SI 507 Lab #3

September 13

New Group Office Hours

Wednesdays 9:30 AM - 11:00 AM

Today's Plan

Lab #3

- Warm-Up Exercise
  - https://bit.ly/3QI8Q2N
- Topic 1: Map/Reduce/Filter
- Topic 2: List & Dictionary Comprehensions

ASCII Standard Character Set

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

Source: www.LookupTables.com

Map/ Reduce/ Filter

Lab #3

`map()`, `reduce()`, and `filter()` allow you to write simpler, shorter code, without necessarily needing to bother about intricacies like loops and branching.

- `map()` and `filter()` come with Python
- `reduce()` needs to be imported from `functools` module

Map

Syntax: `map(func, *iterables)`

- `func` is a function on which each element in `iterables` may be applied on
- `*` means there can be as many iterables as possible, where `func` has that exact number as required input arguments

Map Example

Say we have a list of names, all in lowercase, and we need them in uppercase.

```
name_list = ['emily', 'jeremy', 'keaton', 'harrison']
```

Without `map()`, we could do something like this:

```
4   upper_names = []
5
6   for name in name_list:
7       cap_name = name.upper()
8       upper_names.append(cap_name)
9
10  print(upper_names)
11
```

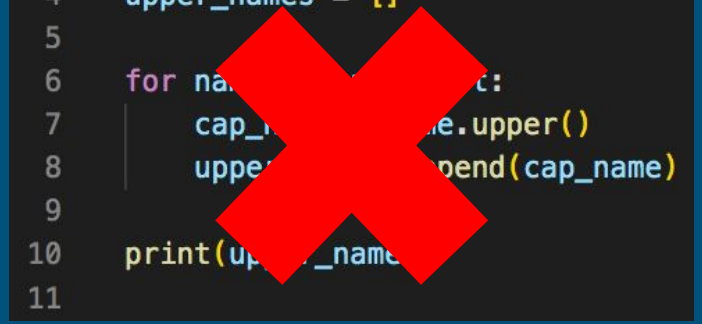
Map Example

Say we have a list of names, all in lowercase, and we need them in uppercase.

```
name_list = ['emily', 'jeremy', 'keaton', 'harrison']
```

With `map()`, we could do something like this:

```
4   upper_names = []
5
6   for na          t:
7       cap_        e.upper()
8       uppe        pend(cap_name)
9
10  print(up        _name
11
```



```
upper_names = list(map(str.upper, name_list))
```

turns the map object into a list

creates a map object

...and make it uppercase

take each string in this list/iterable...

Filter

Syntax: `filter(func, iterable)`

- The `func` argument is required to return a boolean value
- Only one iterable is required
- `filter` passes each element in the iterable through `func` and returns only the ones that evaluate to true.

Filter Example

Say we have a list of test scores on an AP Calculus exam. Let's filter those who passed with scores of more than 75.

```
2   calc_scores = [55, 78, 90, 93, 82, 43, 77, 74]
3
4   def over_75(score):
5       return score > 75
6
7   no_retakes = list(filter(over_75, calc_scores))
8
9   print(no_retakes)
```

returns truth value of this statement

Reduce

Syntax: `reduce(func, iterable[,initial])`

- `reduce` applies a function of **two arguments** cumulatively to the elements of an iterable, optionally starting with an initial argument
- `func` is the function on which each element in the `iterable` gets cumulatively applied to
  - `func` requires two arguments
- `initial` is the optional value that gets placed before the elements of the iterable in the calculation, and serves as a default when the iterable is empty
  - If `initial` is supplied, it becomes the first argument to `func` and the first element in `iterable` becomes the second element

Reduce Example

```
2   from functools import reduce
3
4   numbers = [3, 4, 6, 9, 34, 12]
5
6   def custom_sum(first, second):
7       return first + second
8
9   result = reduce(custom_sum, numbers)
10  print(result)
```

1. `reduce` takes the first and second element in `numbers` and passes them to `custom_sum`
2. `custom_sum` computes their sum and then returns it to `reduce`
3. `reduce` takes the result and applies it as the first element to `custom_sum` and takes the next element (third) in `numbers` as the second element to `custom_sum`
4. The process repeats until `numbers` is exhausted

Reduce Example

Make a prediction:

What happens if we pass 10 as the optional initial parameter?

```
2   from functools import reduce
3
4   numbers = [3, 4, 6, 9, 34, 12]
5
6   def custom_sum(first, second):
7       return first + second
8
9   result = reduce(custom_sum, numbers, 10)
10  print(result)
```

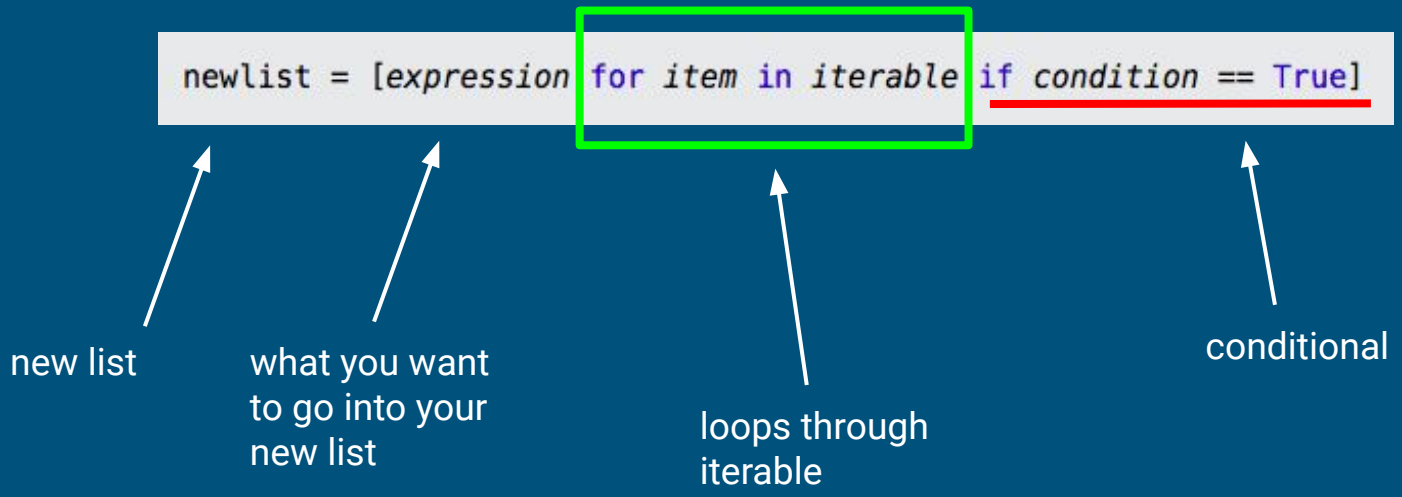
List & Dictionary Comprehensions

Python List Comprehension

List comprehension is a method for transforming one list into another list.

During this transformation, items within the original list can be conditionally included in the new list and each item can be transformed as needed.

List Comprehension Syntax



Python Dictionaries

A collection of items accessed by key, rather than by index.

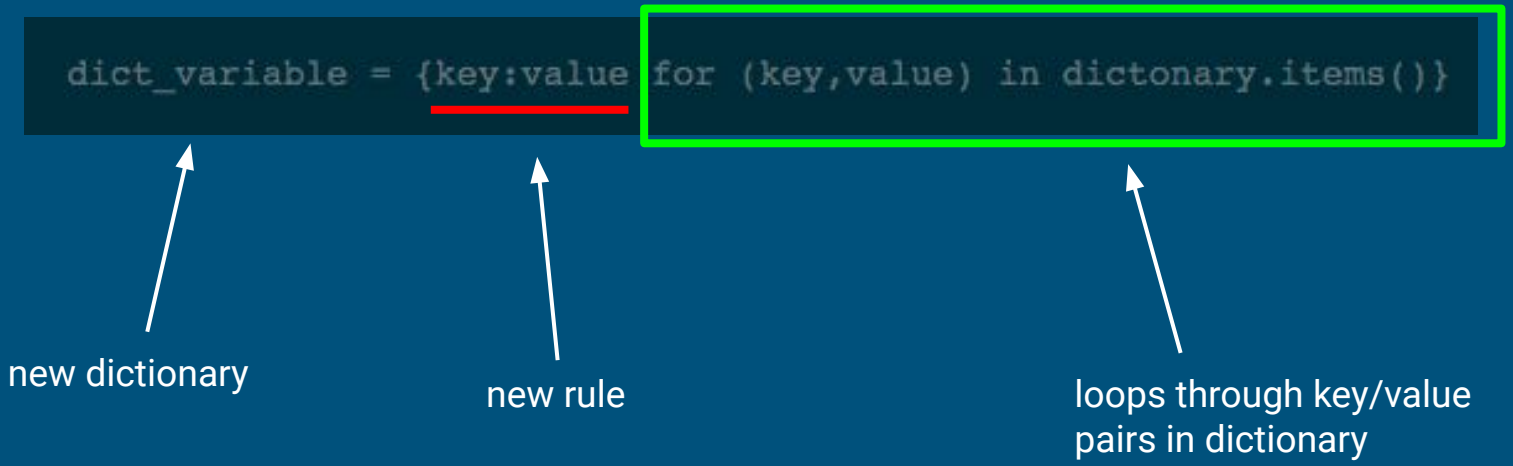
```
a = {'apple': 'fruit', 'beetroot': 'vegetable', 'cake': 'dessert'}
a['doughnut'] = 'snack'
print(a['apple'])
fruit
print(a[0])
KeyError                                  Traceback (most recent call last)

<ipython-input-9-00d4a978143a> in <module>()
----> 1 print(a[0])

KeyError: 0
```

Python Dictionary Comprehension

Dictionary comprehension is a method for transforming one dictionary into another dictionary.

During this transformation, items within the original dictionary can be conditionally included in the new dictionary and each item can be transformed as needed.

Dictionary Comprehension Syntax



Comprehension Exercises

List comprehension exercise

- **https://bit.ly/3Lqq43T**

Dictionary comprehension exercise

- **https://bit.ly/3eAIVfZ**

Homework

No graded assignment this week!

This week we will work on grading HW1.

Sources

https://cs.smu.ca/~porter/csc/ref/ascii.html

https://www.learnpython.org/en/Map%2C_Filter%2C_Reduce

https://www.w3schools.com/python/python_lists_comprehension.asp

https://www.datacamp.com/community/tutorials/python-dictionary-comprehension