2.1

Associative Arrays & Dictionaries

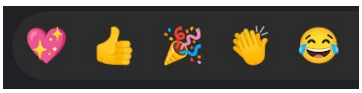
Lesson Plan

- [5] Icebreaker
- [5] Big-O Review
- [5] ADTs, Associative Arrays
- [15] Dictionaries: Access & Mutate, Iteration
- [20] Breakout rooms!
- [10] Counting & defaultdict
- [25] Breakout rooms!
- [10] Problem discussion
- Bonus Material: Nested Dicts
  - Autocomplete Example

Icebreaker

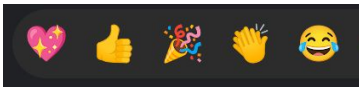
Big-O Review

```
def biggest(lst):
  while len(lst) > 1:
    smallest = min(lst)
    lst.remove(smallest)
  return lst[0]
```



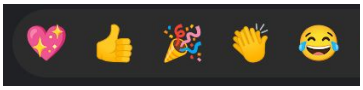
O(1) O(N) O(N^2) O(N^3) O(N^4)

```
def fn1(lst):
  for i in range(len(lst)):
    for j in range(10):
      for k in range(20):
        for m in range(10):
          print('hi')
```



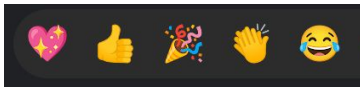
O(1) O(N) O(N^2) O(N^3) O(N^4)

```
def fn2(lst):
  for i in range(len(lst)):
    for j in range(len(lst)):
      print('hi')
      return lst[j]
```



O(1) O(N) O(N^2) O(N^3) O(N^4)

```
def fn3(lst):
  x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  for i in range(len(x)):
    for j in range(len(x)):
      for k in range(len(x)):
        print lst[i + j + k]
```



O(1) O(N) O(N^2) O(N^3) O(N^4)

Data Structures vs ADTs

Abstract Data Types (ADTs)

An **Abstract Data Type (ADT)** is defined by its **supported operations**.

These operations are described from the viewpoint of a **user**.

An ADT does **not** include any details about:

- how those operations are implemented
- the runtime of those operations

Data Structure

An **implementation** of an Abstract Data Type (ADT)

Includes information about:

- How the operations of the ADT are implemented
- The runtime of these operations

Example: **List** is an ADT

Supported Operations:

- **Append** a value to the end of the list
- **Pop** a value from the end of the list
- Return the **size** of the list
- Many more...

The **Python implementation of list** is a data structure

Underneath the hood it looks like an ArrayList.
Another implementation of the list ADT would be a LinkedList.
The **Append** operation works differently and has different runtimes for both.

Associative Arrays: key-value pairs

key	value
person	a human being
marathon	a running race that is about 26 miles
resist	to remain strong against the force
run	to move with haste; act quickly

Example: **Associative Array** is an ADT

Supported Operations:

- **Insert** a (key, value) pair or **Reassign** a key to a new value
- **Lookup** the value (if any) bound to a given key
- **Remove** a (key, value) pair

Example: A bad Data Structure…

ADT: Associative Array

Data Structure: A list of (key, value) tuples

```
grades = [('Mikayla', 78), ('Amy', 95), ('Nick', 100), ('Tristan', 67), ('Mark', 84)]
```

Operation	Description	Runtime
Insert	Add a new key-val pair	?
Replace	Replace the val for a key	?
Lookup	Check if a key exists	?
Remove	Remove a key-val pair	?

Language-Specific Implementations

Language	Implementation
Java	HashMap
C++	unordered_map
Python	dict

Real-Life Examples

Chat waterfall:
What are other real-life examples of when you'd use an associative array?

Song Title → # of Streams

Movie Title → Rating

Dictionaries: Access / Mutate

Cheat Sheet pages 5 & 6

Creating an empty dictionary

```
store = {}
```

Creating a dictionary

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}
```

Size of a dictionary

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

items = len(store) # number of key-value pairs
```

Lookup a value using a key

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

egg_price = store['eggs']
```

Don’t try to access something that isn’t there!

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

kiwi_price = store['kiwi'] # error!
```

Check if a key is in the dictionary

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

if 'kiwi' in store:
 print('found it')
```

Check if a key is not in the dictionary

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

if 'kiwi' not in store:
 print('not there')
```

Insert a new (key, value) pair

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

store['juice'] = 2.68
```

Reassign a key to a new value

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

store['eggs'] = 2.00
```

Remove a (key, value) pair

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

del store['eggs']
```

Don’t delete something that isn’t there!

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

del store['kiwi'] # error!
```

Empty the dictionary

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

store.clear()
```

Dictionaries: Iteration

Looping through the keys

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

# Prints all the keys
for key in store:
 print(key)
```

```
banana
milk
bread
eggs
```

Looping through the keys another way

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

# Prints all the keys
for key in store.keys():
 print(key)
```

```
banana
milk
bread
eggs
```

Looping through the values

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

# Prints all the values
for value in store.values():
 print(value)
```

```
0.24
3.29
3.29
1.54
```

Keys and Values

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

# Prints all the keys and values
for key in store:
 print(key + ' costs ' + str(store[key]))
```

```
banana costs 0.24
milk costs 3.29
bread costs 3.29
eggs costs 1.54
```

Keys and Values - an easier way

```
store = {
 'banana': .24,
 'milk': 3.29,
 'bread': 3.29,
 'eggs': 1.54
}

# Prints all the keys and values
for key, value in store.items():
 print(key + ' costs ' + str(value))
```

```
banana costs 0.24
milk costs 3.29
bread costs 3.29
eggs costs 1.54
```

Iteration Order

In Python 3.7+, (key, value) pairs are iterated in **insertion order**

In older versions of Python, you need to use OrderedDict to get the same behavior

In other languages, the iteration order could be arbitrary - don't rely on it!

Practice Problem: Dictionaries

You will be working in teams of 2 or 3. The goal is to collaboratively find a solution and be able to explain it to the class. Use the table below to figure out what your role is.

Role	Responsibilities	Assignment Criteria
Driver	Copy and share the repl.it, write the code, make sure you're listening to ideas from your teammates	Type "random number between 1 and 100" into Google. Person with the lowest number is the driver
Tester	Play devil's advocate, thinks of edge cases, write unit tests for the driver's code	Person with the highest number is the tester
Presenter	Document the code, be prepared to present the team's design decisions, and share one thing the team learned from the problem	Person with the middlest number is the presenter

If there are only 2 members in your team, the tester will also take on the presenter role.

Dictionaries: Counting and defaultdict

Country	Goals
France	16
Argentina	15
England	13
Portugal	12
Netherlands	10
Spain	9
Brazil	8
Croatia	8
Germany	6
Morocco	6
Senegal	5
Japan	5
Switzerland	5
Serbia	5
Ghana	5
South Korea	5
Australia	4
Ecuador	4
Iran	4
Cameroon	4
Poland	3
Saudi Arabia	3
Costa Rica	3
USA	3

Python Dictionaries: counting

```
dessert_vote_tally = {}

votes = ['pie', 'cake', 'cake', 'cookies', 'pudding', 'pie']

for vote in votes:
  dessert_vote_tally[vote] += 1
```

```
    4
    5 for vote in votes:
----> 6   dessert_vote_tally[vote] += 1

KeyError: 'pie'
```

Python Dictionaries: counting

```
dessert_vote_tally = {}

votes = ['pie', 'cake', 'cake', 'cookies', 'pudding', 'pie']

for vote in votes:
  if vote not in dessert_vote_tally:
    dessert_vote_tally[vote] = 1
  else:
    dessert_vote_tally[vote] += 1

print(dessert_vote_tally)
```

```
{'pie': 2, 'cake': 2, 'cookies': 1, 'pudding': 1}
```

Python Dictionaries: defaultdict

```
from collections import defaultdict

dessert_vote_tally = defaultdict(int)

votes = ['pie', 'cake', 'cake', 'cookies', 'pudding', 'pie']

for vote in votes:
  dessert_vote_tally[vote] += 1

print(dessert_vote_tally)
```

```
defaultdict(<class 'int'>, {'pie': 2, 'cake': 2, 'cookies': 1, 'pudding': 1})
```

Practice Problem: Dictionaries

You will be working in teams of 2 or 3. The goal is to collaboratively find a solution and be able to explain it to the class. Use the table below to figure out what your role is.

Role	Responsibilities	Assignment Criteria
Driver	Copy and share the repl.it, write the code, make sure you're listening to ideas from your teammates	Type "random number between 1 and 100" into Google. Person with the lowest number is the driver
Tester	Play devil's advocate, thinks of edge cases, write unit tests for the driver's code	Person with the highest number is the tester
Presenter	Document the code, be prepared to present the team's design decisions, and share one thing the team learned from the problem	Person with the middlest number is the presenter

If there are only 2 members in your team, the tester will also take on the presenter role.

Bonus Material: Nested Dictionaries

Nested dictionaries

A dictionary where the values are dictionaries!

```
defaultdict(lambda: defaultdict(int))
```

Autocomplete

Given a string of text, construct a dictionary where

- the keys are words
- the values are dictionaries that map each word to frequency of it appearing after the key word

Example Input:

'cats are better **than** **dogs** and dogs are better **than** **lizards**'

Example Output:

{

'cats': {'are': 1},

'are': {'better': 2},

'better': {'than': 2},

'**than**': {'**dogs**': 1, '**lizards**': 1},

'dogs': {'and': 1, 'are': 1},

'and': {'dogs': 1}

}

Autocomplete - Sample Solution

```
from collections import defaultdict

def word_pairs(text):
  words = text.lower().split()
  frequencies = defaultdict(lambda: defaultdict(int))
  for i in range(len(words) - 1):
    word1 = words[i]
    word2 = words[i + 1]
    frequencies[word1][word2] += 1
  return frequencies
```