PYTHON PROGRAMMING

2023/10/04 16:00 PM

CONTENTS

01. FUNCTION

- Function Definition
- Keyword Arguments and Default Values
- Function Calls

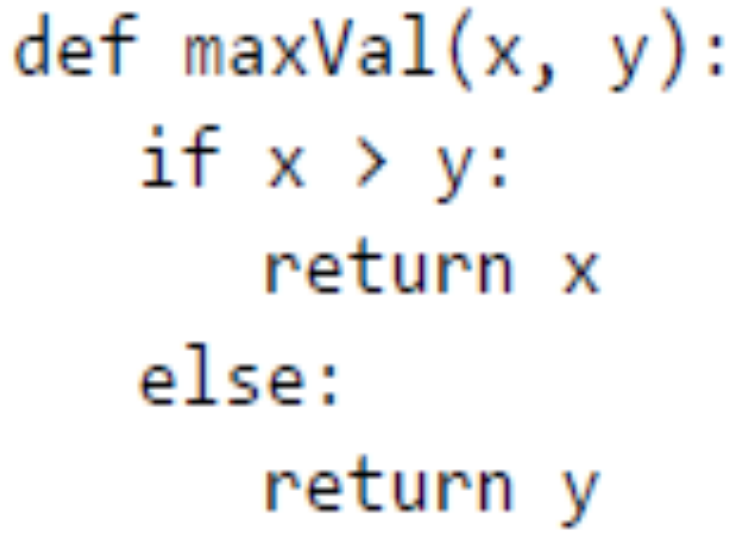
Function

```
def function_name (formal parameter(s)):
        function body
```

- `function_name`: 함수 호출 시 사용할 이름
- `formal parameter`: 함수 실행 시 사용할 인자
- `actual parameter`: 함수 호출 시 (`function call`) **실제로** 전달되는 인자의 값
- `return`: `function body` 내에서 함수 종료
  - ✓ `return` 문이 없다면 함수 호출의 결과값은 None

Function (cont.)

- Function example



```
def maxVal(x, y):
    if x > y:
        return x
    else:
        return y
```

- Function call example

```
ans_1 = maxVal(1, 9)
```

⇒ 1

```
ans_2 = maxVal(3,5) +
        maxVal(10, 3)
```

⇒ 5+10 = 15

Keyword Arguments and Default Values

1) Positional : 함수의 정의된 formal parameter 순서대로 actual parameter 값에 bind

2) Keyword arguments: formal parameter의 이름을 직접 사용하여 actual parameter값에 bind
   - commonly used in conjunction with **default parameter values**

Function Calls – Example

```
def printName(firstName, lastName, reverse):
        if reverse: print(lastName + ', ' + firstName)
        else:
           print(firstName, lastName)
```

- formal parameters
  - ✓ `firstName`: string
  - ✓ `lastName`: string
  - ✓ `reverse`: Boolean

① printName('Jess', 'McCartney')

② printName('Jess', 'McCartney', reverse = False)

③ printName('Jess', lastName = 'McCartney', reverse = False)

④ printName(lastName = 'McCartney', firstName = 'Jess', reverse = False)

⑤ printName('Jess', lastName = 'McCartney', False) (x)

Function Calls – Example (cont.)

- Default value가 정해진 Boolean은 생략 가능 ⇒ True (① == ②)
- Keyword argument 사용 시, 어떤 순서로든 인자 넣기 가능 (④)
- Keyword argument 사용 시, 하나라도 non-keyword argument 사용 시 ERROR (⑤) (③은 옳음)

02. SCOPING

Scope

- Each function defines a new name space, called a **scope**.
- When a function is called, a **stack frame** is created.
  ⇒ It keeps track of all names defined within the function
  and their current bindings.
- If a function is called from within the function body, another stack frame is created.
- When the function completes, its stack frame disappears.

Scope – Example 1

```
def f(x):
    y = 1
    x = x + y
    print(‘x = ‘, x)
    return x
```

함수 `f(x)` 정의
(name *x* used as formal parameter)

```
x = 3
y = 2
z = f(x)
print(‘z =‘, z)
print(‘x =‘, x)
print(‘y =‘, y)
```

z의 값은 `f(x)`의 return 값
위에서 `x = 3` 명시
(value of *x* used as actual parameter)
ie. **`z = f(3) = 4`**

Scope – Example 1 (cont.)

`x = 4`	`f(x)` 내에서 실행된 `print`문
`z = 4`	`print(‘z = ’, z)`
`x = 3`	`print(‘x = ’, x)`
`y = 2`	`print(‘y = ’, y)`

Scope – Example 2

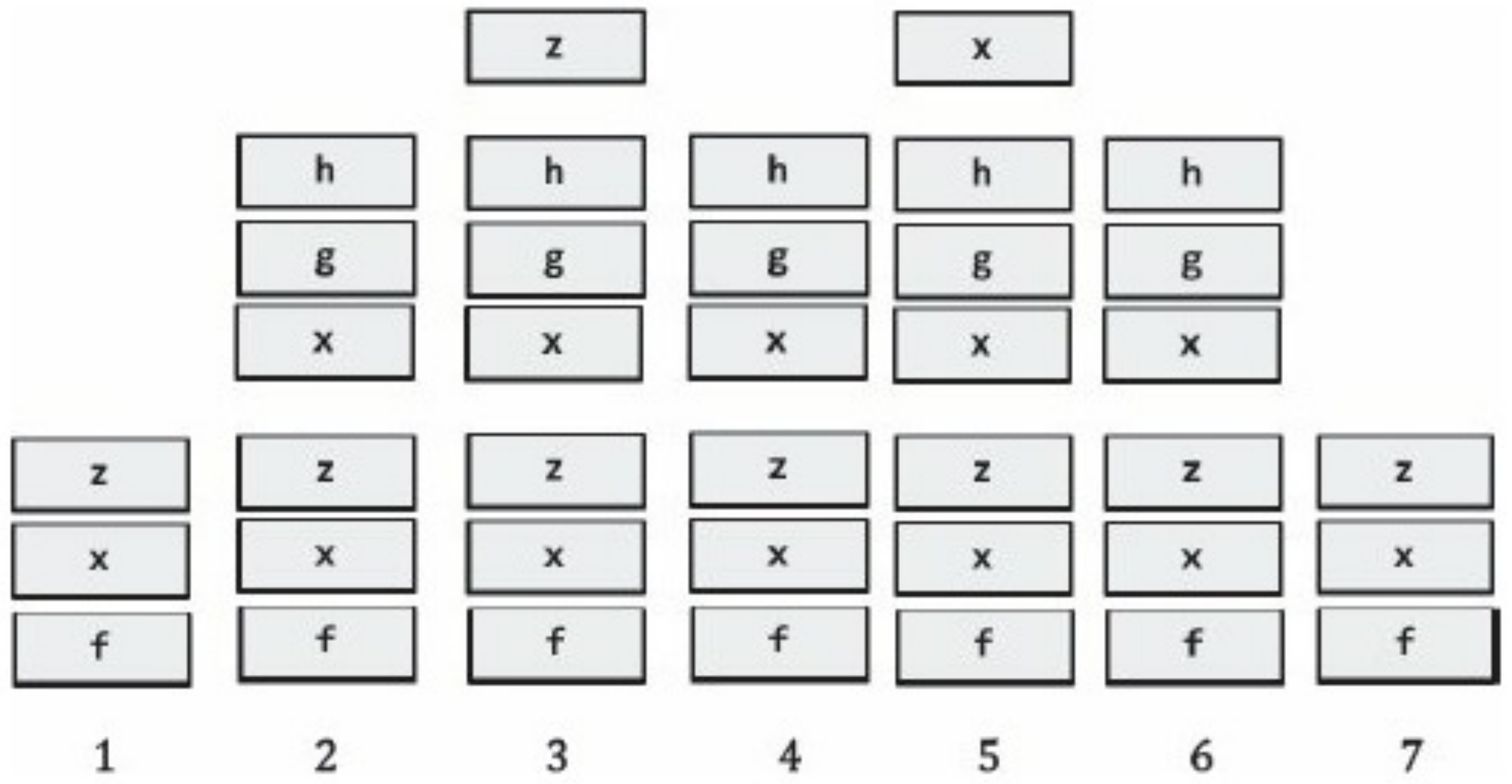
```
def f(x):
    def g( ):
        x = 'abc'
        print('x = ', x)
    def h( ):
        z = x
        print('z = ', z)
    x = x + 1
    print('x = ', x)
    h( )
    g( )
    print('x = ', x)
    return g
```

```
x = 3
z = f(x)
print('x = ', x)
print('z = ', z)
z( )
```

함수 `f(x)`:

- `f(x)` 내에 함수 `g( )`, `h( )`가 nested된 형태
- return 값은 `g`

Scope – Example 2 (cont.)



Scope – Example 2 (cont.)

```
x = 4

z = 4

x = abc

x = 4

x = 3

z = <function f.<locals>.g at 0x1092a7510>

x = abc
```

Scope – Example 3

```
def f():
    print(x)

def g():
    print(x)
    x = 1

x = 3
f( )
x = 3
g( )
```

x = 1이라는 statement를 통해 x를 g( )의 local variable로 만듦

```
3

UnboundLocalError: local variable 'x'
referenced before assignment
```

03. RECURSION

- Factorial
- Fibonacci Sequence
- Palindrome Test
- Function Calls

Recursion

- A recursion definition is made up of two parts:

  ① Base Case: directly specifies the result for a special case.

  ② Recursive Case: defines the answer in terms of the answer to the question on some other input, typically a simpler version of the same problem.

Factorial

n! = n * (n-1)!

```
def factI(n):
        """Assumes n an int > 0
           Returns n!"""
        result = 1
        while n > 1:
                result = result * n
                n -= 1
                return result
```

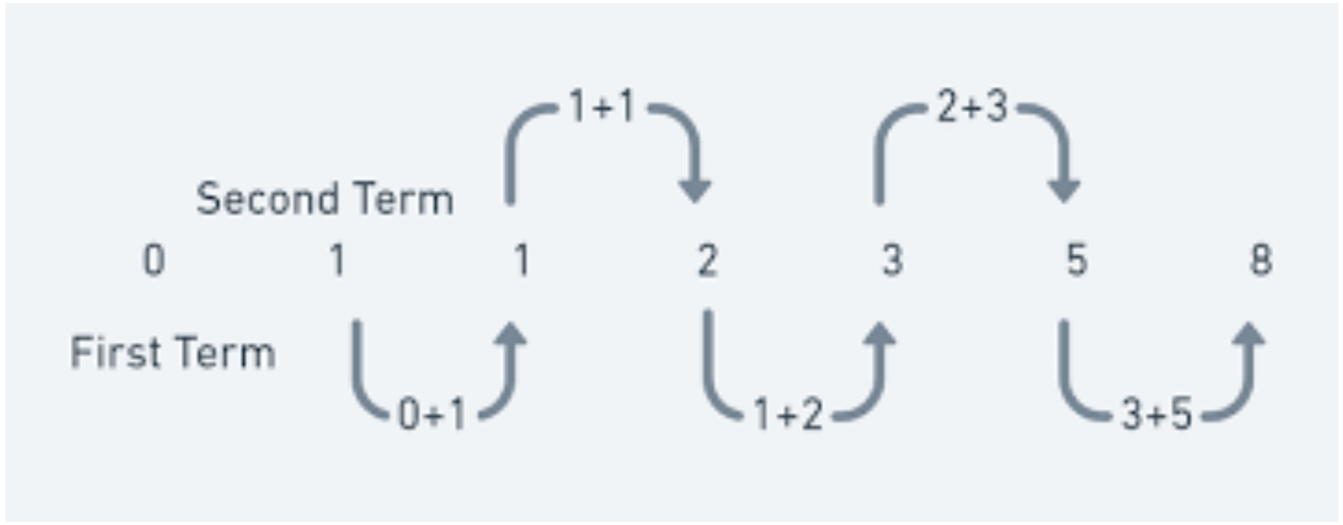
Iterative impletation

```
def factR(n):
        """Assumes n an int > 0
           Returns n!"""
        if n == 1:
                return n
        else:
                return n*factR(n - 1)
```

Recursive implementation

Fibonacci Sequence

$$f_n = f_{n-1} + f_{n-2}$$



Fibonacci Sequence (cont.)

```
def fib(n):
    """Assumes n int >= 0
       Returns Fibonacci of n"""
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)


def testFib(n):
    for i in range(n+1):
    print('fib of', i, '=', fib(i))
```

함수 fib에 n-1 넣었을 때의 return 값과 n-2 넣었을 때의 return 값을 더한 값

Palindrome Test

- Palindrome: 앞으로 읽어도, 뒤로 읽어도 같은 단어 및 문장
  - ex) pop, doggod, 101…

```
def isPalindrome(s):
      """Assumes s is a str
         Returns True if s is a palindrome; False otherwise.
         Punctuation marks, blanks, and capitalization are ignored."""
      def toChars(s):
            s = s.lower()
            letters = ''
            for c in s:
                  if c in 'abcdefghijklmnopqrstuvwxyz':
                        letters = letters + c
            return letters

      def isPal(s):
            print('   isPal called with', s)
            if len(s) <= 1:
                  print('About to return True from base case')
                  return True
            else:
                  answer = s[0] == s[-1] and isPal(s[1:-1])
                  print('   About to return', answer, 'for', s)
                  return answer

      return isPal(toChars(s))
```

Palindrome Test (cont.)

```
def testIsPalindrome():
      print('Try dogGod')
      print(isPalindrome('dogGod'))
      print('Try doGood')
      print(isPalindrome('doGood'))
```

```
Try dogGod
 isPal called with doggod
 isPal called with oggo
 isPal called with gg
 isPal called with
 About to return True from base case
 About to return True for gg
 About to return True for oggo
 About to return True for doggod
True
Try doGood
 isPal called with dogood
 isPal called with ogoo
 isPal called with go
 About to return False for go
 About to return False for ogoo
 About to return False for dogood
False
```

04. STRUCTURED TYPES

- Tuple
- Range
- List
- Dictionary

Tuple

- **Immutable ordered sequence of elements.**
  - Elements do not need to be characters.
  - Literals of tuple are written by enclosing a comma-separated list of elements in parentheses.

```
t1 = ()
t2 = (1, 'two', 3)
print(t1)
print(t2)
```

→

```
()
(1, 'two', 3)
```

  - Repetitions can be used.

```
3*('a', 2)
```

→

```
('a', 2, 'a', 2, 'a', 2)
```

Tuple – Examples

```
t1 = (1, 'two', 3)
t2 = (t1, 3.25)
print(t2)
print((t1 + t2))
print((t1 + t2)[3])
print((t1 + t2)[2:5])
```

```
((1, 'two', 3), 3.25)
(1, 'two', 3, (1, 'two', 3), 3.25)
(1, 'two', 3)
(3, (1, 'two', 3), 3.25)
```

Tuple (cont.)

- Python's multiple assignment statement can be also used in tuples.

$x, y = (3, 4) \longrightarrow$ bind 3 *to* `x`, and 4 to `y`

$a, b, c =$ 'xyz' $\longrightarrow$ bind '*x*' to `a`, '*y*' to `b`, and '*z*' to `c`

Range

```
range (start, stop, step)
```

- Immutable (like strings, tuples)

- Concatenation and repetition cannot be used.

- Range types can be compared whether they represent the same sequence of integers.

```
range(0, 7, 2) == range(0, 8, 2)   →  True  (0,2,4,6)
range(0, 7, 2) = range(6, -1, -2)  →  False (0,2,4,6)!=(6,4,2,0)
```

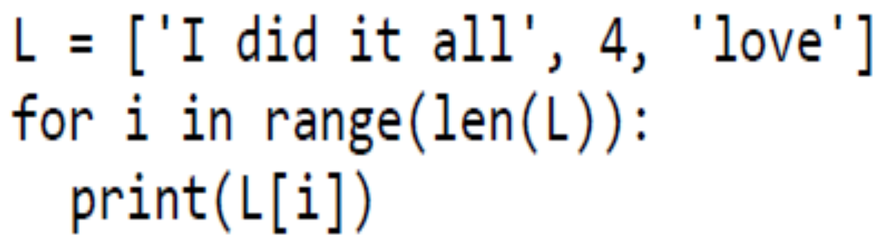
List

list_one = []

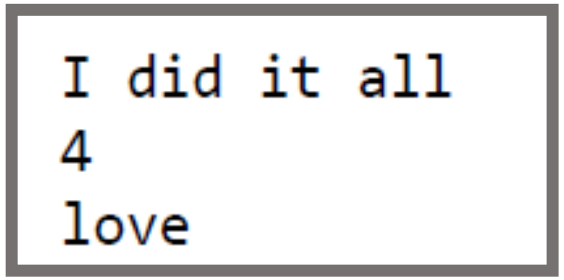
- **an ordered sequence of values**

◦ Mutable ⇒ can be modified after they are created
(cf. string and tuples are immutable)

◦ Each value is identified by an index (0,1,2...)



```
L = ['I did it all', 4, 'love']
for i in range(len(L)):
  print(L[i])
```



```
I did it all
4
love
```

List (cont.)

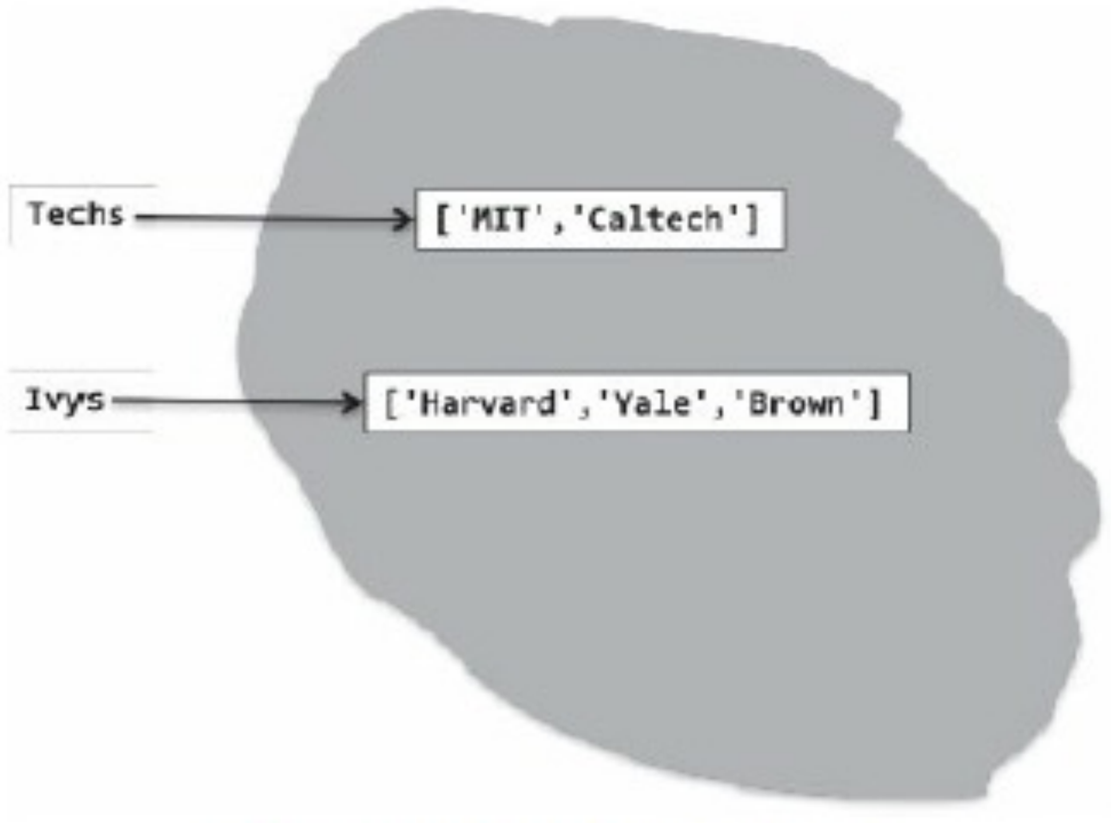
- Methods associated with lists

METHOD	what it does
`L.append(e)`	L의 끝에 객체 e 추가
`L.insert(i, e)`	L의 index i 위치에 객체 e 추가
`L.remove(e)`	L에서 가장 처음 등장하는 e 제거
`L.pop(i)`	L에서 i 위치에 있는 item을 제거하고 item 값 return
`L.count(e)`	L에 e가 몇 개 있는지 return
`L.index(e)`	L에 e가 몇 번 index에 있는지 return
`L.sort( )`	L을 오름차순으로 정렬
`L.reverse( )`	L의 순서를 거꾸로 바꿈
`L.extend(L2)`	L 뒤에 L2를 붙임

List – Mutability

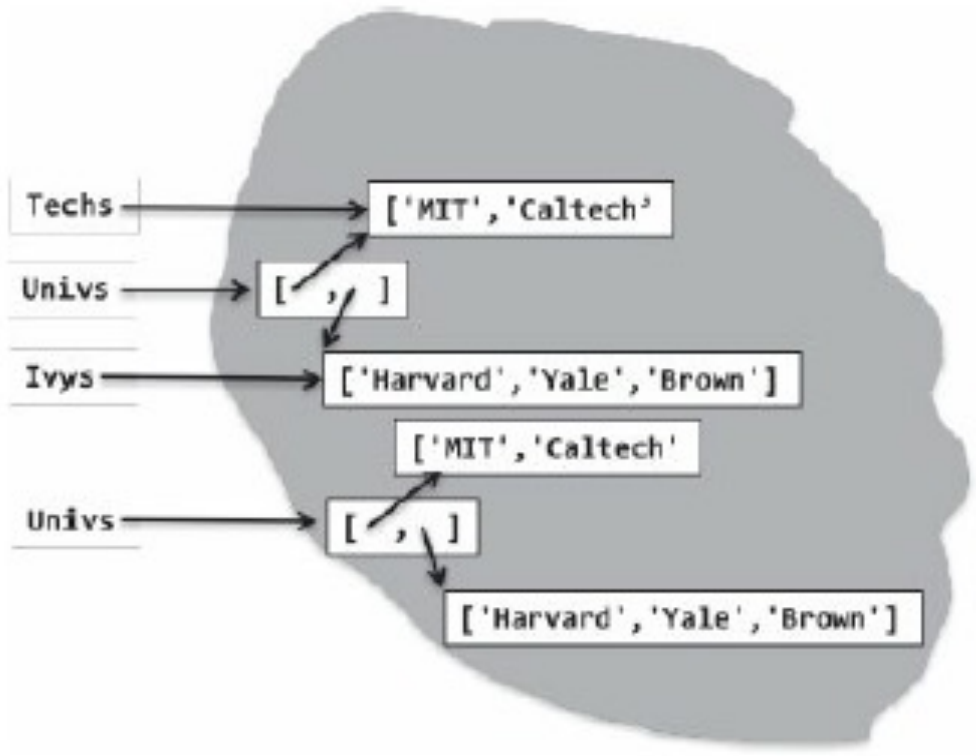
- A variable is merely a name. ⇒ A label that can be attached to an object.

```
Techs = ['MIT', 'Caltech']
Ivys = ['Harvard', 'Yale', 'Brown']
```



List – Mutability (cont.)

```
Univs = [Techs, Ivys]
Univs1 = [['MIT', 'Caltech'],
['Harvard', 'Yale', 'Brown']]
```



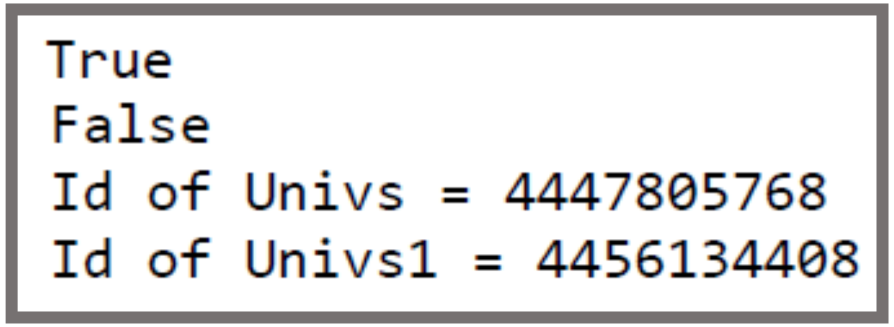
```
print('Univs =', Univs)
print('Univs1 =', Univs1)
print(Univs == Univs1)
```

```
Univs = [['MIT', 'Caltech'],
['Harvard', 'Yale', 'Brown']]
Univs1 = [['MIT', 'Caltech'],
['Harvard', 'Yale', 'Brown']]
True
```

List – Mutability (cont.)

- Value Equality Test vs Object Equality Test
  - `id(`*`type_of_something`*`)`: 객체의 메모리값을 return

```
print(Univs == Univs1)
print(id(Univs) == id(Univs1))
print("Id of Univs = ", id(Univs))
print("Id of Univs1 = ", id(Univs1))
```



```
True
False
Id of Univs = 4447805768
Id of Univs1 = 4456134408
```

List – Cloning

- It is prudent to avoid mutating a list over which one is iterating.

```
def removeDups(L1, L2):
  """Assumes that L1 and L2 are lists.
     Removes any element from L1 that
     also occurs in L2"""
  for e1 in L1:
     if e1 in L2:
        L1.remove(e1)
L1 = [1,2,3,4]
L2 = [1,2,5,6]
removeDups(L1, L2)
print('L1 =', L1)
```

$\longrightarrow$

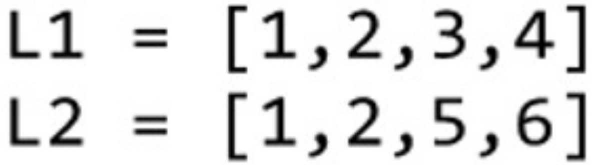
```
L1 = [2, 3, 4]
```

⇒ 원하던 결과인 `L1 = [3, 4]`가 아님.

2가 remove되지 않음!

List – Cloning (cont.)

- During a **for** loop, the implementation of Python keeps track of where it is in the list using an **internal counter** that is incremented at the end of each iteration.



```
L1 = [1,2,3,4]
L2 = [1,2,5,6]
```

① **COUNTER = 0**
check if `L1[0]` is in `L2` → YES**:** removes it (1)
⇒ `L1 = [2, 3, 4]`
⇒ length of `L1` becomes (4 →) **3**.

② **COUNTER = 1**

check if `L1[1]` (3) is in `L2` → NO: does nothing

*this is not the original value of `L1[1]` (2), but the current value of `L1[1]` (3)

List – Cloning (cont.)

- One way to avoid this kinf of problem is to use slicing to **clone** of the list and write for e1 in `L1[:]`.

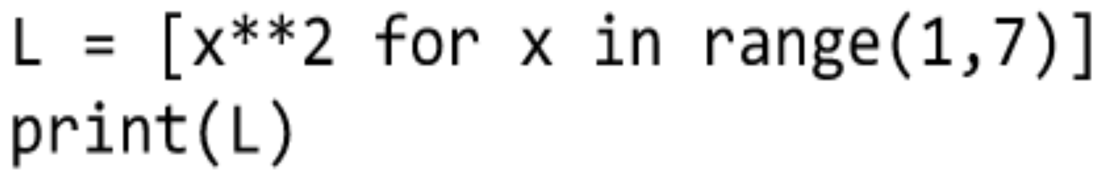
```
def removeDups(L1, L2):
  """Assumes that L1 and L2 are lists.
     Removes any element from L1 that
     also occurs in L2"""
  ~~for e1 in L1:~~
     if e1 in L2:
        L1.remove(e1)
L1 = [1,2,3,4]
L2 = [1,2,5,6]
removeDups(L1, L2)
print('L1 =', L1)
```

```
for e1 in L1[:]
```

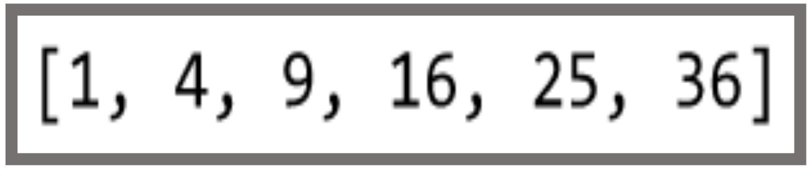
```
L1 = [3, 4]
```

List - Comprehension

- Applies an operation to the values in a sequence, creating a new list.
- This new list contains function-applied values.



```
L = [x**2 for x in range(1,7)]
print(L)
```



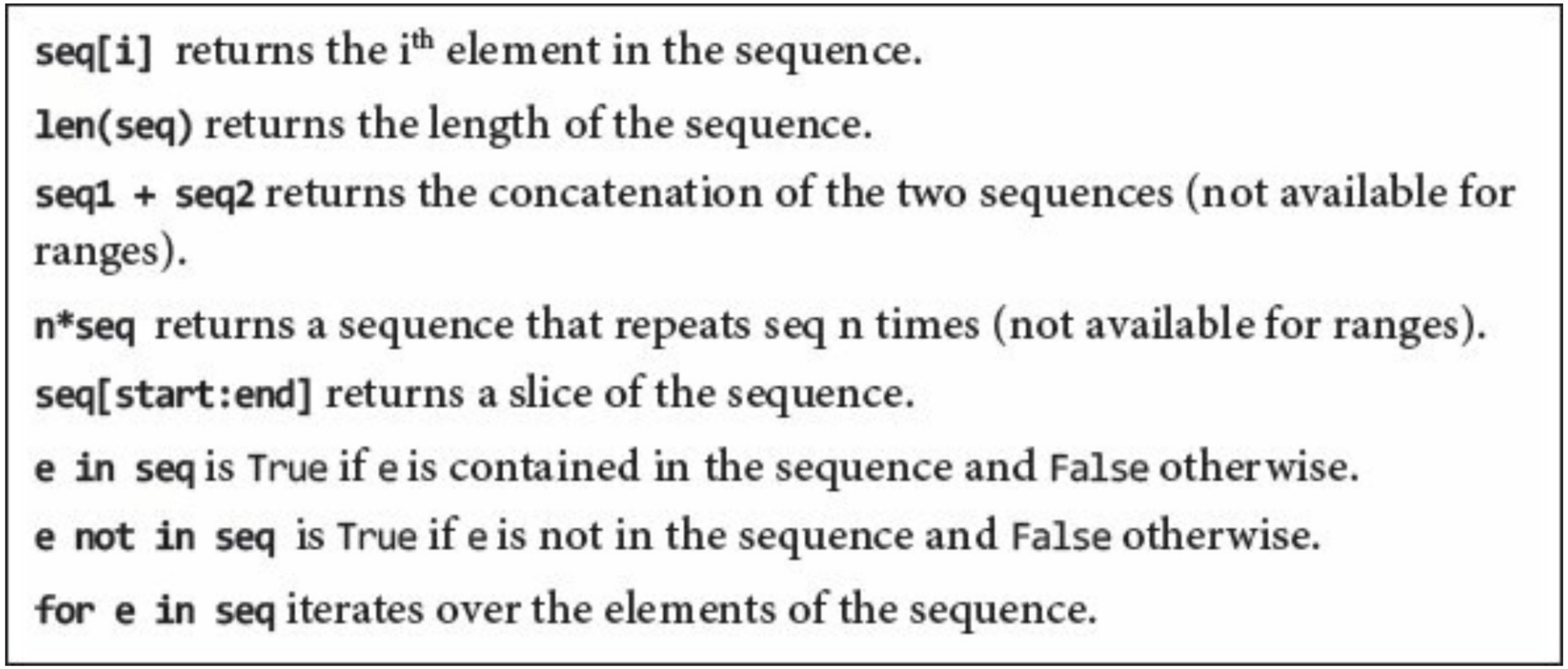
```
[1, 4, 9, 16, 25, 36]
```

05. SEQUENCE TYPES

Sequence Types

- Four different sequence types: `str, tuple, range, list`

- They are similar in that objects of these types can be operated upon.



seq[i] returns the ith element in the sequence.

len(seq) returns the length of the sequence.

seq1 + seq2 returns the concatenation of the two sequences (not available for ranges).

n*seq returns a sequence that repeats seq n times (not available for ranges).

seq[start:end] returns a slice of the sequence.

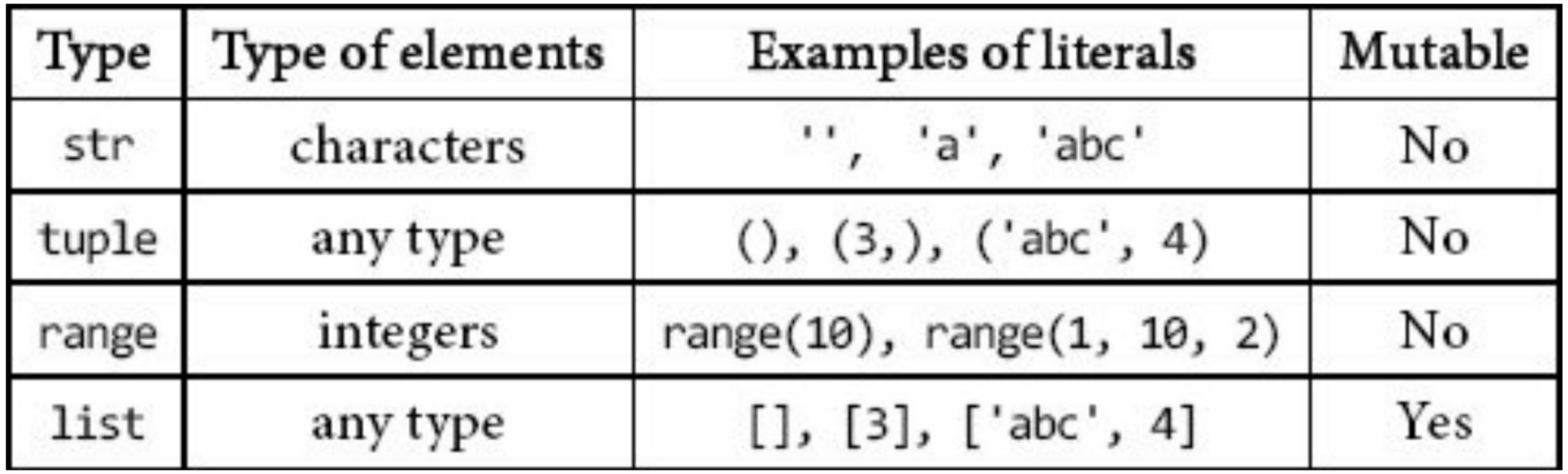
e in seq is True if e is contained in the sequence and False otherwise.

e not in seq is True if e is not in the sequence and False otherwise.

for e in seq iterates over the elements of the sequence.

Sequence Types (cont.)

- They have similarities and differences as summarized below.



Type	Type of elements	Examples of literals	Mutable
str	characters	`''`, `'a'`, `'abc'`	No
tuple	any type	`()`, `(3,)`, `('abc', 4)`	No
range	integers	`range(10)`, `range(1, 10, 2)`	No
list	any type	`[]`, `[3]`, `['abc', 4]`	Yes

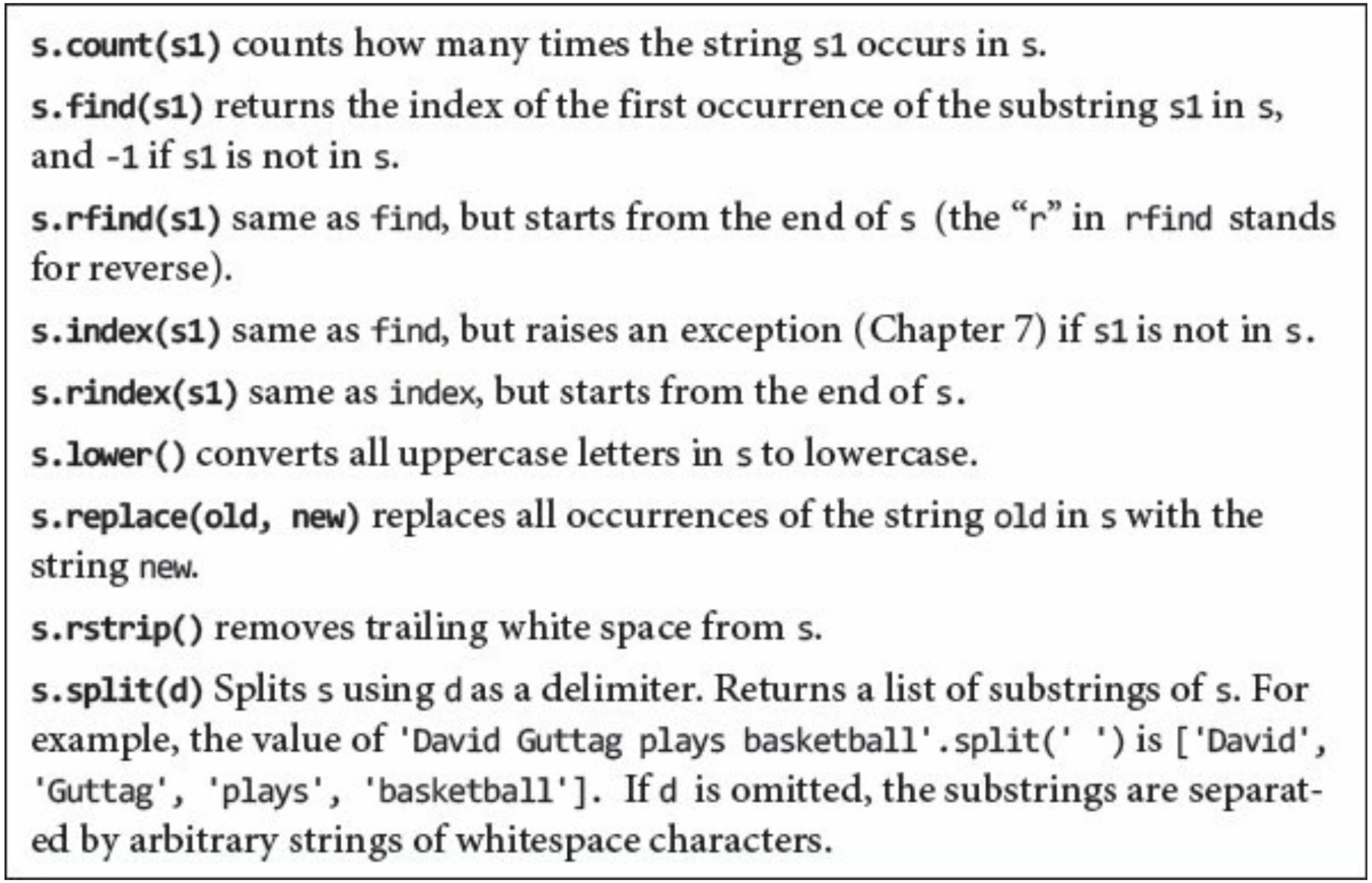
- Python programmers use `lists` far more often than `tuples`.
- Since `lists` are mutable, they can be constructed incrementally during a computation.

Sequence Type – string

- Since `strings` can contain only characters, they are considerably less versatile than `tuples` or `lists`.

- On the other hand, when you are working with a `string` of characters, there are many helpful built-in methods.

Sequence Type – string (cont.)



s.count(s1) counts how many times the string s1 occurs in s.

s.find(s1) returns the index of the first occurrence of the substring s1 in s, and -1 if s1 is not in s.

s.rfind(s1) same as find, but starts from the end of s (the "r" in rfind stands for reverse).

s.index(s1) same as find, but raises an exception (Chapter 7) if s1 is not in s.

s.rindex(s1) same as index, but starts from the end of s.

s.lower() converts all uppercase letters in s to lowercase.

s.replace(old, new) replaces all occurrences of the string old in s with the string new.

s.rstrip() removes trailing white space from s.

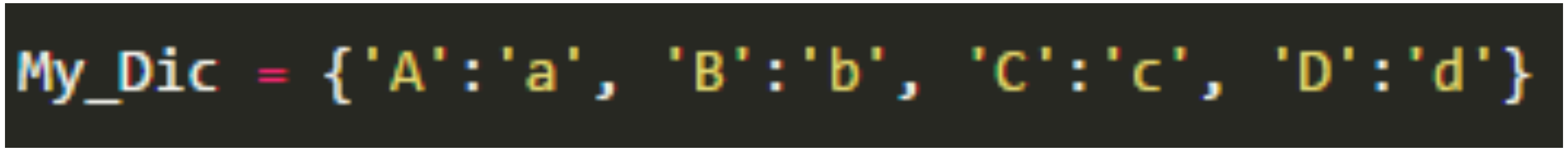
s.split(d) Splits s using d as a delimiter. Returns a list of substrings of s. For example, the value of 'David Guttag plays basketball'.split(' ') is ['David', 'Guttag', 'plays', 'basketball']. If d is omitted, the substrings are separated by arbitrary strings of whitespace characters.

04-5. DICTIONARY

Dictionary

```
My_dict = { }
```

- Object of type `dictionary`, consisting of sets of **`keys`** and **`values`** `{key : value}`
- Similar to `lists`, but uses `keys` to index.
- Mutable (add, modify, delete, etc…)
- The entries in a `dictionary` are unordered and cannot be accessed with an index.



```
My_Dic = {'A':'a', 'B':'b', 'C':'c', 'D':'d'}
```

Dictionary (cont.)

- Some common operations on dictionaries

METHOD	what it does
`dict[key] = value`	dict 사전에 `key:value` 쌍의 entry 입력
`del dict[key]`	dict 사전의 `key:value` 쌍의 entry 삭제
`dict[key]`	dict 사전에서 `key` 에 해당되는 `value`를 return
`dict.get(key, other)`	dict 사전에서 `key`에 해당되는 `value`를 return, 없으면 `other`을 return
`dict.keys()`	dict 사전의 `key`를 return (type: `dict_keys`)
`dict.values()`	dict 사전의 모든 `value` 값들을 return (type: `dict_values`)
`key in dict`	dict 사전에 `key`가 있는지 `True/False`로 return
`len(dict)`	dict 사전의 entry 개수 return
`for key in dict`	dict 사전의 모든 `key`들에 대해 iterate

Dictionary (cont.)

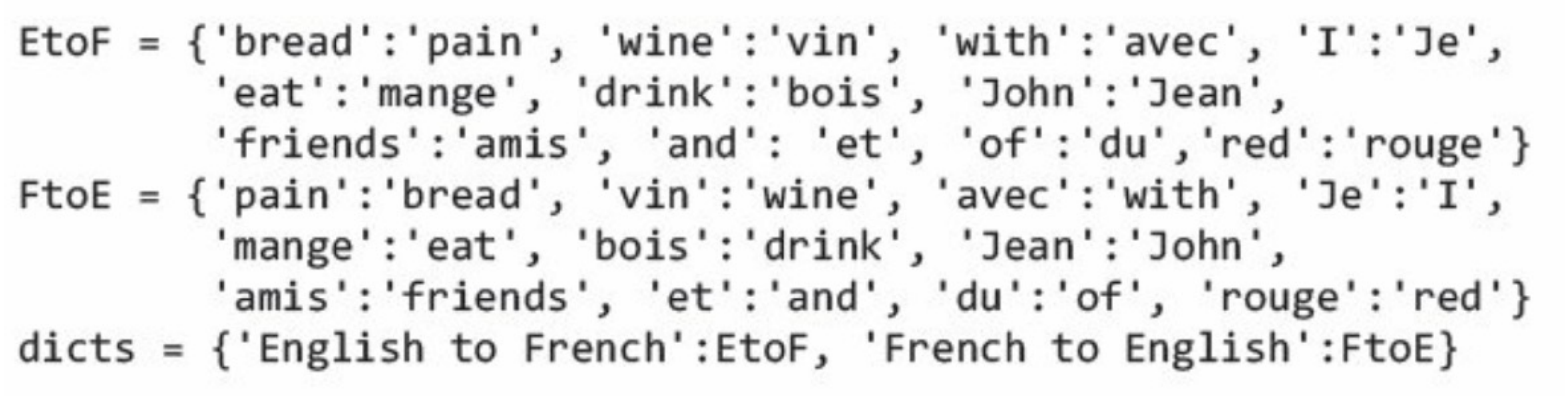
- Example 1

```
monthNumbers = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5,
1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May’}

print('The third month is ' + monthNumbers[3])

dist = monthNumbers['Apr'] - monthNumbers['Jan’]

print('Apr and Jan are', dist, 'months apart')
```

```
The third month is Mar
Apr and Jan are 3 months apart
```

Dictionary (cont.)

- Example 2



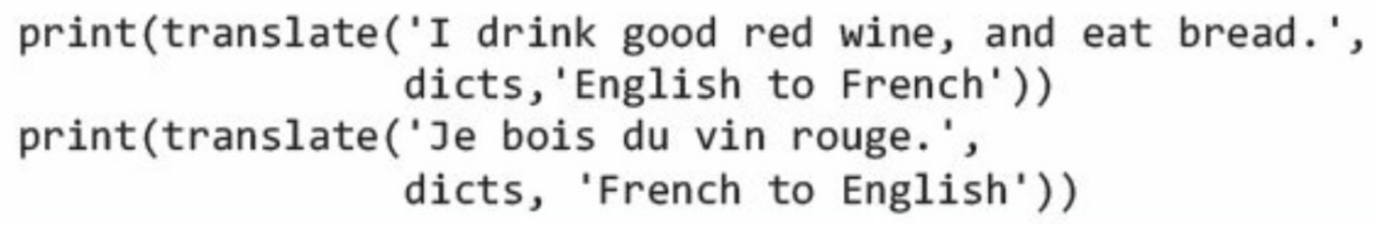
```
EtoF = {'bread':'pain', 'wine':'vin', 'with':'avec', 'I':'Je',
        'eat':'mange', 'drink':'bois', 'John':'Jean',
        'friends':'amis', 'and': 'et', 'of':'du','red':'rouge'}
FtoE = {'pain':'bread', 'vin':'wine', 'avec':'with', 'Je':'I',
        'mange':'eat', 'bois':'drink', 'Jean':'John',
        'amis':'friends', 'et':'and', 'du':'of', 'rouge':'red'}
dicts = {'English to French':EtoF, 'French to English':FtoE}
```

Dictionary (cont.)

```
def translateWord(word, dictionary):
    if word in dictionary.keys():
        return dictionary[word]
    elif word != '':
        return '"' + word + '"'
    return word
```

```
def translate(phrase, dicts, direction):
    UCLetters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    LCLetters = 'abcdefghijklmnopqrstuvwxyz'
    letters = UCLetters + LCLetters
    dictionary = dicts[direction]
    translation = ''
    word = ''
    for c in phrase:
        if c in letters:
            word = word + c
        else:
            translation = translation\
                          + translateWord(word, dictionary) + c
            word = ''
    return translation + ' ' + translateWord(word, dictionary)
```

Dictionary (cont.)



```
print(translate('I drink good red wine, and eat bread.',
                dicts,'English to French'))
print(translate('Je bois du vin rouge.',
                dicts, 'French to English'))
```

```
Je bois "good" rouge vin, et mange pain.

I drink of wine red.
```