

ICS-104

Chapters

6 and 8

Notes

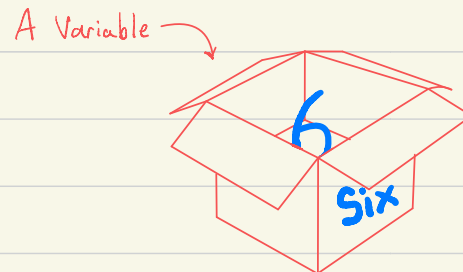
By: Naif Alqahtani

 Twitter: @NaifAlqahtani

 Youtube.com/MWNLLT

• Lists:

- Recall in my chapter 2 notes, I used the following illustration to represent a variable:



- List are essentially a bunch of variable boxes right next each other and packed together

List: →



- each box inside a list is called an **element**. Element are exactly like variables.
- This is how you can create a list in python to store multiple variables:

```
numbers = [8, "ten", 3, -5, "Zero", 4.6]
```

- To access each variable individually we use brackets.

```
print(numbers[0]) → 8
```

```
print(numbers[4]) → Zero
```

```
print(numbers[5]) → 4.6
```

- Remember: Counting Starts From 0
- Our list has a length of 6. If you try to index using any integer outside the range ($0 \leq n \leq 5$) you will get an **error**.
- You can use the **len()** function to find the length of a List.

Scan to try code



How to use: Last Page

• Traversing List Elements:

- Recall that in strings, there were two ways to traverse all elements in a string [see ch4 Notes p:4,5]
- The same exact methods work with strings.
 1. loop i through a range of numbers, use i to index list
 2. Use "in" with a list name to loop through elements.

Scan to try code



How to use: Last Page



```
numbers = [8, "ten", 3, -5, "Zero", 4.6]
```

```
# method one:
```

```
for i in range(len(numbers)):
```

```
    print(numbers[i])
```

→ notice the index

```
print('-----')
```

```
# method two:
```

```
for i in numbers:
```

```
    print(i)
```

→ i is already set to element

```
# both print the same result
```

List Operations:

- Use `.append(element)` to add element to end of List
- Use `.insert(index, element)` to add element at specified index
- Use `.index(element)` to find the index value of element.
- Use `.pop(index)` to remove element at given index. Default is to remove last element
- Use `.remove(element)` to remove element by value.
- Use `+` between two Lists to concatenate them together.
- Use `*` to concatenate the same list multiple times.
- You can compare two Lists using `==` or `!=`.
- Comparing two Lists is `True` when both lists contain the same exact elements in the correct order.
- Here is a common mistake when copying a List

```
List = [3, 4, 5]  
Copy = List
```

← This does NOT copy the List. This only references the List with another Name. This means that Both Copy and List are linked together.

- Instead, to copy a List, we use the `list()` function.

```
Copy = list(List)
```

← This copies all elements into a new and separate List named Copy. This is how you copy a List.

• List Algorithms:

- To swap two values there are two main ways.



```
List = [1,2,3]
```

#Swapping first and last element to get [3,2,1]

#Method 1

#Simultaneous swapping

```
List[0], List[2] = List[2], List[0] ← Fast, Clean.
```

*Not part of the syllabus

#Method 2

#Using temp variable

```
temp = List[0] ← Store value of List[0]
```

```
List[0] = List[2] ← Replace List[0] with the value in List[2]
```

```
List[2] = temp ← Replace List[2] with the value previously saved.
```

- Linear Search a List:



```
List = [2, 2019, "Naif", 1442, 0.45, "&"]
```

```
pos = 0 # position counter
```

```
found = False # flag if found
```

```
for i in List: # loop through elements in list
```

```
    if i == "Naif": # check if current element is equal to "Naif"
```

```
        found = True # set flag to True
```

```
        break # stop searching if found
```

```
    pos += 1 # if not "Naif", then increment the pos counter
```

```
if found:
```

```
    print("Found at position %d" %pos) # print its position
```

```
else:
```

```
    print("Not found") # print not found
```

• Tuples:

- Tuples, unlike lists, cannot be modified
- Tuples can be used to store constant variables.
- Tuples are often used with round brackets().
- Tuples are separate by commas, just like lists.
- Round brackets are optional. You can omit them.
- Tuples are used to return more than 1 variable in functions.

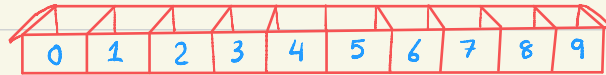
(1, 2, 3) ← Both are valid tuples
1, 2, 3 ←

• Sets:

- Sets store unique values. No two values can be the same in the same set.
- Sets are not ordered.
- You cannot access set values by indexing (No positions)
- Sets resemble mathematical sets. They both have the same operations.
- Sets are faster at doing math operations because they do not have ordered values.
- If two values are the same in a set, python will automatically remove duplicates.
- Sets use curly brackets {} or set()
- Using set() can convert a list into a set.
- Similar to lists and unlike tuples, set can be modified.
- This means sets are mutable. You can add & remove.
- To remove an element from sets you can .discard() or .remove()
 - .discard(): no error if element does not exist.
 - .remove(): raises an error if element is not in the set.

• Dictionaries:

→ Remember this drawing earlier, that represents Lists?



- If you wanted to access a value in the list you would index the list at the **number** position of the list.

eg. `List[3]` ← returns value in box 3.

- Dictionaries allow you to Label and give a name to each element.



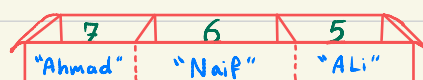
- This means that instead of using numbers to index, you can use the name that you gave that position.

eg. `Dictionary["age"]` ← returns value in "age" box

- Box labels are called Keys.
- Box values are called values.
- You cannot repeat a **Key** in the same dictionary.
- Dictionaries may be called a **map**.
- You use curly brackets `{}`.
- You use a colon `:` to separate **keys** and **values**.
- You use commas `,` to separate multiple **key:value** pairs.
- Colons `:` is what decides if it is a **set** or a **dictionary**.
- Empty curly bracket `{}` is a dictionary.
- **key:value** pairs are not ordered.
- Example:

`ID = {"Ahmad": 7, "Naif": 6, "Ali": 5}`

`ID["Ali"] → 5`



ID dictionary

Dictionary Operations:

- Calling a dictionary with a key that does not exist will raise an error.
- You can use `.get(key)` to get values from a dictionary key
- You can add a second argument `.get(key, default value)`, this will return default value if the provided key is not in the dictionary.

- To modify a value:

`ID["Ahmad"] = 9` → `{"Ahmad": 9, "Naif": 6, "Ali": 5}`

- To add a new key:value pair.

`ID["Khalid"] = 2` → `{"Ahmad": 9, "Naif": 6, "Ali": 5, "Khalid": 2}`

- To remove a key:value pair, use `.pop(key)`

- `.pop()` returns the value of the key removed after removing it.

- `.pop()` raises an error if key does not exist.

- Traversing/looping dictionaries:

→ to loop through keys: `for key in dictionary:`

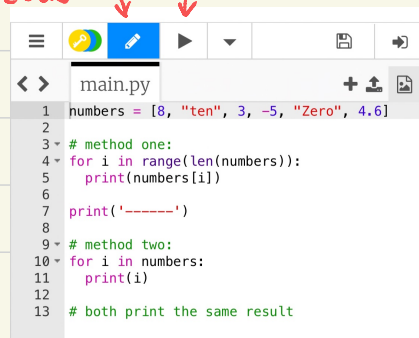
→ to loop through values: `for value in dictionary.values():`

How to use barcodes:

- I added barcodes to programs I mentioned in these notes to help make the notes more interactive & so you can try yourselves.
- I do not know if they will work as intended. (hopefully, they do!)
- I also don't know for how long will they work for (hopefully, for many years)

① To modify code

② To view output or run code



```

1 numbers = [8, "ten", 3, -5, "Zero", 4.6]
2
3 # method one:
4 for i in range(len(numbers)):
5     print(numbers[i])
6
7 print('-----')
8
9 # method two:
10 for i in numbers:
11     print(i)
12
13 # both print the same result
    
```

③ To reset code

④ Download

