w11-Lec

One-Dimensional Lists and Tuples

Assembled for 204111

by Kittipitch Kuptavanich

Lists

- พิจารณาการคำนวณค่าเฉลี่ยของจำนวนจริง n จำนวนที่นำเข้าจาก **Keyboard Input**

```
03 def find_mean(n):
04     sum_value = 0
05     for i in range(n):
06         num = int(input(""))
07         sum_value += num
08
09     return sum_value / n
```

- ในฟังก์ชันนี้เราใช้ตัวแปร `num` เพียง 1 ตัวในการเก็บค่าที่รับเข้ามาทั้ง n ค่าผ่านการ **Reassign (เขียนค่าทับ)**
- ในบางกรณีเช่น การหาค่า **Standard Deviation** การคำนวณต้องใช้แต่ละค่าที่รับเข้ามามากกว่า 1 ครั้ง

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

- จำเป็นต้องเก็บข้อมูล n จำนวน - ตัวแปร n ตัว? ⟶ ไม่สะดวกในการเรียกใช้
- **List** เป็น 1 ในชนิดข้อมูลที่สามารถใช้เก็บข้อมูลหลาย ๆ ค่าในตัวแปร 1 ตัว

Think Python: How to Think Like a Computer Scientist

Lists [2]

- **List** เป็นชนิดข้อมูลแบบประกอบ **(Compound Data Type)** ที่มีลักษณะเป็นรายการข้อมูลที่*มีลำดับ* **(Sequence Type)** คล้ายกันกับ **String**
  - **String** เป็นรายการอักขระ
  - **List** เป็นรายการข้อมูลประเภทใดก็ได้
- เราเรียกข้อมูลแต่ละตัวที่อยู่ใน **List** ว่า **Element** หรือ **Item**
- เราใช้เครื่องหมาย **Bracket** `[]` เพื่อแสดง **List** และคั่นระหว่างแต่ละ **Element** ด้วยเครื่องหมาย **Comma** `,` เช่น

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

https://docs.python.org/3/tutorial/introduction.html#lists

Lists [3]

- โดยมากแต่ละ **Element** ใน **List** จะมีชนิดข้อมูล **(Data Type)** เป็นชนิดเดียวกันทั้งหมด แต่ **List** สามารถประกอบด้วย **Element** ที่มีชนิดข้อมูล ต่างกันก็ได้

```
>>> mixed_list = ['spam', 2.0, 5, [10, 20]]
>>> empty = []
```

  - **List** ด้านบนประกอบด้วย **Element** ชนิด **String, Float, Integer** และ อีก **List** ซ้อนอยู่ข้างใน **(Nested List)**
- เราเรียก **List** ที่ไม่มี **Element** เรียกว่า **Empty List** (ลิสต์ว่าง)

Lists and Strings

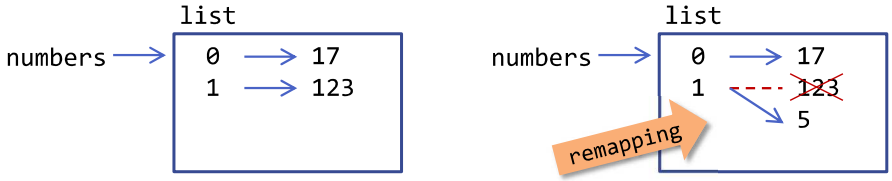
- ฟังก์ชัน `list()` ใช้สร้าง List จาก Iterable อื่น ๆ เช่น String

```
>>> a = list("wahoo!")          # from a string
>>> a
['w', 'a', 'h', 'o', 'o', '!']

>>> b = list(range(5))          # or from a range
>>> b
[0, 1, 2, 3, 4]

>>> e = list()
>>> e
[]                              # same as list("")

>>> s = "".join(a)              # use "".join(a) to convert a list
>>> s                           # of character to a string
'wahoo!'
                                # also works with a list of string
>>> "--".join(['parsley', '', 'is', '', 'gharsley'])
'parsley--is--gharsley'
```


Lists are Mutable

- List มีคุณสมบัติ Mutable กล่าวคือสามารถเปลี่ยนแปลงข้อมูลที่เก็บใน List ได้ (ต่างจาก String ซึ่งมีคุณสมบัติ Immutable)

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```



- เราสามารถพิจารณา List ในลักษณะความสัมพันธ์ระหว่าง Index และ Element เช่น Index 0 สัมพันธ์กับ (Maps to) ค่า 17

Indexing and Slicing

- Indexing และ Slicing ใน List มีลักษณะเดียวกันกับใน String

```
>>> squares = [1, 4, 9, 16, 25, 36, 49]
>>> squares[0]          # indexing returns the item
1

>>> squares[-1]
49

>>> squares[-3:]        # slicing returns a new list

>>> squares[1:5:2]      # slicing with [start:end:step]
[4, 16]

>>> squares[:]          # create a (shallow) copy of a list
[1, 4, 9, 16, 25]
```

Note: Indexing และ Slicing ใช้ได้กับทุก Sequence Type ได้แก่ List, Range, Tuple

Indexing and Slicing [2]

- เราสามารถ Assign ค่าให้กับ Slice ของ List ได้

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']

>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']

>>> # replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

List Properties (len, min, max, sum)

```
# List Property Built-in Functions

>>> a = [2, 3, 5, 2]
>>> a
[2, 3, 5, 2]

>>> len(a)
4

>>> min(a)
2

>>> max(a)
5

>>> sum(a)
12
```


List Operations

- **The + operator concatenates lists:**

```
>>> a = [5, 3]
>>> b = [2, 1] + a
>>> print(b)
[2, 1, 5, 3]
```

- **Similarly, the * operator repeats a list:**

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```


The del Statement

- เราใช้คำสั่ง `del` ประกอบกับ **Slicing** เพื่อลบสมาชิกบางตัวหรือทุกตัวได้

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> del letters[2:5]              # remove some elements
>>> letters
['a', 'b', 'f', 'g']

>>> del letters[:]                # remove all elements
>>> letters
[]                                # empty list

>>> del letters
>>> letters                       # no reference to the list
...
NameError: name 'letters'is not defined
```


Adding Elements

- **List เป็น Data Type ประเภท Mutable Sequence Type**
  - **สามารถใช้ Method ของ Mutable Sequence Type ได้**

เพิ่ม Elements **– List เดิม (Destructively)**

```
>>> a = [2, 3]                  # Create a list

>>> a.append(7)      # Add an item with method list.append(item)
>>> a
[2, 3, 7]
>>> a.append([4, 5])
>>> a
[2, 3, 7, [4, 5]]

>>> a += [11, 13]    # Add a list of items with list += list2
>>> a
[2, 3, 7, [4, 5], 11, 13]
```

Adding Elements [2]

เพิ่ม *Elements* – List เดิม (Destructively) [2]

```
>>> a
[2, 3, 7, 11, 13]

# Add a list of items with method list.extend(list2)
>>> a.extend([17, 19])
>>> a
[2, 3, 7, 11, 13, 17,19]

# Insert an item with method list.insert(index, element)
>>> a.insert(2, 5)          # at index 2, insert a 5
>>> a
[2, 3, 5, 7, 11, 13, 17, 19]
```


Adding Elements [3]

เพิ่ม *Elements* – สร้าง List ใหม่ (Non-destructively)

```
>>> a
[2, 3, 7, 11]

# Add an item with list1 + list2
b = a + [13, 17]
>>> a
[2, 3, 7, 11]
>>> b
[2, 3, 7, 11, 13, 17]

# Insert an item at a given index (with list slices)
>>> b = a[:2] + [5] + a[2:]
>>> a
[2, 3, 7, 11]
>>> b
[2, 3, 5, 7, 11]
```


Finding Elements

ค้นหา *Elements*

```
>>> a = [2, 3, 5, 2, 6, 2, 2, 7]

>>> 2 in a          # Check for list membership:  in
True
>>> 4 in a
False

>>> 4 not in a      # Check for list non-membership:  not in
True

# Count occurrences with method list.count(item)
>>> a.count(1)
0
>>> a.count(2)
4
```


Finding Elements [2]

ค้นหา *Elements* [2]

```
>>> a = [2, 3, 5, 2, 6, 2, 2, 7]

# Find index of item with method list.index(item)
>>> a.index(6)
4
>>> a.index(2)
0                              # Index of the first item found

# list.index(item, start)
>>> a.index(2, 1)
3                              # Start looking at index 1
>>> a.index(2, 4)
5
>>> a.index(8)
...
ValueError: 8 is not in list
```

Removing Elements

ลบ *Elements* – List เดิม (Destructively)

```
>>> a = [2, 3, 5, 3, 7, 5, 11, 13]

# Remove an item with method list.remove(item)
>>> a.remove(5)
>>> a
[2, 3, 3, 7, 5, 11, 13]  # Remove only the first occurrence

>>> a.remove(5)
>>> a
[2, 3, 3, 7, 11, 13]

>>> a.remove(5)
...
ValueError: list.remove(x): x not in list
```


Removing Elements [2]

ลบ *Elements* – List เดิม (Destructively)

```
>>> a = [2, 3, 5, 8, 7, 5, 11, 13]

# Remove an item at a given index with method list.pop(index)
>>> item = a.pop(3)
>>> item
8
>>> a
[2, 3, 5, 7, 5, 11, 13]

# Remove last item with list.pop()
>>> item = a.pop()
>>> item
13
>>> a
[2, 3, 5, 7, 5, 11]
```


Removing Elements [3]

ลบ *Elements* – สร้าง List ใหม่ (Non-destructively)

```
>>> a = [2, 3, 5, 7, 11]

# Remove an item at a given index (with list slices)
>>> b = a[:2] + a[3:]
>>> a
[2, 3, 5, 7, 11]
>>> b
[2, 3, 7, 11]
```


List Alias

```
# Create a list
>>> a = [2, 3, 5, 7]

# Create an alias to the list
>>> b = a

# We now have two references (aliases) to the SAME list
>>> a[0] = 42
>>> b[1] = 99

>>> print(a)
[42, 99, 5, 7]
>>> print(b)
[42, 99, 5, 7]

>>> a is b
True
```

List Alias [2]

```
>>> a = [2, 3, 5, 7]
>>> b = a               # Create an alias to the list

# Create a different list with the same elements
>>> c = [2, 3, 5, 7]

# a and b are references (aliases) to the SAME list
# c is a reference to a different but EQUAL list

>>> a == b
True
>>> a is b
True
>>> a == c
True
>>> a is c
False
```


List Alias [3]

```
# Function parameters are aliases, too!
>>> def f(a):
...     a[0] = 42

>>> a = [2, 3, 5, 7]


>>> f(a)
>>> print(a)
[42, 3, 5, 7]
```


Looping over Lists

```
>>> a = [2, 3, 5, 7]

# Looping with: for item in list
>>> for item in a:
...     print(item, end=" ")

2 3 5 7

# Looping with:  for index in range(len(list))
>>> for index in range(len(a)):
...     print("a[", index, "] = ", a[index], sep="")

a[0] = 2
a[1] = 3
a[2] = 5
a[3] = 7
```


Looping over Lists [2]

```
# Looping backward a = [2, 3, 5, 7]
>>> for index in range(len(a)):
...     revIndex = len(a) - 1 - index
...     print("a[", revIndex, "] = ", a[revIndex], sep="")

a[3] = 7
a[2] = 5
a[1] = 3
a[0] = 2

# Hazard!!:  Modifying While Looping
>>> for index in range(len(a)):
...     if (a[index] == 3):
...         a.pop(index)

3
IndexError: list index out of range
```

Using Lists with Functions

- **List Parameters**
  - **Example:** `count_odds(list)`

```
09 def count_odds(a):
10     count = 0
11     for item in a:
12         if (item % 2 == 1):
13             count += 1
14     return count
15
16 print(count_odds([2, 3, 7, 8, 21, 23, 24]))
17 # 4
```


Using Lists with Functions [2]

- **Modifying list elements is visible to caller:**

  `fill(list, value)`

```
>>> def fill(a, value):
...     for i in range(len(a)):
...         a[i] = value

>>> a = [1, 2, 3, 4, 5]
>>> fill(a, 42)
>>> a
[42, 42, 42, 42, 42]
```


Using Lists with Functions [3]

- **List Return Type**
  - **Example:** `numbers_with_3s()`

```
02 def numbers_with_3s(lo, hi):
03     result = []
04
05     for x in range(lo, hi):
06         if ("3" in str(x)):
07             result.append(x)
08     return result
09
10 print(numbers_with_3s(250, 304))
11 # [253, 263, 273, 283, 293, 300, 301, 302, 303]
```


Think Python: How to Think Like a Computer Scientist

Map, Filter and Reduce

- ฟังก์ชัน `sum()` ดำเนินการบน Element ทุกตัวใน List แล้วให้ Return Value เป็นผลรวมของ แต่ละ Element
  - เราเรียกการดำเนินการโดยใช้ค่าของ Element หลาย ๆตัวใน List แล้วให้ผลลัพธ์เป็นค่า*เพียงหนึ่งค่า*ว่า **Reduce**
- พิจารณาฟังก์ชัน `only_upper()` ที่สร้าง List ใหม่ จากคำใน List ที่เป็น Upper Case เท่านั้น

```
02 def only_upper(word_list):
03     result = []
04     for word in word_list:
05         if word.isupper():
06             result.append(word)
07     return result
```

  - Operation ในลักษณะนี้เรียกว่า **Filter** เนื่องจาก*เลือก*เฉพาะสมาชิก*บางตัว*จาก List และคัดกรองบางตัวทิ้งไป

Map, Filter and Reduce [2]

- พิจารณาฟังก์ชัน `capitalize_all()` ที่สร้าง List ใหม่ที่ประกอบด้วยสมาชิกของเดิมทุกตัวในรูป Capitalized

```
09 def capitalize_all(word_list):
10     result = []
11     for word in word_list:
12         result.append(word.capitalize())
13     return result
```

  - Operation ในลักษณะนี้เรียกว่า **Map** เนื่องจากสมาชิกแต่ละตัวใน List ผลลัพธ์ เกิดจากการดำเนินการ (ในกรณีนี้ `str.capitalize()`) ลงบนสมาชิกแต่ละตัวของ List เดิม (1:1)

```
>>> before = ['star', 'wars:', 'the', 'force', 'awakens']
>>> after = capitalize_all(before)
>>> after
['Star', 'Wars:', 'The', 'Force', 'Awakens']
```


The `map()` Function

- Python มีฟังก์ชัน built-in `map()` เพื่อใช้ดำเนินการ Map ฟังก์ชันใด ๆ ไปที่แต่ละ Element ของ List (หรือ Iterable อื่น ๆ)

```
>>> from math import pi as PI
>>> def circle_area(radius):
...     return PI * radius ** 2

>>> a = [1, 2, 3, 4, 5]
>>> result = map(circle_area, a)
>>> result
<map object at 0x039AF210>

>>> list(result)
[3.141592653589793, 12.566370614359172, 28.274333882308138,
50.26548245743669, 78.53981633974483]

# or result_list = list(map(circle_area, a))
```

ต้องเป็นฟังก์ชันที่ทำงานกับพารามิเตอร์ตัวเดียว

The `map()` Function [2]

- พิจารณาการหาความยาวหลักของสมาชิกใน List จำนวนเต็มบวก

```
>>> def digit_count(x):
        return len(str(x))

>>> a = [3197, 69, 73948, 8216, 4, 982660, 58]
>>> b = list(map(digit_count, a))
>>> b
[4, 2, 5, 4, 1, 6, 2]

>>> d = list(map(lambda x: len(str(x)), a))
>>> d
[4, 2, 5, 4, 1, 6, 2]
```

parameter

expression

- `lambda` statement ใน Python มีหน้าที่เปลี่ยน Parameter และ Expression ให้เป็นฟังก์ชันที่ไม่มีชื่อ โดยฟังก์ชันจะมีหน้าที่คืนค่าที่ evaluate ได้ตาม Expression ที่ระบุ

The `filter()` Function

- Python มีฟังก์ชัน built-in `filter()` เช่นกัน

```
>>> def positive(x):    # To use with filter() the function
...     return x > 0    # must return True or False only

>>> a = [1, -2, -3, 4, 5]
>>> list(filter(positive, a)))
[1, 4, 5]
```

- ฟังก์ชัน `reduce()` ถูกถอดออกจากฟังก์ชัน built-in ใน Python 3 เนื่องจากทำให้ Code อ่านและเข้าใจยากกว่าการใช้ Loop ปกติ และย้ายไปอยู่ใน Module `functools`

Sorting Elements

เรียง *Elements* – List เดิม (Destructively)

```
>>> a = [7, 2, 5, 3, 5, 11, 7]

# Sort item destructively with method list.sort()
>>> a.sort()
>>> a
[2, 3, 5, 5, 7, 7, 11]
```

เรียง *Elements* – สร้าง List ใหม่ (Non-destructively)

```
>>> a = [7, 2, 5, 3, 5, 11, 7]
>>> b = sorted(a) # non-destructively with sorted(list)
>>> a
[7, 2, 5, 3, 5, 11, 7]
>>> b
[2, 3, 5, 5, 7, 7, 11]
```


Reverse Elements

กลับลำดับ *Elements* – List เดิม (Destructively)

```
>>> a = [1, 2, 3, 4, 5, 6, 7]

>>> # Sort item destructively with method list.reverse()
>>> a.reverse()
>>> a
[7, 6, 5, 4, 3, 2, 1]
```

กลับลำดับ *Elements* – สร้าง List ใหม่ (Non-destructively)

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = list(reversed(a)) # non-destructively with reversed(list)
>>> a
[1, 2, 3, 4, 5, 6, 7]
>>> b
[7, 6, 5, 4, 3, 2, 1]
```


Swapping Elements

```
>>> a = [2, 3, 5, 7]

>>> a[0] = a[1]              # Failed swap
>>> a[1] = a[0]
>>> a
[3, 3, 5, 7]

>>> a = [2, 3, 5, 7]         # Swap with a temp variable
>>> temp = a[0]
>>> a[0] = a[1]
>>> a[1] = temp
>>> a
[3, 2, 5, 7]

>>> a = [2, 3, 5, 7]
>>> a[0], a[1] = a[1], a[0]  # Swap with tuple assignment
>>> a
[3, 2, 5, 7]
```

Tuple swap

```
x, y = y, x
```


Comparing Lists

```
>>> a = [2, 3, 5, 3, 7]
>>> b = [2, 3, 5, 3, 7]       # same as a
>>> c = [2, 3, 5, 3, 8]       # differs in last element
>>> d = [2, 3, 5]             # prefix of a

>>> a == b
True
>>> a == c
False
>>> a != b
False
>>> a != c
True

>>> a < c
True
>>> a < d
False
```

เทียบทีละ Element
ลักษณะเดียวกับการเทียบคำในภาษาอังกฤษ ทีละตัวอักษร

```
bat < cat       #True
```

https://docs.python.org/3/library/stdtypes.html#typesseq-mutable

List Operation Summary

Operation	Result	Notes
s[i] = x	item *i* of *s* is replaced by *x*	
s[i:j] = t	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*	
del s[i:j]	same as *s*[*i*:*j*] = []	
s[i:j:k] = t	the elements of *s*[*i*:*j*:*k*] are replaced by those of *t*	
del s[i:j:k]	removes the elements of s[*i*:*j*:*k*] from the list	
s.append(x)	appends *x* to the end of the sequence (same as *s*[len(*s*):len(*s*)] = [*x*])	
s.clear()	removes all items from *s* (same as del *s*[:])	
s.copy()	creates a shallow copy of *s* (same as *s*[:])	
s.extend(t)	extends *s* with the contents of *t* (same as *s*[len(*s*):len(*s*)] = *t*)	
s.insert(i, x)	inserts *x* into *s* at the index given by *i* (same as *s*[*i*:*i*] = [*x*])	
s.pop([i])	retrieves the item at *i* and also removes it from *s*	
s.remove(x)	remove the first item from *s* where *s*[*i*] == *x*	
s.reverse()	reverses the items of *s* in place	

Tuples

- **Tuple เป็นรายการข้อมูลที่มีลำดับ (Sequence Data Type) เช่นเดียวกันกับ List, String, และ Range* ที่มีลักษณะ Immutable**
  - **เราใช้เครื่องหมาย Comma , คั่นระหว่างแต่ละ Element และใช้เครื่องหมายวงเล็บ () ล้อมรอบ เช่น (1, 2, 3)**
  - **Tuple ต้องมี Comma เสมอ – แต่ไม่จำเป็นต้องมีเครื่องหมายวงเล็บ**

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
```

*https://docs.python.org/3/library/stdtypes.html#typesseq

https://docs.python.org/3/tutorial/datastructures.html

Tuples

- **กรณีเป็น Tuple ว่างเราจำเป็นต้องใส่เครื่องหมายวงเล็บ**
- **หากไม่ใช่ Tuple ว่าง ต้องใส่ Comma *ทุกกรณี***

```
>>> ()                          # Empty tuple, needs parentheses
()
>>> 1,                          # Singleton (One Element)
(1,)

>>> t = 12345, 54321, 'python!'# Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u ((12345, 54321, 'python!'), (1, 2, 3, 4, 5))

>>> t[0] = 88888                        # Tuples are immutable:
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```


Think Python: How to Think Like a Computer Scientist

Tuples Assignment

```
>>> tuple('hello') # Creating from iterables using tuple()
('h', 'e', 'l', 'l', 'o')

>>> a = tuple([2, 3, 5, 7])  # tuple from list
(2, 3, 5, 7)

>>> a, b, c = 'cat'      # multiple assignment
>>> a
'c'
>>> b
'a'
>>> c
't'

>>> a, b = [2, 8]
>>> a
2
>>> b
8
```

Immutability

```
>>> t[0] = 88888                    # Tuples are immutable:
TypeError: 'tuple'object does not support item assignment
>>> # but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])

>>> t = ('a', 'b', 'c', 'd', 'e')
>>> # Cannot modify but can replace one tuple with another:
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

- เราไม่สามารถเปลี่ยน Element ใน Tuple ได้ แต่เราสามารถ Assign ค่าใหม่ได้ (ย้าย Reference ไปชี้ที่ Tuple ใหม่)

Tuple Swap

```
>>> a, b, c = [2, 8, 5]
>>> a
2
>>> b
8
>>> c
5

>>> # Swapping
>>> b, c, a = a, b, c
>>> a
5
>>> b
2
>>> c
8
```


Tuples as Return Values

- ฟังก์ชันใด ๆ สามารถคืนค่าได้เพียงค่าเดียว หากต้องการคืนค่ามากกว่าหนึ่งค่า เราสามารถคืนค่าเป็น Sequence Type เช่น Tuple (หรือ List) ได้
  - เช่น ในการหารจำนวนเต็มการคำนวณหาผลหาร (Quotiant) และเศษ (Remainder) ทั้งสองค่าในคราวเดียวกัน เป็นวิธีที่มีประสิทธิภาพมากกว่าโดยการใช้ฟังก์ชัน built-in `divmod()`

```
>>> t = divmod(7, 3)
>>> t
(2 , 1)
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```


Strings, Lists and Tuples

`seq[i]` returns the i^{th} element in the sequence.
`len(seq)` returns the length of the sequence.
`seq1 + seq2` returns the concatenation of the two sequences.
`n * seq` returns a sequence that repeats seq n times.
`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.

Figure 5.6 Common operations on sequence types

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

Figure 5.7 Comparison of sequence types

Reference

- http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-1d-lists.html
- https://docs.python.org/3/tutorial/introduction.html#lists
- https://docs.python.org/3/tutorial/datastructures.html#more-on-lists
- https://docs.python.org/3.3/tutorial/datastructures.html#tuples-and-sequences
- https://docs.python.org/3/library/stdtypes.html#typesseq-mutable
- https://docs.python.org/3/library/stdtypes.html#tuple
- Guttag, John V ***Introduction to Computation and Programming Using Python, Revised***