Python Notes:

(List, dictionary, tuple, set)

—----------

List:

1. Use [ ] to define list or sequence of objects.
2. You can have objects of any type in a list like:
   letters = [“a”, “b”, “c”]
   sample = [1, “b”, 10, “some letters”, [“g”, 8], true]

3. By using start (*) in a list, you can repeat items in a list. i.e.
   numbers = [0] * 3 // result would be [0, 0, 0]

4. It is possible to combine multiple lists by using +
   combined = numbers + letters
   print(combined) // [0, 0, 0, “a”, “b”, “c”]

5. One other way to create list in python is by using **list** function.
   Note: list function can accept iterable.
   Iterable means anything you can loop over, like **range** function.
   e.g.
   numbers = list(range(8)) // [1, 2, 3, 4, 5, 6, 7]
   chars = list(“Hello World”) // [‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘ ’, ‘W’, ‘o’, ‘r’, ‘l’, ‘d’]

6. To find number of items in a list, use **len** function.
   numbers = [1, 3, 5]
   print(len(numbers)) // 3

Accessing Items in a List

7. By using bracket and index number of an element in a list
   numbers = [1, 7, 10, 2]
   print(numbers[0]) // 1
   print(numbers[1]) // 7
   print(numbers[-1]) // 2
   print(numbers[-2]) // 10

8. For changing specific element in a list:
   numbers[0] = “A”
   print(numbers) // [“A”, 7, 10, 2]

Slicing a list

9. To slice a list, we can use two indices:
   numbers = [1, 7, 10, 2]
   print(numbers[0:3]) // [1, 7,10]

   note: if first index is omitted, then it means 0 index
   print(numbers[:3]) // [1, 7, 10]

   note: Similarly, if you don't include the end index, then by default the length of list is used.
   print(numbers[:]) // [1, 7, 10, 2]

10. In list slicing, we can you third index to specify step length. For example:
    To select every second element in the list:
    sample_numbers = list(range(11)) // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print(sample_numbers [::2]) // [0, 2, 4, 6, 8, 10]

11. Little trick to have a list in reverse order. Just use -1 as step size.
    sample_numbers = list(range(11)) // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print(sample_numbers [::-1]) // [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

List unpacking

12. It is possible to assign each element of a list like:
    numbers = [1, 2, 3]
    first = numbers[0] // 1
    second = numbers[1] // 2
    third = numbers[2] // 3

    Instead you can use list unpaking technique:
    first, second, third = numbers

    print(first) // 1
    print(second) //2
    print(third) //3

    note: the number of variables on the left side must be equal to number of list elements, otherwise we have error.

note: if we want only first and second items in the list?

```
numbers = [1, 2, 3, 4, 5, 6]
first, second, *others = numbers

print(first)        // 1
print(second)    // 2
print(others)     // [3, 4, 5, 6]
```

note: if we want only first and last items in the list?

```
numbers = [1, 2, 3, 4, 5, 6]
first, *others, last = numbers

print(first)        // 1
print(last)         // 6
print(others)     // [2, 3,  4,  5]
```

note: if any of items is not so important, then use underscore for that item.

```
numbers = [1, 2, 3]
first, _ , last = numbers

print(first)        // 1
print(last)         // 3
```

Iterating over list

13. For iterating over a list, use for loop:

```
numbers = [10, 20, 30]
for number in numbers:
    print(number)
```

14. If index of each item is needed in the loop, then use **enumerate** function:

```
numbers = [10, 20, 30]
for number in enumerate(numbers):
    print(number)   // (0,10)  (1,20)  (2,30)
```

note: in above code, you can unpack the tuple like:

```
numbers = [10, 20, 30]
for index, item in enumerate(numbers):
    print(index, item)   // 0  10      1   20        2   30
```

Add/Remove items in a list

15. To add an item at the end of list, use **append** method.

```
numbers = [1,7, 10]
numbers.append(11)
print(numbers)   // [1, 7, 10, 11]
```

16. To add item at specific index, use **insert** method.

```
numbers = [1,7, 10]
numbers.inset(2, 9)
print(numbers)   // [1, 7, 9, 10]
```

17. To remove an item from end of list, use **pop** method.

```
numbers = [1,7, 10]
numbers.pop()
print(numbers)   // [1, 7]
```

18. To remove an item at specific index, use **pop** method with target index.

```
numbers = [1,7, 10]
numbers.pop(1)
print(numbers)   // [1, 10]
```

19. To remove specific item based on its value,use **remove** method.

```
numbers = [1,7, 10]
numbers.remove(7)
print(numbers)   // [1, 10]
```

20. Another way to remove an item by its index is to use **del** function.

```
numbers = [1,7, 10]
del   numbers[0]
print(numbers)   // [7, 10]
```

21. To remove range of items based on index, use **del** with desired index range.

```
numbers = [1,7, 10, 12, 15]
del   numbers[0:3]
print(numbers)   // [12, 15]
```

side note: everything in python is object.

Finding Items

22. For finding an index of target object in a list, use **index** method.

```
letters = [ 'a',  'b',  'c',  'd' ]
print(letters .index('a'))        // 0
print(letters .index('b'))        // 1
```

If given object does not exist in the list, then using **letters.index** throws an error !
So first of all check if given object exists in the list and then call **letters.index** on it.

```
letters = [ 'a',  'b',  'c',  'd' ]
If 'a'  in  letters:
    print(letters.index('a'))
```

23. To count number of occurances of an item, use **count** method.

```
letters =  ['a',  'b', 'c', 'a', 'c', 'a']
letters.count('a')   // 3
```

Sorting Lists

More info on sorting: [Link]

24. For sorting items of a list, we can use **sort** method.
    - sort method, modifies the original list.

```
numbers = [35, 1, 10, 67]
numbers.sort()
print(numbers)   // [1, 10, 35, 67]
```

25. For sorting items in reverse order:

```
numbers = [35, 1, 10, 67]
numbers.sort(reverse=True)
print(numbers)   // [67, 35, 10, 1]
```

26. Python has built-in function called **sorted**.
    - Sort function, does not modify original list but it returns new sorted list.

```
numbers = [35, 1, 10, 67]
print(sorted(numbers))    // [1, 10, 35, 67]
```

27. **Sorted** function can reverse the order of items.

```
numbers = [35, 1, 10, 67]
print(sorted(numbers, reverse=True))    // [67, 35, 10, 1]
```

28. If list has complex items, how can sort this list. Something like:

```
items = [
        (“product1” , 10),
        (“product2”, 21),
        (“product3” , 3)
    ]
```

In this situations, we should define a function to tell python how to sort the list.

```
def  sort_item(item):
     return  item[1]

items.sort(key=sort_item)
print(items)
```

Another way: using lambda function for specifying target key for sorting

```
items = [
        (“product1” , 10),
        (“product2”, 21),
        (“product3” , 3)
    ]
items.sort(key=lambda item: item[1])
print(items)
```

note: sorted function can apply on any iterable(like: dictionary, set, tuple, list, strings)

29. For filtering items of a list, use **filter** function which is built-in function in python.
30. More notes which should be learn about lists in python is:
    a. Built-in map function
    b. Built-in zip function
    c. List comprehension

Stack:

1. Stack = LIFO: Last in - First out

Queue:

2. queue = FIFO: First in - First out

Tuple:

Tuple is a read-only list.
In tuple we can not add a new item, remove the existing item or change the items of the tuple.

1. All possible ways to define tuple:
   points = (1, 2)
   points = 1, 2 → when defining tuple we could excluce the paranthesis
   points = 1, → must have trailing comma, otherwise this interpret as integer
   points = () → empty tuple

note: to check the type of any variable in python, use **type** function like: **print(type(points))**

2. For concatinating two tuples, use +
   points = (1, 2) + (3, 4)
   print(points) // (1, 2, 3, 4)

3. To repeat a tuple, use *
   points = (1, 2) * 3
   print(points) // (1, 2, 1, 2, 1, 2)

4. We could convert any iterable to tuple by using **tuple** function.
   points = tuple([1, 2])
   print(points) // (1, 2)
   print(type(points)) // tuple

5. Like lists, to access an item in a tuple we use:
   points = (1, 2, 3)
   Points[0] // 1

6. To slice, just like list:
   print(points[0:1])

7. To unpack a tuple:
   x, y, z = points

8. To check if an item exists in a tuple:
   If 10 in points:
   print("exists")

Arrays:

If you are dealing with large sequence of items(ten thousands or more) and you encounter performance problems, we have a more efficient data type in python called **array**.

from array import array
numbers = array("i", [1,2,3]) → **"i" is type code**
numbers.append(5)
numbers.pop()
numbers.insert(3, 19)

note: eventy item in the array should have typecode which is specified as first argument of array function.

Set:

A collection with unique values(without duplications).
Set is defined by {}

```
numbers = [1, 1, 2, 2, 3]
uniques = set(numbers)    // [1,2,3]
second = {1, 4}
second.add(5)
second.remove(5)
len(second)  // 2
```

1. Union of multiple sets:
   ```
   first = set([1, 1, 2, 2, 3])
   second = {1, 5}
   print(first | second)   // {1, 2, 3, 5}
   ```

2. Intersection of two sets:
   ```
   first = set([1, 1, 2, 2, 3])
   second = {1, 5}
   print( first  &  second)   // {1}
   ```

3. Find all elements which exists in first set and not in second set:
   ```
   first = set([1, 1, 2, 2, 3])
   second = {1, 5}
   print( first  -  second)   // {2,3}
   ```

4. the symmetric difference of two sets is a set that contains elements that are in either of the sets but not in their intersection. In other words, it includes all the elements that are unique to each of the sets.
   ```
   set1 = {1, 2, 3, 4}
   set2 = {3, 4, 5, 6}

   sym_diff = set1 ^ set2
   print(sym_diff)     // Output: {1, 2, 5, 6}
   ```

   note: set does not support indexing. Means items in set are not in order.

note: sets are **unordered collections of unique elements.** Because sets are unordered, they do not support indexing, slicing, or other sequence-like behavior. You cannot directly access an item in a set by its position. However, you can check for the presence of an item, iterate through the items, and convert the set to a list if you need to access items by index.

```
my_set = {1, 2, 3, 4, 5}
print(3 in my_set)   // Output: True
print(6 in my_set)   // Output: False
```

Dictionary:

Dictionaries are collection of key-value pairs.

1. Two ways to defined dictionary in python:
   - **points = { 'x':1, 'y':2 }**
   - **points = dict(x=1, y=2)**

2. Accessing an item in dictionary:
   print(points['x'])

3. Add new item to dictionary:
   points['x'] = 10
   points['z'] = 20

4. To access an item in a dictionary there is two way:

   Way1: first check if item exists in dictionary and then access to it:
   ```
   If  'a'  in  points:
       print(points['a'])
   ```

   Way2: using get method:
   print(points.get('a'))

   note: when using **get** method, if item does not exists in dictionary, then the Output will be **None.**

5. We can define default value for an item when does not exists in dictionary:
   print(points.get('a', 0))

6. For deleting an item from dictionary, use del function:
   del points['x']

7. How iterate over a dictionary:
   for key in points:
   print(key, points[key])

   And another way:
   for item in points:
   print(item) // tuple of key-values

   for key,value in points:
   print(key, value)

8. We can use comprehension for list, set and dictionary. i.e.
   values = {}
   for x in range(5):
   values[x] = x

   Instead of above code, we can use dictionary comprehension:
   values = { x: x * 2 for x in range(5) }

   note: this code creates a generator object:
   values = (x * 2 for x in range(5))
   print(values) // generator object

Unpacking Operator:

With this operator, we can unpack any iterable
numbers = [1, 2, 3, 4]
print(*numbers) // 1 2 3 4

Example:
values = list(range(5))
Or
values = [*range(5)]

values = [*range(5), *"Hello"]

Example:

```
first = [1, 2]
second = [2,3,4]
result = [*first,  *second, *”something”]
```

Example:

```
first = {‘x’: 1}
second = {‘y’: 2}
result = {**first, **second, *’something”, ‘z’:1}
```