Lecture 8

Topics to be covered:

- String basics
- Lists basics
- Set basics
- Dictionary basics

String basics

A *string* is a sequence of characters
Example: abrakadabra,
We can store it and associate a name/variable with it.

To strings like "hi" or 'broom' we refer to as a *string literal* in a Python program.

String basics

A *string* is a sequence of characters
Example: abrakadabra,
We can store it and associate a name/variable with it.

To strings like "hi" or 'broom' we refer to as a *string literal* in a Python program.

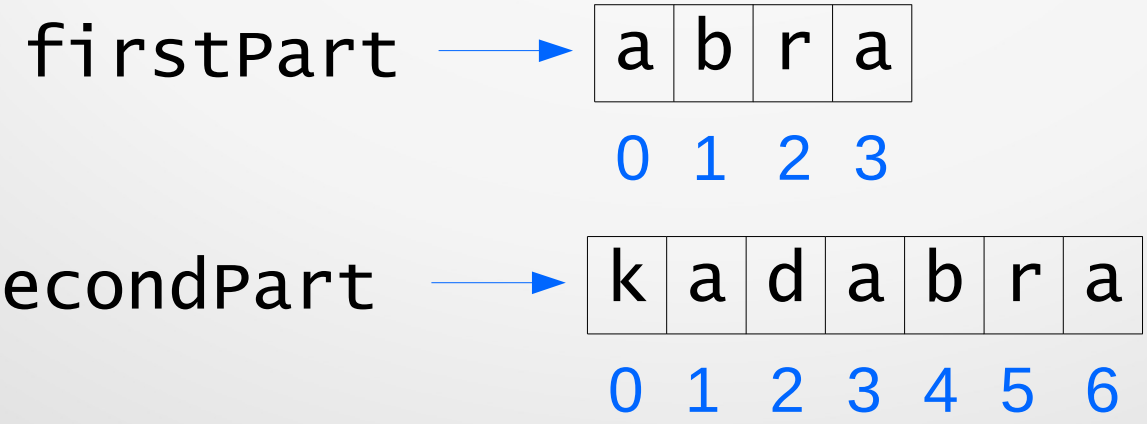
```
>>> firstPart = “abra”
>>> secondPart = “kadabra”
```

String basics

A *string* is a sequence of characters
Example: abrakadabra,
We can store it and associate a name/variable with it.

To strings like "hi" or 'broom' we refer to as a *string literal* in a Python program.

```
>>> firstPart = “abra”
>>> secondPart = “kadabra”
```



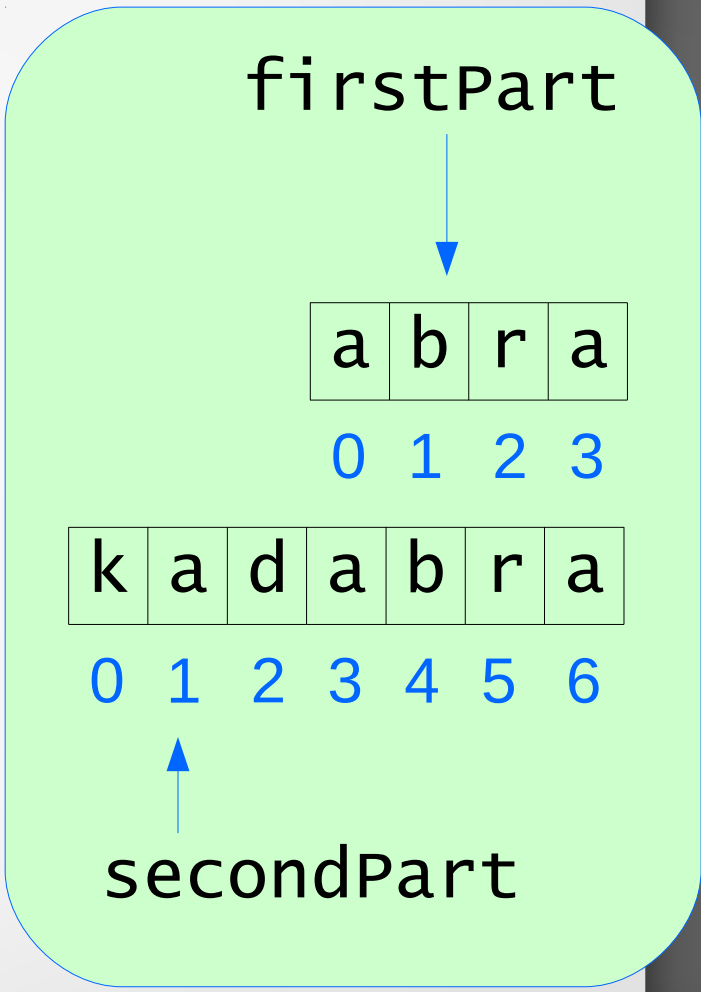
String basics

What can I do with strings in Python?

String basics

What can I do with strings in Python?

```
>>> firstPart = “abra”
>>> secondPart = “kadabra”
```

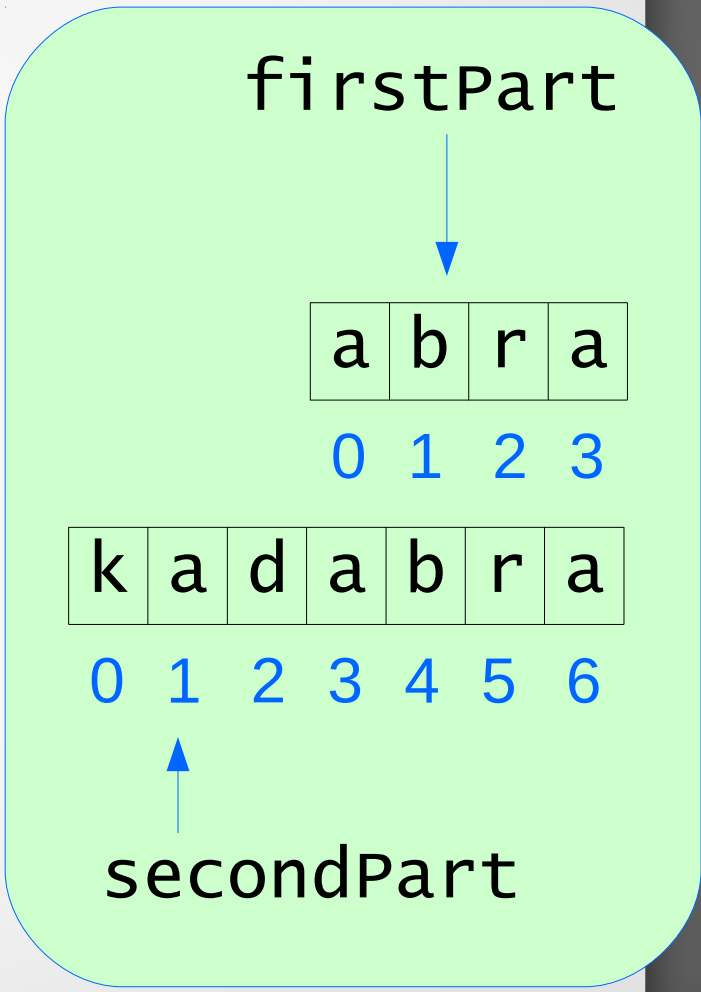


String basics

What can I do with strings in Python?

```
>>> firstPart = “abra”
>>> secondPart = “kadabra”

>>> firstPart[2]
‘r’
>>> secondPart[0]
‘k’
>>> secondPart[2:]
‘dabra’
>>> secondPart[1:4]
‘ada’
>>> firstPart + secondPart
‘abrakadabra’
>>> len(firstPart)
4
```



String basics

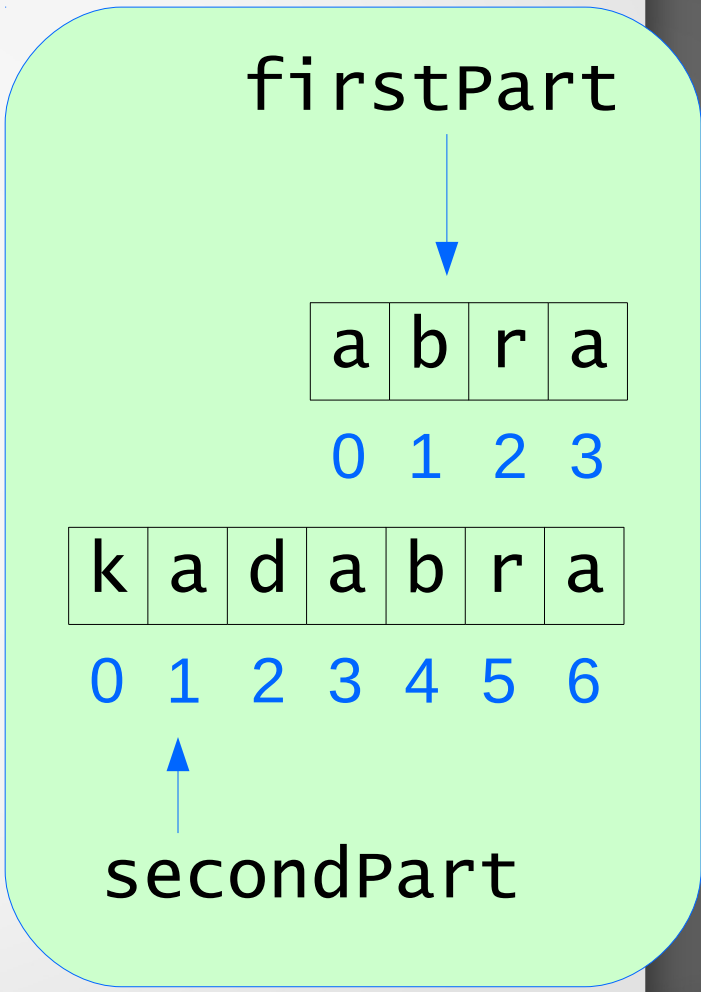
What can I do with strings in Python?

```
>>> firstPart+"cloud"
'abracloud'

>>> firstPart+'/'+secondPart
'abra/kadabra'

>>> firstPart*3
'abraabraabra'

>>> firstPart
'abra'
```



String basics: in-class work

Do the activities 1-3 in the in-class activity handout

Lists basics

A *container* is a construct used to group related values together and contains references to other objects instead of data.

A `list` is a container created by surrounding a sequence of variables or literals with brackets `[]`.

Example: `my_list = [10, 'abc',2.5]`

creates a new list variable `my_list` that contains the three items: 10, `'abc'`, and `2.5`

A list item is called an *element* .

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with 0.

```
>>> myList=[1,2,3,"a","hello",8]
>>> myList[2]
3
```

3 is at position 2

position 2

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with 0.

```
>>> myList=[1,2,3,"a","hello",8]
>>> myList[3]
3
>>> my_list = [ ]
```

3 is at position 2

position 2

create an empty list

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList=[1,2,3,"a","hello",8]
>>> myList[3]
3
```

3 is at position 2

position 2

```
>>> my_list = [ ]
```

create an empty list

```
>>> names = ["Jane","Margo","Sally"]
>>> names
```

print all elements of the list names

```
['Jane', 'Margo', 'Sally']
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList=[1,2,3,”a”,”hello”,8]
>>> myList[3]
3
```

3 is at position 2

position 2

```
>>> my_list = [ ]
```

create an empty list

```
>>> names = ["Jane","Margo","Sally"]
>>> names
```

print all elements of the list names

```
['Jane', 'Margo', 'Sally']
```

```
>>> names.append("Sam")
```

add another element to the list

```
>>> names
```

display the list

```
['Jane', 'Margo', 'Sally', 'Sam']
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList2=[1,2,3,5]
>>> myList2[2] = 7
>>> myList2
[1,2,7,5]
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList2=[1,2,3,5]
>>> myList2[2] = 7
>>> myList2
[1,2,7,5]

>>> len(myList2)          get the length of the list
4
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList2=[1,2,3,5]
>>> myList2[2] = 7
>>> myList2
[1,2,7,5]

>>> len(myList2)          get the length of the list
4

>>> myList2.pop()         remove the last element
5
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList2=[1,2,3,5]
>>> myList2[2] = 7
>>> myList2
[1,2,7,5]

>>> len(myList2)          get the length of the list
4

>>> myList2.pop()         remove the last element
5

>>> myList2.pop(1)     remove the element with index 1
>>> myList2
[1,7]
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with 0.

```
>>> myList3=[10,9,8,7,6,5,4,6,3,2,1]
>>> myList3.remove(6)  finds and removes first occurrence of value 6 in the list
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList3=[10,9,8,7,6,5,4,6,3,2,1]
>>> myList3.remove(6)  finds and removes first occurrence of value 6 in the list

>>> myList3
[10,9,8,7,5,4,6,3,2,1]
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList3=[10,9,8,7,6,5,4,6,3,2,1]
>>> myList3.remove(6)  finds and removes first occurrence of value 6 in the list

>>> myList3
[10,9,8,7,5,4,6,3,2,1]

>>> myList3.index(8)   get the position/index of value 8
2
```

Lists basics

A `list` is also a sequence (just like `string`), meaning the contained elements are ordered by position in the list, known as the *element's index*, starting with `0`.

```
>>> myList3=[10,9,8,7,6,5,4,6,3,2,1]
>>> myList3.remove(6)  finds and removes first occurrence of value 6 in the list

>>> myList3
[10,9,8,7,5,4,6,3,2,1]

>>> myList3.index(8)    get the position/index of value 8
2

>>> myList3.insert(3,12)
>>> myList3
[10,9,8,12,7,5,4,6,3,2,1]
```

List basics: in-class work

Do the items 4-5 in the in-class activity handout

Set basics

A `set` is an *unordered collection of unique elements*.

Set basics

A `set` is an *unordered* collection of *unique* elements.

```
>>> mySet=set()  creates an empty set
```

Set basics

A `set` is an *unordered* collection of *unique* elements.

```
>>> mySet=set()
```
creates an empty set

```
>>> mySet = set([10,20,30,40])
or
>>> mySet = {10,20,30,40}
```
- creates a set with values 10, 20, 30, 40

Set basics

A `set` is an *unordered* collection of *unique* elements.

```
>>> mySet=set()
```
creates an empty set

```
>>> mySet = set([10,20,30,40])
or
>>> mySet = {10,20,30,40}
```
- creates a set with values 10, 20, 30, 40

```
>>> mySet2 = {10,10,20,30,30,30,40,56}
>>> mySet2
{40, 10, 20, 56, 30}
```
- all duplicates are removed, no order

Set basics

A `set` is an *unordered* collection of *unique* elements.

```
>>> mySet=set()
```
creates an empty set

```
>>> mySet = set([10,20,30,40])
or
>>> mySet = {10,20,30,40}
```
- creates a set with values 10, 20, 30, 40

```
>>> mySet2 = {10,10,20,30,30,30,40,56}
>>> mySet2
{40, 10, 20, 56, 30}
```
- all duplicates are removed, no order

```
>>> len(mySet2)
5
```
returns the length of the set mySet2

Set basics: modifying sets

Sets are *mutable*, therefore elements can be added or removed using set methods.

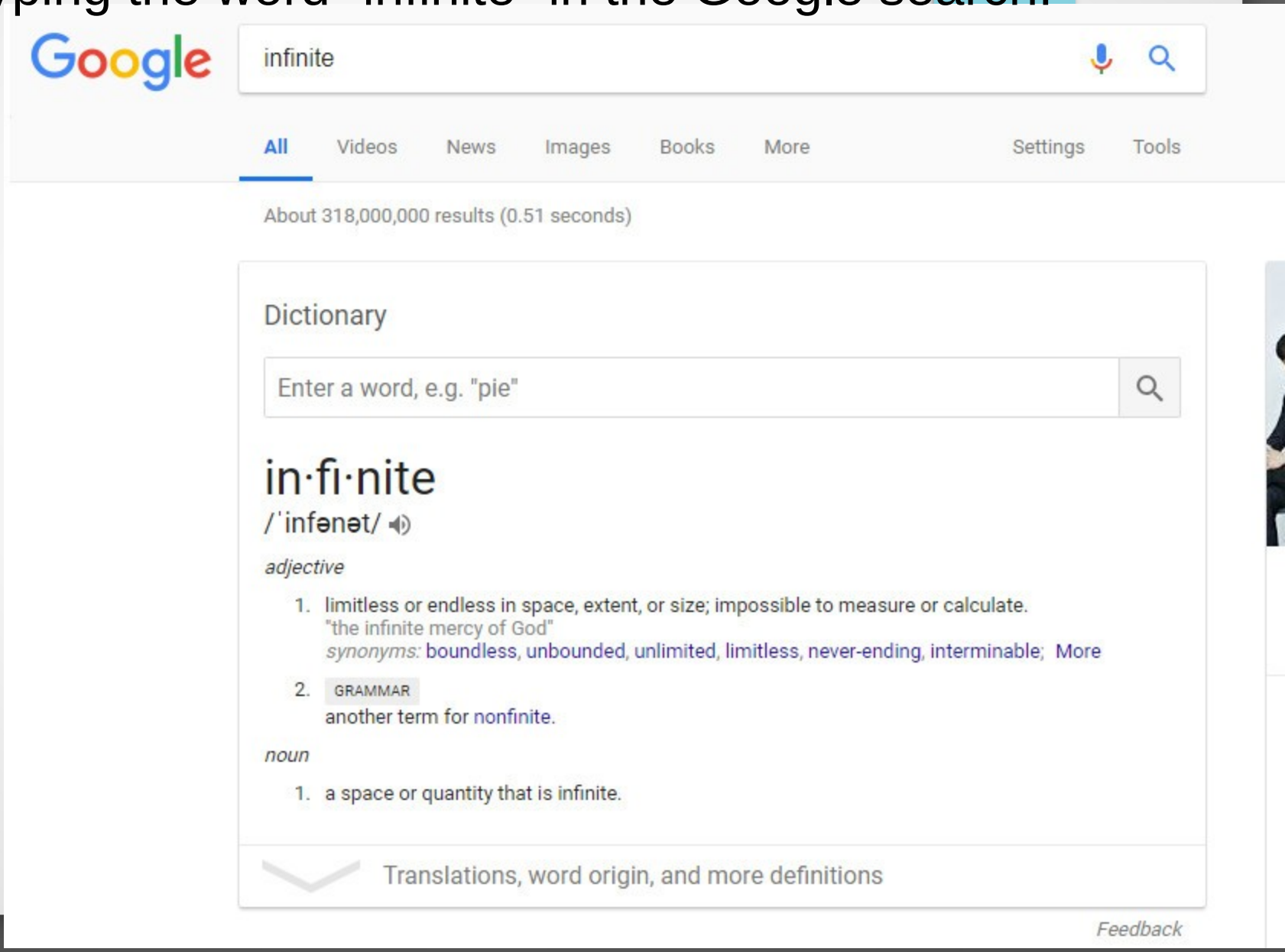
Operation	Description
`len(set)`	return the number of the elements in the set
`set1.update(set2)`	add the elements from the `set2` to the `set1`
`set.add(value)`	add the element `value` to the `set`
`set.remove(value)`	remove value from the `set`. Raises KeyError if value is not found.
`set.pop()`	remove a random element from the `set`
`set.clear()`	remove all the elements from the `set`

Set basics: in-class work

Do the items 6-7 from the in-class work handout

Dictionary basics

Consider typing the word “infinite” in the Google search:



Dictionary basics

A *dictionary* is a Python *container* used to describe associative relationships.

A *dictionary* is represented by the **`dict`** object type.

A *dictionary associates* (or "*maps*") *keys* with *values*.

A *key* is a term that can be located in a *dictionary*, such as the word "infinite" in the Google search.

A *value* describes some data associated with a *key*, such as a definition.

A *key* can be any immutable type, such as a number, string, or tuple; a *value* can be any type.

Dictionary basics

A **`dict`** object is created using curly braces `{}` to surround the `key:value` pairs that comprise the dictionary contents.

Example:

```
myDict = {
   “street address”: ”2155 University Avenue”,
   “city”: “Bronx”,
   “state” : ”New York”,
   “zip code” : 10453,
   “phone” : “(718) 289-5100”,
   “admissions” : “(718) 289-5895”}
```

Dictionary basics

Dictionaries are typically used in place of lists when an *associative relationship* exists.

Example: If a program contains a collection of anonymous student test scores, those scores should be stored in a list. However, if each score is associated with a student name, a dictionary could be used to associate student names to their score.

Dictionary basics: in-class work

I have 5 students: Cute Princess, Fairy Queen, Evil Don, Fussy Cat, and Lazy Daisy.

I also have the record of 4 of their test scores.

Let's create a dictionary `students`, where student's IDs will serve as *`keys`*, and the *`value`* will be a *`list`* with five elements/members: student's name, and 4 test scores.

Name	ID	Test 1	Test 2	Test 3	Test 4
Cute Princess	846563	89	67	98	100
Fairy Queen	736542	76	56	83	99
Evil Don	287563	52	81	79	27
Fussy Cat	294512	27	38	100	75
Lazy Daisy	975321	88	99	66	77

Download and save file Dict1.py from our web-site

Dictionary basics: in-class work

`students` dictionary we got:

keys
846563
736542
287563
294512
975321

0	1	2	3	4
“Cute Princess”	89	67	98	100
“Fairy Queen”	76	56	83	99
“Evil Don”	52	81	79	27
“Fussy Cat”	27	38	100	75
“Lazy Daisy”	88	99	66	77

Dictionary basics: in-class work

Now, let's print some information: put the following lines into *Dict1.py:*

```
print(students[975321])
print(students[846563])
```

See what happens!

keys	0	1	2	3	4
846563	"Cute Princess"	89	67	98	100
736542	"Fairy Queen"	76	56	83	99
287563	"Evil Don"	52	81	79	27
294512	"Fussy Cat"	27	38	100	75
975321	"Lazy Daisy"	88	99	66	77

Dictionary basics: in-class work

Let's now calculate Lazy Daisy's average test score: add the following lines of code into *Dict1.py*

```
s = students[975321]
averageTestScore = (s[1]+s[2]+s[3]+s[4])/4
print("Lazy Daisy average test score is",
   averageTestScore)
```

keys	0	1	2	3	4
846563	"Cute Princess"	89	67	98	100
736542	"Fairy Queen"	76	56	83	99
287563	"Evil Don"	52	81	79	27
294512	"Fussy Cat"	27	38	100	75
975321	"Lazy Daisy"	88	99	66	77

s

s[0] s[1] s[2] s[3] s[4]

Dictionary basics: in-class work

Now, let's add one more record and display the dictionary:

Name	ID	Test 1	Test 2	Test 3	Test 4
"Glad Lad"	625342	98	76	48	80

By adding the following line in *Dict1.py*:

```
students[625342] = ["Glad Lad",98,76,48,80]
print(students)
```

keys	0	1	2	3	4
846563	"Cute Princess"	89	67	98	100
736542	"Fairy Queen"	76	56	83	99
287563	"Evil Don"	52	81	79	27
294512	"Fussy Cat"	27	38	100	75
975321	"Lazy Daisy"	88	99	66	77
625342	"Glad Lad"	98	76	48	80

Dictionary basics: in-class work

Now, let's delete the record about Fussy Cat from the dictionary
By adding the following line in *Dict1.py*:

```
del students[294512]
print(students)
```

keys	0	1	2	3	4
846563	"Cute Princess"	89	67	98	100
736542	"Fairy Queen"	76	56	83	99
287563	"Evil Don"	52	81	79	27
~~294512~~	~~"Fussy Cat"~~	~~27~~	~~38~~	~~100~~	~~75~~
975321	"Lazy Daisy"	88	99	66	77
625342	"Glad Lad"	98	76	48	80

Dictionary basics: in-class work

Do the items 9-10 in the in-class handout