w10-Lec

# Strings

Assembled for 204111

by Kittipitch Kuptavanich



## Strings

- **Besides numbers, Python can also manipulate strings. We can define strings using:**
  - **single quotes (`'...'`)**
  - **double quotes (`"..."`)**
- **Also `\` can be used to escape quotes**





## Displaying Strings

- **ในกรณีที่ String ประกอบด้วย Escaped Characters เราสามารถใช้ฟังก์ชัน `print()` ในการแสดง Output ที่อ่านง่ายขึ้น**

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.

>>> s = 'First line.\nSecond line.'
>>> s                 # \n is included in the output
'First line.\nSecond line.'

>>> print(s)          # with print(), \n produces a new line
First line.
Second line.
```





## Displaying Strings [2]

- **หากไม่ต้องการให้ตัวอักขระที่ตามหลัง Backslash ถูกตีความว่าเป็นอักขระพิเศษ หรือ Escaped Sequence**
  - **เราสามารถบังคับให้แสดงผลแบบ Raw String ได้โดยการเพิ่มตัว `r` ก่อนการเปิดเครื่องหมายคำพูด**

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
  ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

# Multi-Line String Literals

- String Literals สามารถมีความยาวข้ามบรรทัดได้ โดยการใช้ """
  - โดยจะมีการเพิ่ม End-of-line Character (EOL) ให้อัตโนมัติ
- สามารถใช้เครื่องหมาย \ กันเพื่อไม่ให้มีการเพิ่ม EOL ได้

```
>>> s = """
multi-line
text!
"""
>>> print(s)
multi-line
text!
>>>
```

```
>>> s = """\
multi-line
text!\
"""
>>> print(s)
multi-line
text!
>>>
```





# Concatenation

- เราเรียกการนำ String มากกว่าหนึ่ง String มาเชื่อมต่อกันว่า Concatenation
- เราสามารถเชื่อม String โดยใช้เครื่องหมาย + และ, ทำซ้ำด้วย *

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
```

- เมื่อวาง String Literal ไว้ติดกันจะเกิดการ Concatenate โดยอัตโนมัติ (ต้องเป็น Literals ทั้งคู่ - ใช้กับ variable ไม่ได้)

```
>>> 'Py' 'thon'
'Python'
>>> prefix = 'Py'
>>> prefix 'thon'
 ...
SyntaxError: invalid syntax
```





# String Indexing

- String คืออักขระหลาย ๆ ตัวมาวางต่อกัน (Text *Sequence Type*)
  - เราสามารถเข้าถึงอักขระแต่ละตัวได้ โดยการใช้เครื่องหมาย Bracket [] เช่น a[1] (aka. Subscript Notation)

```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

- Statement ในบรรทัดที่ 2 ดึงอักขระตัวที่ 1 จากตัวแปร fruit แล้ว assign ให้กับตัวแปร letter
- ตัวเลขที่อยู่ภายในเครื่องหมาย Bracket เรียกว่า Index
- แต่ผลลัพธ์อาจไม่ใช่อย่างที่คิด

```
>>> print(letter)
a
```





# String Indexing [2]

```
>>> print(letter)
a
```

- สำหรับคนทั่วไป อักขระตัวแรกใน 'banana' คือ b ไม่ใช่ a
  - แต่สำหรับ Computer Scientist
    - Index คือ Offset จากต้น String
    - อักขระตัวแรกจึงมี Index เท่ากับ 0

```
>>> letter = fruit[0]
>>> print(letter)
b
```

- Index ต้องเป็นเลขจำนวนเต็มเท่านั้น

```
>>> letter = fruit[1.5]
TypeError: string indices must be integers
```

# String Indexing [3]

- ใน Python อักขระ 1 ตัวถือว่าเป็น String ขนาด 1 ตัวอักษร
- เราสามารถใช้ฟังก์ชัน `len()` เพื่อบอกจำนวนอักขระใน String

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

- หากต้องการ อักขระตัวสุดท้ายของ String
  - เนื่องจาก Index เริ่มจาก 0 และมีอักขระทั้งหมด 6 ตัว
  - Index ของอักขระตัวสุดท้ายจึงมีค่าเป็น_____

```
>>> length = len(fruit)
>>> last = fruit[__________]
>>> print(last)
a
```





# String Indexing [4]

- Index ใน Python สามารถมีค่าเป็นจำนวนลบได้ โดย Index ที่ `-1` จะเป็น Index ของอักขระตัวสุดท้าย
  - ในทำนองเดียวกัน Index ที่ `-2` จะเป็นอักขระตัว*รอง*สุดท้าย

```
>>> word = 'Python'
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```





# Slicing

- ในขณะที่ Index ใช้เพื่อเข้าถึงค่าอักขระแต่ละตัวใน String
  - Slicing เป็น Operation ที่ทำให้เราเข้าถึงอักขระ (หลายตัว) ในลักษณะ Substring (สายอักขระย่อย) ได้ โดยการใช้เครื่องหมาย Colon `:` ในรูปแบบ `string[start_index:end_index]`

```
>>> word[0:2]
'Py'
>>> word[2:5]
'tho'
```

```
>>> word[0:5]
'__________'
>>> word[0:____]
'Python'
```

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

- เราอาจมอง Index ในลักษณะเป็นเส้นกั้นระหว่างอักขระ
- จำนวนอักขระของ Slicing `[i:j]` คือ $j-i$





# Slicing [2]

- สังเกตว่าอักขระตัวที่ระบุด้วย Start Index จะถูกรวมไว้เสมอ ในขณะที่ อักขระตัวที่ระบุโดย End Index จะไม่ปรากฏในผลลัพธ์
  - เพื่อที่ `s[:i] + s[i:]` จะเท่ากับ `s` เสมอ

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

```
>>> word[:]
'Python'
>>> word[0:6]
'Python'
```

- หากไม่ระบุ Index ที่ใช้ การทำ Slicing จะใช้ค่า Default
  - Start Index ใช้ค่า ***0***, End Index จะใช้ค่า***ความยาว String***

```
>>> word[:2]  # 0 included to 2 excluded
'Py'
>>> word[4:]  # 4 (included) to the end
'on'
>>> word[-2:] # second-last (included) to the end
'on'
```

# Slicing [3]

- การใช้ indexing ที่มีความยาวมากกว่าความยาว String จะเกิด Error

```
>>> word = 'Python'
>>> word[42]  # the word only has 6 characters

IndexError: string index out of range
```

- แต่ใน Slicing Operation การใช้ตัวเลข Index ค่าที่มากกว่าความยาว String สามารถทำได้

```
>>> word[4:42]
'on'
>>> word[42:]
''
```





# Slicing [4]

- นอกจากนี้เรายังสามารถใช้ Slicing Operation ในรูปแบบ `string[start_index:end_index:step]` โดยมีลักษณะการทำงานเช่นเดียวกันกับฟังก์ชัน `range()` ใน for loop

```
>>> word = 'Hello Python'
>>> word[2:9:2]
'loPt'

# negative steps
>>> word[len(word)-1::-1]        # or word[-1::-1]
'nohtyP olleH'

>>> word[len(word)::-1]          # or word[::-1]
'nohtyP olleH'
```





# Immutability

- Python strings *cannot* be changed — they are immutable
  - ไม่สามารถเปลี่ยนแปลงค่าของ String ที่มีอยู่แล้วได้

```
>>> word[0] = 'J'
    ...
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
    ...
TypeError: 'str' object does not support item assignment
```

- ถ้าต้องการ String ที่ต่างจากเดิมให้สร้าง String ใหม่แทน

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```





# Immutability

- อย่างไรก็ตาม หากต้องการเปลี่ยนแปลง String แล้ว Assign ไปที่ Variable ตัวเดิม *ไม่ควร*ใช้เครื่องหมาย +

```
>>> a = 'Py'
>>> b = 'thon'
>>> a += b                  # NO
```

```
>>> a = 'Py'
>>> b = 'thon'
>>> a = a + b               # NO
```

- เนื่องจากจะทำให้เกิดปัญหาใน Python Interpreter อื่น ๆ ที่ไม่ใช่ CPython
  - ในกรณีนี้ให้ใช้ *String Method* `str.join()`

```
>>> a = "".join([a, b])
>>> a
'Python'
```

```
>>> a = "***".join([a, b])
>>> a
'Py***thon'
```

# Traversal with a `while` Loop

- ในการเขียนโปรแกรม ในหลาย ๆ กรณี เราจำเป็นต้องเข้าถึงอักขระใน **String** ทีละตัว **(Traversal)**
- หนึ่งในวิธีที่เป็นไปได้คือการใช้ `while` **loop**

```
02 fruit = "banana"
03 index = 0
04
05 while index < len(fruit):
06     letter = fruit[index]
07     print(letter)
08     index = index + 1
```





# Traversal with a `for` Loop

- `for` **loop with indexes:**

```
11 s = "abcd"
12 length = len(s)
13 for i in range(length):
14     print(i, s[i])
15
```

- `for` **loop without indexes**

```
17 s = "abcd"
18 for c in s:        # similar to for i in range(n)
19     print(c)
```

  - เราเรียก **Data Type** ที่สามารถเข้าถึงแต่ละหน่วยย่อย (เช่น 1 ตัวอักษร) ได้ครั้งละ 1 หน่วย (เช่น `range` หรือ `string`) ว่า **Iterable**





# Searching

- พิจารณาการทำงานของฟังก์ชันดังต่อไปนี้

```
01 def find(word, letter):
02     index = 0
03     while index < len(word):
04         if word[index] == letter:
05             return index
06         index = index + 1
07     return -1
```

- ลักษณะการทำงานตรงกันข้ามกับเครื่องหมาย `[]`
  - รับอักขระ *`letter`* มาแล้วหา **Index** ใน **String** *`word`*
  - หากอักขระ *`letter`* ไม่ปรากฏใน *`word`* จะคืนค่า **-1**





# Exercise 1

- **Exercise 1:** ปรับแก้ฟังก์ชัน `find()` ให้มี **parameter** ตัวที่ 3 เพื่อระบุว่าควรเริ่ม **search** จาก **index** ตำแหน่งไหน

# Example: Looping and counting

- ฟังก์ชันด้านล่างนับจำนวนครั้งที่อักขระ *key* ปรากฏใน String *word*

```
02 def count_letter(word, key):
03     count = 0
04
05     for letter in word:
06         if letter == key:
07             count = count + 1
08     print(count)
09
10 count_letter('banana', 'a')
```





# The in Operator

- Operator `in` เป็น Boolean Operator ที่รับ Operands เป็น String 2 ตัว แล้ว return `True` ถ้า String แรก เป็น ***Substring*** ของ String ที่สอง

```
>>> 'a' in 'banana'
True
>>> 'z' not in 'banana'
True
>>> 'seed' in 'banana'
False
```





# Example: in_both()

- ฟังก์ชันด้านล่างแสดงผลอักขระที่ซ้ำใน String *word1* และ *word2*

```
01 def in_both(word1, word2):
02     for letter in word1:
03         if letter in word2:
04             print(letter)
```

เมื่อเปรียบเทียบ String `'apple'` และ `'orange'`

```
>>> in_both('apples', 'oranges')
a
e
s
```





# String Comparison

- เครื่องหมาย Relational Operator (`==`, `<`, `>`, `!=`, `<=`, `>=`) ใช้กับ String ได้
  - เครื่องหมาย `==` และ `!=` ใช้เพื่อเปรียบเทียบ String ทั้งสองว่าเหมือนหรือต่างกัน
  - เครื่องหมายอื่น ๆ ใช้เปรียบเทียบ String ตามลำดับอักษร (Alphabetical Order)

```
>>> 'bat' <= 'cat'
True
>>> 'rat' < 'cat'
False
>>> 'apple' < 'Apple'                # A comes before a
False
```

# String-related Built-in Functions

- **bin(*x*)**
  - เปลี่ยนเลขจำนวนเต็ม *x* เป็น **Binary String**

```
>>> bin(3)
'0b11'
```

- **chr(*i*)**
  - คืนค่า **String** แทนอักขระที่มีรหัส **Unicode** เป็นจำนวนเต็ม *i*

```
>>> chr(97)
'a'
```

- **eval(*expression*[, *globals*[, *locals*]])**
  - คืนค่าผลลัพธ์ของการ **evaluate String** *expression*

```
>>> x = 1
>>> print(eval('x + 1'))
2
```

https://docs.python.org/3/library/functions.html





# String-related Built-in Functions [2]

- **hex(*x*)**
  - เปลี่ยนเลขจำนวนเต็ม *x* เป็น **Hexadecimal String**

```
>>> hex(18)
'0x12'
```

- **oct(*x*)**
  - เปลี่ยนเลขจำนวนเต็ม *x* เป็น **Octal String**

```
>>> oct(9)
'0o11'
```

- **ord(*c*)**
  - คืนค่ารหัส **Unicode** ของ **String** ความยาวหนึ่งอักขระ *c*

```
>>> ord('a')
97
```

https://docs.python.org/3/library/functions.html





# String-related Built-in Functions [3]

- **str(*object*="")**
  - เปลี่ยน **object** ให้เป็น **String** ที่เหมาะกับการแสดงผล

```
>>> str(18)
'18'
>>> str(0x35)
'53'
>>> str(None)
'None'
>>> str(print)
'<built-in function print>'
>>> str(hello)        # user written function hello()
'<function hello at 0x03390C90>'
```

https://docs.python.org/3/library/functions.html





# String Constants

```
>>> import string
>>> string.digits
'0123456789'
```

| | |
|---|---|
| string.ascii_letters | 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' |
| string.ascii_lowercase | 'abcdefghijklmnopqrstuvwxyz' |
| string.ascii_uppercase | 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' |
| string.digits | '0123456789' |
| string.hexdigits | '0123456789abcdefABCDEF' |
| string.octdigits | '01234567' |
| string.punctuation | '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{\|}~' |
| string.printable | digits + letters + punctuation + whitespace |
| string.whitespace | space + tab + linefeed + return + formfeed + vertical tab (on most systems) |

https://docs.python.org/3/library/string.html

# Basic String Methods

- **Method มีลักษณะคล้ายฟังก์ชัน**
  - รับค่า **Argument** และมีการคืนค่าผลลัพธ์
  - แต่ **Syntax** การเรียกใช้ต่างจากฟังก์ชัน
- **ตัวอย่างเช่น Method `upper()` รับค่า String แล้ว Return String ใหม่ที่เป็นตัวพิมพ์ใหญ่ทั้งหมด (Uppercase)**
  - แทนที่จะเรียกใช้ด้วย **Syntax** `upper(word)`
  - **Syntax ที่ถูกต้องของ String Method คือ** `word.upper()`

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print(new_word)
BANANA
```





# Basic String Methods [2]

- `str.count(sub[, start[, end]])`
  - นับจำนวนครั้ง **(non-overlapping) ที่ Substring** *sub* ปรากฏใน *str* โดยสามารถใช้ **Optional Parameter** *start* และ *end* เพื่อระบุช่วง **index ที่ค้นหาโดยตีความในลักษณะเดียวกันกับ slicing**

```
>>> 'banana'.count('na')
2
```

```
>>> 'aaaaa'.count('aa')
2
```

- `str.endswith(suffix[, start[, end]])`
  - คืนค่า **True** ถ้า *str* ลงท้ายด้วย *suffix*

```
>>> 'Quadruple'.endswith('uple')
True
```

- `str.find(sub[, start[, end]])`
  - **คืนค่า index แรกที่พบ Substring** *sub* **ใน** *str* **และ -1 หากไม่พบ**





# Basic String Methods [3]

- `str.isalpha()`
  - คืนค่า **True** ก็ต่อเมื่อไม่ใช่ **String** ว่างและอักขระทุกตัวเป็นตัวอักษร **(Alphabetic)**
- `str.isdigit()`
  - คืนค่า **True** ก็ต่อเมื่อไม่ใช่ **String** ว่างและอักขระทุกตัวเป็นตัวเลข **(Numeric)**
- `str.islower()`
  - คืนค่า **True** ก็ต่อเมื่อมีอักษรชนิดที่มีแยกตัวพิมพ์เล็ก-ใหญ่ **(Cased Characters)** อย่างน้อย **1** ตัว และทุกตัวเป็นตัวพิมพ์เล็ก **(Lowercase)**





# Basic String Methods [4]

- `str.isspace()`
  - คืนค่า **True** ก็ต่อเมื่อไม่ใช่ **String** ว่างและอักขระทุกตัวเป็นอักขระ **Whitespace**
- `str.isupper()`
  - คืนค่า **True** ก็ต่อเมื่อมี **Cased Character** อย่างน้อย **1** ตัว และทุกตัวเป็นตัวพิมพ์ใหญ่ **(Uppercase)**
- `str.replace(old, new[, count])`
  - สร้าง **String** ใหม่จาก *str* โดยแทนที่ **Substring** *old* ด้วย **Substring** *new* โดยสามารถใช้ **Optional Parameter** *count* เพื่อระบุจำนวนครั้งที่ทำการแทนที่

https://docs.python.org/3/library/stdtypes.html#string-methods

# Basic String Methods [5]

- *str*.**split**([*sep*[, *maxsplit*]])
  - สร้าง **List** ของ **String** ย่อยที่เกิดจากการตัด *str* ด้วย **String** *sep* **(Separator)** โดยสามารถใช้ **Optional Parameter** *maxsplit* เพื่อจำกัดจำนวนครั้งที่ทำการตัด

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2 3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

**Note**: ถ้าไม่ระบุ *sep* หรือ *sep* มีค่า **None** การตัดจะถือว่าอักขระ **whitespace** ที่ติดกันทั้งหมด เป็น **Separator** ตัวเดียว

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '   1  2 3   '.split()
['1', '2', '3']
```





# Basic String Methods [6]

- *str*.**splitlines**(*keepend*=False)
  - สร้าง **List** ของ **String** ย่อยที่เกิดจากแยกบรรทัด *str*

```
01 s = """
02 This is a sample
03 multi-line
04 string
05 """
06
07 print("Lines with splitlines():")
08 for line in s.splitlines():
09     print(" line:", line)
10
11 print("==================")
12
13 print("Lines with splitLines(True):")
14
15 for line in s.splitlines(True):
16     print(" line:", line)
17
```

```
>>>
Lines with splitlines():
 line:
 line: This is a sample
 line: multi-line
 line: string
==================
Lines with splitLines(True):
 line:

 line: This is a sample

 line: multi-line

 line: string
```

https://docs.python.org/3/library/stdtypes.html#string-methods





# Basic String Methods [7]

- *str*.**startswith**(*prefix*[, *start*[, *end*]])
  - คืนค่า **True** ถ้า *str* ขึ้นต้นด้วย *prefix*
- *str*.**strip**([*chars*])
  - สร้าง **String** ใหม่จาก *str* โดยลบ อักขระทุกตัวใน **String** *chars* ออกจากตำแหน่งหัวและท้ายของ *str* **(ถ้ามี)**
  - ถ้าไม่ระบุ *chars* หรือ *chars* เป็น **None** จะทำการลบอักขระ **whitespace** ที่ตำแหน่งหัวและท้ายของ *str* แทน

```
>>> 'www.example.com'.strip('cmowz.')
'example'
>>> '   spacious   '.strip() # if char is omitted
'spacious'
```

**Note**: ยังมี **Method** *str*.**lstrip()** และ *str*.**rstrip()** ที่ทำงานในลักษณะเดียวกันโดย **lstrip()** จะลบเฉพาะอักขระทางด้านซ้าย และ **rstrip()** จะลบเฉพาะทางขวาเท่านั้น

https://docs.python.org/3/library/stdtypes.html#string-methods





# Basic String Method [8]

- *str*.**upper()**
  - สร้าง **String** ใหม่จาก *str* โดยเปลี่ยน **Case Character** ทุกตัวให้เป็นตัวพิมพ์ใหญ่
- *str*.**lower()**
  - สร้าง **String** ใหม่จาก *str* โดยเปลี่ยน **Case Character** ทุกตัวให้เป็นตัวพิมพ์เล็ก

https://docs.python.org/3/library/stdtypes.html#string-methods

# References


- https://docs.python.org/3/tutorial/introduction.html#strings
- https://docs.python.org/3/reference/lexical_analysis.html#literals
- https://docs.python.org/3/library/functions.html#built-in-funcs
- https://docs.python.org/3/library/string.html
- https://docs.python.org/3/library/stdtypes.html#string-methods
- http://www.greenteapress.com/thinkpython/html/thinkpython003.html#toc19
- http://www.greenteapress.com/thinkpython/html/thinkpython009.html
- http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-strings.html