returns new list**

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- **append() method**
  - **Adding at the end of the list**
  - **Modifying the list**

```
>>> squares.append(55)
[1, 4, 9, 16, 25, 55]
```

Lists

- **Assignment to slices**

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters[2:5] = ['C', 'D', 'E']
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> letters[2:5]  = []
['a', 'b', 'f', 'g']
>>> letters[:] = []
[]
```

- **Length**

```
>>> len(letters)
0
```

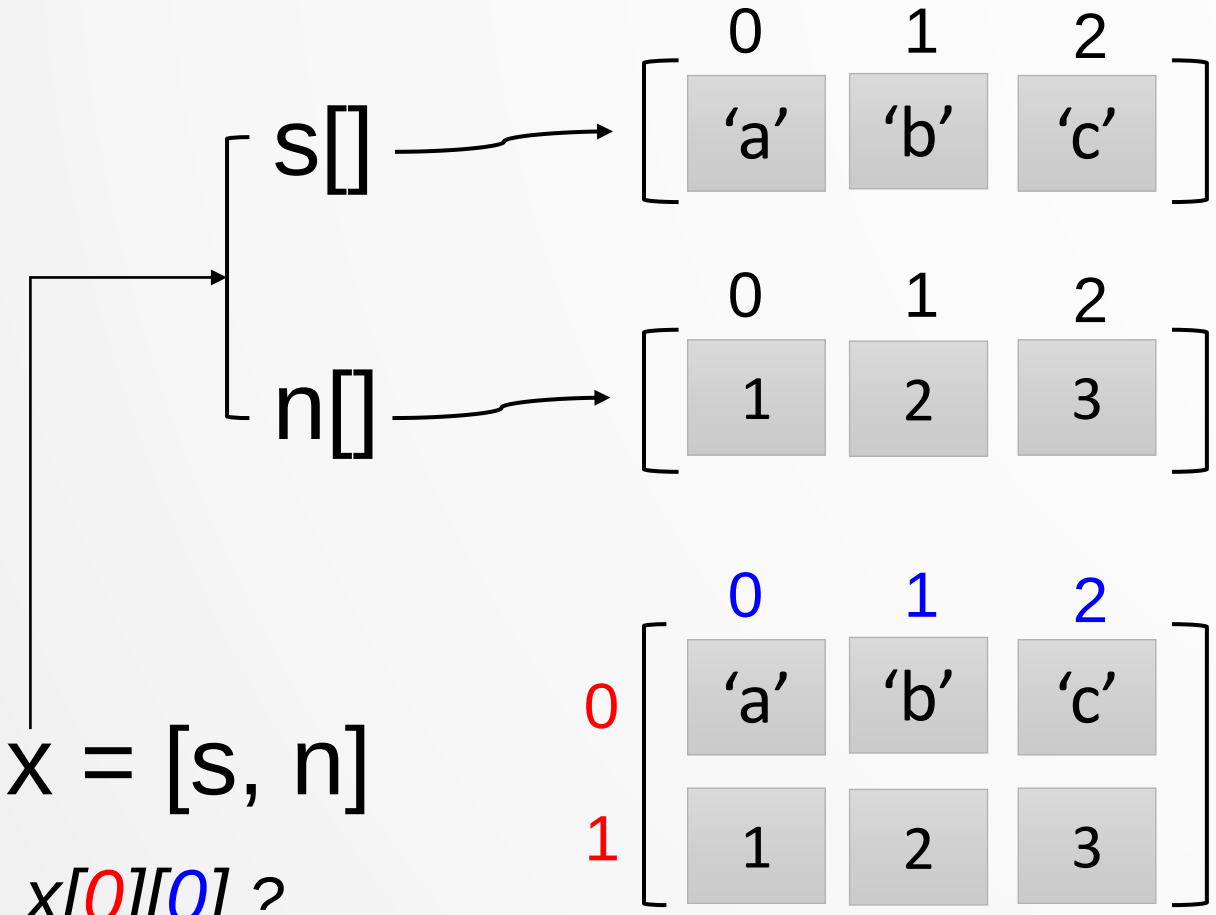
Lists

- **Lists in lists**

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> print(x)
[['a', 'b', 'c'], [1, 2, 3]]
```

Lists

- **Lists in lists**



Lists

- Lists can contain elements of different types, including other lists, as illustrated here:

```
>>> small_birds = ['hummingbird', 'finch']

>>> extinct_birds = ['dodo', 'passenger pigeon',
                     'Norwegian Blue']

>>> carol_birds = [3, 'French hens', 2, 'turtledoves']

>>> all_birds = [small_birds, extinct_birds, 'macaw',
                 carol_birds]
```

Lists

- **Lists in lists**

```
>>> all_birds
[['hummingbird', 'finch'], ['dodo', 'passenger pigeon', 'Norwegian Blue'], 'macaw', [3, 'French hens', 2, 'turtledoves']]
```

List

- ['hummingbird', 'finch'] — *List*
- ['dodo', 'passenger pigeon', 'Norwegian Blue'] — *List*
- 'macaw' — *String*
- [3, 'French hens', 2, 'turtledoves'] — *List*

all_birds[1]? *all_birds[1][1]?* *all_birds[1][1][1]?*

Lists

```
['hummingbird', 'finch']

['dodo',  'passenger pigeon', 'Norwegian Blue']

'macaw'

[3, 'French hens', 2, 'turtledoves']
```

```
>>> all_birds[0]
>>> all_birds[1][0]
>>> all_birds[2]
>>> all_birds[2][3]
>>> all_birds[1][2][3:5]
```

Lists – append(), extend()

- **Add an Item to the End with append()**

```
>>> countries = ['Kazakhstan', 'Korea', 'Russia']
```

We want to add one item (India) to the list

```
>>> countries.append('India'); print(countries)
['Kazakhstan', 'Korea', 'Russia', 'India']
>>> w_countries = ['USA', 'Canada']

1) >>> countries.extend(w_countries); print(countries)
2) >>> countries  = countries + w_countries
['Kazakhstan', 'Korea', 'Russia', 'India', 'USA',
'Canada']
```

Lists – insert()

- The append() function adds items only to the end of the list.

```
>>> countries = [‘Kazakhstan’, ‘Korea’, ‘Russia’]

>>> countries.insert(1, ‘USA’)
[‘Kazakhstan’, ‘USA, ‘Korea’, ‘Russia’]

>>> countries.insert(10, ‘China’)
[‘Kazakhstan’, ‘USA, ‘Korea’, ‘Russia’, ‘China’]
```

Lists - remove(), pop()

- If you're not sure or don't care where the item is in the list, use remove() to delete it by value.

```
>>> countries = ['Kazakhstan', 'Korea', 'Russia']
>>> countries.remove('Korea'); print(countries)
['Kazakhstan', 'Russia']
```

- You can get an item from a list and delete it from the list at the same time by using pop().

```
                    pop(0)          pop(1)   pop(), pop(-1)
>>> countries = ['Kazakhstan', 'Korea', 'Russia']
>>> countries.pop(); print(countries)
'Russia'
['Kazakhstan', 'Korea']
```

Lists – index(), count()

```
>>> countries = [‘Kazakhstan’, ‘Korea’, ‘Russia’]
>>> countries.index(‘Korea’)
1
>>> countries.count(‘Korea’)
1
```

Lists - del()

- **del statement**

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del(a[0])
>>> print(a)
[ 1, 66.25, 333, 333, 1234.5 ]
>>> del(a[2:4])
[ 1, 66.25, 1234.5]
>>> del(a[:])
>>> print(a)
[]
```

Del() is a Python statement, not a list method
a.del(a[0])-> this is wrong

```
>>> del(a)
>>> print(a)
error
```

Convert other data type to lists

- Python's list() function converts other data types to lists.

```
>>> str1 = 'hello world' ; print(str1)
'hello world'
>>> str1.split(' ')
['hello', 'world']
>>> str2 = list(str1) ; print(str2)
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> birthday = '1/6/1952'
>>> birthday.split('/')
['1', '6', '1952']
```

Lists – index(), count(), join()

- join() is a string method, not a list method

```
>>> ''.join(countries)
'KazakhstanKoreaRussia'
>>> ' '.join(countries)
'Kazakhstan Korea Russia'
>>> str1 = ', '.join(countries); print(str1)
'Kazakhstan, Korea, Russia'
>>> list1 = str1.split(', '); print(str1)
['Kazakhstan', 'Korea', 'Russia']
```

Tuples

- Similar to lists, tuples are sequences of arbitrary items.
- Unlike lists, tuples are ***immutable***, meaning you can't add , delete, or change items after the tuple is defined.

```
>>> alist = ['a', 'b', 'c']
>>> atuple = ('a', 'b', 'c')
>>> atuple2 = 'a', 'b', 'c'
>>> alist[0] = 'A'
>>> print(A)
['A', 'b', 'c']
>>> atuple[0] = 'A'
error
```

Functions: tuple(), list()

- The tuple() conversion function makes tuples from other things.
- The function list() is also same.

```
>>> data_list = [‘data1’, ‘data2’, ‘data3’]
>>> data_tuple  = tuple(data_list)
>>> type(data_tuple)
<class 'tuple’>
>>> data_list = list(data_tuple)
>>> type(data_list)
<class ‘list’>
```

Mutable/immutable

Course code	Session	Semester	Location	
CSCI 115	2019/08/28	2	<Any>	Apply

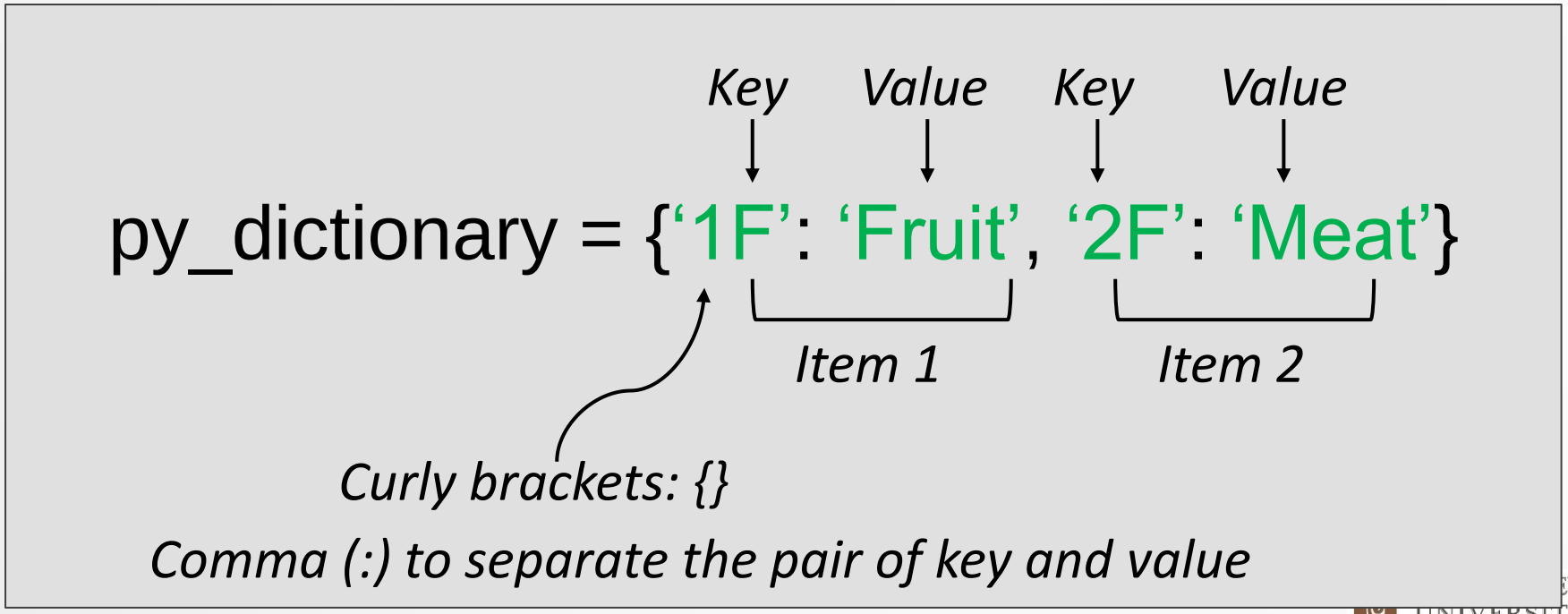
Student. No	Surname	First Name	Middle Name	CA1	CA2
1110404668	ABDULLAHI	BELLO	GWADABAWA	12	14
11104041002	Abdulmumin	Abubakar	I	10	15
1110404719	Abubakar	Bilyaminu		8	15
1110404387	Ahmad	Usman		15	8
1110404582	Ahmad	Usman	Usman	7	13
1110404274	Aminu	Tijjani	Muhammad	18	13
1110404026	Balarabe	Ibrahim	Tukur	3	15

List vs. Tuple

- You can often use tuples in place of lists, but they have m any fewer functions (no append(), insert(), and so on)
  - Tuples use less space.
  - You can prevent to modify items by mistake.
  - You can use tuples as dictionary keys.
  - *Named tuples* (see Named Tuples) can be a simple alternative to objects.
  - Function arguments are passed as tuples (see Functions).

Dictionaries {}

- A *dictionary* is similar to a list, but the order of items doesn't matter
- Instead of offset, they have ***a unique key*** to associate with each value
- The type is mostly ***string***, but it can be any of Python's im mutable types: ***boolean, integer, float, tuple***, and others



Dictionaries: update()

- **add or change the new items**

```
>>> dictionary = {'1F': 'Fruit', '2F': 'Meat', '3F':'Electronics'}
>>> dictionary['1F']              # str1 = dictionary['1F']
'Fruit'
>>> dictionary.get('1F')
'Fruit'
>>> dictionary['6F']
Error
>>> dictionary.get('6F')
..
>>> dictionary.get('6F', 'Not in dictionary')
'Not in dictionary'
```

Dictionaries: key(), value()

- We can get the key and its value

```
>>> dictionary.keys()
dict_keys(['1F', '2F', '3F'])
>>>tm1 = list(dictionary.keys());    print(tm1)
['1F', '2F', '3F']

>>> dictionary.values()
dict_values(['Fruit', 'Meat', 'Electronics'])
>>> tm2 = list(dictionary.values());    print(tm2)
['Fruit', 'Meat', 'Electronics']
```

Dictionaries: update()

- **add or change the new items**

```
>>> dictionary = {'1F': 'Fruit', '2F': 'Meat', '3F':'Electronics'}
>>> dictionary.update({'4F':'Clothes'});    print(dictionary)
 {'1F': 'Fruit', '2F': 'Meat', '3F': 'Electronics', '4F': 'Clothes'}
>>> dictionary2 = {'5F': 'Fish', '6F': 'Toy'};)

>>> dictionary.update(dictionary2);    print(dictionary)
{'1F': 'Fruit', '2F': 'Meat', '3F': 'Electronics', '4F': 'Clothes', '5F
': 'Fish', '6F': 'Toy'}

>>> dictionary['2F'] = 'Computer';    print(dictionary)
{'1F': 'Book', '2F': 'Computer', '3F': 'Electronics', '4F': 'Clothe
s', '5F': 'Fish', '6F': 'Toy'}
```

Compare data structures

- Square bracket for List **[]**
- Parenthesis(or none) for tuple **()**
- Curly bracket for dictionary **{}**, and pair of ***key:value***
- In each case, you access a single element with square brackets

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> marx_dict = {'Groucho': 'banjo', 'Chico': 'piano',
'Harpo': 'harp'}
>>> marx_list[2]                    Integer Offset
'Harpo'
>>> marx_tuple[2]                   Integer Offset
'Harpo'
>>> marx_dict['Harpo']                  Key
'harp'
```

Data structure

- A data structure is a particular way of **organizing data** in a computer so that it can be used effectively.

	math	language	science
Jane	10	50	78
Tom	12	35	60
Julia	13	22	40
Amy	14	15	66
Peter	15	80	80

How can we properly/efficiently organize this data in python

Data structure – parsing

	math	language	science
Jane	10	50	78
Tom	12	35	60
Julia	13	22	40
Amy	14	15	66
Peter	15	80	80

```
math language science
Jane 10 50 78
Tom 12 35 60
Julia 13 22 40
Amy 14 15 66
Peter 15 80 80
```

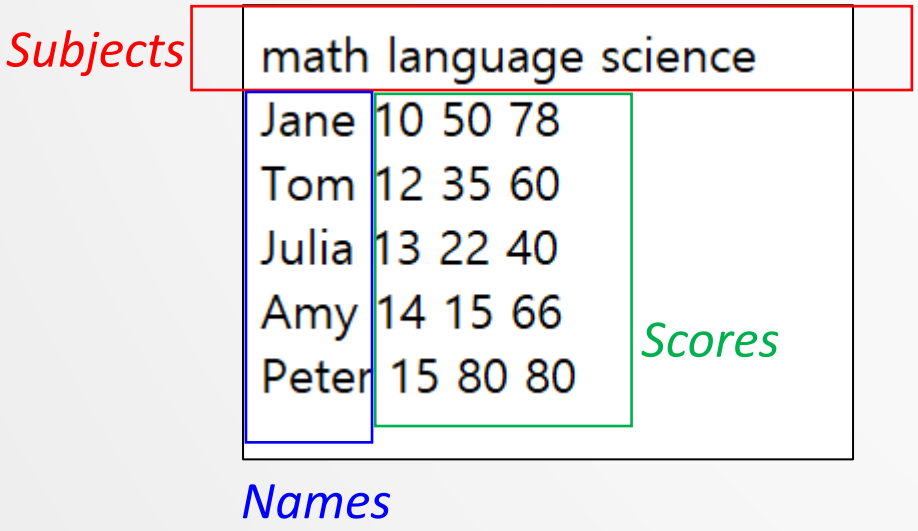
L_1 = ‘math language science’

L_2 = ‘Jane 10 50 78’

⋮

Data structure – parsing

	math	language	science
Jane	10	50	78
Tom	12	35	60
Julia	13	22	40
Amy	14	15	66
Peter	15	80	80



L_1 = 'math language science'

L_2 = 'Jane 10 50 78'

⋮

Data structure – organizing

```
{'math': 10, 'language': 50, 'science': 78}
```

Dictionary

```
{
'Jane': {'math': 10, 'language': 50, 'science': 78},
'Tom': {'math': 12, 'language': 35, 'science': 60},
'Julia': {'math': 13, 'language': 22, 'science': 40},
'Amy': {'math': 14, 'language': 15, 'science': 66},
'Peter': {'math': 15, 'language': 80, 'science': 80}
}
```

Dictionary in the dictionary

Data structure – organizing

```
dic_subject = ('math', 'language', 'science')
dic_name = ('Jane', 'Tom', 'Julia', 'Amy', 'Peter')
dic_lecture = [
    {'math':10, 'language':50, 'science': 78},
    {'math':12, 'language':35, 'science': 60},
    {'math':13, 'language':22, 'science': 40},
    {'math':14, 'language':15, 'science': 66},
    {'math':15, 'language':80, 'science': 80},
]
dic_grade = {
            dic_name[0]:dic_1[0], dic_name[1]:dic_1[1],
            dic_name[2]:dic_1[2], dic_name[3]:dic_1[3],
            dic_name[4]:dic_1[4]
}
```

Data structure – analysis

```
for i in dic_grade:
    tm = (dic_grade[i]['math']+dic_grade[i]['language']\
          +dic_grade[i]['science'])/len(dic_class)
    print(i,'\'s ','average score is', tm)
```

Output:

```
Jane 's  average score is 46.0
Tom 's  average score is 35.6
Julia 's  average score is 25.0
Amy 's  average score is 31.6
Peter 's  average score is 58.3
```

Data structure – extension