PROGRAMMING FOR AI

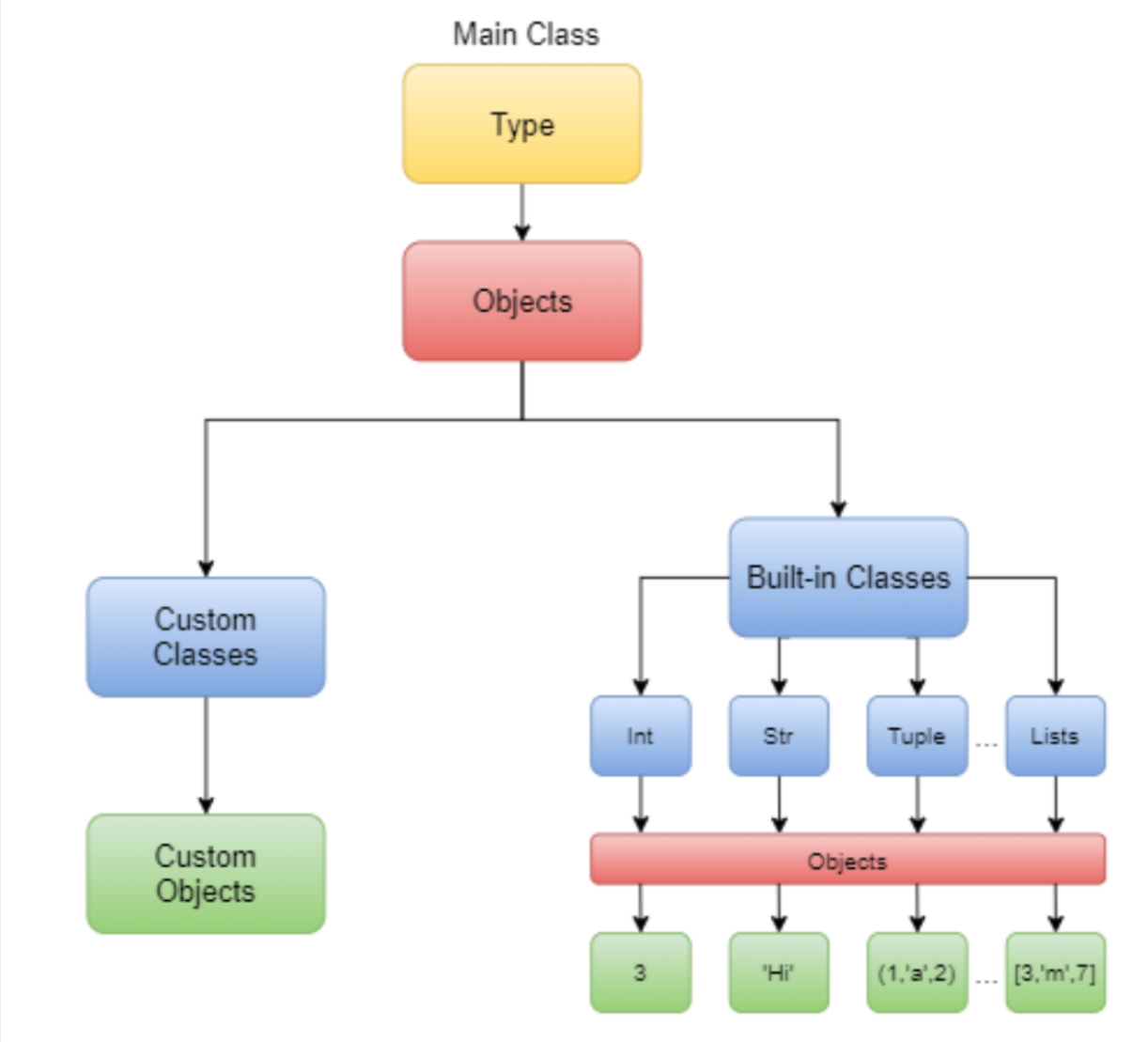
Lecture 3

Data Structure & Algorithms

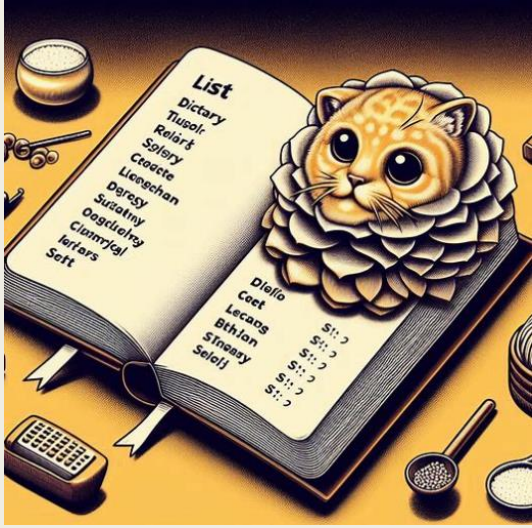
Instructor: Zafar Iqbal

Agenda

- Data Structures for AI: Lists, Tuples, Sets, Dictionaries, Arrays, DataFrames
- Advanced Data Structures: Stacks, Queues, Heaps, Trees
- Essential Algorithms: Sorting, Searching, Recursion


Main Class
Type
Objects
Custom Classes
Custom Objects
Built-in Classes
Int
Str
Tuple
...
Lists
Objects
3
'Hi'
(1,'a',2)
...
[3,'m',7]

Data Structure



Built-in Data Structures	User-defined Data Structures
list — Ordered, mutable collection of items	stack — LIFO (Last In, First Out) structure
tuple — Ordered, immutable collection of items	queue — FIFO (First In, First Out) structure
set — Unordered collection of unique items	linked list — Sequential nodes pointing to the next item
dict — Key-value pairs for fast lookups	tree — Hierarchical, parent-child relationship
	graph — Nodes connected by edges

Lists

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Length

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

String, int and boolean data types

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
print(list1)
print(list2)
print(list3)
```

Python - Access List Items

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
thislist =
["apple", "banana", "cherry", "orange",
 "kiwi", "melon", "mango"]
print(thislist[2:5])
```

```
thislist =
["apple", "banana", "cherry", "orange",
 "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
thislist =
["apple", "banana", "cherry", "orange",
 "kiwi", "melon", "mango"]
print(thislist[2:])
```

```
thislist =
["apple", "banana", "cherry", "orange",
 "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

```
thislist =
["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the
fruits list")
```

Python - Change List Items

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a Range of Item Values

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Change the second value by replacing it with two new values

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

Change the second and third value by replacing it with one value

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

The insert() method inserts an item at the specified index

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Python - Add List Items

Using the append() method to append an item

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Add the elements of tropical to thislist

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.)

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Python - Remove List Items

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Remove the first occurrence of "banana"

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Remove the second item

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Python - Loop Lists

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Through the Index Numbers

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

A short hand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Python - List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

Python Collections (Arrays)

- There are four collection data types in the Python programming language:
  - *List is a collection which is ordered and changeable. Allows duplicate members.*
  - *Tuple is a collection which is ordered and unchangeable. Allows duplicate members.*
  - *Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.*
  - *Dictionary is a collection which is ordered** and changeable. No duplicate members.*

Tuple()

- Tuples are used to store multiple items in a single variable
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuples are written with round brackets.
- Tuple features
  - *Access Tuples*
  - *Update Tuples*
  - *Unpack Tuples*
  - *Loop Tuples*
  - *Join Tuples*
  - *Tuple Methods*

Tuple()

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Allow Duplicates

```
thistuple =
("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Tuple Items - Data Types

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
print(tuple1)
print(tuple2)
print(tuple3)
```

Access Tuple Items

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Negative Indexing

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Range of Indexes

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

Access Tuple Items

Range of Negative Indexes

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Check if Item Exists

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Update Tuples

- Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Update Tuples

- **Add Items**
- Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.
- 1. **Convert into a list:** Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.
- Convert the tuple into a list, add "orange", and convert it back into a tuple

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)

print(thistuple)
```

Update Tuples

- 2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple
- Create a new tuple with the value "orange", and add that tuple

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

Convert the tuple into a list, remove "apple", and convert it back into a tuple

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Unpacking a Tuple

- Packing a tuple
  - *fruits = ("apple", "banana", "cherry")*
- In Python, we are also allowed to extract the values back into variables. This is called "unpacking“
- Unpacking a tuple

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Unpacking a Tuple

- **Using Asterisk***
- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
print(red)
```

Loop Tuples

- Loop through the tuple items by using a for loop

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Print all items, using a while loop to go through all the index numbers

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Join Tuples

- To join two or more tuples you can use the + operator

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Multiply Tuples

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

Task

- Create a Python program that does the following using tuples:
  - *Create a tuple named vegetables containing at least five different vegetable names.*
  - *Print the entire tuple.*
  - *Access and print the third element in the tuple.*
  - *Try to modify one element in the tuple and handle the resulting error.*
  - *Use tuple unpacking to assign the first three elements of the tuple to three different variables and print them.*

Task

```
# Step 1: Creating a tuple
vegetables = ("Carrot", "Potato", "Tomato", "Cabbage", "Spinach")

# Step 2: Print the entire tuple
print("Tuple of vegetables:", vegetables)

# Step 3: Access and print the third element
print("Third vegetable in the tuple:", vegetables[2])

# Step 4: Attempt to modify an element and handle the error
try:
    vegetables[1] = "Broccoli"  # Tuples are immutable
except TypeError as e:
    print("Error:", e)

# Step 5: Tuple unpacking
veg1, veg2, veg3, _, _ = vegetables  # Using underscore to ignore remaining values
print("Unpacked vegetables:", veg1, veg2, veg3)
```

Set

- Sets are used to store multiple items in a single variable.
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
- Sets are written with curly brackets.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Duplicates Not Allowed

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

True and 1 are considered the same value in sets, and are treated as duplicates

```
thisset = {"apple", "banana", "cherry", True, 1, 2}
print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

```
thisset = {"apple", "banana", "cherry", False, True, 0}
print(thisset)
```

Access Set Items

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
thisset = {"apple", "banana", "cherry"}

print("banana" not in thisset)
```

Add Set Items

Cannot change but can Add an item to a set, using the add() method

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

---

object in the update() method

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

Remove Set Items

Remove "banana" by using the remove() method

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

Remove "banana" by using the discard() method

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

The pop() method to remove an item

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

The clear() method empties the set

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

Loop Sets

Loop through the set, and print the values

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Join Sets

- There are several ways to join two or more sets in Python.
- The union() and update() methods joins all items from both sets.
- The intersection() method keeps ONLY the duplicates.
- The difference() method keeps the items from the first set that are not in the other set(s).
- The symmetric_difference() method keeps all items EXCEPT the duplicates.

Join Sets (Union)

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

You can use the | operator instead of the union() method

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1 | set2
print(set3)
```

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}
myset = set1.union(set2, set3, set4)
print(myset)
```

Join Sets (Intersection, Difference)

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.intersection(set2)
print(set3)
```

& operator instead of the intersection()

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 & set2
print(set3)
```

---

difference() method will return a new set

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.difference(set2)
print(set3)
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 - set2
print(set3)
```

Python Dictionaries

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Python Dictionaries

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
 thisdict = {
   "brand": "Ford",
   "electric": False,
   "year": 1964,
   "colors":
 ["red", "white", "blue"]
 }
```

Access Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

---

method called get() that will give you the same result

```
thisdict ={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

Access Items

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.keys()
print(x) #before the change
car["color"] = "white"
print(x) #after the change
```

Update Dictionary

change the value of a specific item by referring to its key name

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

---

update() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020}
```

Removing Items

There are several methods to remove items from a dictionary:

pop() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

---

popitem() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

del keyword:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

---

clear() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Loop Dictionaries

Print all key names in the dictionary, one by one using a for loop:

```
for x in thisdict:
  print(x)
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

Print all values in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

use the values() method to return values:

```
for x in thisdict.values():
  print(x)
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

use the keys() method to return the keys

```
for x in thisdict.keys():
  print(x)
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)
```

Loop through both keys and values, by using the items() method:

```
for x, y in thisdict.items():
  print(x, y)
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Copied Dictionaries

. There are two ways to make a copy:

- built-in function dict() .
- built-in Dictionary method copy().

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Nested Dictionaries

A dictionary can contain dictionaries is called nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}
myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

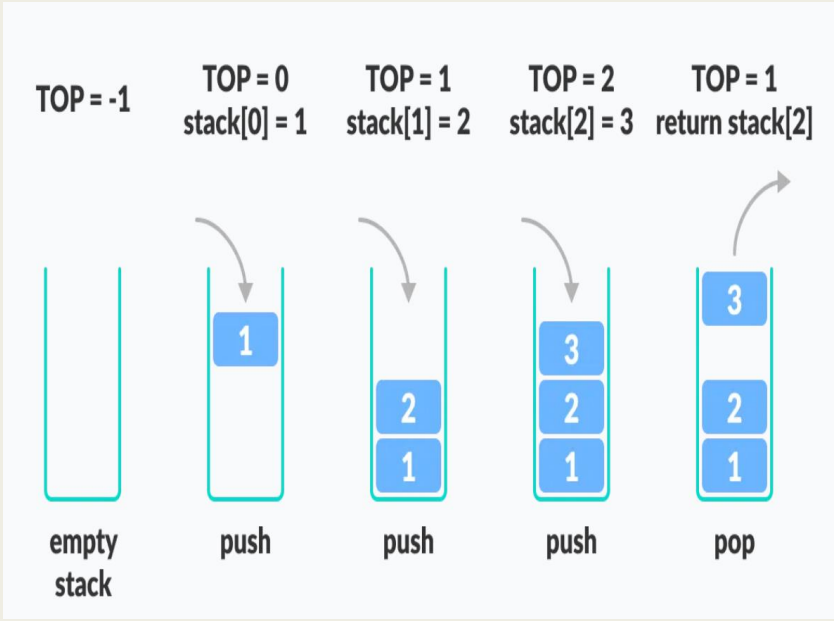
Dictionaries Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Stack implementation in python

```
# Creating a stack
def create_stack():
    stack = []
    return stack
# Creating an empty stack
def check_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("pushed item: " + item)
# Removing an element from the stack
def pop(stack):
    if (check_empty(stack)):
        return "stack is empty"
    return stack.pop()
stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
print("popped item: " + pop(stack))
print("stack after popping an element: " + str(stack))
```



Trace table

Step	Operation	Stack State	Output
1	`create_stack()`	`[]`	
2	`push(1)`	`[1]`	`pushed item: 1`
3	`push(2)`	`[1, 2]`	`pushed item: 2`
4	`push(3)`	`[1, 2, 3]`	`pushed item: 3`
5	`push(4)`	`[1, 2, 3, 4]`	`pushed item: 4`
6	`pop()`	`[1, 2, 3]`	`popped item: 4`
7	Print Stack	`[1, 2, 3]`	`stack after popping an element: [1, 2, 3]`

Queue in Python

```
queue = []
queue.append('a')
queue.append('b')
queue.append('c')
print("Initial queue")
print(queue)
print("\nElements dequeued from queue")
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))
print("\nQueue after removing elements")
print(queue)
```

Task 1: Swapping Two Variables

Write python code to swap two variables using a tuple

Task 1: Swapping Two Variables

Write python code to swap two variables using a tuple

```
a, b = 5, 10
a, b = b, a
print(f'a: {a}, b: {b}')  # Output: a: 10, b: 5
```

Task 2: Find Common Elements

Write a python code to find common elements between two sets

Task 2: Find Common Elements

Write a python code to find common elements between two sets

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
common = set1.intersection(set2)
print(common)  # Output: {3, 4}
```

Task 3: Remove Duplicates

Write a python code to remove duplicates from a list while maintaining order

Task 3: Remove Duplicates

Write a python code to remove duplicates from a list while maintaining order

```
lst = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(dict.fromkeys(lst))
print(unique_list)  # Output: [1, 2, 3, 4, 5]
```

Task 4: Count Word Frequency

Write a python code to count the frequency of each word in a string

Task 4: Count Word Frequency

Write a python code to count the frequency of each word in a string

```
text = 'hello world hello Python'
words = text.split()
word_count = {}
for word in words:
    word_count[word] = word_count.get(word, 0) + 1
print(word_count)  # Output: {'hello': 2, 'world': 1, 'Python': 1}
```

Task 5: Find the Second Largest Element

Write a python code to find the second largest number in a list

Task 5: Find the Second Largest Element

- **Write a python code to find the second largest number in a list**

```
nums = [10, 20, 4, 45, 99]
second_largest = sorted(set(nums))[-2]
print(second_largest)  # Output: 45
```