Listiness

Or, things that behave like lists

Univ.AI

Python puts great stock in the idea of having **protocols** or mechanisms of behavior, and identifying cases in which this behavior is common.

One of the most important ideas is that of things that behave like a *list of items*.

These include, well, lists, but also strings and files.

And many other data structure in Python are made to behave like lists as well, so that their contents
might be iterated through, in addition to their own native behavior.

We shall see all of these soon.

Lists

Python is 0-indexed. This means that the first index is 0 not 1.

Create a lazy sequence of numbers from 0 to 9 (Why lazy?):

```
seq = range(10)
```

Create a list:

```
seq = range(10)
lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
lst = list(seq) # same as above
```

Add to a list:

```
lst.append(10)
print(lst)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Get the third element above(value 2):

```
lst[2] # the 2 inside is called an index.
```

List Indexing and Ops

```
lst = ['hi', 7, 'c'].
```

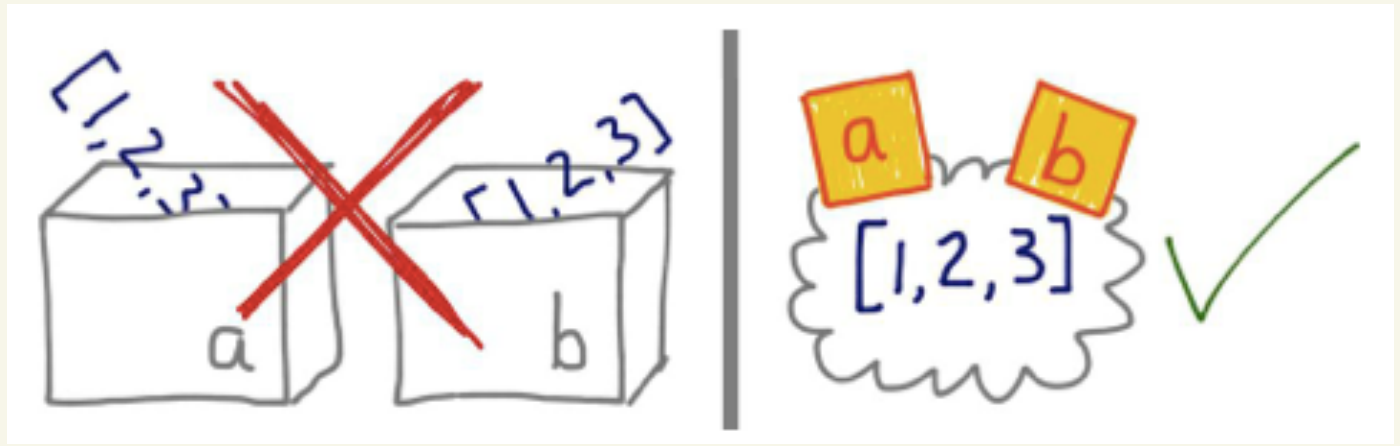
Univ.AI

Variables as labels, again

```
In [1]: a = [1, 2, 3]

In [2]: b = a

In [3]: print(a)
[1, 2, 3]

In [4]: print(b)
[1, 2, 3]

In [5]: b[1] = 5

In [6]: print(a)
[1, 5, 3]

In [7]: print(b)
[1, 5, 3]
```

Values are stored in memory locations. Variables are **pointers**, or **labels** of this location. The function `id` tells us this location.

(this image is from Fluent Python)



```
In [8]: id(a)
Out[8]: 4571342600

In [9]: id(a) == id(b)
Out[9]: True
```

Comprehensions and conditionals

```
nums = [1, 4, 7, 9, 12]
doubles = [2*element for element in nums] # simple comprehension
print(doubles)
```

Output: `[2, 8, 14, 18, 24]`

Loop with conditional

```
evens = []
for num in nums:
    if num%2 == 0:
        evens.append(num)
print(evens)
```

Comprehension

```
evens = [e for e in nums if e%2 == 0]
print(evens)
```

Output (in both cases): `[4, 12]`

Strings: immutable collection of characters

```python
mystring, substr, newstr, sep  = 'Hi world', 'world', 'World', ' '
```

Iteration over a list or string

```
lst, mystring = ['hi', 7, 'c', 2.2], 'Hi !'
```

```
for element in lst:
    print(element)
```

Output:

```
hi
7
c
2.2
```

```
for character in mystring:
    print(character)
```

Output:

```
H
i

!
```

Files

```python
fd = open("data/Julius Caesar.txt")
counter = 0
```

First open the file.

```python
for line in fd:
    if counter < 10: # print first 10 lines
        print("<<", line, ">>")
    else:
        break # break out of for loop
    counter = counter + 1 # also writeable as counter += 1
```

Now treat the open file object like a "list" and just iterate over it, getting one line at a time. Here we read 10 lines only to save memory.

```python
fd.close()
```

Finally close the file.

File Methods

Python Code	What it does
`fhandle = open('filetoread')`	Opens file `filetoread` for reading
`thetext = fhandle.read()`	Read all the contents of the file
`thelines = fhandle.readlines()`	Read each line (with its newline character `\n`) into a list item
`fhandle2 = open('filetowrite', "w")`	Opens file for writing
`fhandle3 = open('filetowrite', "a")`	Opens file for appending to the end.
`fhandle2.write(thetext)`	Write `thetext` into a file opened for writing
`fhandle2.writelines(thelines)`	Write each string from a list into a file
`fhandle.close()`	Close file.

Read a file, parse lines, and get all words

```
# make a list with all words in documents
# the words can occur more than once
wordlist = []
fd = open("filename")
lines = fd.readlines()
fd.close()
# strip newline characters and other whitespace off the edges
cleaned_lines = [line.strip() for line in lines]
# make a list of lists.
# each inner list if the list of words on that line
list_of_lines_words = [line.split() for line in lines]
# Take each list of words, and get all the words
for lines_words in list_of_lines_words:
    wordlist = wordlist + lines_words # update the wordlist using the new list.
print(wordlist)
```