Title: ตำราวิชา Problem Solving and Computer Programming (PSCP) - PSCP Book

Author: รศ.ดร. โชติพัชร์ ภรณวลัย

Language: th-TH

Date: 10 กรกฎาคม 2567

Ref: Think Python 2nd edition: Chapter 8

Chapter 5: String

Index

String คือข้อมูลแบบ Sequence ชนิดหนึ่ง เป็นข้อมูลแบบมีลำดับของตัวอักษร โดยสามารถล้อมรอบชุดตัวอักษรเหล่านั้นได้ด้วย single quote หรือ double quote ก็ได้ เช่น 'banana' หรือ "banana"

ลำดับในที่นี้หมายความว่า ลำดับของตัวอักษรใน String เช่น คำว่า 'banana' ตัวอักษร 'b' มาก่อนตัวอักษร 'a'

เราสามารถเข้าถึงตัวอักษรแต่ละตัวใน string นั้นได้ โดยใช้ index ซึ่งจะเริ่มจากหมายเลข 0 เสมอ เช่น ดังรูปตัวอย่างด้านล่าง การอ้าง index จะใช้ bracket ล้อมรอบค่า index เช่น `fruit[0]` หมายถึง index 0 ของ ตัวแปร fruit

```
>>> fruit = 'banana'
>>> fruit[0]
'b'
>>> fruit[1]
'a'
>>> fruit[2]
'n'
>>> fruit[3]
'a'
>>> fruit[4]
'n'
>>> fruit[5]
'a'
>>> fruit[6]
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    fruit[6]
IndexError: string index out of range
>>> fruit[-1]
'a'
>>> fruit[-2]
'n'
>>> fruit[-3]
'a'
>>> fruit[-4]
'n'
>>> fruit[-5]
'a'
>>> fruit[-6]
'b'
>>> fruit[-7]
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    fruit[-7]
IndexError: string index out of range
>>>
```

โดยค่า index 0 จะหมายถึงตัวอักษรตัวแรก ได้แก่ 'b' และ index 1 หมายถึงตัวอักษรตัวที่ 2 ได้แก่ 'a' ไปเรื่อยๆ จนถึง index 6 จะได้ Error ที่เรียกว่า IndexError เนื่องจากตัวอักษรตัวสุดท้ายคือ 'a' มี index 5 คำว่า 'banana' มีขนาด 6 ตัวอักษร แต่ index จะมีค่าตั้งแต่ 0 ถึง 5

ค่า index สามารถเป็นค่าลบได้ โดยค่า -1 จะหมายถึงตัวอักษรตัวสุดท้าย และ -2 จะหมายถึงตัวรองสุดท้าย ในกรณีตัวแปร fruit นี้ ตัวอักษรตัวแรกจะมีค่า index เป็น -6 และ ค่า index เป็น -7 จะได้ Error ที่เรียกว่า IndexError

ค่า index เป็นจำนวนเต็มได้เท่านั้น ดังนั้น `fruit[1.5]` จะได้ Error ที่เรียกว่า `TypeError` กลับมา

```
>>> fruit[1.5]
Traceback (most recent call last):
  File "<pyshell#188>", line 1, in <module>
    fruit[1.5]
TypeError: string indices must be integers
>>>
```

len Function and in Operator

มี builtin function ชื่อ `len` ที่เราสามารถใช้ในการตรวจสอบขนาดของ string ได้ ในตัวอย่างด้านบน `len(fruit)` จะได้ค่า 6 เนื่องจากตัวแปร `fruit` เก็บค่า `banana` ซึ่งมีขนาด 6 ตัวอักษร อย่างไรก็ตามโปรดระวังว่า index จะเริ่มจาก 0 ดังนั้นหากต้องการเข้าถึงตัวอักษรตัวสุดท้ายจะต้องลบขนาดของ string นั้นด้วย 1 ก่อน เช่น `fruit[len(fruit)-1]` จะได้ผลลัพธ์เป็น 'a' ซึ่งเป็นตัวอักษรตัวสุดท้าย

```
>>> fruit = 'banana'
>>> len(fruit)
6
>>> fruit[len(fruit)]
Traceback (most recent call last):
  File "<pyshell#184>", line 1, in <module>
    fruit[len(fruit)]
IndexError: string index out of range
>>> fruit[len(fruit)-1]
'a'
>>>
```

เราสามารถใช้ตรวจสอบว่า substring หรือส่วนของ string เป็นส่วนของ string หรือไม่ ด้วย `in` operator เช่น

```
'a'
>>> 'a' in 'banana'
True
>>> 'na' in 'banana'
True
>>> 'nan' in 'banana'
True
>>> 'nab' in 'banana'
False
>>> 'c' in 'banana'
False
```

Traveral with a for Loop

เราสามารถเข้าถึงตัวอักษรแต่ละตัวตามลำดับ โดยใช้ **for** loop ได้

วิธีแรกคือใช้ `range` function

```
>>> for i in range(len(fruit)):
        print(fruit[i])


b
a
n
a
n
a
>>>
```

วิธีที่สอง ไม่จำเป็นต้องใช้ `range` หรือใช้ index และจะเขียนง่ายกว่า

```
>>> for ch in 'banana':
        print(ch)


b
a
n
a
n
a
```

Traveral with a while Loop

สามารถใช้ while loop ได้เช่นเดียวกัน แต่จะมีแค่วิธีเดียวคือการใช้ index ร่วมด้วย

ดังตัวอย่างในรูปด้านล่าง จะมีการสร้าง index i ไว้ก่อนเข้า Loop while โดยค่า i นี้จะทำให้เงื่อนไขใน while เป็นจริง เพื่อเข้าไปใน while และจะมีการ update ค่า i ใน Loop while จนกระทั่งเงื่อนไขใน while เป็นเท็จ จึงจะออกจาก Loop while

```
>>> i = 0
>>> while i < len(fruit):
        print(fruit[i])
        i = i + 1


b
a
n
a
n
a
>>>
```

String Slicing

เราสามารถ slice เหมือนเป็นการหั่น string ออกเป็นส่วนย่อยๆ เรียกว่า substring ได้ โดยการอ้างอิง index ที่มีทั้งหมด 3 ค่า ได้แก่ start stop และ step ลักษณะเดียวกันกับ range function

กรณีต้องการใช้ index 2 ค่า จะได้แก่ค่า start และ stop ลักษณะการเขียนจะเป็น [start: stop] โดยค่า step จะมีค่าเป็น 1

ถ้าหากไม่กำหนดค่า start ไว้ จะมีค่า default ของ start เป็น 0 และหากไม่ได้กำหนดค่าของ stop ไว้ จะมีค่า default เป็นขนาดของ str นั้นๆ ดังนั้นเราจะเห็นได้ว่า fruit และ fruit[0:6] และ fruit[:6] และ fruit[0:] และ fruit[:] มีค่าเท่ากันทั้งหมด ถ้าหากค่า start มีค่ามากกว่าหรือเท่ากับ stop ก็จะได้ค่า str ว่าง หรือ empty string คืนกลับมา ซึ่งเขียนได้เป็น ' '

```
>>> fruit
'banana'
>>> fruit[0:6]
'banana'
>>> fruit[:6]
'banana'
>>> fruit[0:]
'banana'
>>> fruit[:]
'banana'
>>> fruit[2:6]
'nana'
>>> fruit[2:]
'nana'
>>> fruit[3:3]
''
>>>
```

เวลาทำ string slice ด้วยค่า start และ ค่า stop ทั้ง ค่า start และ ค่า stop สามารถมีค่าเกิดขอบเขตของค่า index ที่เป็นไปได้ ยกตัวอย่างเช่น ค่า index ที่เป็นไปได้ของคำว่า 'banana' คือ -6 ถึง 5 แต่กรณีทำ string slice ค่า start สามารถน้อยกว่า -6 ได้ และค่า stop สามารถมีค่ามากกว่า 5 ได้ โดยจะไม่เกิด Error

```
>>> fruit
'banana'
>>> fruit[-10:]
'banana'
>>> fruit[:10]
'banana'
>>> fruit[2:10]
'nana'
>>> fruit[-10:3]
'ban'
>>>
```

กรณีมี index 3 ค่า จะได้แก่ค่า start และ stop และ step โดยหลักการจะคล้ายๆกับ function `range` แต่จะเขียนด้วยรูปแบบ [start:stop:step]

ดังตัวอย่างด้านล่างนี้ `fruit` จะมีค่าเท่ากับ `fruit[:]` และ `fruit[::1]` เนื่องจากค่า default ของ step คือ 1 เมื่อค่า step มากกว่า 1 เช่นเมื่อ step = 2 ก็จะกระโดดข้ามตัวอักษร 1 ตัว คำว่า `fruit[::2]` จะมีค่าเท่ากับ `'bnn'` ข้ามตัวอักษร a มา 3 ครั้ง และเมื่อค่า step = 3 ก็จะข้ามตัวอักษร 2 ตัวได้แก่ 'an' และ 'na' เป็นเช่นนี้ไปเรื่อยๆ

แต่ค่า `step` ไม่สามารถเป็น 0 ได้ แต่เป็นค่าติดลบได้ กรณี step = -1 จะเป็นย้อนตัวอักษรกลับ สังเกตว่า `fruit[::-1] มีค่าเป็น 'ananab' ซึ่งก็คือ banana' อ่านจากขวามาซ้าย

```
>>> fruit
'banana'
>>> fruit[::]
'banana'
>>> fruit[::1]
'banana'
>>> fruit[::2]
'bnn'
>>> fruit[::3]
'ba'
>>> fruit[::4]
'bn'
>>> fruit[::5]
'ba'
>>> fruit[::6]
'b'
>>> fruit[::7]
'b'
>>> fruit[::8]
'b'
>>> fruit[::0]
Traceback (most recent call last):
  File "<pyshell#115>", line 1, in <module>
    fruit[::0]
ValueError: slice step cannot be zero
>>> fruit[::-1]
'ananab'
>>>
```

ข้อความที่เป็น palindrome คือข้อความที่เขียนจากมาซ้ายหรือขวามาซ้ายจะได้ข้อความเดียวกัน เช่นคำว่า 'refer' เขียนจากซ้ายมาขวา หรือขวามาซ้ายก็เป็นข้อความเดียวกัน ดังนั้นเราสามารถตรวจสอบได้ว่าข้อความใดเป็น palindrome หรือไม่ โดยการใช้ step = -1 เพื่อให้ดึงตัวอักษรจากด้านขวามาด้านซ้าย ดังตัวอย่างด้านล่าง

```
>>> s = 'abcd'
>>> s[::-1]
'dcba'
>>> if s == s[::-1]:
        print(s, "is palindrome")


>>> s = 'refer'
>>> s[::-1]
'refer'
>>> if s == s[::-1]:
        print(s, "is palindrome")


refer is palindrome
>>>
```

String is Immutable Type

หมายความว่า string ไม่สามารถเปลี่ยนค่าข้อมูลได้ ยกตัวอย่างเช่น

```
>>> fruit
'banana'
>>> fruit[0]
'b'
>>> fruit[0] = 'c'
Traceback (most recent call last):
  File "<pyshell#120>", line 1, in <module>
    fruit[0] = 'c'
TypeError: 'str' object does not support item assignment
>>> new_fruit = 'c' + fruit[1:]
>>> new_fruit
'canana'
>>>
```

จากรูปค่า `fruit[0]` คือ 'b' หากเราต้องการเปลี่ยนค่า 'b' เป็น 'c' เพื่อจะสร้าง string ว่า `cnanana` จะไม่สามารถทำได้ เนื่องจากเป็นการแก้ไข string เดิม หากต้องการ string 'canana' จะต้องสร้าง string ใหม่จาก string เดิม หรือกำหนดค่า string ใหม่ที่ต้องการให้กับตัวแปรเดิม

```
>>> fruit = 'canana'
>>> fruit
'canana'
>>>
```

ในตัวอย่างด้านบนนี้เป็นการกำหนดให้ 'fruit' เก็บค่าใหม่เลยคือ 'canana' และค่าเดิม 'banana' จะไม่สามารถอ้างถึงได้อีก และจะถูกจัดการลบออกจากหน่วยความจำภายหลังโดยอัตโนมัติ

Builtin Function for String

เราสามารถแปลงค่าตัวอักษร string ไปเป็นตัวเลขตามรหัส Ascii ได้ ด้วย function `ord()` และกลับกัน สามารถแปลงตัวเลขเป็นตัวอักษรได้ด้วย function `chr()`

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharithms.com

ดังตัวอย่างในรูปด้านล่าง ค่า `ord('a')` จะมีค่า 97 ซึ่งตรงกับค่า DEC หรือ Decimal ในตาราง Ascii ในรูปด้านบน

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(97)
'a'
>>>
```

เมื่อมีการ compare string กัน จะทำการเปรียบเทียบตามตาราง Ascii นี้ ยกตัวอย่างเช่น

```
>>> 'a' > 'A'
True
>>>
```

จะเห็น 'a' > 'A' เป็นจริง เพราะ 'a' มีค่า 97 และ 'A' มีค่า 65

String Methods

method มีลักษณะที่คล้าย function แต่ว่าจะผูกติดกับ object นั้น ยกตัวอย่างเช่นสมมุติว่าเรามี object ในที่นี้คือตัวแปรเพื่อเก็บ string ข้อความ 'banana' เราสามารถเรียกใช้ method ต่างๆที่มากับ string เช่น lower, upper เป็นต้น ดังตัวอย่างในรูปด้านล่าง

```
>>> fruit
'banana'
>>> fruit.upper()
'BANANA'
>>> fruit
'banana'
>>> fruit2 = fruit.upper()
>>> fruit2
'BANANA'
>>> fruit
'banana'
>>> fruit2.lower()
'banana'
>>>
```

String is immutable

หากเราใช้ upper() method จะทำการสร้าง string ใหม่ ที่เป็นตัวพิมพ์ใหญ่ทั้งหมด

เนื่องจากว่า string เป็น immutable type ดังนั้น การเปลี่ยนแปลงใดๆของ string ที่เกิดจากการใช้ method เช่นการเปลี่ยนตัวพิมพ์เล็กเป็นตัวพิมพ์ใหญ่ จะเป็นการสร้าง string ใหม่เสมอ ไม่ใช่เป็นการเปลี่ยนแปลง string เดิม

ดังนั้นหากต้องการเก็บค่า 'BANANA' ไว้ จำเป็นจะต้องมีตัวแปรอีกตัว ในที่นี้ชื่อ fruit2 มาเก็บค่า

นอกจาก upper() ยังมี lower() ที่ใช้ในการสร้าง string ตัวพิมพ์เล็กจาก string ข้อความเดิมได้

เราสามารถดูว่า method อะไรบ้างที่สามารถใช้กับ string ได้ โดยการเรียก `help(str)`

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
```

หรือหาดูจาก Internet เช่นที่ [w3cschool.com] ที่สรุปไว้ให้ครบ และสามารถดูตัวอย่างและทดลองได้ใช้งานได้ด้วย

อย่างไรก็ตาม อย่าลืมว่าตอนสอบ จะสามารถใช้ help(str) ได้เท่านั้น ดังนั้นควรศึกษาและอ่าน help ให้เป็นด้วย