

# Week 5 Course Review

## Composite Data Types in Python



**Instructor: Asst. Prof. LIN Shang-Wei**

**Email: [shang-wei.lin@ntu.edu.sg](mailto:shang-wei.lin@ntu.edu.sg)**

Abstraction in Data: **Data Structures**  
(Strings, Lists, Tuples, Dictionaries, etc.)

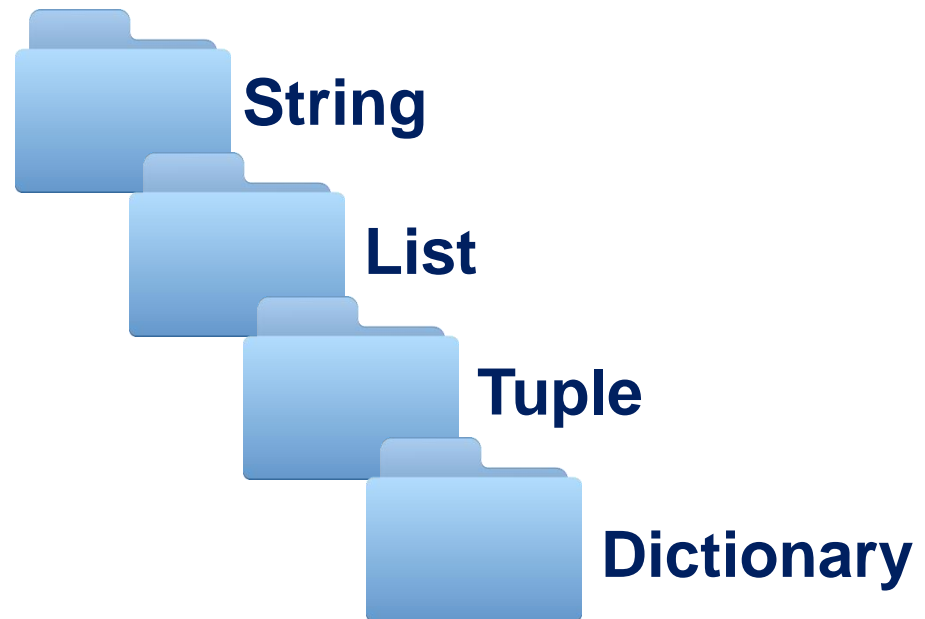


**Programs = Algorithms + Data Structures**

*(1976, Niklaus Wirth)*

Abstraction in Algorithms:  
**Functions**

# Review Outline



# Python Strings



```
myStr = "Hello World"
```

Characters	H	e	l	l	o		W	o	r	l	d
Indices	0	1	2	3	4	5	6	7	8	9	10
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

We can use `[]` to access particular characters in a string.

```
print(myStr[10])
```

```
print(myStr[-1])
```

# will print 'd'

```
print(myStr[11])
```

❗ Error

```
myStr = "Hello World"
```

**Syntax:** [ **start** : **finish** : **step** ]



specifies the step size to jump along the sequence

**Default Value:**

- **start**: beginning
- **finish**: end
- **step**: 1

```
newStr = myStr[:]           # To copy a string
print(newStr)               # Will print 'Hello World'

print(myStr[::-1])          # To reverse a string      # Will print 'dlroW olleH'

print(myStr[0:11:-1])        # Will print ''
```

# Q1: What is the output of the following Python program?

```
myStr = "Hello World"  
print(myStr[2:-4:2])
```



- A. 'el '
- B. 'el o'
- ✓ C. 'loW'
- D. 'loWr'
- E. 'lo W'

Characters  
Indices

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

# Basic Operations

```
opStr = "Basic"
```

Length of a string: `len()`

e.g. `len(opStr)` → **5**

Concatenate strings: `+`

e.g. `opStr + " operations"` → **'Basic operations'**

Repeat String: `*`

e.g. `opStr * 3` → **'BasicBasicBasic'**



## Q2: What is the output of the following Python program?

```
str1 = "ABC"  
str2 = "a b"  
newStr = str1 * len(str2)  
print(newStr)
```



- A. 'a b'
- B. 'a ba b'
- C. 'a ba ba b'
- D. 'ABCABC'
- E. 'ABC'
- ✓ F. 'ABCABCABC'



## Is one string contained in another?

- Operator: **in**
- **a in b**: True if string **a** is contained in string **b**

```
myStr = "abcdefg"
```

```
'c' in myStr    → true
```

```
'cde' in myStr  → true
```

```
'cef' in myStr  → false
```

```
myStr in myStr  → true
```

### Q3: What is the output of the following Python program?

```
str1 = "ababc"
str2 = "ab"

if str2 * len(str2) in str1:
    print("case1")

elif str2 in str1:
    print("case2")

else:
    print("case3")
```



A. 'case1'  
    'case2'


B. 'case2'

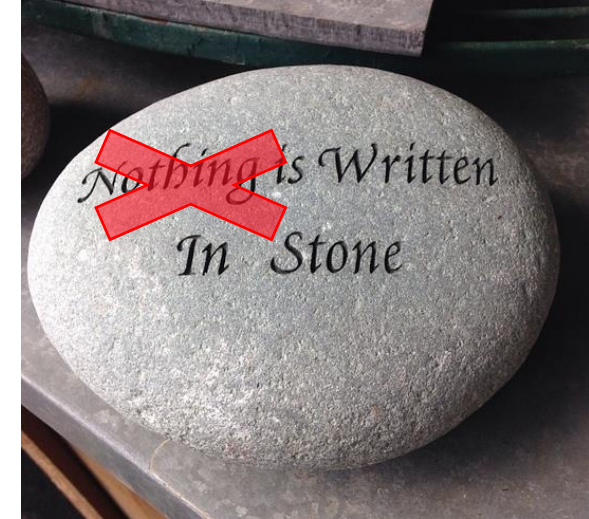
✓ C. 'case1'

D. 'case3'

E. 'case2'  
    'case3'

# Strings are Immutable

- Strings are immutable, i.e., you cannot change one once you make it.
  - `aStr = 'spam'`
  - `aStr[1] = 1` →  Error
- However, you can use it to make another string (copy it, slice it, etc.).
  - `newStr = aStr[:1] + 'l' + aStr[2:]`
  - `newStr` → `'slam'`
  - `aStr` → `'spam'`



## Q4: What is the output of the following Python program?

```
str1 = "couple"  
str2 = "tea"  
newStr = str2[:1] + str1[2:]  
  
print(newStr, str1)
```



- A. `'tuple'`
- B. `'couple'`
- C. `'tea'`
- D. `'tuple'` `'co'`
- E. `'tuple'` `'uple'`
- ✓ F. `'tuple'` `'couple'`

# String Method: `find()`

`find()` is another string method.

```
myStr = "Find in a string"  
myStr.find('d') → 3
```

- Input: a single character or a string
- Output: the **index** of the character/string (first seen from left to right)
- If the character/string is not found, **-1** is returned

## Q5: What is the output of the following Python program?

```
str1 = "couple"  
  
newStr = str1[str1.find('ou'):str1.find('l')]  
  
print(newStr)
```



- A. 'couple'
- B. 'ouple'
- C. 'coupl'
- ✓ D. 'oup'
- E. 'oupl'

## Q6: What is the output of the following Python program?

```
str1 = "couple"  
str2 = "t"  
  
newStr = str1[::-str1.find(str2)]  
  
print(newStr)
```



- A. 'couple'
- B. ''
- C. 'coupl'
- ✓ D. 'elpuoc'
- E. 'elpuo'



# Python Lists



# 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 = [...]
```

```
aList = list('abc')    # ['a', 'b', 'c']  
newList = [1, 3.14159, 'a', True]
```

# Operations on Lists

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

## Q7: What is the output of the following Python program?

```
list1 = [1, "Python", [3, 4], True]

list2 = list1[::-1] + list1[2]

print(list2)
```



- A. [1, "Python", [3, 4], True]
- B. [1, "Python", [3, 4], True, [3, 4]]
- C. [1, "Python", [3, 4], True, 3, 4]
- ✓ D. [True, [3, 4], 'Python', 1, 3, 4]
- E. [True, [3, 4], 'Python', 1, [3, 4]]
- F. Error

## Q8: What is the output of the following Python program?

```
list1 = [1, "Python", [3, 4], True]

if 3 in list1:
    list2 = list1[2] * len(list1[2])
    print(list2)

elif [3, 4] in list1:
    print(list1[2][1])

else:
    print(list1[2])
```



- A. [3, 4, 3, 4]
- B. [3, 4]
- C. [4]
- ✓ D. 4
- E. 3
- F. [3]

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()
```

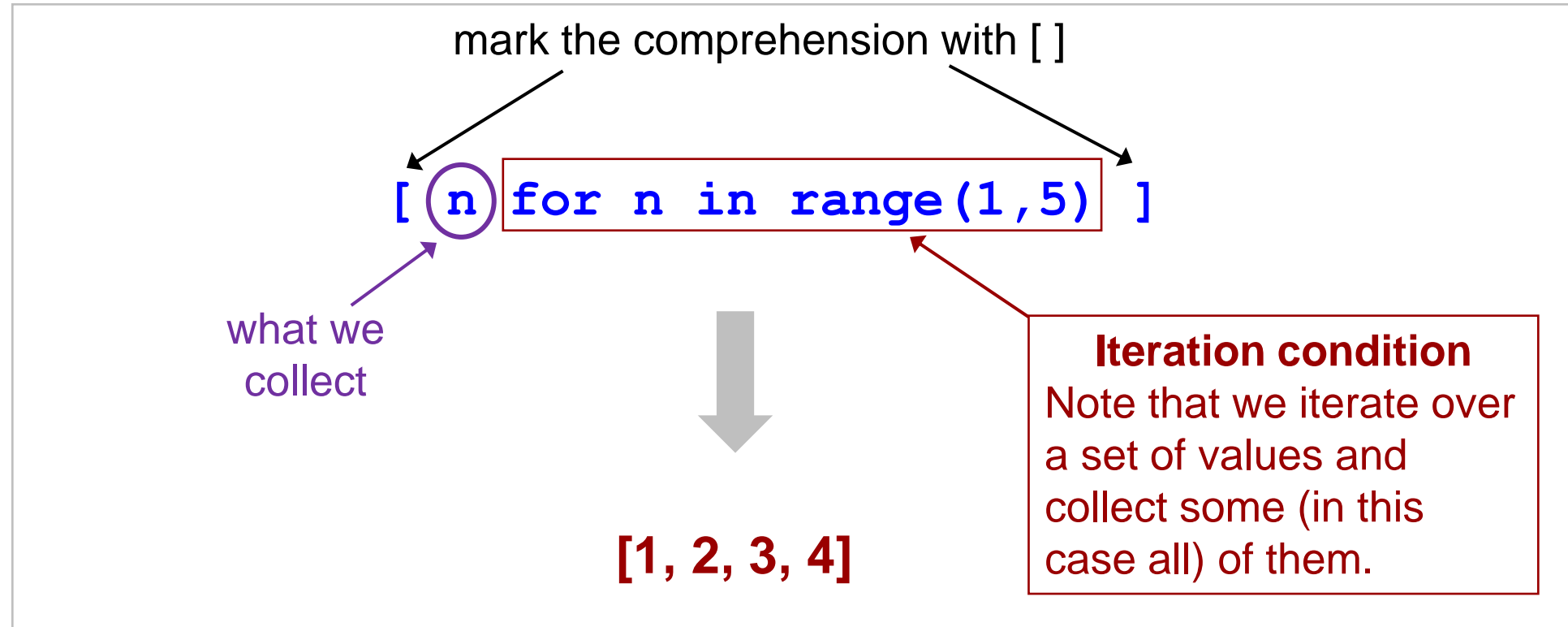
## Q9: What is the output of the following Python program?

```
list1 = ['d', 'c']  
list2 = ['a', 'b']  
  
list1.reverse()  
list2.reverse()  
  
list3 = list1.extend(list2)  
  
print(list3)
```



- A. ['c', 'd']
- B. ['b', 'a']
- C. ['d', 'c', 'a', 'b']
- D. ['a', 'b', 'c', 'd']
- E. ['c', 'd', 'b', 'a']
- ✓ F. None

**List comprehension:** syntactic structure for concise construction of lists





## Q10: What is the output of the following Python program?

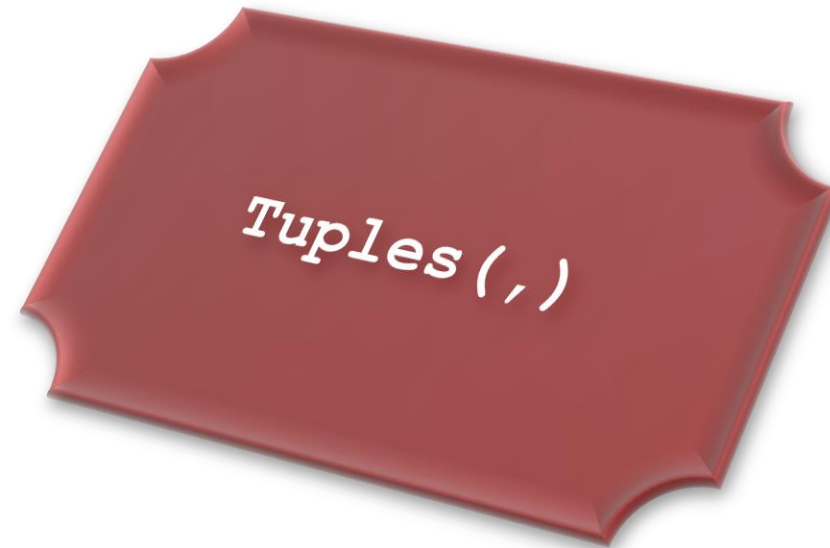
```
list1 = ['d', 'c', 'A', 3]
list2 = ['A', 'b', '3']

result = [item for item in list1 if item in list2]

print(result)
```

- A. ['3']
- B. 'A'
- C. [3]
- ✓ D. ['A']
- E. ['A', 3]
- F. []

# Python 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.

## Q11: What is the output of the following Python program?

```
myTuple = (4, 2, 3, [6, 5])  
  
myTuple[0] = 7  
  
print(myTuple)
```



- A. (4, 2, 3, [6, 5])
- B. [7, 2, 3, [6, 5]]
- C. []
- ✓ D. Error

## Q12: What is the output of the following Python program?

```
tuple1 = (3, 2, 6, ['a', 'b'])  
  
tuple2 = tuple1[::-2]  
  
print(tuple2[0][0])
```



- A. ([ 'a' , 'b' ] , 2)
- ✓ B. 'a'
- C. 'b'
- D. 3
- E. Error

## Q13: What is the output of the following Python program?

```
tuple1 = ('a', 'b', 'c')
list1 = ['A', 'B', 'b', 'D', 'a']

result = []

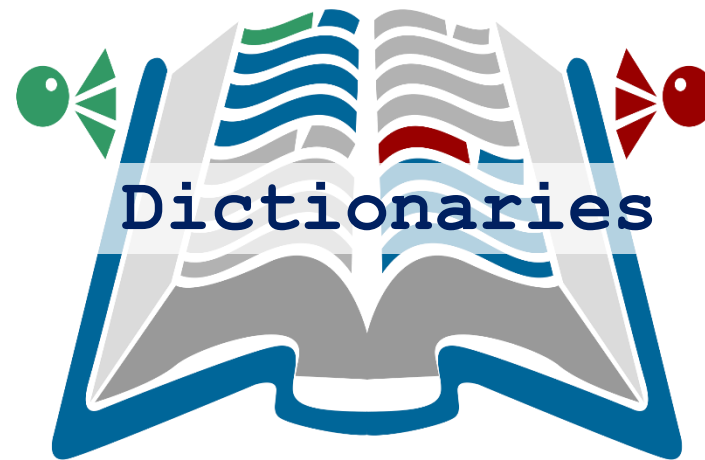
for i in range(len(tuple1)):
    if tuple1[i] in list1:
        result.append(i)

print(result)
```



- A. (0,1)
- B. ('a', 'b')
- ✓ C. [0,1]
- D. ['a', 'b']
- E. []
- F. Error

# Python Dictionary

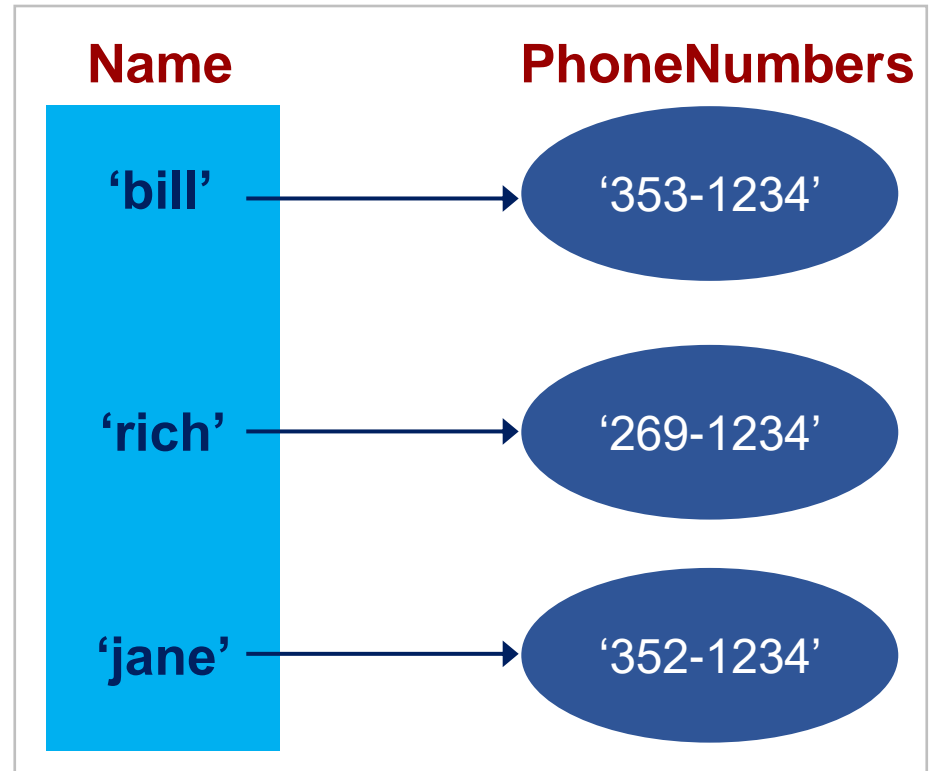




**{ }** **marker**: used to create a dictionary

**:** **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'}
```



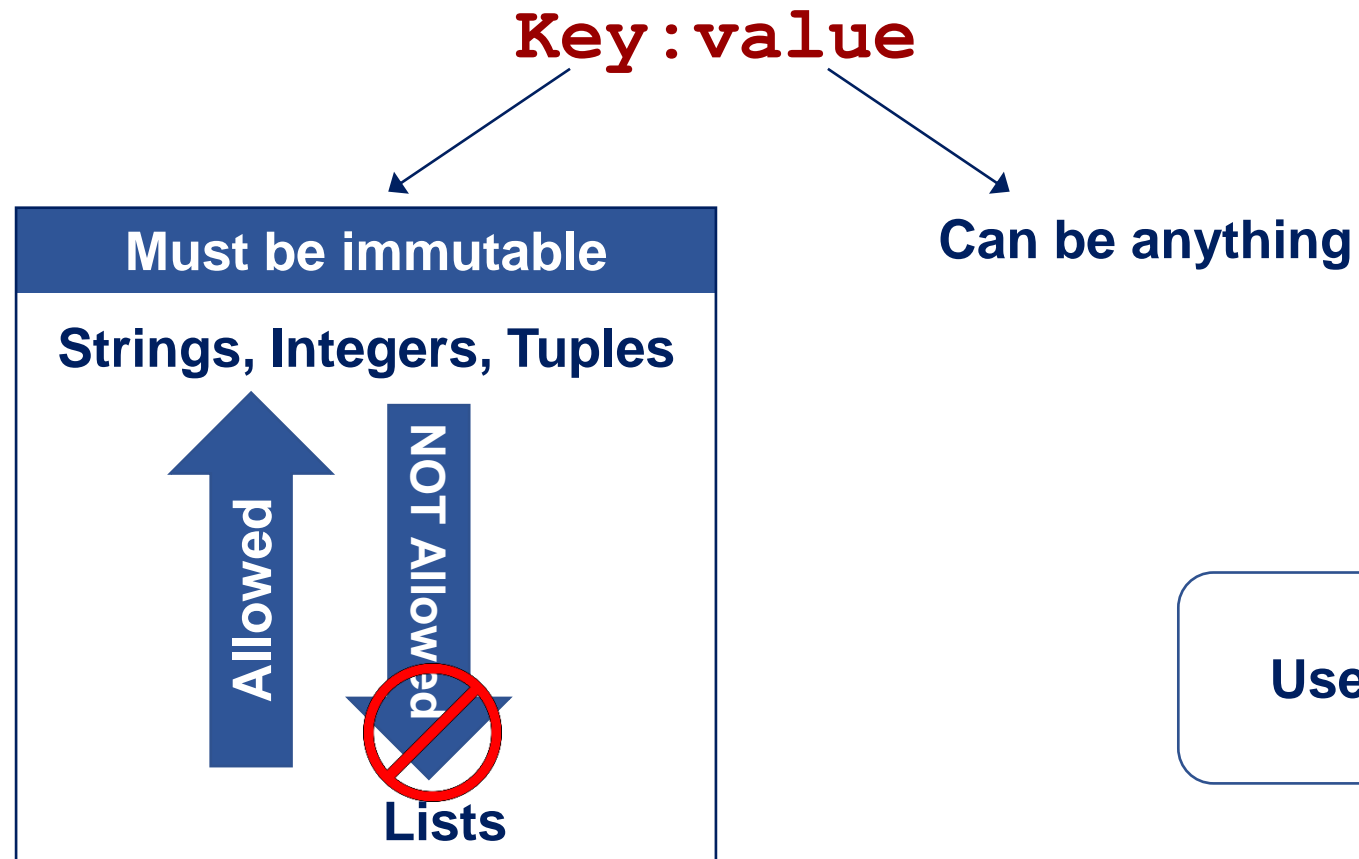
## Q14: What is the output of the following Python program?

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



- ✓ A. 3  
B. 6  
C. Error

# What are Keys and Values?



Use **Keys** to access the values


## Q15: What is the output of the following Python program?

```
contacts = {  
    ['Bill', 'Male']: '353-1234',  
    ['Rich', 'Male']: '269-1234',  
    ['Jane', 'Femal']: '352-1234'  
}  
  
print(len(contacts))
```



- A. 3
- B. 6
- ✓ C. Error

## Q16: What is the output of the following Python program?

```
tuple1 = (1, 2, [1], [1, 2])   
  
dict1 = {  
    'a': [1],  
    'b': [2],  
}  
  
for key in dict1:  
    if dict1[key] in tuple1:  
        dict1[key] = 'hit'  
  
print(dict1)
```



- A. {'a': [1], 'b': [2]}
- B. {'a': 'hit', 'b': 'hit'}
- C. {'hit': [1], 'hit': [2]}
- D. {'hit': [1], 'b': [2]}
- E. {'a': 'hit', 'b': [2]}
- F. Error

# Methods on Dictionaries

`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

## Q17: What is the output of the following Python program?

```
dict1 = {  
    'a': [1],  
    'b': [2],  
}  
  
dict2 = {  
    'c': [3]  
}  
  
dict1.update(dict2)  
  
print(dict1)
```



- A. { 'a' : [1] , 'b' : [2] }
- B. { 'c' : [3] }
- C. { 'a' : [1] , 'b' : [2] , 'c' : [1] }
- D. { 'a' : [1] , 'b' : [2] , 'c' : [3] }
- E. Error

## Q18: What is the output of the following Python program?

```
dict1 = {  
    'a': [1],  
    'b': [2],  
}  
  
dict2 = {  
    'a': [7],  
    'c': [3]  
}  
  
dict1.update(dict2)  
  
print(dict1)
```



- A. { 'a' : [1] , 'b' : [2] , 'c' : [3] }
- B. { 'a' : [7] , 'c' : [3] }
- C. { 'a' : [1] , 'b' : [2] }
- D. { 'a' : [7] , 'b' : [2] , 'c' : [3] }
- E. Error