Composite Types in Python

Lesson Objectives

At the end of this lesson, you should be able to:

- Discuss the concept of composite types
- Explain the importance of composite types
- Use composite types in Python to solve problems

Topic Outline

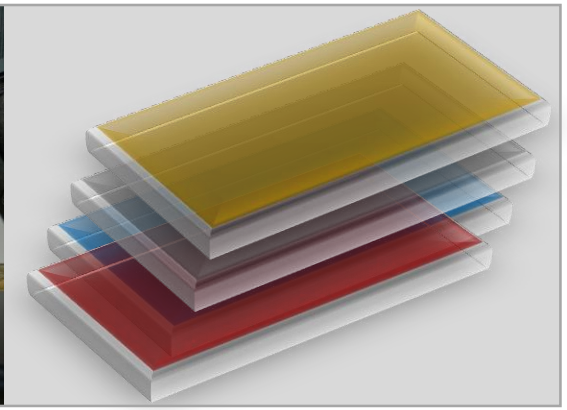
What are Composite Types/ Data Structure?

Why are Composite Types/ Data Structure Important?

Three Common Data Structures in Python (and Their Operations):

- **List**
- **Tuples**
- **Dictionaries**

What is a Composite Type?

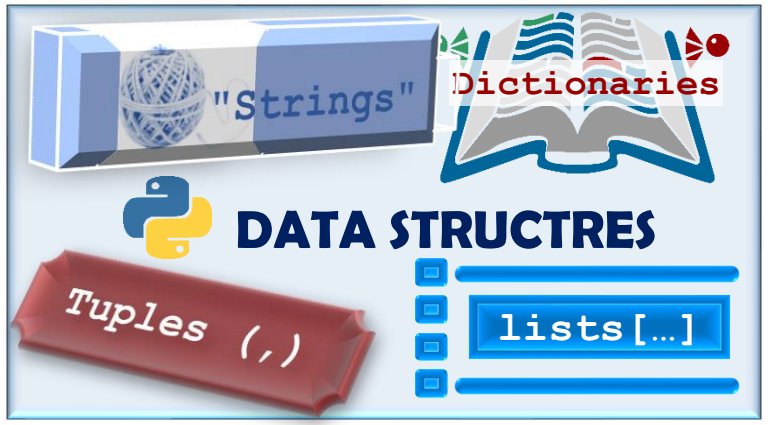


- A **data type**, which is constructed (composed) **using primitive** and **other composite types**.

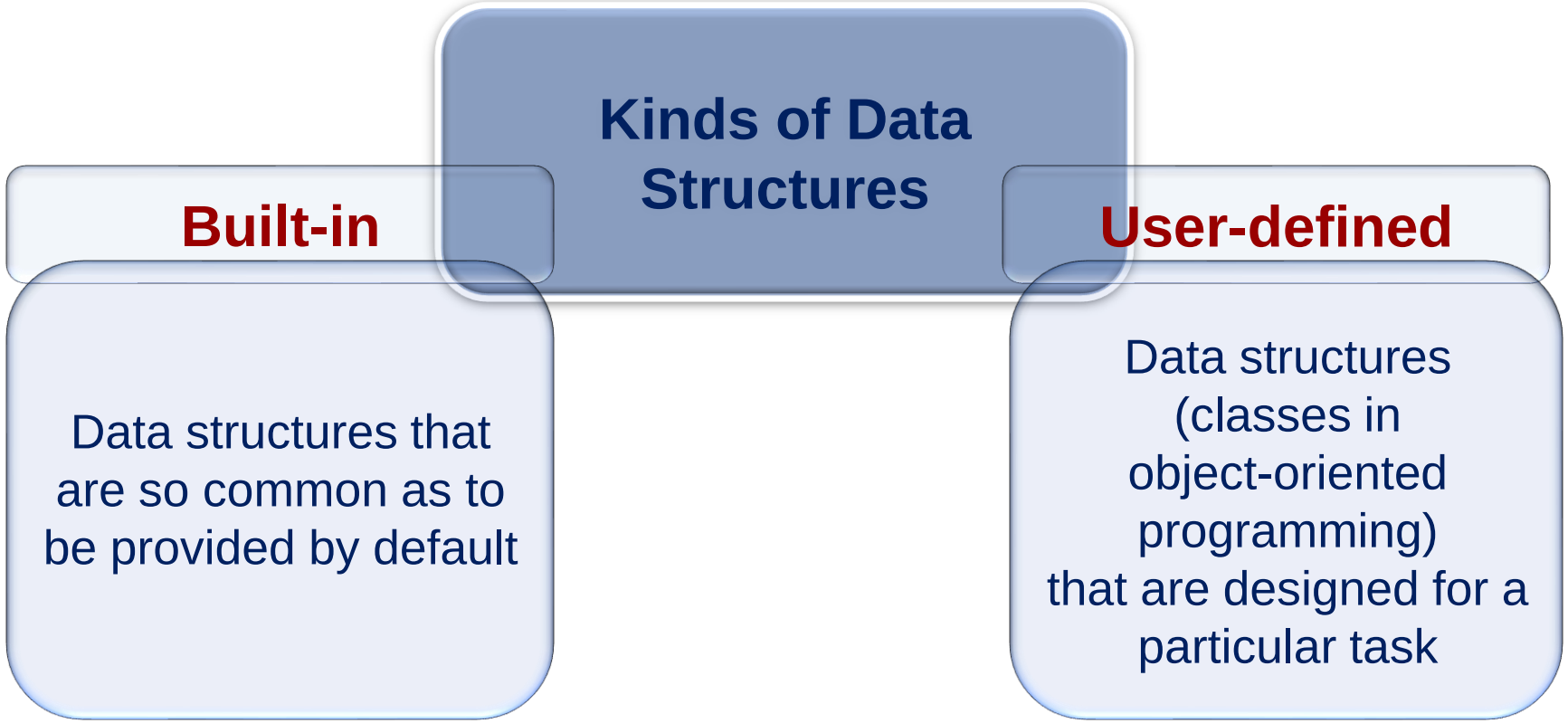
- A **new data type** made from existing ones.

Data Structures

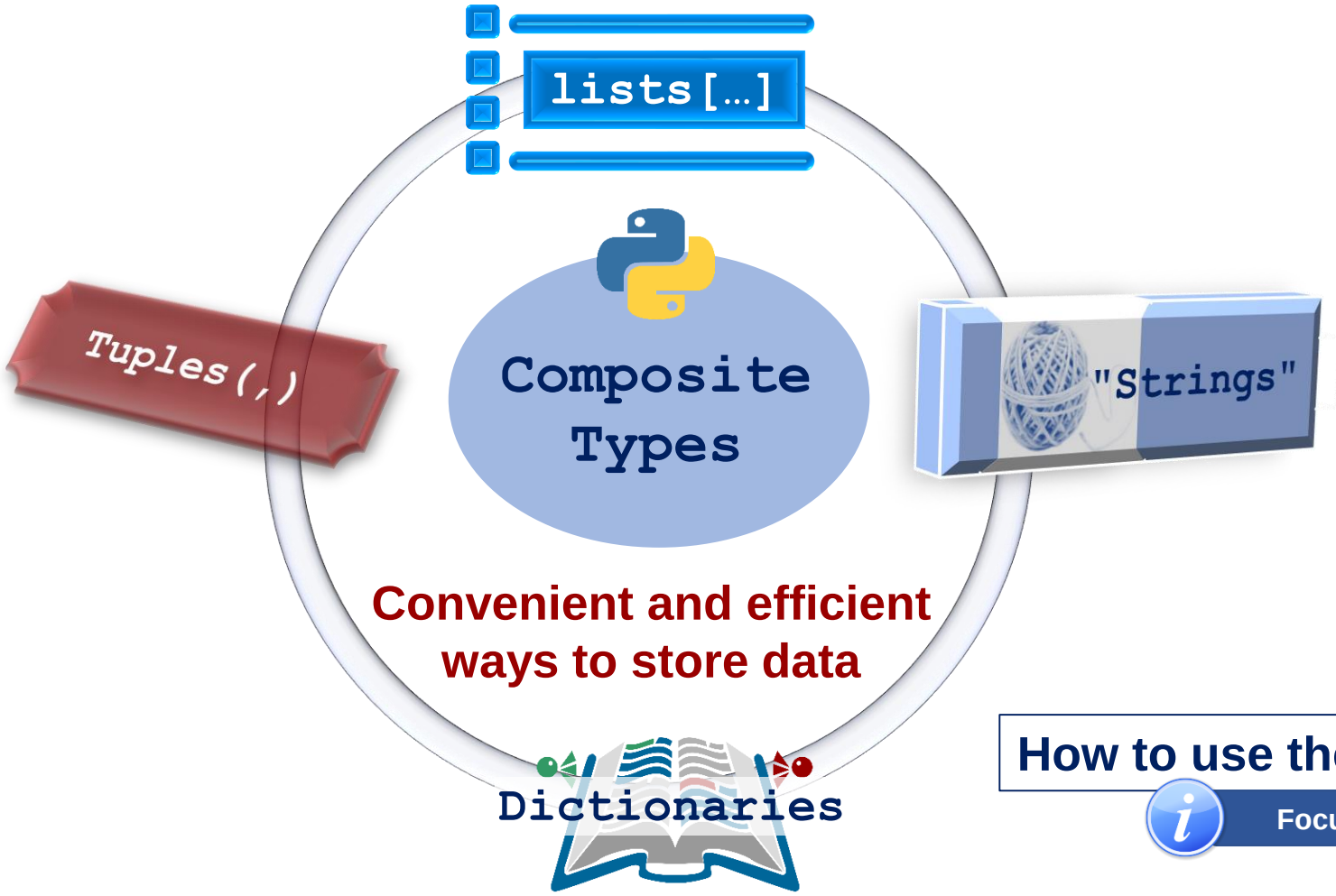
- **Particular ways of storing data** to make some operations easier or more efficient
  - They are tuned for certain tasks, and they are often associated with algorithms
- Different data structures have different characteristics
  - One suited to solving a certain problem may not be suited for another problem



Data Structures (Cont'd)



Composite Types in Python



How to use them?

Focus of this lesson


lists[…]

Python Lists

Python List is an **ordered sequence of items**.

Recall

We have already covered a type of sequence: **Strings**

- A string is a sequence of characters.

Creating a List

- As with all data structures, lists have a **constructor**.
- **Constructors** have the same name as the data structures.

```
l = list()
```
→ Creates an empty list

```
l = list(arg)
```
→ Takes an iterable data structure as an argument and add each item of **`arg`** to the constructed list **`l`**

- **Shortcut**: use of **square brackets `[]`** to indicate explicit items. → `l = […]`

Creating a List: Example

```
aList = list('abc')
  aList ⇒['a','b','c']

newList = [1, 3.14159, 'a', True]
```

Lists: Similarities with Strings

- **concatenate**: `+` (only for lists – not string + list)
- **repeat**: `*`
- **indexing**: the `[ ]` operator), e.g., `lst[3]` → 4th item in the list
- **slicing**: `[:]`
- **membership**: the `in` operator
- **length**: the `len()` function

Lists: Differences with Strings

- Lists can contain **a mixture of python objects (types)**; strings can **only hold characters**.

  E.g. `l = [1, 'bill', 1.2345, True]`

- Lists are **mutable**; their values can be changed, while strings are **immutable**.

- Lists are designated with `[ ]`, with elements separated by commas; strings use `""`.

List Structure

```
myList = [1, 'a', 3.14159, True]
```

myList	1	'a'	3.14159	True
Index Forward	0	1	2	3
Index Backward	–4	–3	–2	–1

`myList[1]` → **'a'**

`myList[:3]` → **[1, 'a', 3.14159]**

[] ? Indexing on Lists

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

[] means a list and it is also used to retrieve index.

```
['a', 'b', 'c'][1]  ➡ 'b'
[0, 1, 2][0]        ➡ 0
[0][0]              ➡ 0
```

Content is important!

Index is always at the end of the expression and is preceded by something (variable, **sequence**).

Lists of Lists

```
myLst = ['a', [1, 2, 3], 'a']
```

What is the second element of the list?

```
myLst[1][0]    #apply from left to right
```

`myLst[1]` ➡ **[1, 2, 3]**

`[1, 2, 3][0]` ➡ **1**

`[1, [2, [3, 4]], 5][1][1][0]` ➡ **?**

Operators

`+`

e.g. `[1, 2, 3] + [4]` → **[1, 2, 3, 4]**

`*`

e.g. `[1, 2, 3] * 2` → **[1, 2, 3, 1, 2, 3]**

`in`

e.g. `1 in [1, 2, 3]` → **True**

List Functions

`len(lst)` Number of elements in list (top level)

e.g. `len([1, [1, 2], 3])`  **3**

`min(lst)` Minimum element in the list

`max(lst)` Maximum element in the list

`sum(lst)` Sum of the elements, numeric only

Iterate on the List


```
for element in [1, [1, 2], 'a', True]:
    print(element)
```

What do you think is the print output?

Answer:

1
[1, 2]
‘a’
True

Mutable vs. Immutable

Immutable (Strings): Examples

```
myStr = 'abc'
myStr[0] = 'z' #not possible
newStr= myStr.replace('a', 'z') #make a new string
```

Lists as Mutable

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

The object's contents can be changed.


```
myLst = [1, 2, 3]
myLst[0] = 127
print(myLst)
```

What do you think is the output?

Answer: **[127, 2, 3]**

List Methods

A list is mutable and can be changed:

```
myList[0] = 'a'      #index assignment
myList.append(e)     // e: element to append
myList.extend(L)     // L: a list
myList.pop(i)        // i: index (default: -1)
myList.insert(i,e)
myList.remove(e)
myList.sort()
myList.reverse()
```

List Methods: Example

```
myList = [1,3]                  [1, 3]
myList[0] = 'a'                 ['a', 3]
myList.append(2)                ['a', 3, 2]

lst = [6,5]
myList.extend(lst)              ['a', 3, 2, 6, 5]

myList.extend(5)                ERROR!

element = myList.pop()          ['a', 3, 2, 6]
print(element)                  5

myList.append([8,9])            ['a', 3, 2, 6, [8,9]]
```

List Methods: Example (Cont'd)

```
                                ['a', 3, 2, 6]
myList.insert(0, 'b')           ['b', 'a', 3, 2, 6]
myList.insert(-1, 'b')          ['b', 'a', 3, 2, 'b', 6]
myList.insert(10, 'c')          ['b', 'a', 3, 2, 'b', 6, 'c']
myList.remove('b')              ['a', 3, 2, 'b', 6, 'c']

myList.sort()                   TypeError!!

myList.remove('b')              ['a', 3, 2, 6, 'c']
myList.remove('a')              [3, 2, 6, 'c']
myList.remove('c')              [3, 2, 6]

myList.remove('d')              ValueError!!

myList.sort()                   [2, 3, 6]

myList.reverse()                [6, 3, 2]
```

Return Values

- When compared to string methods, most of these list methods do not return a value.
- This is because lists are mutable so the methods modify the list directly; there is no need to return a new list.

Remember the python standard is your friend!

Warning about Results

```
myLst = [4, 7, 1, 2]
myLst = myLst.sort()
myLst ⇒ None      #what happened?
```

What is the return value of `myLst.sort()`?

Be careful of what you are assigning!

WARNING

String Method: `split()`

- The string method **`split()`** generates a sequence of characters by splitting the string at certain split-characters.
  - Default split-character: **white space**.
- The string method, **`split()`**, returns a list.

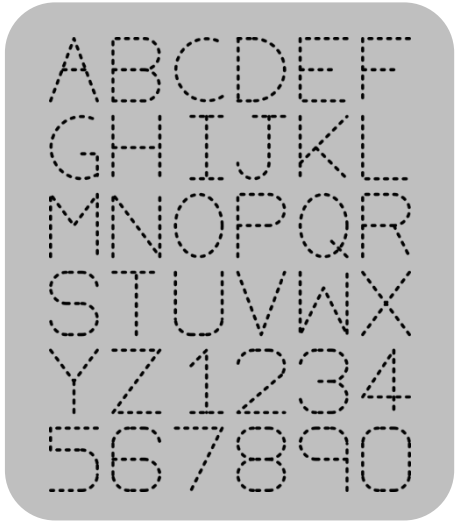
```
splitLst = 'this is a test'.split()
print(splitLst)
```

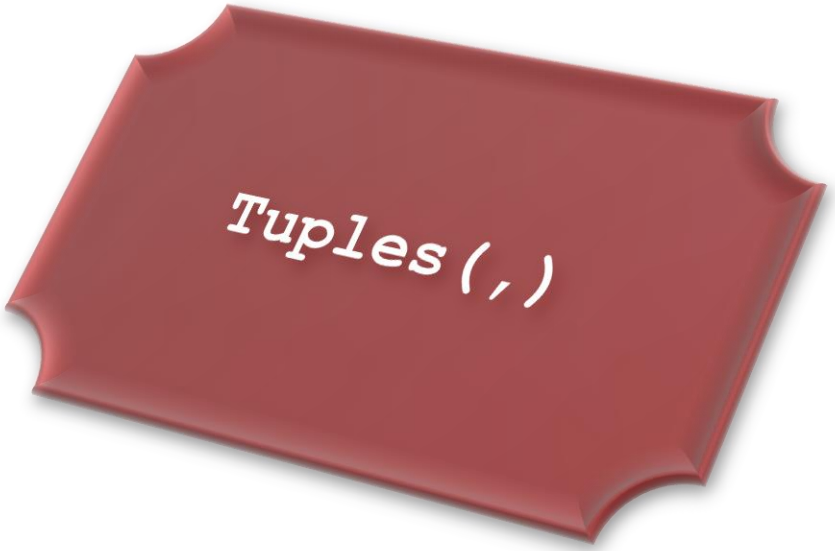
['this', 'is', 'a', 'test']

Sorting

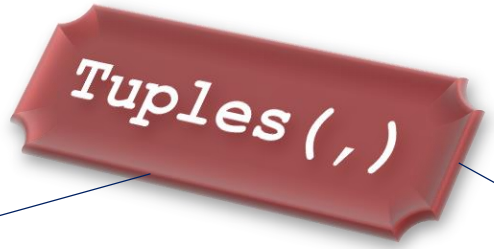
- Only lists have a **built-in sorting method**.
- Thus, data could be converted to a list if it needs sorting.

```
myLst = list('xyzabc') #iterable to constructor
myLst  ➡ ['x', 'y', 'z', 'a', 'b', 'c']
myLst.sort()  ➡ ['a', 'b', 'c', 'x', 'y', 'z']
# convert back to a string
sortStr = ''.join(myLst)  ➡ 'abcxyz'
```




Tuples(,)

Tuples



Tuples are **immutable** lists.

Why Immutable Lists?

- Provides a data structure with some integrity and some permanency
- To avoid accidentally changing one

They are designated with **(,)**.

Example:

```
myTuple = (1, 'a', 3.14, True)
```

Lists vs. Tuples

Everything that works for a list works for a tuple **except** methods that modify the tuple.

What works?

- **indexing**
- **slicing**
- `len()`
- `print()`

What doesn't work?

Mutable methods

- `append()`
- `extend()`
- `remove(), etc.`

Commas Create Tuples

For tuples:

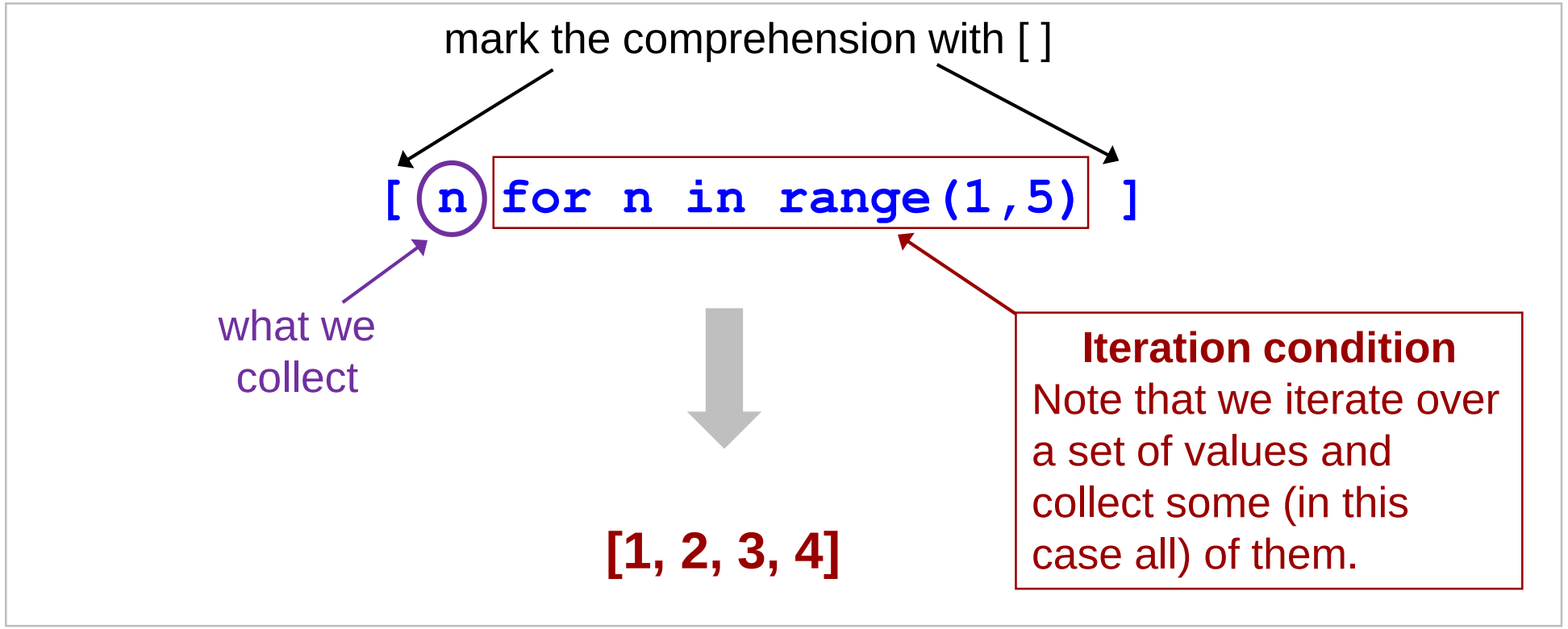
- **comma** can be thought of as the **operator** that makes a tuple
- while the **round bracket ( )** simply acts as a **grouping**

```
myTuple = 1,2        # creates (1,2)
myTuple = (1,)       # creates (1)
myTuple = (1)        # creates 1 not (1)
myTuple = 1,         # creates (1)
```

List Comprehension

Constructing List

List comprehension: syntactic structure for concise construction of lists



Other Examples

```
[n**2 for n in range(1,6)]
```

[1, 4, 9, 16, 25]

```
[x + y for x in range(1,5) for y in range (1,4)]
```

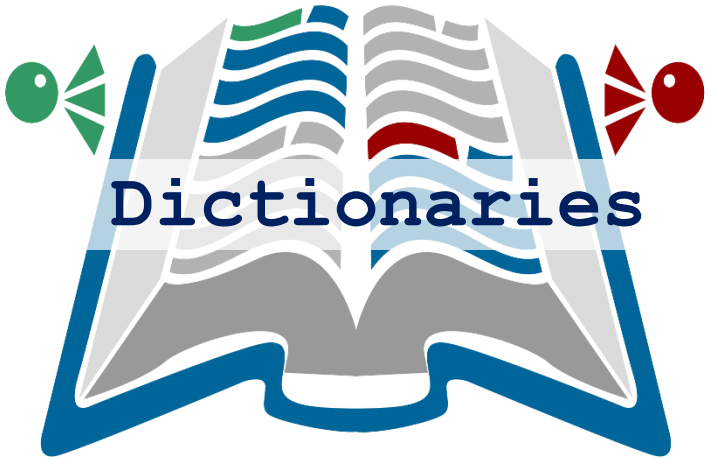
?

It is as if we had done the following:

```
myList = [ ]
for x in range (1,5):
  for y in range (1,4):
    myList.append(x+y)
```

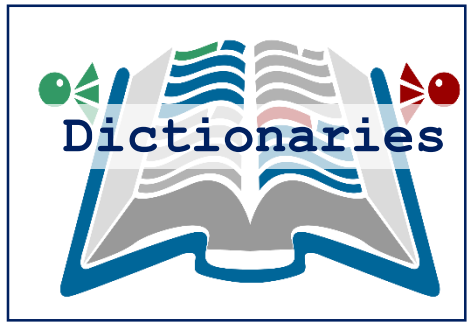
```
[c for c in "Hi There Mom" if c.isupper()]
```

['H', 'T', 'M']


Dictionaries

What is Dictionary?

- In data structure terms, a **dictionary** is better termed as an **associative array**, or **associative list**, or a **map**.
- You can think of it as a **list of pairs**.
  - The **key**, which is the **first element** of the pair, is used to retrieve the **second element**, which is the **value**.
- Thus, we map a key to a value.



Key:Value

- The **key** acts as a “lookup” to find the associated value.
- Just like a dictionary, you look up a word by its spelling to find the associated definition.
- A dictionary can be searched to locate the value associated with a key.

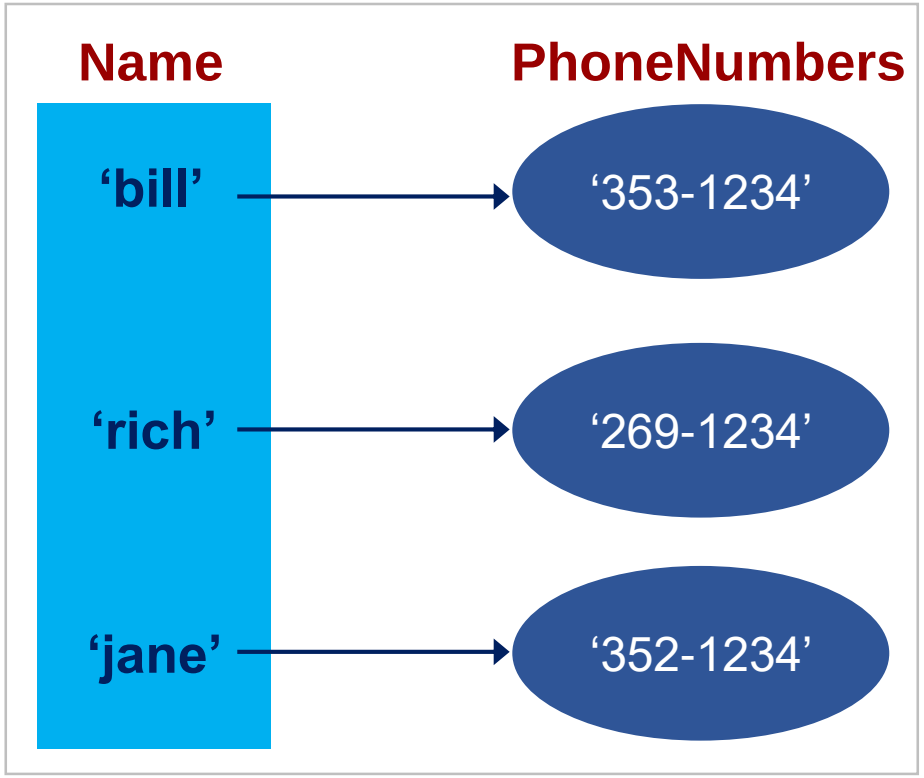
Python Dictionary

{ } marker: used to create a dictionary

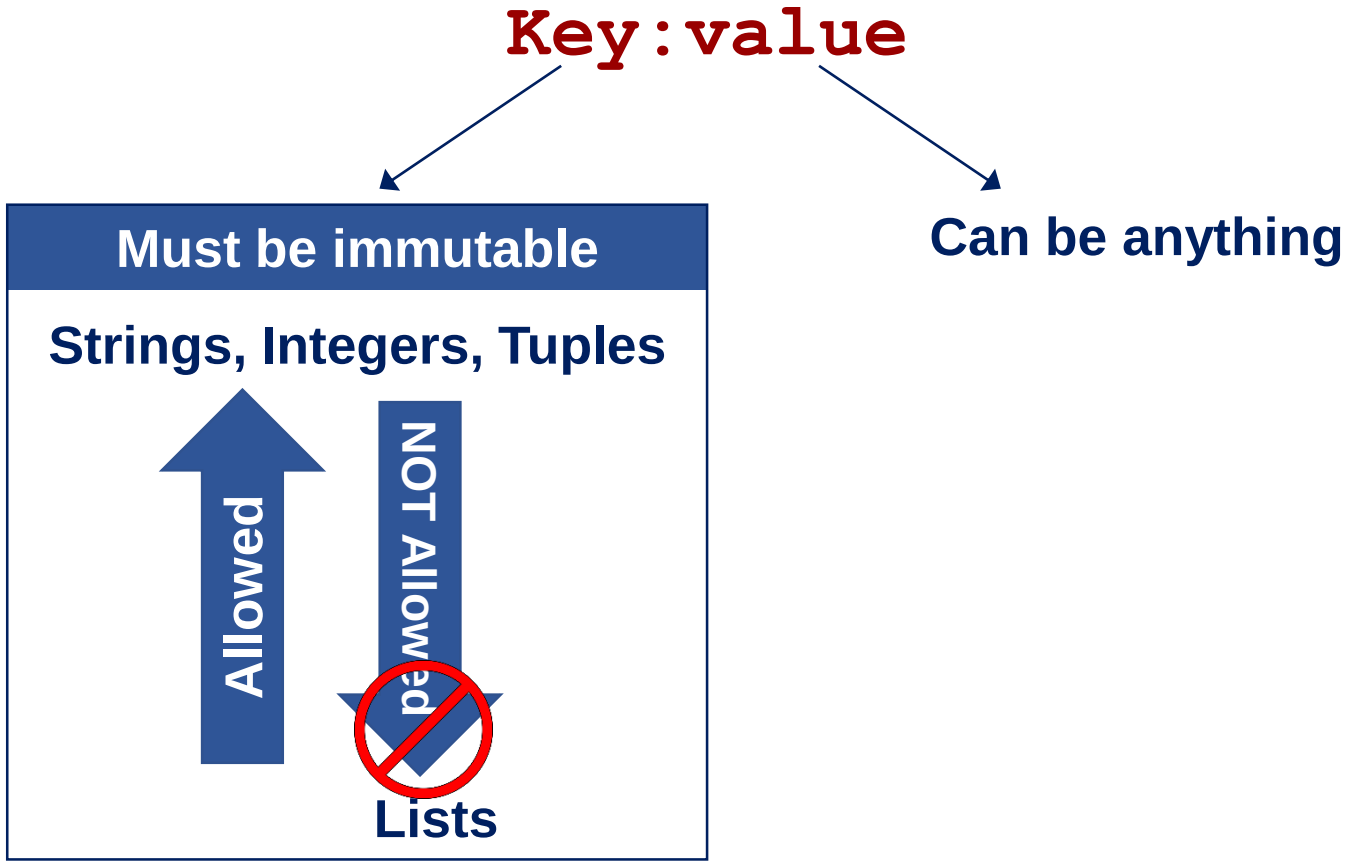
: marker: used to create **key:value** pairs

```
contacts = {'bill': '353-1234',
            'rich': '269-1234',
            'jane': '352-1234'}

print(contacts)  {'jane': '352-1234',
                  'bill': '353-1234',
                  'rich': '269-1234'}
```



What are Keys and Values?



Collection vs. Sequence

Dictionaries are **collections** but they are **not sequences** like lists, strings, or tuples.

- There is **no order** to the elements of a dictionary.
- In fact, the order (for example, when printed) might change as elements are added or deleted.

So, how do you access dictionary elements?

Access to Dictionary

Access requires **[ ]** and the **key** is the index.

`myDict = {}` → **an empty dictionary**

`myDict['bill'] = 25` → **add the pair `'bill':25`**

`print(myDict['bill'])` → **print 25**

`del myDic['bill']` → **remove the pair `'bill':25`**

Dictionaries are Mutable

Like lists, dictionaries are mutable.

- You can change the object via various operations, such as index assignment.

```
myDict = {'bill':3, 'rich':10}

print(myDict['bill'])          # prints 3
myDict['bill'] = 100           # change value
print(myDict['bill'])          # prints 100

del myDict['rich']             # remove 'rich':10
del myDict['rich']             # KeyError
```

Dictionary Operations

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Like others, dictionaries respond to these:

`len(myDict)` → **number** of **key:value pairs** in the dictionary

`element in myDict` → boolean; is **element** a **key** in the dictionary?

`for key in myDict` → iterate through the **keys** of a dictionary

Other Methods and Operations

`myDict.items()` → return all the **key:value** pairs

`myDict.keys()` → return all the keys

`myDict.values()` → return all the values

`myDict.clear()` → empty the dictionary

`myDict.update(yourDict)` → for each key in `yourDict`, update `myDict` with that **key:value** pair

Iterating on a Dictionary

```
for key in myDict:
    print(key)
```
 prints all the keys

```
for key,value in myDict.items():
    print(key, value)
```
 prints all the key:value pairs

```
for value in myDict.values():
    print(value)
```
 prints all the values

Summary

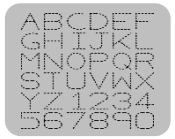
In this lesson, we have learnt:

- The concept of composite types
- Built-in composite types in the Python programming language:
  - List
  - Tuple
  - Dictionary

References for Images

No.	Slide No.	Image	Reference
1	5		Gabovitch, I. (2014). AV Out In HDMI In Jack Plug Red White Yellow Audio and Video Mixer Backside [Online Image]. Retrieved May 17, 2018 from https://www.flickr.com/photos/qubodup/12248078123.
2	6		Python Logo [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/language-logo-python-2024210/.
3	6, 8, 37, 38		By Ephemeron - Own work, based on File:Dynamic Dictionary Logo.png, CC BY-SA 3.0, retrieved May 18, 2018 from https://commons.wikimedia.org/w/index.php?curid=7361291.
4	6, 8		String [Online Image]. Retrieved April 24, 2018 https://pixabay.com/en/string-twine-ball-twined-isolated-314346/.
5	10		Search [Online Image]. Retrieved April 18, 2018 from https://pixabay.com/en/database-search-database-search-icon-2797375/.

References for Images

No.	Slide No.	Image	Reference
6	17, 20, 22, 42		Question problem [Online Image]. Retrieved April 18, 2018 from https://pixabay.com/en/question-problem-think-thinking-622164/.
7	21		Survey icon [Online Image]. Retrieved April 18, 2018 from https://pixabay.com/en/survey-icon-survey-icon-2316468/.
8	26		Smiley 11 [Online Image]. Retrieved April 18, 2018 from http://www.publicdomainfiles.com/show_file.php?id=13545100814144.
9	27		By Unknown - From the Open Clip Art Gallery - http://openclipart.org/, CC0, retrieved May 16, 2018 from https://commons.wikimedia.org/w/index.php?curid=1849852.
10	29		Alphabet [Online Image]. Retrieved May 17, 2018 from https://pixabay.com/en/alphabet-letters-numbers-digits-40515/.