Loops

what it means for an object to be iterable?

An **iter*able*** is anything you're able to loop over.

Examples:

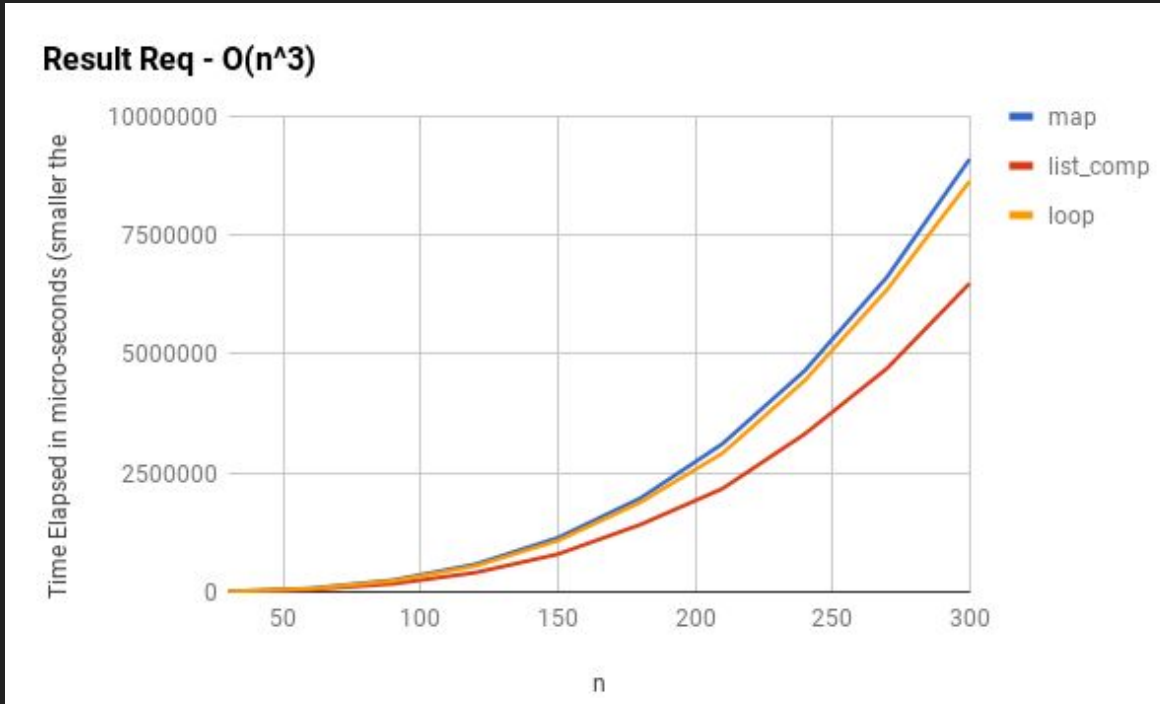
Iterable are string, list, tuple, dict, set, and frozenset types.

```
file = open("somefile.txt")
next(file)
for lines in file …
```

What is an iterator?

An ***iterator*** is the object that does the actual iterating.

If an object is iterable, it can be passed to the built-in Python function iter(), which returns something called an iterator.
Iterators help make more more memory-efficient code.


Result Req - O(n^3)
Time Elapsed in micro-seconds (smaller the
10000000
7500000
5000000
2500000
0
50
100
150
200
250
300
n
map
list_comp
loop

```
summation = 0
for i in range(100000000):
    summation = summation + i
```

each steps:

1. Dereference summation (find the place in memory that is associated with the variable summation and read the data there)
2. Read that data to figure out the type of summation (in this case, it's an integer).
3. Dereference i
4. Read that data to figure out the type of i (in this case, it's an integer).
5. Lookup how it should interpret + given that summation is an integer and i is an integer.
    - Remember: if summation and i were strings, it wouldn't be doing addition, it would be concatenating! And because integers and floating point numbers are different things from the perspective of a computer, if i were an integer and summation were a floating point number, + would actually require (a) converting i to a floating point number, then (b) adding the floating point version of i to summation.
6. Compile low-level binary code (referred to as "machine code") that can be read by the computer's processor to tell it to execute an addition of two integers AND checks for integer overflows.
7. Run that code.

List comprehensions perform better here because they don’t need to load the append attribute off of the list and call it as a function.

Instead, in a comprehension, generate a specialial LIST_APPEND bytecode to append onto the result list.

This is the for loop

```
list_a = [1, 2, 3, 4]
list_b = [2, 3, 4, 5]

common_num = []

for a in list_a:
  for b in list_b:
          if a == b:
          common_num.append(a)
```

List comprehension

```
list_a = [1, 2, 3, 4]
list_b = [2, 3, 4, 5]

common_num = [
for value1 in list_a
       for value2 in list_b
           if value1 == value2
            common_num.append(a)
           ]
```

```
1.  array([ 0,        1,        2, ..., 999997, 999998, 999999])
2.  %%timeit
3.  b = numbers.copy()
4.  for i in range(len(b)):
        b[i] = numbers[i] * 2
```

357 ms ± 10.3 ms per loop

```
1.  import numpy as np
2.  numbers = np.arange(1000000)
3.  numbers
4.  %%timeit
5.  b = numbers * 2
```

1.09 ms ± 40.6 µs per loop

Encoding and decoding Base64

1. Take the ASCII value of each character in the string
2. Calculate the 8-bit binary equivalent of the ASCII values
3. Convert the 8-bit chunks into chunks of 6 bits by simply re-grouping the digits (Base64 characters only represent 6 bits of data) mind the padding.
4. Convert the 6-bit binary groups to their respective Base64 decimal values.Using a base64 encoding table, assign the respective base64 character for each decimal value.
5. convert these decimals into the appropriate Base64 character using the Base64 conversion table

1. 80, 121, 116, 104, 111, 110
2. '01010000', '01111001', '01110100', '01101000', '01101111', '011101110'
3. 101000 011110 011110 100110 100011 011111 101110
4. 20 7 37 52 26 6 61 46
5. UHl0aG9u

1. ord()
2. `bin(ord(x))[2:]`
3. `grouping.join(tuple(binary)) split every 6 (+ padding if needed)`
4. `Find the integer value of the eg:` 101110 and compare to the table

Base64 Encoding Table

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

how do you convert 8-bit binary numbers into their ASCII characters?

References

- https://www.kdnuggets.com/2019/06/speeding-up-python-code-numpy.html
- https://www.practicaldatascience.org/html/performance_understanding.htm
- https://treyhunner.com/2016/12/python-iterator-protocol-how-for-loops-work/l
- https://treyhunner.com/2018/06/how-to-make-an-iterator-in-python/
- https://www.practicaldatascience.org/html/performance_understanding.html
- https://github.com/treyhunner/lazy-looping
- https://www.youtube.com/watch?v=q8czHnhcVGE