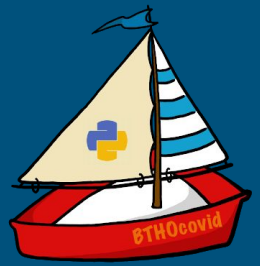



# ShiP.py

## Learn to Py while Shelter-in-Place

## L4B: Dictionaries and Sets



A volunteering educational initiative during COVID-19

# ShiP Crew

JD

Teddy

Chinmay

Pratik

Siddharth

Umang

Waseem

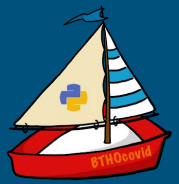
A volunteering educational initiative during COVID-19

# Topics

## PHASE I: Foundations

**All times are in CDT (GMT-5)**

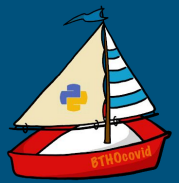
| | | |
|---|---|---|
| 1. | Variables, Expressions, Simple I/O | Sat, April 18 (11 am-12 noon) |
| 2. | Boolean Decisions (branching) | Wed, April 22 (9 pm-10 pm) |
| 3. | Repetitions (loops) | Sat, April 25 (11 am-12 noon) |
| 4A. | Collective Data Structures (Lists and Tuples) | Wed, April 29 (9 pm-10 pm) |
| 4B. | CDS (Dictionaries and Sets) | Sat, May 02 (11 am-12 noon) |
| 5. | Functions | Wed, May 06 (9 pm-10 pm) |
| 6. | File I/O | Sat, May 09 (11 am-12 noon) |



# Lecture 4B

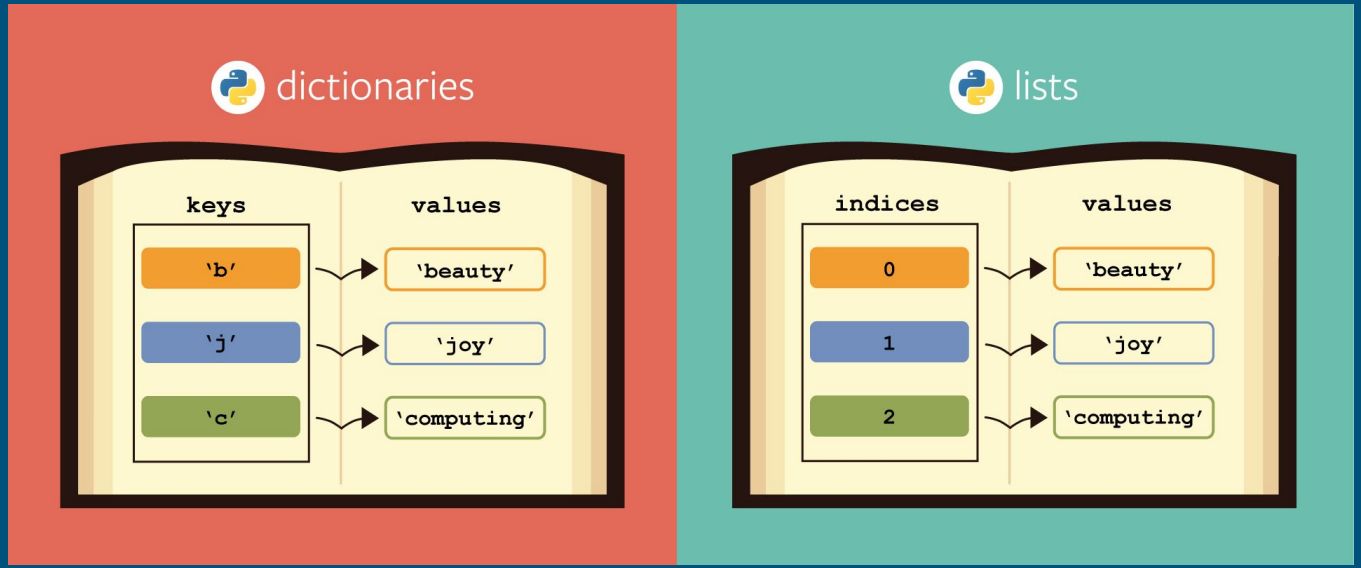
## AGENDA

- Dictionaries
- Sets
- Membership operators
- Revisiting Iterables
- Comprehension

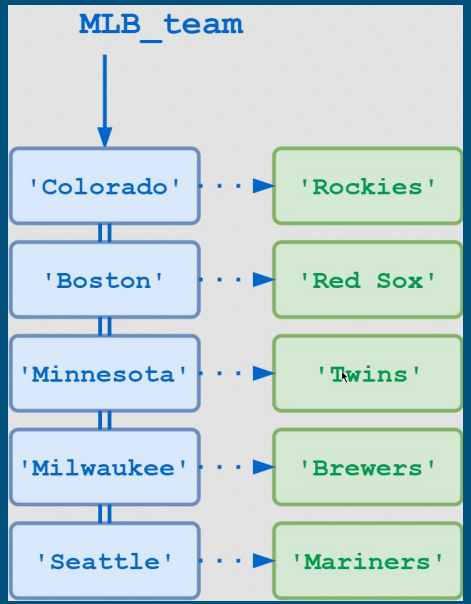


# Dictionaries

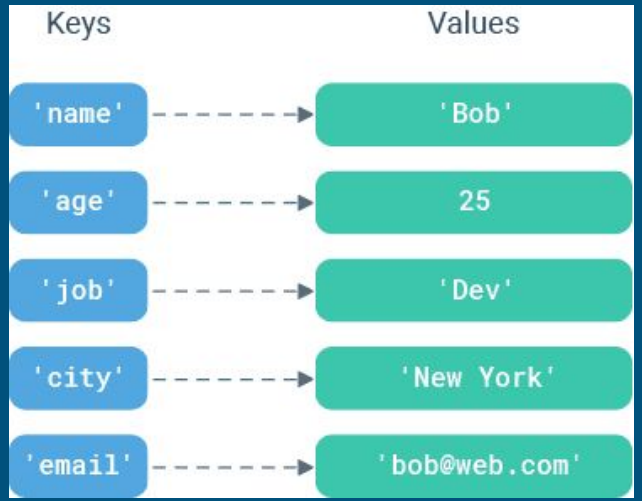
A dictionary consists of a collection of **key (unique) - value** pairs



# Dictionaries: Examples




source




source

value 'Rockies' is mapped to 'Colorado' key
value 'Red Sox' is mapped to 'Boston' key

value 'Bob" is mapped to 'name' key
value 'New York' is mapped to 'city' key

# Syntax and accessing elements

Dictionary created using {} curly brackets, each key-value pair separated by a comma

```
#sets are written using {} curly brackets
new_dict = {'Tamu': 'Aggies', 'Ut': 'Longhorns', 'Georgia': 'Bulldogs', 'Clemson': 'Tigers'}
type(new_dict)
```

```
dict
```

**Key** **Value**

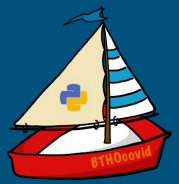
Values are accessed using keys (unlike numeric indexing 0,1,.. in lists)

```
new_dict['Tamu']
```

```
'Aggies'
```

```
new_dict['Clemson']
```

```
'Tigers'
```



# Examples

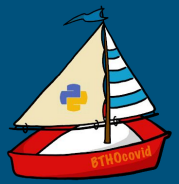
```
state_abbr = {'Texas':'TX', 'New York':'NY', 'California':'CA', 'Arizona':'AZ', 'Florida':'FL'}
state_abbr
```

```
{'Arizona': 'AZ',
 'California': 'CA',
 'Florida': 'FL',
 'New York': 'NY',
 'Texas': 'TX'}
```

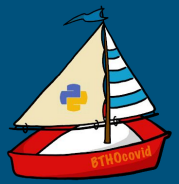
Creating a dictionary using `dict( )` function

```
cost_dict = dict(flour=1.59, sugar=0.99, salt=0.50, pepper=1.00)
cost_dict
```

```
{'flour': 1.59, 'pepper': 1.0, 'salt': 0.5, 'sugar': 0.99}
```



| | List | Dictionary |
|---|---|---|
| **Usage** | - List is a container of data which is referred with a numeric index.<br>- Useful when the order of the data items is important.<br>- Can store heterogeneous data but mostly used for data of same type. | - Dictionary map key to values.<br>- Each key is unique and the order of data is not important.<br>- They are versatile and powerful enough to handle heterogeneous data types. |
| **Variable Declaration** | - Declare using Square brackets [ ]<br>month = [‘Jan’, ‘Feb’, ’Mar’] | - Declare using curly brackets { }<br>capitals = {'Italy':'Rome', 'Germany':'Berlin'} |
| **Element Reference** | - Each element is referred with a numeric index starting from zero.<br>month [1] = ‘Feb’ | - Each element is referred with a unique key.<br>capitals['Italy'] = 'Rome' |
| **Recalling elements** | Print month [1] | Print capitals['Italy'] |
| **Adding elements** | A- Add items to the last.<br>i. month.append('Apr')<br>ii. month.extend('Apr')<br>iii. month + ['June']<br>B- Add to selected position<br>month.insert(1, ‘Jul’) | Order isn’t important<br>capitals['Pak'] = 'Islamabad' |
| **Replacing elements** | month [0] = 'Jan' | capitals['Italy'] = 'Venice' |
| **Delete elements** | A- Delete by index<br>del month[1]<br>B- Delete by value<br>month.remove(‘Jun’)<br>Deletion shall cause reordering of elements. | del capitals['Italy']<br>- No reordering. |
| **Sorting** | month.sort() | - Not applicable |
| **Slicing** | month [Start:End] – Specific elements<br>month[:] – All elements<br>month[-1] – Last Element | - Not applicable |



# Sets

A set is collection of unique objects in an unordered fashion

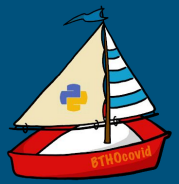
Duplicate elements are not allowed

You can add more items - given it is not already present in a set

Once a set is created, you cannot change its items but you can add items

```
#sets are written using {} curly brackets
new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
type(new_set)
```

```
set
```



# Examples

```
new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
print(new_set)
new_set.add('aggies')
print(new_set)
```

```
{'c', 'howdy', 13.2, 14, 'a', 'game', 25}
{'c', 'howdy', 13.2, 14, 'a', 'game', 25, 'aggies'}
```

```
mySet = set('Apple')
print(mySet)
myNewSet = set('Howdy')
print(myNewSet)
```

```
{'A', 'p', 'l', 'e'}
{'o', 'w', 'H', 'd', 'y'}
```

```
# clear method
new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
new_set.clear()
print(new_set)
```

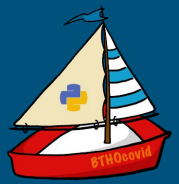
```
set()
```

```
# discard method
new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
new_set.discard('game')
print(new_set)
```

```
{'c', 'howdy', 13.2, 14, 'a', 25}
```

```
# del method
new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
del(new_set)
print(new_set)
```

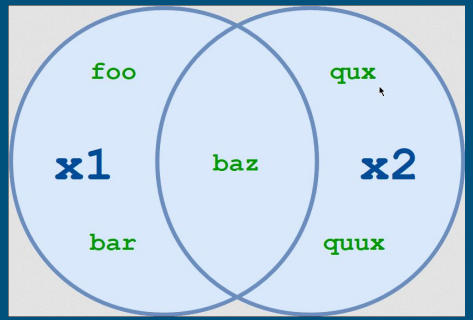
```
---------------------------------------------------------------------
NameError                                 Traceback (most recent
<ipython-input-7-47b88d90fc9a> in <module>()
      1 new_set = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
      2 del(new_set)
```



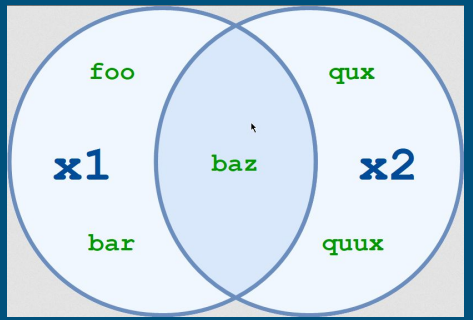
# Set operations

Works similar to set operations in mathematics

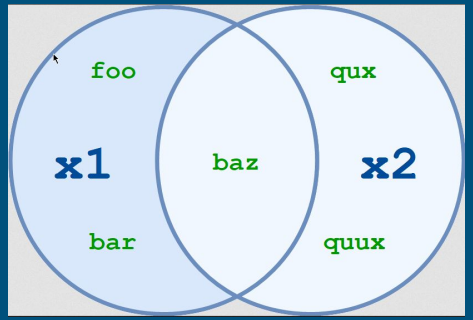
Union




Intersection




Difference




# Example of set operations

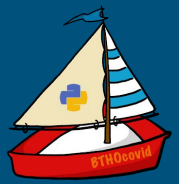
```
set1 = {'a', 'c', 'game', 14, 25, 13.20, 'howdy'}
set2 = {'d', 'f', 14, 'game', True, 'football'}

#union
print('Union set:', set1.union(set2))

#intersection
print('Intersection set:', set1.intersection(set2))

#difference
print('Difference set:', set1.difference(set2))
```

```
Union set: {'c', True, 'howdy', 13.2, 14, 'a', 'football', 'd', 'game', 25, 'f'}
Intersection set: {14, 'game'}
Difference set: {'c', 'howdy', 13.2, 'a', 25}
```



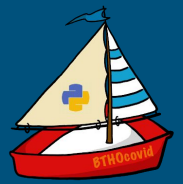

For a full list of set operations and methods

https://docs.python.org/3.8/library/stdtypes.html#set

# List Vs Set Vs Dictionary Vs Tuple

| Lists | Sets | Dictionaries | Tuples |
|---|---|---|---|
| List = [10, 12, 15] | Set = {1, 23, 34}<br>Print(set) -> {1, 23,24}<br>Set = {1, 1}<br>print(set) -> {1} | Dict = {"Ram": 26, "mary": 24} | Words = ("spam", "egss")<br>Or<br>Words = "spam", "eggs" |
| Access: print(list[0]) | Print(set).<br>Set elements can't be indexed. | print(dict["ram"]) | Print(words[0]) |
| Can contains duplicate elements | Can't contain duplicate elements. Faster compared to Lists | Can't contain duplicate keys, but can contain duplicate values | Can contains duplicate elements. Faster compared to Lists |
| List[0] = 100 | set.add(7) | Dict["Ram"] = 27 | Words[0] = "care" ->TypeError |
| Mutable | Mutable | Mutable | Immutable - Values can't be changed once assigned |
| List = [] | Set = set() | Dict = {} | Words = () |
| Slicing can be done<br>print(list[1:2]) -> [12] | Slicing: Not done. | Slicing: Not done | Slicing can also be done on tuples |
| Usage:<br>Use lists if you have a collection of data that doesn't need random access.<br>Use lists when you need a simple, iterable collection that is modified frequently. | Usage:<br>- Membership testing and the elimination of duplicate entries.<br>- when you need uniqueness for the elements. | Usage:<br>- When you need a logical association b/w key:value pair.<br>- when you need fast lookup for your data, based on a custom key.<br>- when your data is being constantly modified. | Usage:<br>Use tuples when your data cannot change.<br>A tuple is used in comibnation with a dictionary, for example, a tuple might represent a key, because its immutable. |





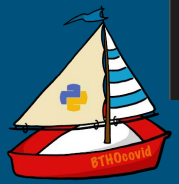
# Membership operators

| Operator | Returns |
| --- | --- |
| `in` | Returns True if value is in the collection |
| `not in` | Returns True if value is not in the collection |

**Syntax**

```
value in collection
```

```
value not in collection
```



# Examples

```
my_list = [3, 5, 4.65, 'Howdy', 'John']
'Howdy' in my_list
```

```
True
```

```
my_list = [3, 5, 4.65, 'Howdy', 'John']
'Dora' in my_list
```

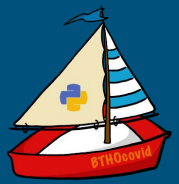
```
False
```

```
my_list = [3, 5, 4.65, 'Howdy', 'John']
name = 'John'
age = 35

if name in my_list:
  print('John is an item in the list')

if age in my_list:
  print('Age is an item in the list')
else:
  print('Age is not an item in the list')
```

```
John is an item in the list
Age is not an item in the list
```



# Revisiting Iterables

List, Tuples, Sets and Dictionaries are iterables in python

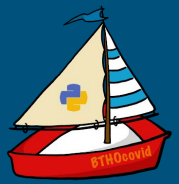
A for loop can be used to iterate of over the elements

```
for item in list:
    Statement..
    statement..
```

```
for item in tuple:
    Statement..
    Statement..
```

```
for item in set:
    Statement..
    Statement..
```

```
#To iterate over values
for values in dict.values():
    Statements
#To iterate over keys
for keys in dict.keys():
    Statements
#To iterate over both keys & values
for key, value in dict.items():
    Statements
```



# Examples

## List as an iterable

```
numList = list(range(1,9))
print(numList)
for number in numList:
  print(number**3, end='  ')
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
1  8  27  64  125  216  343  512
```

## Tuple as an iterable

```
myTuple = ('a', 'c', 'h', 'b', 'g', 'z')
print(myTuple)
for char in myTuple:
  print(char*3, end='  ')
```

```
('a', 'c', 'h', 'b', 'g', 'z')
aaa  ccc  hhh  bbb  ggg  zzz
```

## Set as an iterable

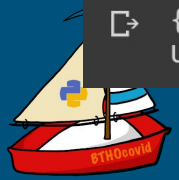
```
newSet = set('hullabaloo')
print(newSet)
for char in newSet:
  print(char.upper(), end='  ')
```

```
{'u', 'o', 'h', 'b', 'a', 'l'}
U  O  H  B  A  L
```

## Dictionary as an iterable

```
myDict = {'TX':'Texas', 'NY':'New York', 'AL':'Alabama'}
print(myDict)
for key, value in myDict.items():
  if key == 'TX':
    print(f'{key} corresponds to the state {value}')
```

```
{'TX': 'Texas', 'NY': 'New York', 'AL': 'Alabama'}
TX corresponds to the state Texas
```



# enumerate( )

Another way to shorten your code

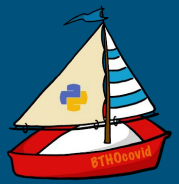
Enumerate function adds a counter to the iterable and returns it along with item

```
enumerate(collection, startcount)
```

```
myList = ['howdy', 'aggie', 'yell', 'bonfire', 'muster']

for index, value in enumerate(myList, 1):
  print(f'{index} : {value}')
```

```
1 : howdy
2 : aggie
3 : yell
4 : bonfire
5 : muster
```



# zip( )

If you wanna iterate through two or more collections simultaneously

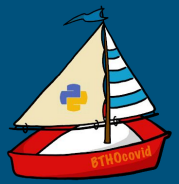
Zip ( ) takes in 2 or more iterables and returns an iterator

```
zip(iterable1, iterable2, ...)
```

```
listA = ['howdy', 'aggie', 'yell', 'bonfire', 'muster']
listB = [1, 2, 3, 4, 5]
listC = ['H', 'O', 'W', 'D', 'Y']

for a, b, c in zip(listA, listB, listC):
  print(f'{a} : {b} : {c}')
```

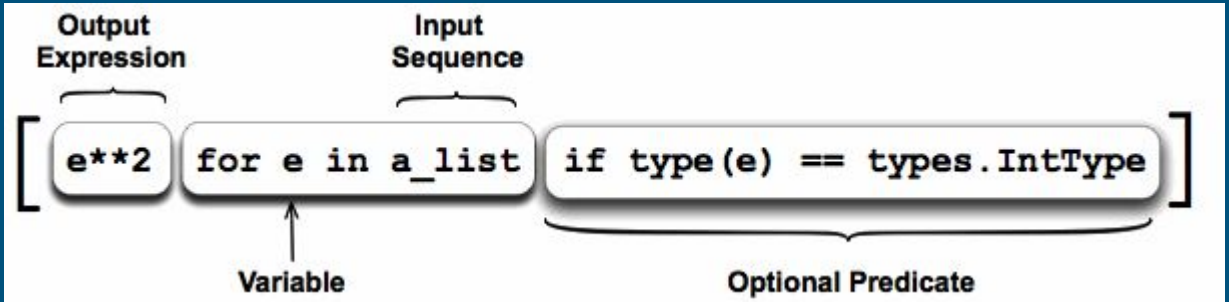
```
howdy : 1 : H
aggie : 2 : O
yell : 3 : W
bonfire : 4 : D
muster : 5 : Y
```



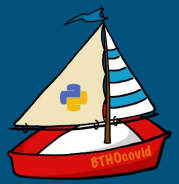
# Comprehension

Comprehension is a technique to shorten your code

Used when you need to create a collection from an existing collection




Note that comprehension can be used with dictionaries & sets



# Example of list comprehension

Classical Method

```
myList = list(range(1,9))
print(myList)
newList = []

for item in myList:
    if item % 2 == 0:
        newList.append(item)

print(newList)
```

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

Comprehension

```
myList = list(range(1,9))
print(myList)

newList = [item for item in myList if item%2 == 0]
print(newList)
```

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

# Nested List Comprehension

```python
#creating a nested list
nestList =[]
for i in range(5):
  #adding a blank element to list
  nestList.append([])
  for j in range(5):
    nestList[i].append(j)

print(nestList)
```

```python
nestList = [[j for j in range(5)] for i in range(5)]
print(nestList)
```

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

# Another Example

```
#Select a subset of planets if length > 6
planets = [['Mercury', 'Venus', 'Earth'],
           ['Mars', 'Jupiter', 'Saturn'],
           ['Uranus', 'Neptune', 'Pluto']]

subset = []
for sublist in planets:
  for planet in sublist:
    if len(planet) > 6:
      subset.append(planet)

print(subset)
```

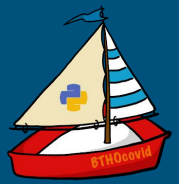
```
['Mercury', 'Jupiter', 'Neptune']
```

Outer Loop

Inner Loop

```
subset = [planet for sublist in planets for planet in sublist if len(planet)>6]
print(subset)
```

```
['Mercury', 'Jupiter', 'Neptune']
```



# Example of Dictionary Comprehension

Classical Method

```
myList = [1, 2, 3, 4, 5, 6, 7]
outputDict = {}

for item in myList:
  if item%2 != 0:
    outputDict[item] = item**2

print('Output dictionary:', outputDict)
```

```
Output dictionary: {1: 1, 3: 9, 5: 25, 7: 49}
```

Comprehension

```
myList = [1, 2, 3, 4, 5, 6, 7]

outputDict = {item:item**2 for item in myList if item%2 != 0}

print('Output dictionary:', outputDict)
```

```
Output dictionary: {1: 1, 3: 9, 5: 25, 7: 49}
```

# Next Lecture

## Functions

Wed, May 06 (9 pm-10 pm CDT)