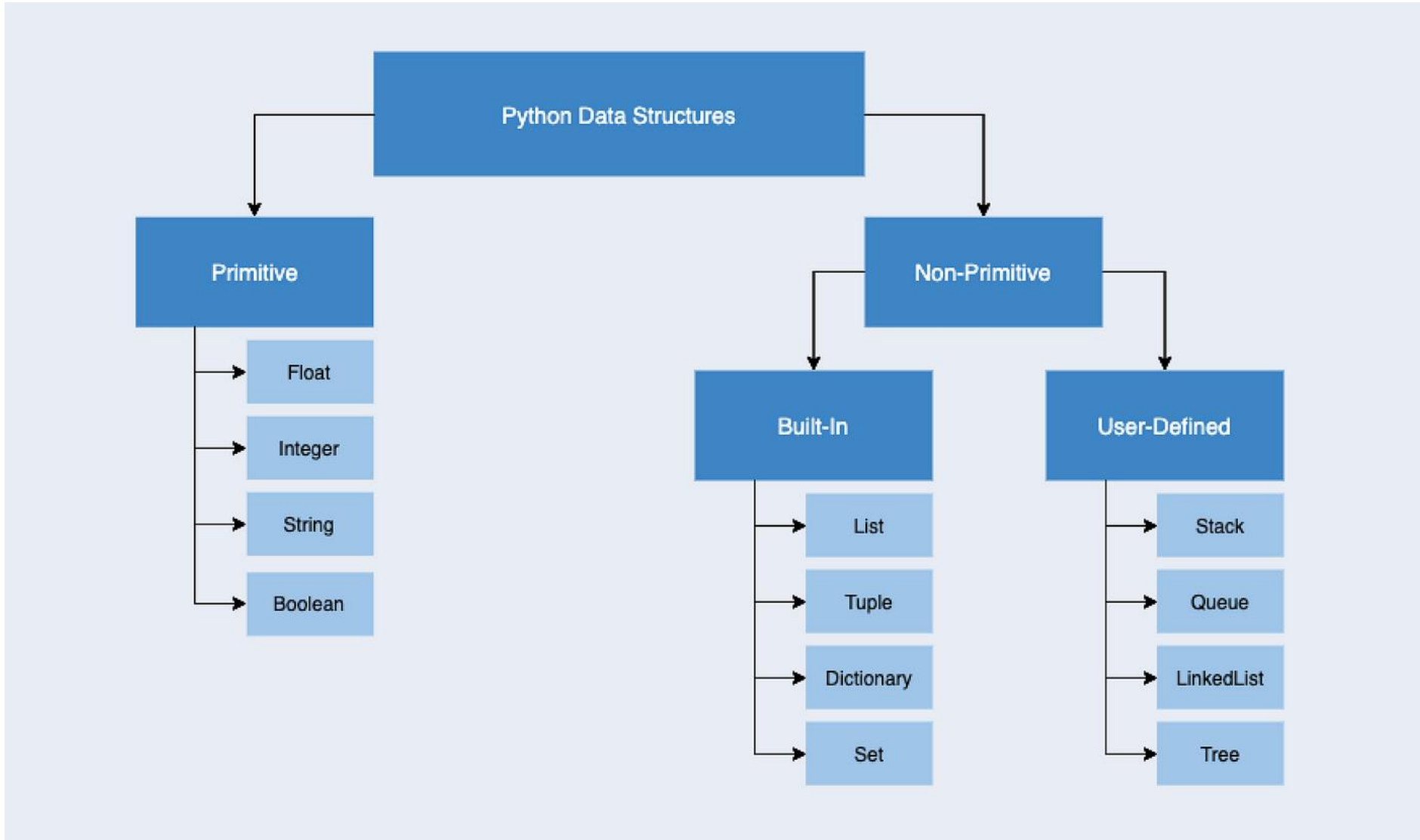
AI VIETNAM
All-in-One Course

Data Structure

Tuple, Set & Dictionary

Nguyen Dinh Vinh
Ph.D in Computer Science

Year 2023

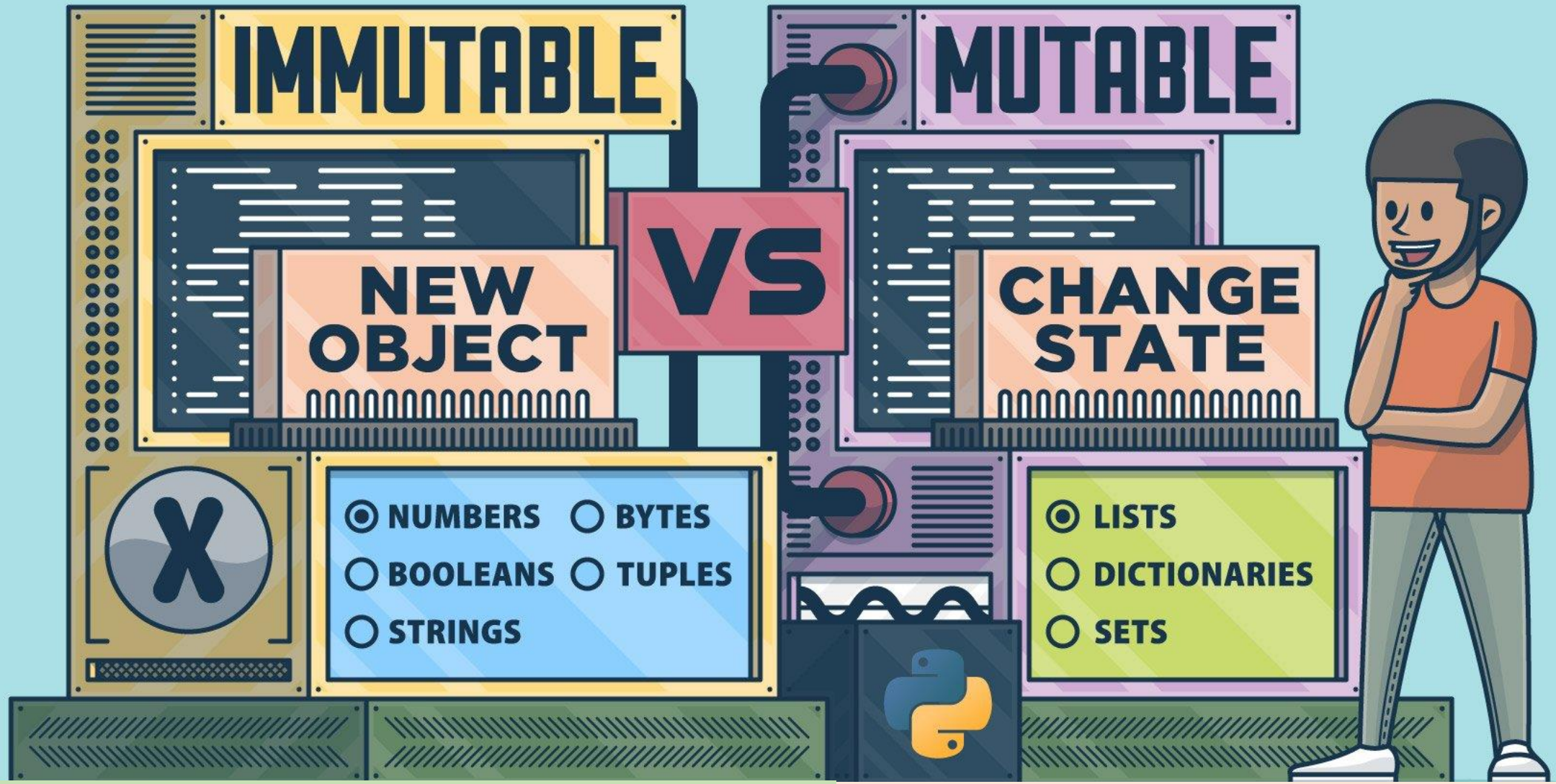

Python Data Structures
Primitive
Float
Integer
String
Boolean
Non-Primitive
Built-In
List
Tuple
Dictionary
Set
User-Defined
Stack
Queue
LinkedList
Tree

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Further study

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Further study



- **Immutable**:
  - Nhóm này gồm có: Numbers, String, Tuple, và các kiểu immutable khác do người dùng định nghĩa
  - Giá trị **không thể thay đổi**

- **Mutable**:
  - Nhóm này gồm có: List, Set, Dict và các kiểu mutable khác do người dùng định nghĩa
  - Giá trị **có thể thay đổi**

Problem in using List

```
def getFirstLastName(name):
  list = name.split()
  return [list[2], list[0]]
```

```
name = "Nguyen Dinh Vinh"
result = getFirstLastName(name)
result[0] = "Unknown"
result[1] = "Unknown"
print(result)
```

```
['Unknown', 'Unknown']
```

Are there any data struture to prevent a user to modify data?

Tuples

Solve by Tuples

```
def getFirstLastName(name):
  list = name.split()
  return (list[2], list[0])

name = "Nguyen Dinh Vinh"
result = getFirstLastName(name)
result[0] = "Unknown"
result[1] = "Unknown"
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-a562679c17a9> in <cell line: 7>()
      5 name = "Nguyen Dinh Vinh"
      6 result = getFirstLastName(name)
----> 7 result[0] = "Unknown"
      8 result[1] = "Unknown"
      9 print(result)

TypeError: 'tuple' object does not support item assignment
```

What is a Tuples?

Tuples Are Like Lists

Tuples are another kind of sequence that functions much like a list - they have elements which are indexed starting at 0

```
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print(x[2])
Joseph
>>> y = ( 1, 9, 2 )
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```

```
>>> for iter in y:
...     print(iter)
...
1
9
2
>>>
```

but... Tuples are “immutable”

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>>[9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback:'str' object does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple' object does
not support item
Assignment
>>>
```

Built-in Functions in List and Tuple

```
>>> l = list()
>>> dir(l)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

>>> t = tuple()
>>> dir(t)
['count', 'index']
```

Tuple

❖ Structure

tuple_name = (element-1, …, element-n)

Create a tuple

```
1.  t = (1, 2, 3)
2.
3.  print(t[0])
4.  print(t[1])
5.  print(t[2])
```

```
1
2
3
```

Tuple unpacking

```
1  x1,y1,z1   = ('a','b','c')
2  (x2,y2,z2) = ('a','b','c')
3  print(x1)
4  print(x2)
```

```
a
a
```

Tuple

❖ Structure

tuple_name = (element-1, …, element-n)

( ) can be removed

```
1.  t = 1, 2
2.  print(t)
```

```
(1, 2)
```

Tuple with one element

```
1  var1 = (1 + 2) * 5
2  print(type(var1), ' ', var1)
3
4  var2 = (1)
5  print(type(var2), ' ', var2)
6
7  var3 = (1,)
8  print(type(var3), ' ', var3)
```

```
<class 'int'>   15
<class 'int'>   1
<class 'tuple'>   (1,)
```

Do you remember zip() function

Oh No, a new terminology? What is it?

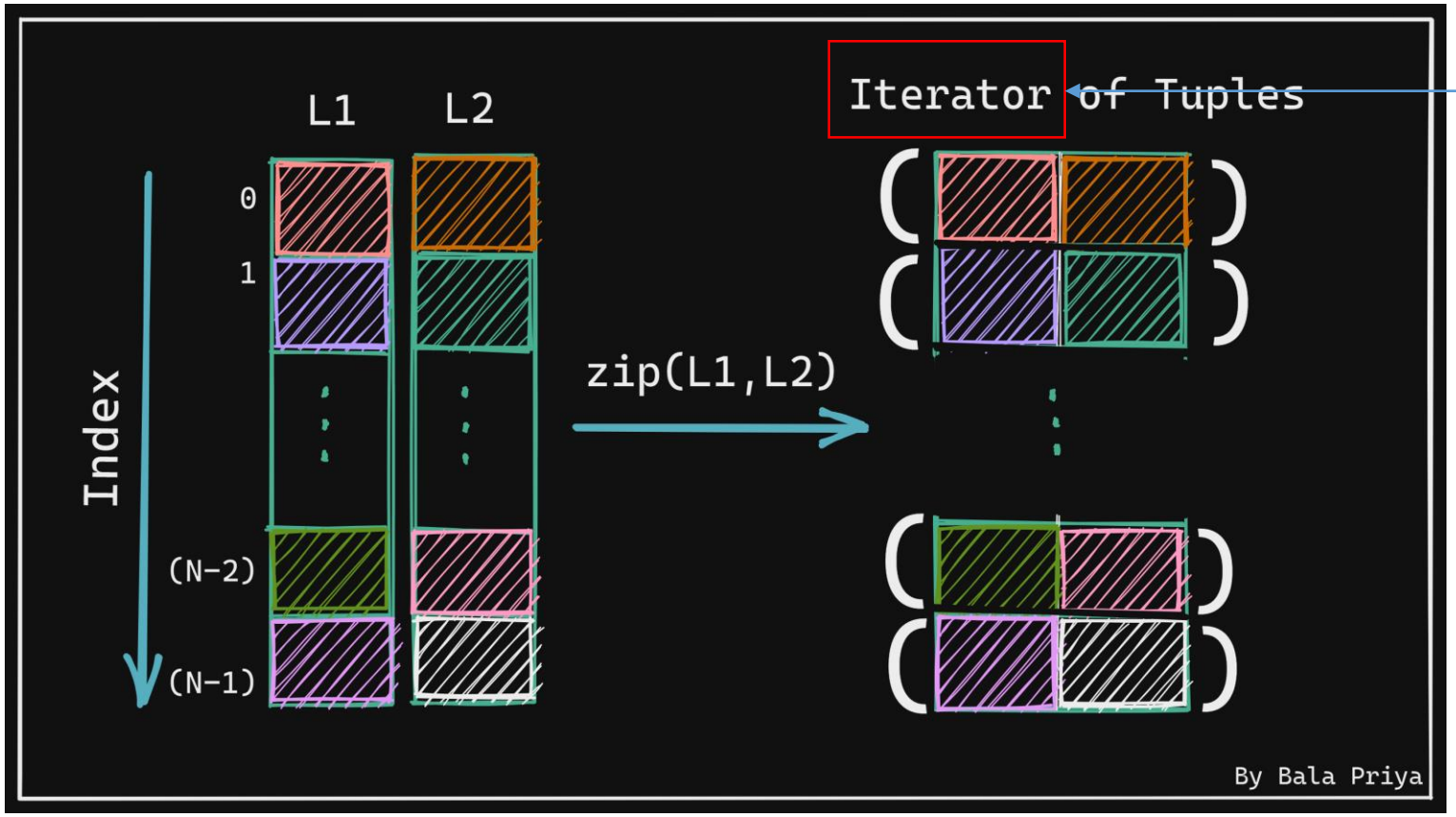


Image credit: https://www.freecodecamp.org/

Look back to Iteration in Python

When writing computer programs, you often need to repeat a given piece of code multiple times. To do this, you can follow one of the following approaches:

Repeating the target code as many times as you need in a sequence

```
# Get five input numbers from the user
number1 = input("Please enter the number:")
number2 = input("Please enter the number:")
number3 = input("Please enter the number:")
number4 = input("Please enter the number:")
number5 = input("Please enter the number:")
numbers = [number1, number2, number3, number4, number5]
```

Putting the target code in a loop that runs as many times as you need

```
1  numbers = []
2  list = [1 2 3 4 5]
3  for i in list:
4      numbers.append(input("Please enter the number:"))
```

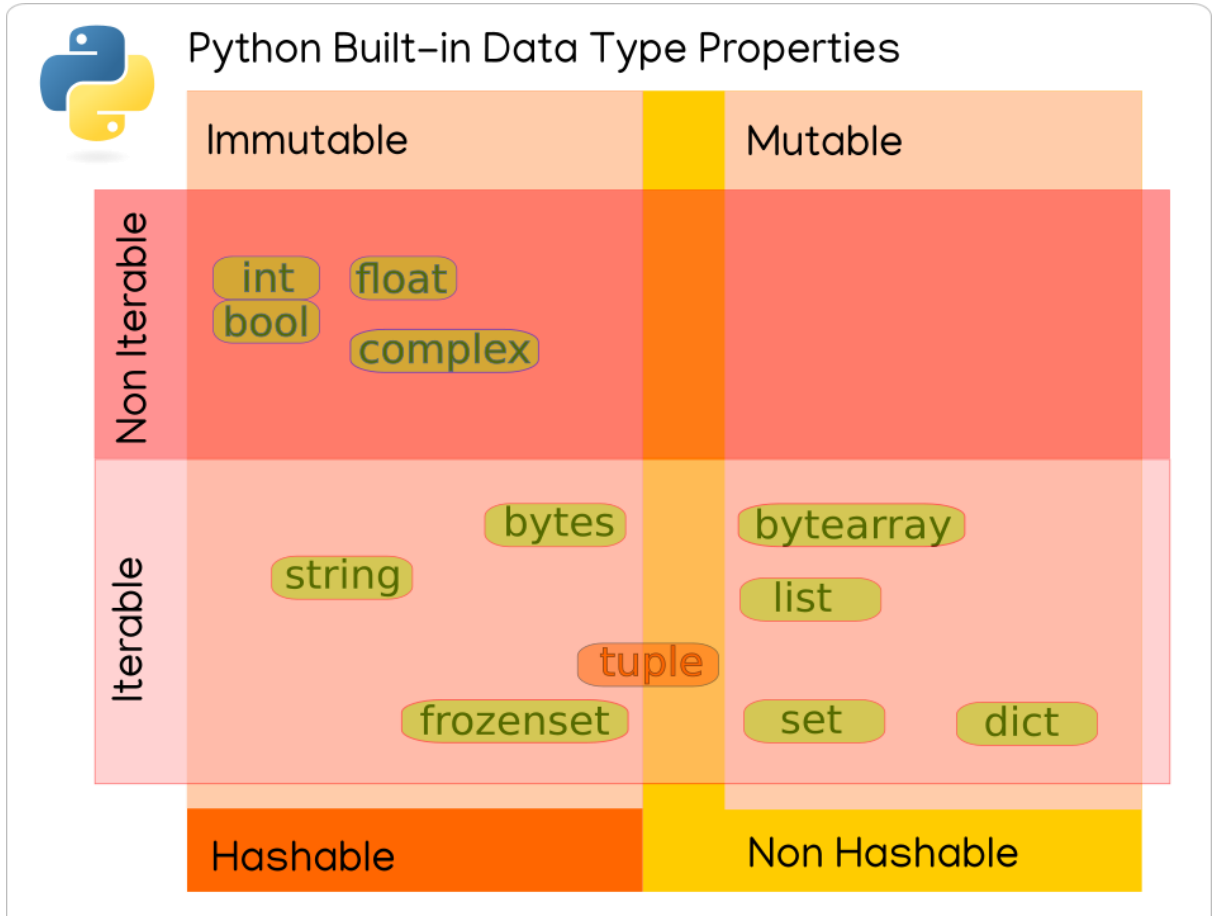
Limitations?

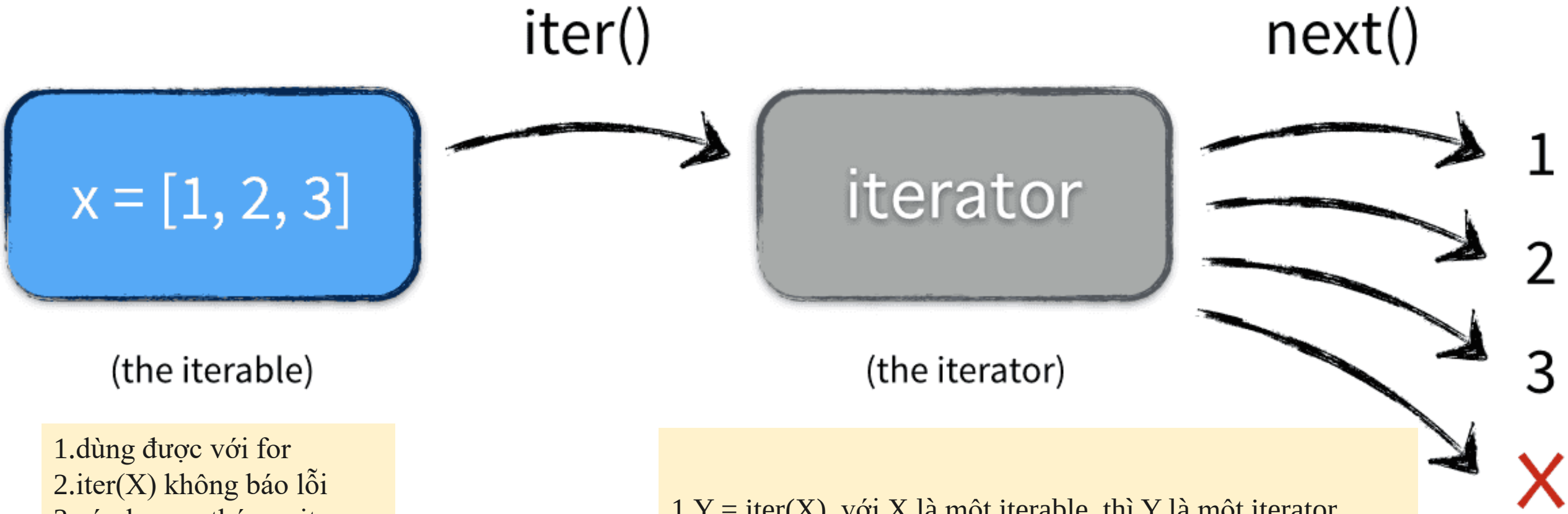
iterables

Iteration

Oh No, a new terminology? What is it?

When it comes to iteration in Python, you'll often hear people talking about **iterable objects** or just **iterables**. As the name suggests, an iterable is an object that you can *iterate* over. To perform this iteration, we might use a for loop.

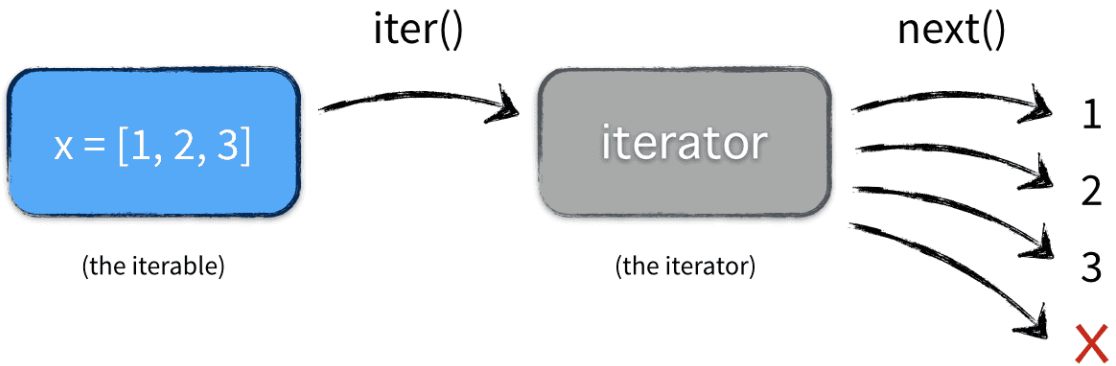




1.dùng được với for
2.iter(X) không báo lỗi
3.có phương thức __iter__

1.Y = iter(X), với X là một iterable, thì Y là một iterator
2.next(Y) sẽ trả về giá trị phần tử tiếp theo hoặc ngoại lệ StopIteration, thì Y là iterator.

Other solution to access elememements in the List rather than using for loop?



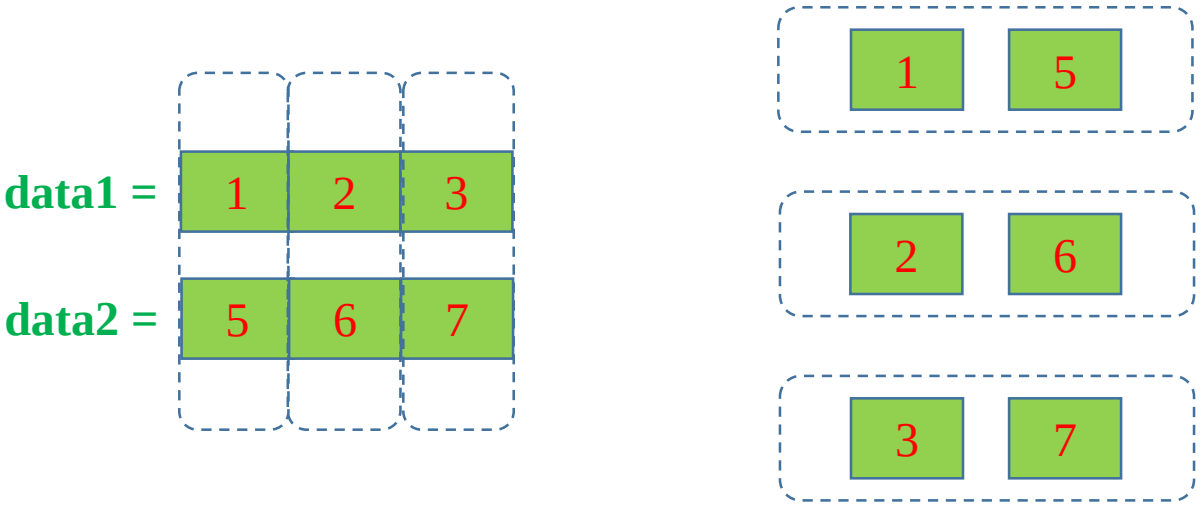

```python
# define a iterable such as a list
list1 = [0, 1, 2]

# get an iterator using iter()
iter1 = list1.__iter__()

#iterate the item using __next__method
print(iter1.__next__())

#iterate the item using __next__method
print(iter1.__next__())

#iterate the item using __next__method
print(iter1.__next__())
```

Discuss more detail in next lecture

Built-in Functions

zip()

the **iterators are also iterables** which act as their **own iterators**



zip(*iterables). The function takes in iterables as arguments and returns an **iterator**

zip() function to perform parallel iterations on multiple iterables

```
l1 = [1, 2, 3]
l2 = [5, 6, 7]

# print in pairs
length = len(l1)
for i in range(length):
    print(l1[i], l2[i])
```

```
1 5
2 6
3 7
```

```
l1 = [1, 2, 3]
l2 = [5, 6, 7]

# print in pairs
for v1, v2 in zip(l1, l2):
    print(v1, v2)
```

```
1 5
2 6
3 7
```

```python
from itertools import count

#Tao ra danh sach so chan [0, 2, 4, 6, ..., infinive]
list_1 = list[count(start = 0, step = 2)]
print(list_1)

```

```python
from itertools import count

#Tao ra danh sach so chan [0, 2, 4, 6, ..., infinive]
iterator = (count(start = 0, step = 2))

print("1st element:", next(iterator))
print("2nd element:", next(iterator))
print("3rd element:", next(iterator))
print("4th element:", next(iterator))
```

Tuple

❖ + and * operators

```
1.  t1 = (1, 0)
2.  print(t1)
3.
4.  t1 += (2,)
5.  print(t1)
```

```
(1, 0)
(1, 0, 2)
```

```
1.  t = (1,) * 5
```

```
(1, 1, 1, 1, 1)
```

count() - đếm số lần xuất hiện của một giá trị

index() - tìm vị trí xuất hiện của một giá trị

```
1.  t = (1,2,3,1)
2.  count = t.count(1)
3.  index = t.index(2)
4.
5.  print(count)
6.  print(index)
```

```
2
1
```

https://colab.research.google.com/drive/1yZIgm8v5pkCk0u2kksV9qiP_-ELGrw3q?usp=sharing

Tuple

len() - Tìm chiều dài của một tuple

```
1.  t = (1, 2, 3, 4)
2.  len(t)
```

```
4
```

Lấy giá trị min và max của một tuple

```
1.  t = (1, 2, 3, 4, 5)
2.
3.  print(min(t))
4.  print(max(t))
5.  print(sum(t))
```

```
1
5
15
```

Dùng hàm zip() cho tuple

```
1.  t1 = (1, 2, 3, 4, 5)
2.  t2 = ('a', 'b', 'c', 'd', 'e')
3.
4.  print(t1)
5.  print(t2)
6.
7.  t3 = zip(t1, t2)
8.  for x,y in t3:
9.      print(x, y)
```

```
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd', 'e')
1 a
2 b
3 c
4 d
5 e
```

Sắp xếp các giá trị trong một tuple

```
1.  t = (4, 7, 3, 9, 6)
2.  t_s = sorted(t)
3.  print(t_s)
```

```
[3, 4, 6, 7, 9]
```

Tuple

❖ Immutable

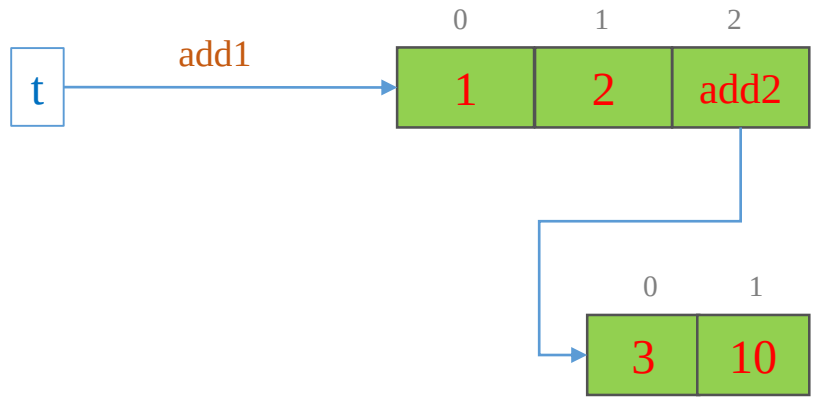
```
1. t = (1, 2, 3, 4, 5)
2. t[2] = 9
3. print(t)
```

```
---------------------------------------------------------------
TypeError                                 Traceback (most
<ipython-input-30-b1f85e7d332a> in <module>
      1 t = (1, 2, 3, 4, 5)
----> 2 t[2] = 9
      3 print(t)

TypeError: 'tuple' object does not support item assignment
```

however

```
1. t = (1, 2, [3, 10])
2. t[2][1] = 4
3. print(t)
```

```
(1, 2, [3, 4])
```



Tuple Examples

Swapping two variables

```
def swap(v1, v2):
    (v2, v1) = (v1, v2)
    return (v1, v2)
```

```
v1 = 2
v2 = 3
(v1, v2) = swap(v1, v2)

# print
print(v1)
print(v2)
```

```
3
2
```

Memory requirement

```
# memory comparison
import sys

aList  = [3, 4, 5, 6, 7]
aTuple = (3, 4, 5, 6, 7)

print(sys.getsizeof(aList))
print(sys.getsizeof(aTuple))
```

```
120
80
```

Tuple slicing

```
data = (1, 2, 3, 4, 5)
print(data[2:])
print(data[::-1])
```

```
(3, 4, 5)
(5, 4, 3, 2, 1)
```

list2tuple

```
# convert from list to tuple
aList  = [3, 4, 5, 6, 7]
aTuple = tuple(aList)

print(aTuple)
print(type(aTuple))
```

```
(3, 4, 5, 6, 7)
<class 'tuple'>
```

tuple2list

```
# convert from tuple to list
aTuple  = (3, 4, 5, 6, 7)
aList = list(aList)

print(aList)
print(type(aList))
```

```
[3, 4, 5, 6, 7]
<class 'list'>
```

AI VIETNAM
All-in-One Course

Tuple

❖ Example: Solve quadratic equation

```python
import math

def quadratic_equation(a, b, c):
    '''
    This function aims at solving the quadratic equation
    a, b, c --- three parameters and a =! 0
    '''

    # compute delta
    delta = b*b - 4*a*c

    if delta < 0:
        return ()
    elif delta == 0:
        x = (-b+math.sqrt(delta))/2*a
        return (x,)
    else:
        x1 = (-b+math.sqrt(delta))/(2*a)
        x2 = (-b-math.sqrt(delta))/(2*a)
        return (x1,x2)

```

Case 1: delta<0

```python
result = quadratic_equation(a=5, b=0, c=1)
print(type(result))
print(len(result))
print(result)
```

```
<class 'tuple'>
0
()
```

Case 2: delta>0

```python
result = quadratic_equation(a=5, b=5, c=1)
print(type(result))
print(len(result))
print(result)
```

```
<class 'tuple'>
2
(-0.276393202250021, -0.7236067977499789)
```

Case 3: delta=0

```python
result = quadratic_equation(a=4, b=4, c=1)
print(type(result))
print(len(result))
print(result)
```

```
<class 'tuple'>
1
(-8.0,)
```

Data is protected

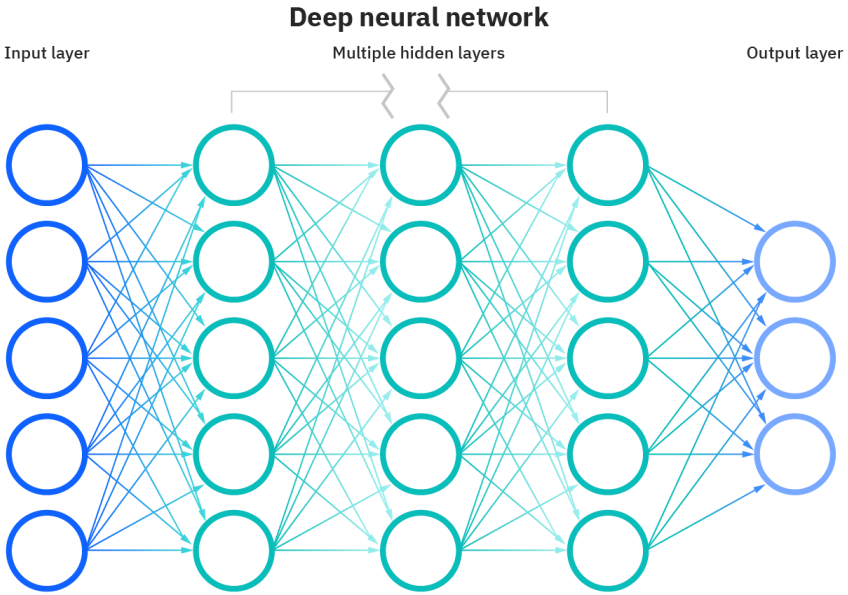
```python
result = quadratic_equation(a=4, b=4, c=1)
result[0] = 1
```

```
---------------------------------------------------------------
TypeError                                 Traceback (most
<ipython-input-21-55f394e5ddc8> in <module>
      1 result = quadratic_equation(a=4, b=4, c=1)
----> 2 result[0] = 1

TypeError: 'tuple' object does not support item assignment
```

Case study

Tuple in Object Detection and Localization

Input/Đầu vào

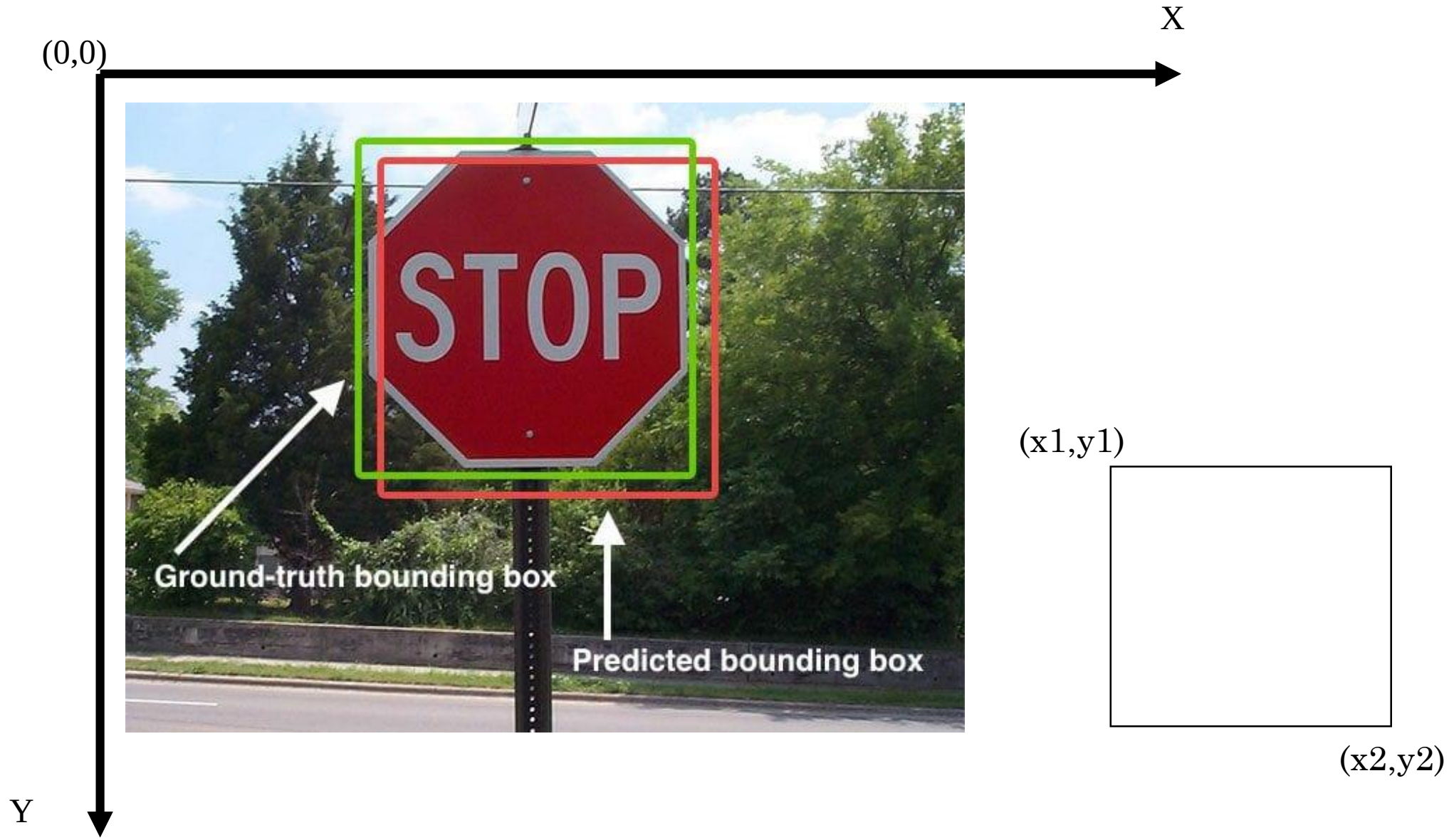


AI algorihtm/Giải thuật
trí tuệ nhân tạo

Output/Kết quả chương
trình

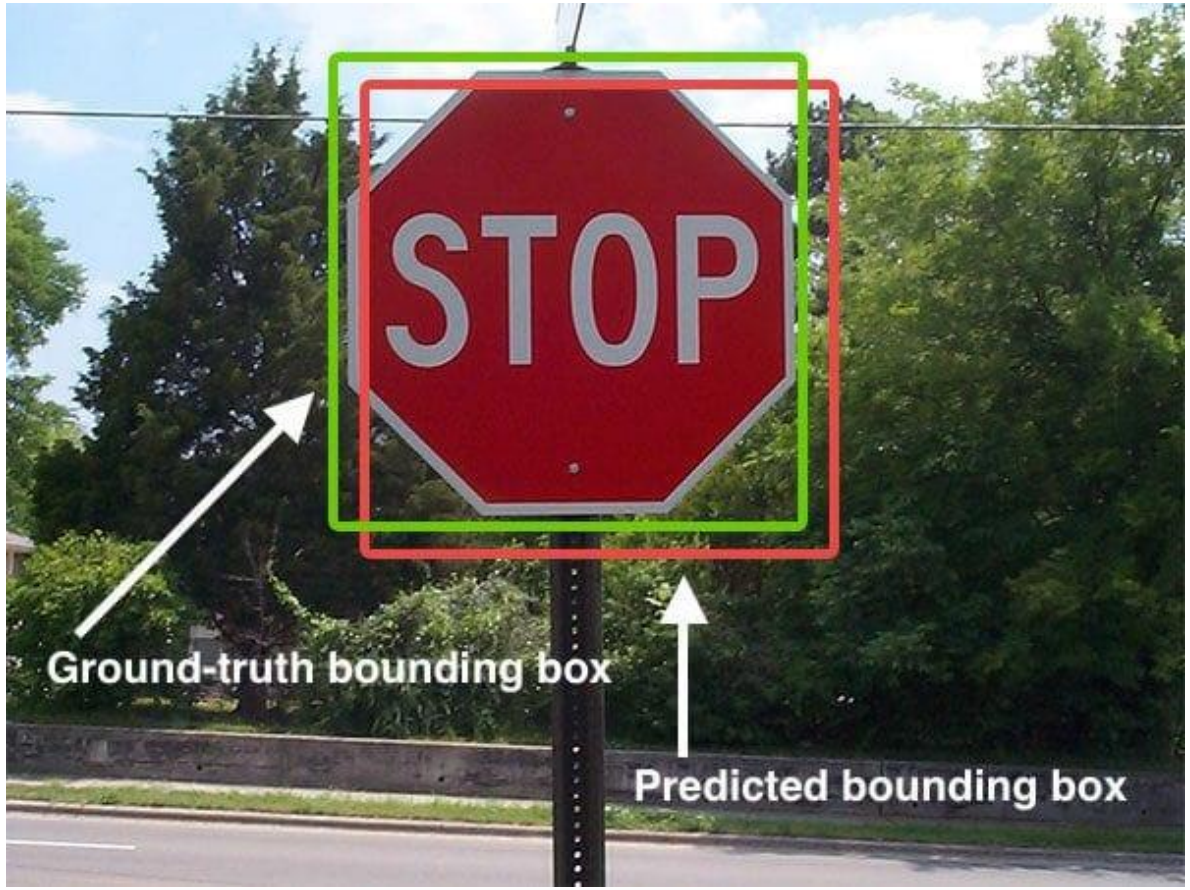
Ground Truth / Kết quả mong đợi

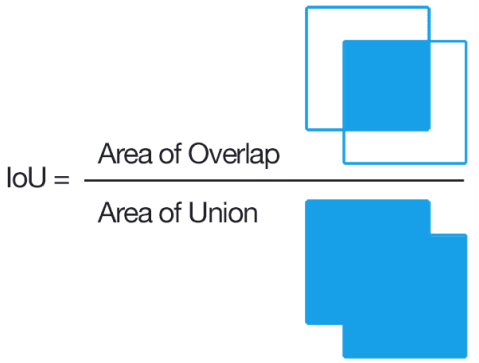
Image coordinate system



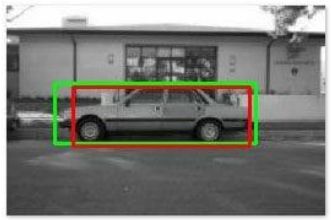
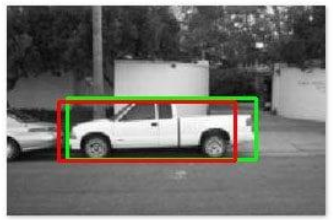
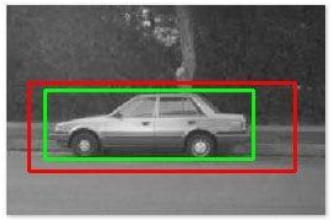
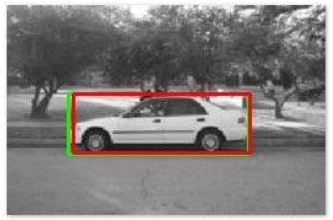
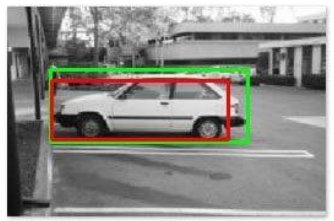
Case study

Tuple in Object Detection and Localization





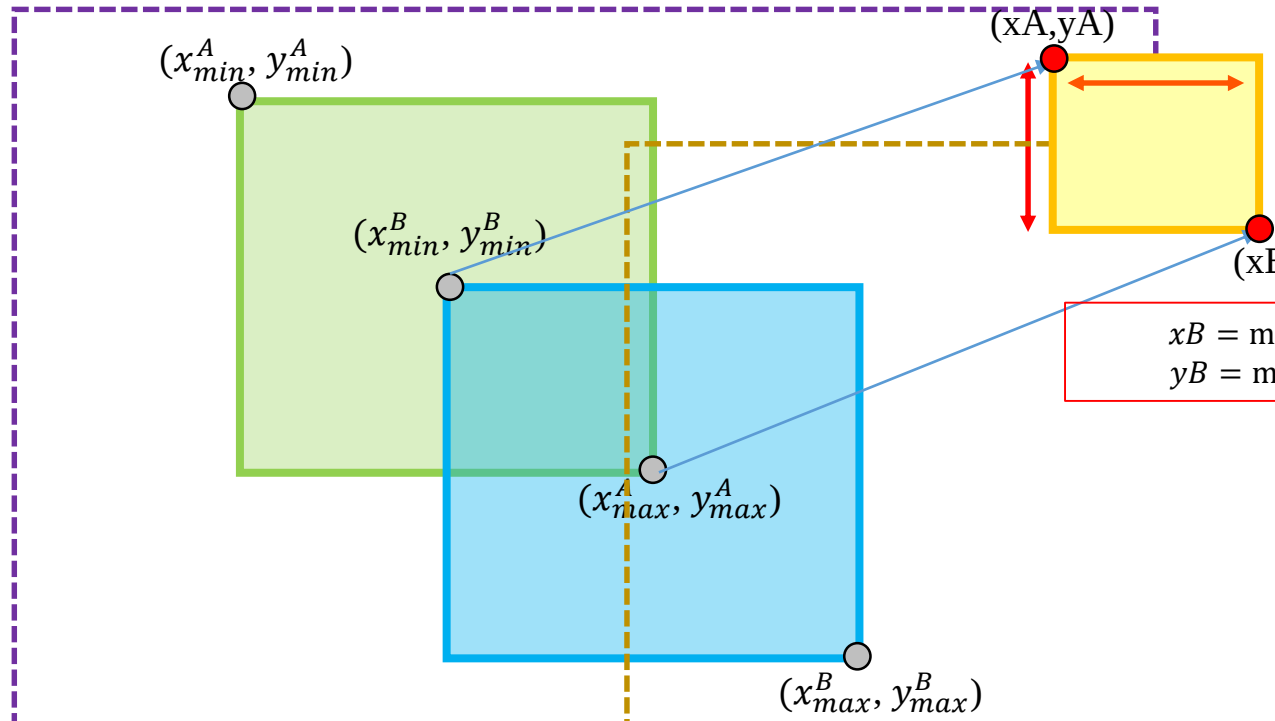
$$\frac{|A \cap B|}{|A \cup B|}$$

Giả sử ta có 2 boxes với thông tin như sau:

- **Box A** có tọa độ $(x^A_{min}, y^A_{min}, x^A_{max}, y^A_{max})$ => Tuple
- **Box B** có tọa độ $(x^B_{min}, y^B_{min}, x^B_{max}, y^B_{max})$ => Tuple

$$xA = \max(x^A_{min}, x^b_{min}) = x^b_{min}$$
$$yA = \max(y^A_{min}, y^b_{min}) = y^b_{min}$$

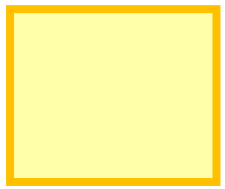


$$xB = \min(x^A_{max}, x^b_{max}) = x^A_{max}$$
$$yB = \min(y^A_{max}, y^b_{max}) = y^A_{max}$$

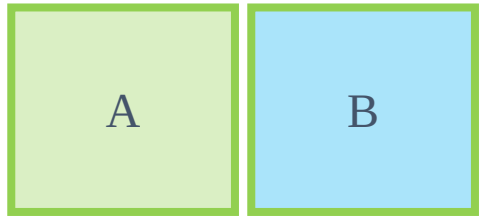
$A \cap B = W * H$

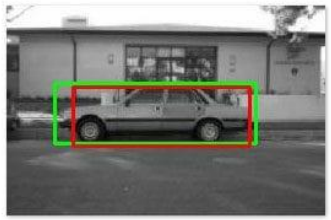
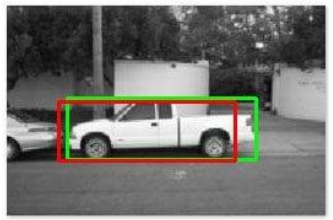
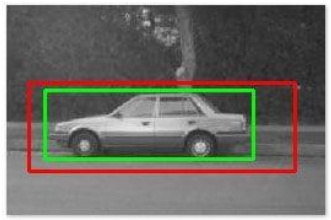
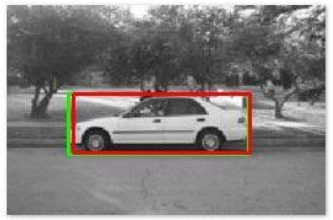
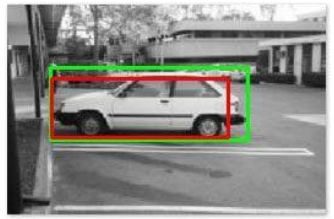
$W = xB - xA$

H = yB – yA



Area

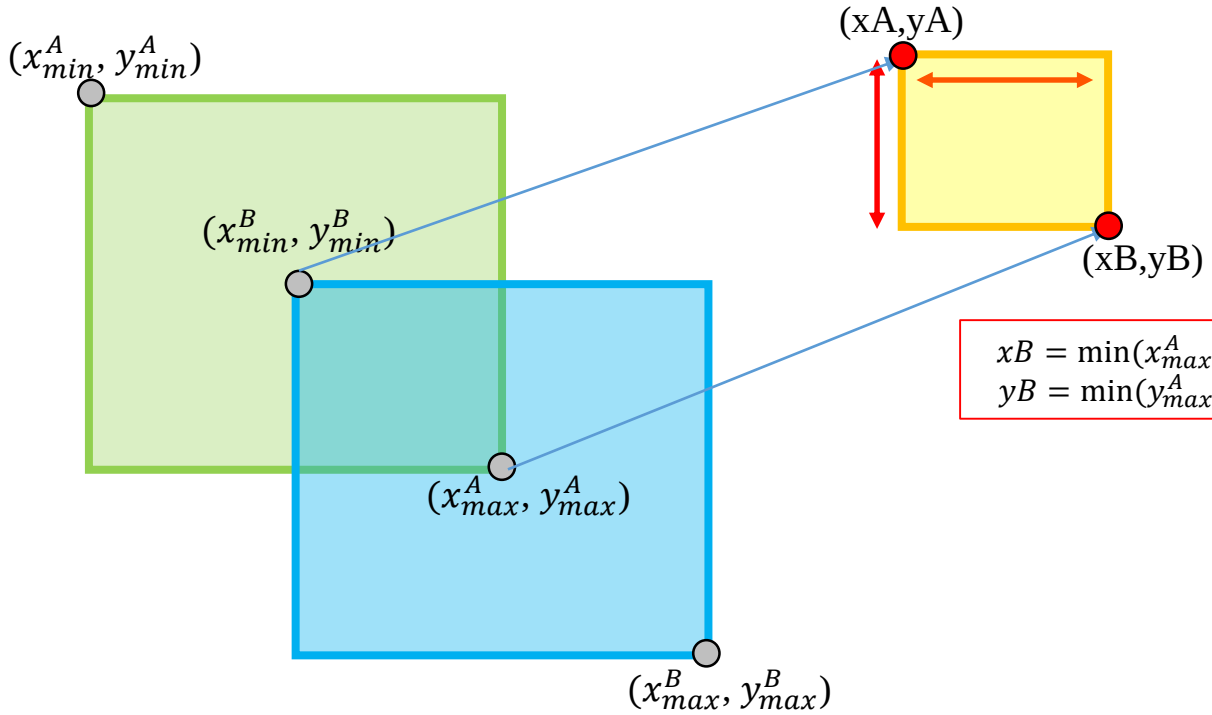


Giả sử ta có 2 boxes với thông tin như sau:

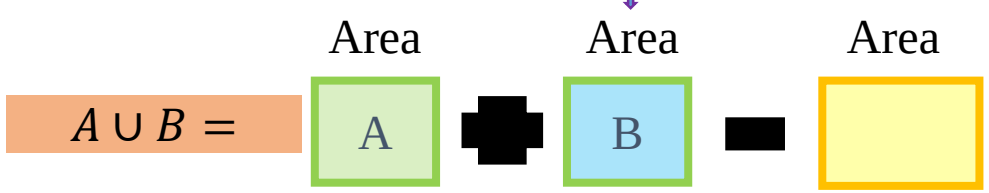
- **Box A** có tọa độ $(x^A_{min}, y^A_{min}, x^A_{max}, y^A_{max})$ => Tuple
- **Box B** có tọa độ $(x^B_{min}, y^B_{min}, x^B_{max}, y^B_{max})$ => Tuple

$$xA = \max(x^A_{min}, x^b_{min}) = x^b_{min}$$
$$yA = \max(y^A_{min}, y^b_{min}) = y^b_{min}$$



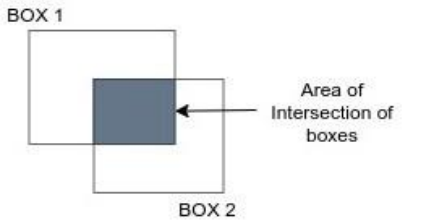
$$xB = \min(x^A_{max}, x^b_{max}) = x^A_{max}$$
$$yB = \min(y^A_{max}, y^b_{max}) = y^A_{max}$$

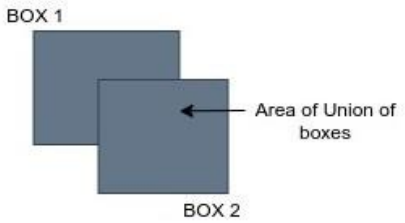
$$(x^B_{max} - x^B_{min}) * (y^B_{max} - y^B_{min})$$



$$(x^A_{max} - x^A_{min}) * (y^A_{max} - y^A_{min})$$

IOU =





$$\frac{|A \cap B|}{|A \cup B|}$$

Tính diện tích của 2 box A, B

area1 = $(x^A_{max} - x^A_{min}) * (y^A_{max} - y^A_{min})$

area2 = $(x^B_{max} - x^B_{min}) * (y^B_{max} - y^B_{min})$

Tìm tọa đọ của vùng giao nhau (Intersection)

$$xA = \max(x^A_{min}, x^b_{min}) = x^b_{min}$$
$$yA = \max(y^A_{min}, y^b_{min}) = y^b_{min}$$

$$xB = \min(x^A_{max}, x^b_{max}) = x^A_{max}$$
$$yB = \min(y^A_{max}, y^b_{max}) = y^A_{max}$$

Tính diện tích vùng giao nhau

$W = xB - xA$

H = yB – yA

intersection_area = w*h

Tính diện tích phần hợp nhau

union_area = area1 + area2 - intersection_area

Dựa trên phần giao và phần hợp để tính IoU

IoU = intersection_area / union_area

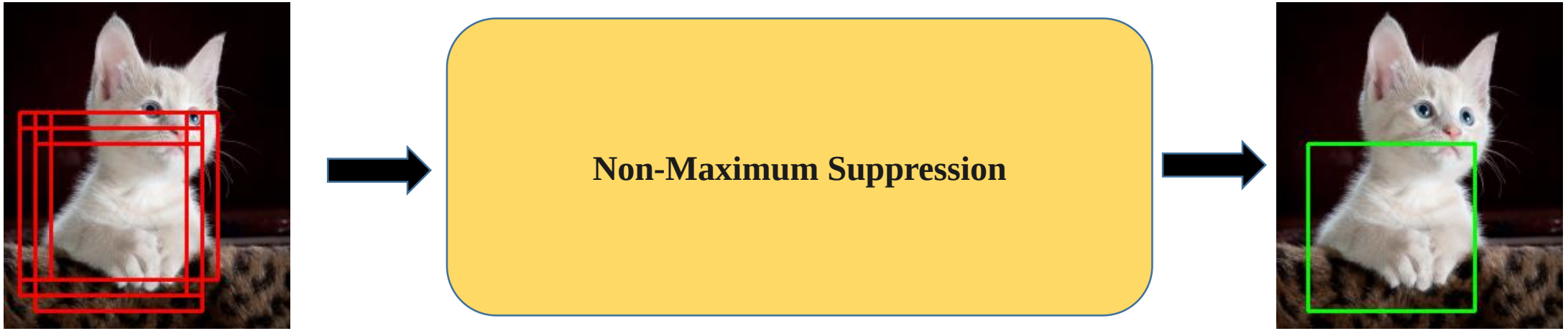
```python
def computeIoU(boxA, boxB):
  # determine the (x, y)-coordinates of the intersection rectangle
  xA = max(boxA[0], boxB[0])
  yA = max(boxA[1], boxB[1])
  xB = min(boxA[2], boxB[2])
  yB = min(boxA[3], boxB[3])

  # compute the area of intersection rectangle
  interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
  # compute the area of both the prediction and ground-truth
  # rectangles
  boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
  boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
  # compute the intersection over union by taking the intersection
  # area and dividing it by the sum of prediction + ground-truth
  # areas - the interesection area
  iou = interArea / float(boxAArea + boxBArea - interArea)
  # return the intersection over union value
  return iou
```

```python
boxA = (0,0,5,5)
boxB = (2.5,2.5,7.5,7.5)
iou = computeIoU(boxA,boxB)
print(iou)
```

```
0.20502092050209206
```

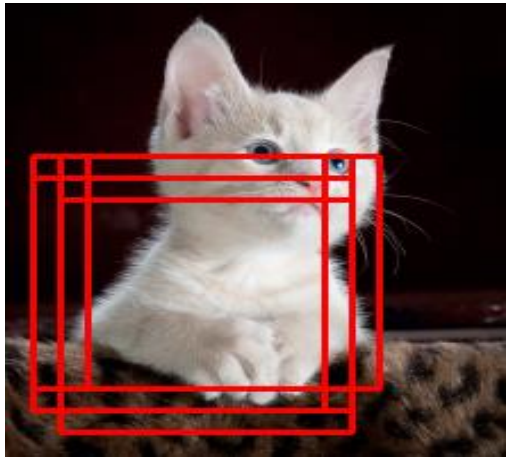
Case study:

(List and Tuple for Non-maximum Suppression)



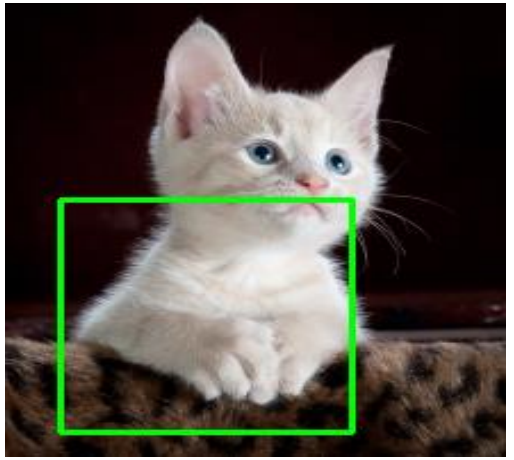
Input

List P of prediction BBoxes of the form (x1,y1,x2,y2,c), where (x1,y1) and (x2,y2) are the location of the BBox and c is the predicted confidence score of the model.



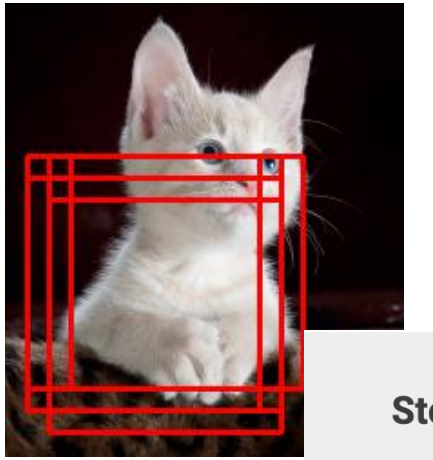
Ouput

Return a list keep of filtered prediction BBoxes.

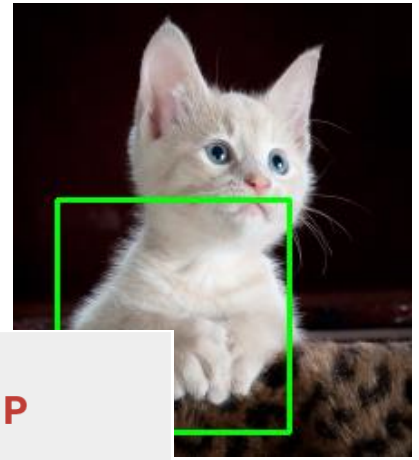


Case study

(List and Tuple for Non-maximum Suppression)



Non Maximum Suppression



Step 1 : Select the prediction **S** with highest confidence score and remove it from **P** and add it to the final prediction list **keep**. (**keep** is empty initially).

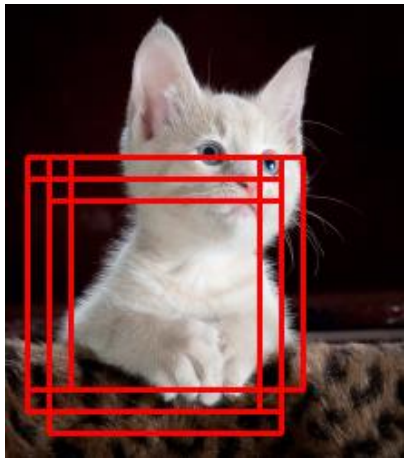
Step 2 : Now compare this prediction **S** with all the predictions present in **P**. Calculate the IoU of this prediction **S** with every other predictions in **P**. If the IoU is greater than the threshold **thresh_iou** for any prediction **T** present in **P**, remove prediction **T** from **P**.

Step 3 : If there are still predictions left in **P**, then go to **Step 1** again, else return the list **keep** containing the filtered predictions.

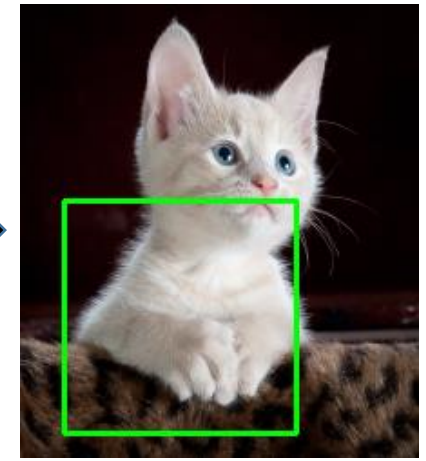
https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/

Case study

(List and Tuple for Non-maximum Suppression)



Non Maximum Suppression



List

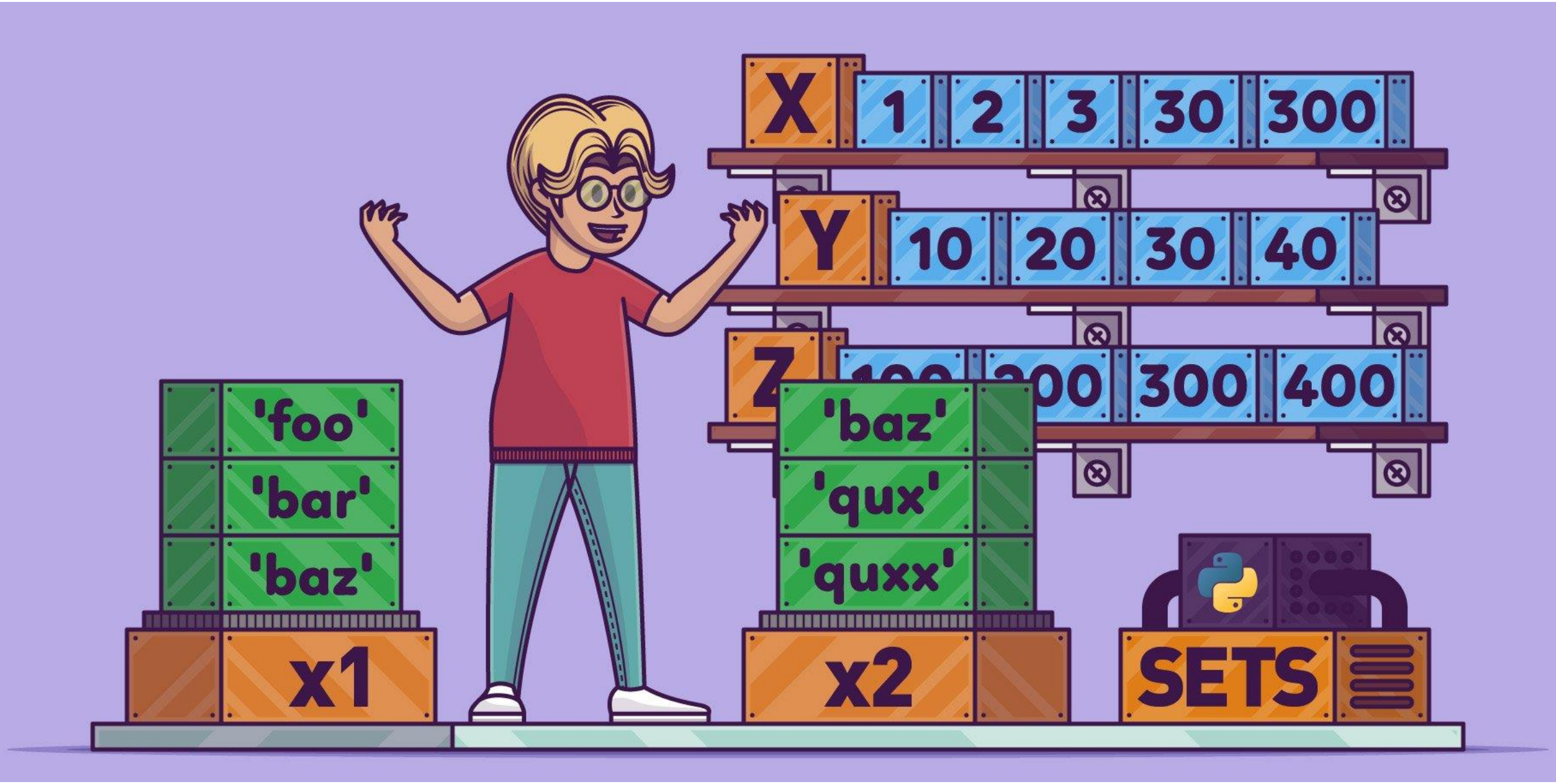
Tuple

```
# import the necessary packages
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

boundingBoxes = np.array([
  (12, 84, 140, 212, 0.3),
  (24, 84, 152, 212, 0.4),
  (36, 84, 164, 212, 0.5),
  (12, 96, 140, 224, 0.6),
  (24, 96, 152, 224, 0.7),
  (24, 108, 152, 236, 0.8)])
```

https://colab.research.google.com/drive/1hqSe6gRXhrp-S6G-kLC2arNuk4cQrHTA#scrollTo=-KCCLTpGxiqH

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Further study


X
1
2
3
30
300
Y
10
20
30
40
Z
300
400
'foo'
'bar'
'baz'
'baz'
'qux'
'quxx'
x1
x2
SETS

Problem in List

how to define a list that only includes unique elements

```
list = []
list.append(1)
list.append(1)
list.append(1)
list.append(2)
print(list)
```

```
[1, 1, 1, 2]
```

Problem in List

how to define a list that only includes unique elements

```
#how to define a list that only includes unique elements
def addElement(list, element):
  if(element not in list):
    list.append(element)
  return list

list = []
list = addElement(list, 1)
list = addElement(list, 1)
list = addElement(list, 1)
list = addElement(list, 2)
print(list)
```

```
[1, 2]
```

Are there any efficient data struture?

Problem in List

how to define a list that only includes unique elements

```
#how to define a list that only includes unique elements => set
set_test = set()
set_test.add(1)
set_test.add(1)
set_test.add(1)
set_test.add(2)
print(set_test)
```

```
{1, 2}
```

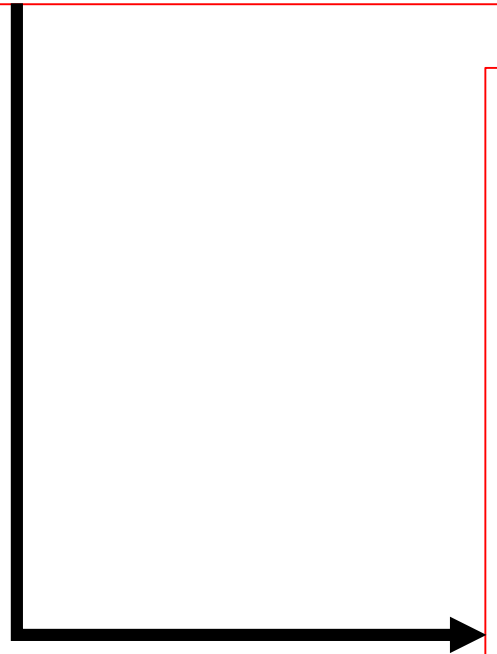
Use Set for storing data

data.txt

```
Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code
readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.  It supports
multiple programming paradigms, including structured, object-oriented and functional programming.
```

Case study 1:

Set in Finding List of Uniques Words Appearing in the Paragraph



```
0 supports
1 structured
2 and
3 typed
4 code
5 general
6 its
7 readability
8 garbage
9 collected
10 multiple
11 the
12 indentation
13 with
14 python
15 high
16 including
17 level
18 purpose
19 emphasizes
20 dynamically
21 paradigms
22 oriented
23 philosophy
24 significant
25 design
26 use
27 interpreted
28 object
29 of
30 language
31 it
32 a
33 is
34 functional
```

Set

❖ Create a set

Using curly brackets

```
# create a set
animals = {"cat", "dog", "tiger"}

print(type(animals))
print(animals)
```

```
<class 'set'>
{'dog', 'cat', 'tiger'}
```

Items with different data types

```
# create a set
a_set = {"cat", 5, True, 40.0}

print(type(a_set))
print(a_set)
```

```
<class 'set'>
{40.0, 'cat', 5, True}
```

Set comprehension

```
# set comprehension

a_set = {i*i for i in range(10)}
print(a_set)
```

```
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

https://colab.research.google.com/drive/1GDLtw5mI0ZC6y-lBqJZXw3khkfee_lrY?usp=sharing

Set

Access the items of a set

```
# accessing items
animals = {"cat", "dog", "tiger"}
for animal in animals:
    print(animal)
```

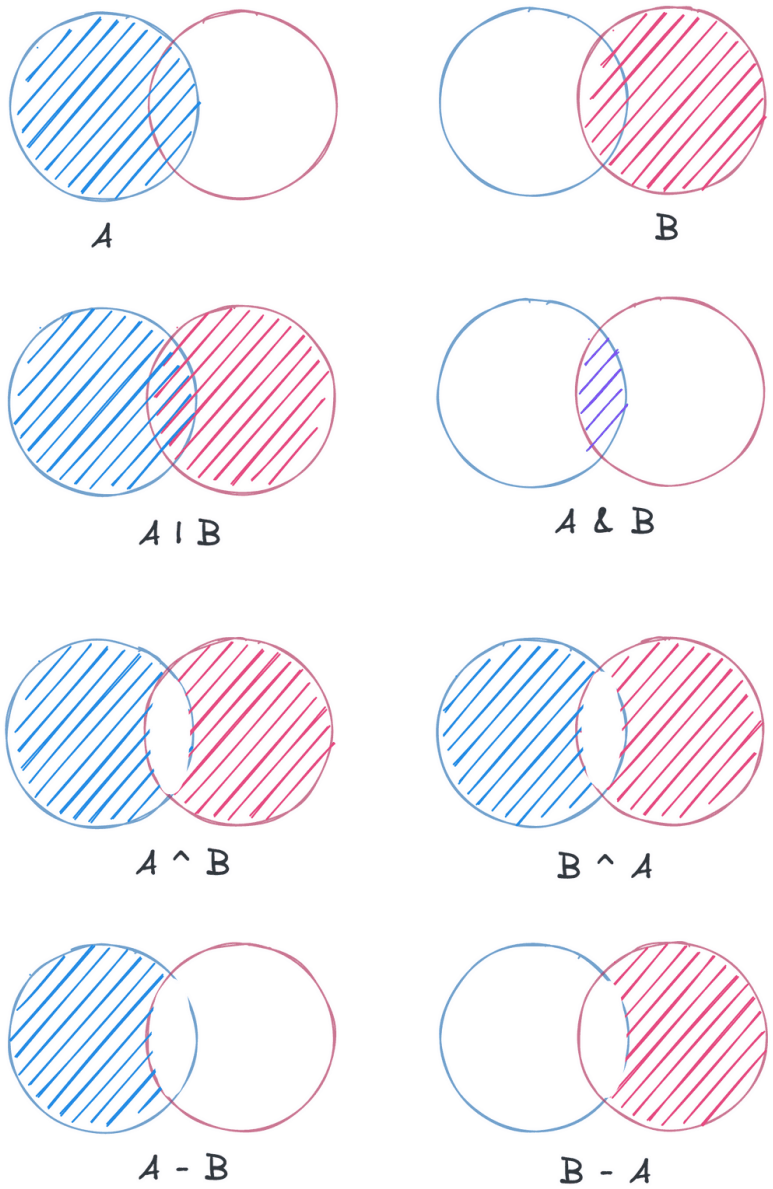
```
dog
cat
tiger
```

Copy a set

```
# copy
animals = {"cat", "dog", "tiger"}
print("Animals:", animals)

a_copy = animals.copy()
print("Copy:", a_copy)
```

```
Animals: {'dog', 'cat', 'tiger'}
Copy: {'dog', 'cat', 'tiger'}
```

List of the set operations available in Python



Set

Add an item

```
# add an item
animals = {"cat", "dog", "tiger"}
animals.add("bear")
print(animals)
```

```
{'dog', 'bear', 'cat', 'tiger'}
```

Insert a set to another set

```
# insert a set to another set
animals = {"cat", "dog", "tiger"}
animals.update({"chicken","Duck"})
print(animals)
```

```
{'Duck', 'tiger', 'dog', 'cat', 'chicken'}
```

Join two sets

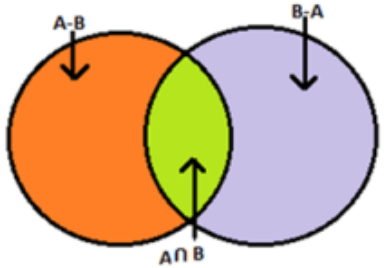
```
# join two sets
set1 = {"cat", "dog"}
set2 = {"duck", "tiger"}

set3 = set1.union(set2)
print(set3)
```

```
{'duck', 'dog', 'cat', 'tiger'}
```

Not allow duplicate values

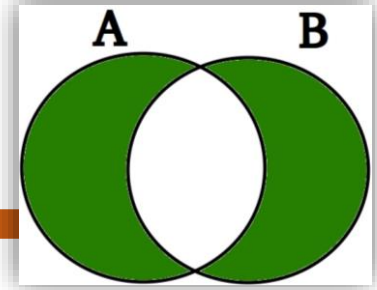
```
# No duplication
animals = {"cat", "dog", "tiger"}
print(animals)

animals.add("cat")
print(animals)
```

```
{'tiger', 'cat', 'dog'}
{'tiger', 'cat', 'dog'}
```



e

Set



difference function

```
# difference

set1 = {"apple", "banana", "cherry"}
set2 = {"pineapple", "apple"}

set3 = set1.difference(set2)

print(set3)
```

```
{'cherry', 'banana'}
```

symmetric_difference

```
# symmetric_difference

set1 = {"apple", "banana", "cherry"}
set2 = {"pineapple", "apple"}

set3 = set1.symmetric_difference(set2)

print(set3)
```

```
{'pineapple', 'cherry', 'banana'}
```

difference_update function

```
# difference_update

set1 = {"apple", "banana", "cherry"}
set2 = {"pineapple", "apple"}

set1.difference_update(set2)

print(set1)
```

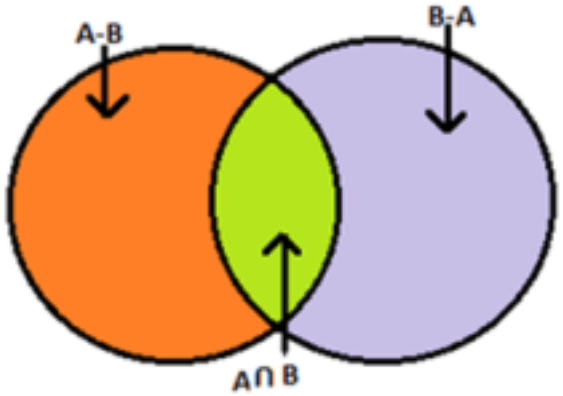
```
{'cherry', 'banana'}
```

symmetric_difference_update

```
# symmetric_difference_update

set1 = {"apple", "banana", "cherry"}
set2 = {"pineapple", "apple"}

set1.symmetric_difference_update(set2)

print(set1)
```

```
{'pineapple', 'cherry', 'banana'}
```

Set



❖ Bitwise operator

```python
# AND (&)

set1 = {1, 2, 3}
set2 = {3, 4, 5}

print(set1 & set2)
```

```
{3}
```

```python
# XOR (^)

set1 = {1, 2, 3}
set2 = {3, 4, 5}

print(set1 ^ set2)
```

```
{1, 2, 4, 5}
```

```python
# OR (|)

set1 = {1, 2, 3}
set2 = {3, 4, 5}

print(set1 | set2)
```

```
{1, 2, 3, 4, 5}
```

```python
# subtraction (-)

set1 = {1, 2, 3}
set2 = {3, 4, 5}

print(set1 - set2)
```

```
{1, 2}
```

Set

❖ Remove an item

remove(item)

Remove an item from the set.

```
# remove an item
animals = {"cat", "dog", "tiger"}
animals.remove("dog")
print(animals)
```

```
{'cat', 'tiger'}
```

discard(item)

Remove an item from the set if it is present.

```
# remove an item
animals = {"cat", "dog", "tiger"}
animals.discard("tiger")
print(animals)
```

```
{'dog', 'cat'}
```

Set comprehension

```
# set comprehension

aSet = {i*i for i in range(10)}
print(aSet)
```

```
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

https://docs.python.org/3/library/stdtypes.html?t#set

Set

❖ Remove an item

remove(item)

Remove an item from the set.

Raises KeyError if elem is not contained in the set.

```
1 # remove an item
2 animals = {"cat", "dog", "tiger"}
3 animals.remove("duck")
4 print(animals)
```

```
---------------------------------------
KeyError
<ipython-input-29-604e724f3515> in <module
      1 # remove an item
      2 animals = {"cat", "dog", "tiger"}
----> 3 animals.remove("duck")
      4 print(animals)

KeyError: 'duck'
```

discard(item)

Remove an item from the set if it is present.

```
1 # remove an item
2 animals = {"cat", "dog", "tiger"}
3 animals.discard("duck")
4 print(animals)
```

```
{'dog', 'cat', 'tiger'}
```

https://docs.python.org/3/library/stdtypes.html?t#set

Set

❖ Create a set

Unordered and unindexed

```
1  # not support indexing
2  animals = {"cat", "dog", "tiger"}
3  print(animals[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-a1b489f88deb> in <module>
      1 # not support indexing
      2 animals = {"cat", "dog", "tiger"}
----> 3 print(animals[1])

TypeError: 'set' object is not subscriptable
```

Trời ơi, lại 1 thuật ngữ mới? unhashable là gì đây ta

Cannot contain unhashable types

```
1  # create a set
2  a_list = [1, 2, 3]
3  a_set  = {"cat", a_list}
4  print(a_set)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-35efb5d3ad33> in <module>
      1 # shallow copy
      2 a_list = [1, 2, 3]
----> 3 a_set  = {"cat", a_list}
      4 print(a_set)

TypeError: unhashable type: 'list'
```

Previous topic
"Why is Python Installed on my Computer?" FAQ

Next topic
About these documents

This Page
Report a Bug
Show Source

hashable

An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

Hashability makes an object usable as a dictionary key and a set member, because these data structures use the hash value internally.

Most of Python's immutable built-in objects are hashable; mutable containers (such as lists or dictionaries) are not; immutable containers (such as tuples and frozensets) are only hashable if their elements are hashable. Objects which are instances of user-defined classes are hashable by default. They all compare unequal (except with themselves), and their hash value is derived from their `id()`.

A hashable Python object is any object that has a hash value — an integer identificator of that object which never changes during its lifetime. Most of Python's immutable built-in objects are hashable; mutable containers (such as lists or dictionaries) are not;

```
# hash for integer unchanged
print('Hash for 181 is:', hash(181))

# hash for decimal
print('Hash for 181.23 is:',hash(181.23))

# hash for string
print('Hash for Python is:', hash('Python'))
```

```
181

5303438921191261 97

5256052183123979819
```

Set

Set ←→ List and Tuple

```python
# convert from set to list
aSet = {1, 2, 3, 4, 5}

aList = list(aSet)
print(aList)
print(type(aList))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

```python
# convert from list to set
aList = [1, 2, 3, 2, 1]

aSet = set(aList)
print(aSet)
print(type(aSet))
```

???

```python
# convert from set to tuple
aSet = {1, 2, 3, 4, 5}

aTuple = tuple(aSet)
print(aTuple)
print(type(aTuple))
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
```

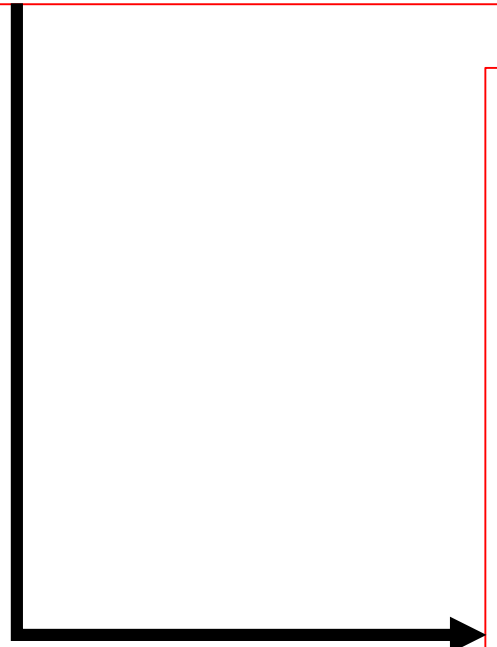
```python
# convert from tuple to set
aTuple = (1, 2, 3, 2, 1)

aSet = set(aTuple)
print(aSet)
print(type(aSet))
```

???

	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

data.txt

```
Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code
readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.  It supports
multiple programming paradigms, including structured, object-oriented and functional programming.
```



```
0 supports
1 structured
2 and
3 typed
4 code
5 general
6 its
7 readability
8 garbage
9 collected
10 multiple
11 the
12 indentation
13 with
14 python
15 high
16 including
17 level
18 purpose
19 emphasizes
20 dynamically
21 paradigms
22 oriented
23 philosophy
24 significant
25 design
26 use
27 interpreted
28 object
29 of
30 language
31 it
32 a
33 is
34 functional
```

Case study 1:

Finding a List of Uniques Words Appearing in the Paragraph

Set

❖ Convert text to numbers

```
# kết nối với file
a_file = open('data.txt','r')

# read content
data = a_file.read()
print(data)

# Đóng kết nối với file
a_file.close()
```

```
data = data.replace('.', '')
print(data)
```

```
data = data.replace(',', '')
print(data)
```

```
data = data.replace('-', ' ')
print(data)
```

```
data = data.lower()
print(data)
```

```
data = data.split()
print(data)
print(len(data))
```

```
data = set(data)
print(data)
print(len(data))
```

```
for index, value in enumerate(data):
    print(index, value)
```

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Case study

Problem in a List, Set

- We want to store following parameters for YOLO machine learning-based algorithms: batch size, learning rate, momentum, decay. Which data structure we can use?

```
name_list = ["learning_rate", "batch_size", "momentum", "decay"]
value_list = [0.001, 1, 0.9, 0.0005]

def getValueByName(parater_name, name_list, value_list):
  index = name_list.index(parater_name)
  return value_list[index]

learning_rate = getValueByName('learning_rate', name_list, value_list)
print(learning_rate)
```

```
0.001
```

Limitations by using List

Problem in a List, Set

- We want to store following parameters for YOLO machine learning-based algorithms: batch size, learning rate, momentum, decay. Which data structure we can use?

```
dict = {"learning_rate":0.001, "batch_size":1, "momentum":0.9, "decay":0.0005}
learning_rate = dict["learning_rate"]
print(learning_rate)
```

```
0.001
```

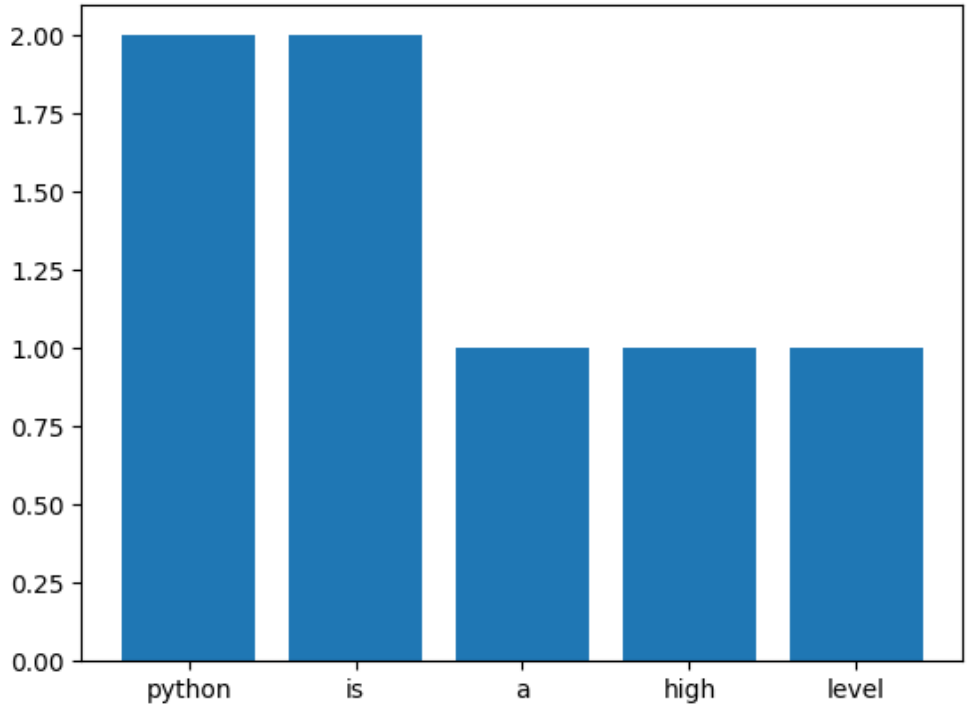
Solve by Dictionary

data.txt

```
Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code
readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.  It supports
multiple programming paradigms, including structured, object-oriented and functional programming.
```

Case study:

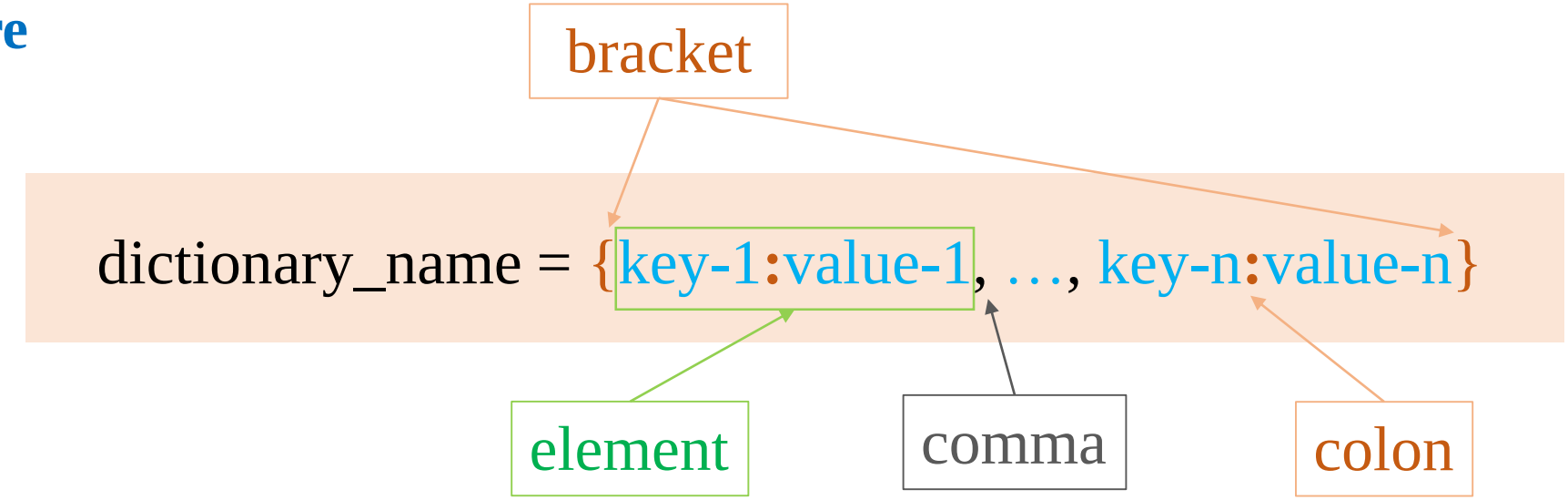
Find frequency of each word in a paragraph




```
{'python': 2, 'is': 2, 'a': 1, 'high': 1, 'level': 1, 'interpreted': 1, 'general': 1, 'purpose': 1, 'programming': 3
```

Dictionary

❖ Structure



Create a dictionary

```
1  parameters = {'learning_rate': 0.1,
2                'optimizer': 'Adam',
3                'metric': 'Accuracy'}
4
5  print(parameters)
6  print(type(parameters))
```

```
{'learning_rate': 0.1, 'optimizer': 'Adam', 'metric': 'Accuracy'}
<class 'dict'>
```

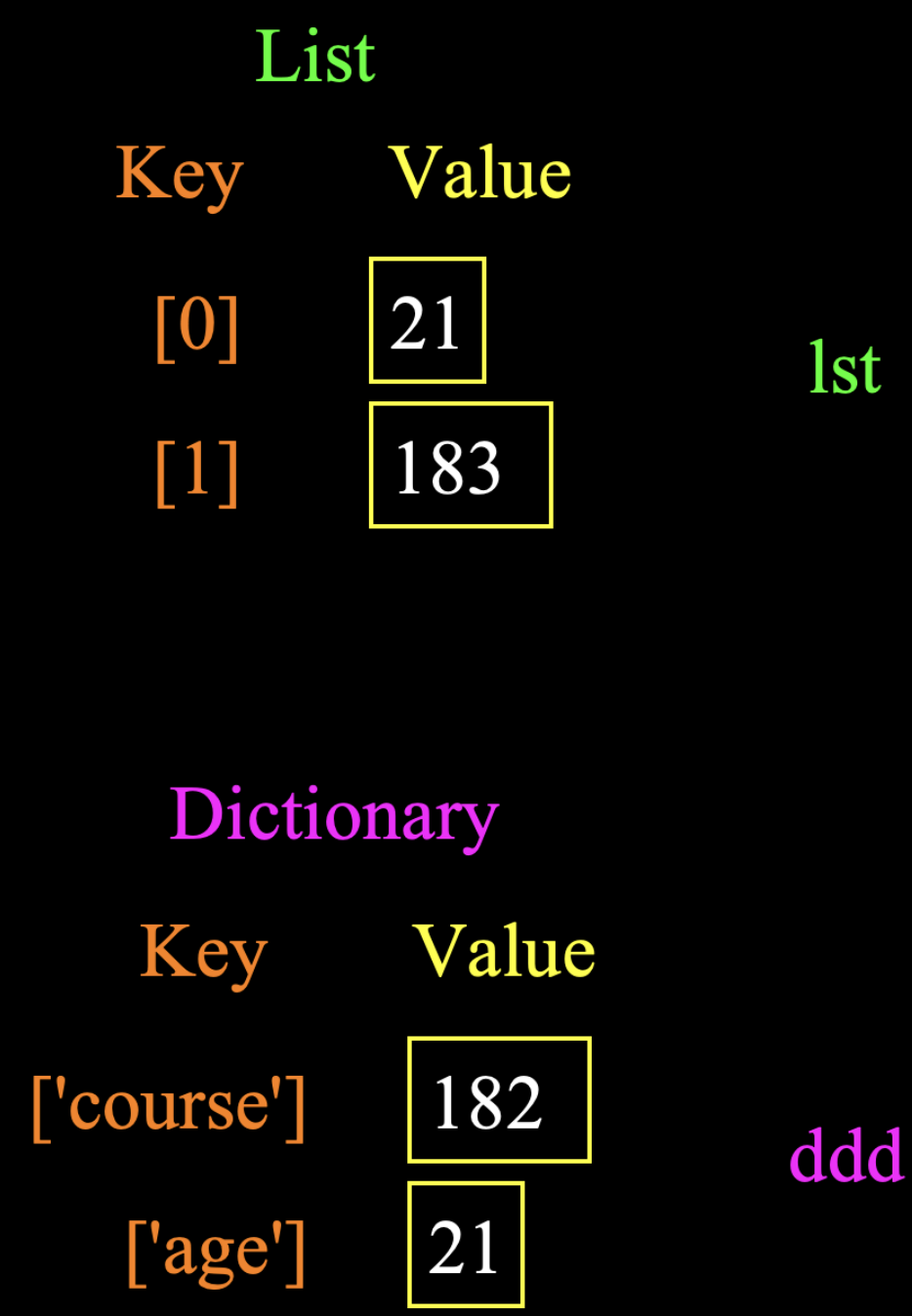
Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]

>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```



Dictionary

❖ Create a Dictionary

```python
# dic comprehension

a_dict = {str(i):i for i in range(5)}
print(a_dict)
```

```
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
```

```python
# from zip

tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

a_dict = dict(zip(tuple1, tuple2))
print(type(a_dict))
print(a_dict)
```

```
<class 'dict'>
{1: 4, 2: 5, 3: 6}
```

```python
# from zip

set1 = {1, 2, 3}
set2 = {4, 5, 6}

a_dict = dict(zip(set1, set2))
print(type(a_dict))
print(a_dict)
```

```
<class 'dict'>
{1: 4, 2: 5, 3: 6}
```

```python
# from zip

list1 = [1, 2, 3]
list2 = [4, 5, 6]

a_dict = dict(zip(list1, list2))
print(type(a_dict))
print(a_dict)
```

```
<class 'dict'>
{1: 4, 2: 5, 3: 6}
```

Dictionary

❖ Update a value

```
parameters = {'learning_rate': 0.1,
              'metric': 'Accuracy'}

parameters['learning_rate'] = 0.2
print(parameters)
```

```
{'learning_rate': 0.2, 'metric': 'Accuracy'}
```

❖ Copy a dictionary

```
parameters = {'learning_rate': 0.1,
              'metric': 'Accuracy'}

a_copy = parameters.copy()
a_copy['learning_rate'] = 0.2

print(parameters)
print(a_copy)
```

```
{'learning_rate': 0.1, 'metric': 'Accuracy'}
{'learning_rate': 0.2, 'metric': 'Accuracy'}
```

Dictionary

❖ Hàm copy() chỉ sao chép kiểu shallow

```
d1 = {'a': [1,2], 'b': 5}
d2 = d1.copy()

# thay đổi giá trị d2 sẽ ảnh hưởng đến d1
d2['a'][0] = 3
d2['a'][1] = 4

print('d1:', d1)
print('d2:', d2)
```

```
d1: {'a': [3, 4], 'b': 5}
d2: {'a': [3, 4], 'b': 5}
```

❖ Sử dụng hàm deepcopy() trong module copy

```
import copy

d1 = {'a': [1,2], 'b': 5}
d2 = copy.deepcopy(d1)

# thay đổi giá trị d2
d2['a'][0] = 3
d2['a'][1] = 4

print('d1:', d1)
print('d2:', d2)
```

```
d1: {'a': [1, 2], 'b': 5}
d2: {'a': [3, 4], 'b': 5}
```

Dictionary

❖ Get keys and values

Get keys

```
keys = parameters.keys()
for key in keys:
    print(key)
```

```
learning_rate
optimizer
metric
```

Get values

```
values = parameters.values()
for value in values:
    print(value)
```

```
0.1
Adam
Accuracy
```

Get keys

```
parameters = {'learning_rate': 0.1,
              'optimizer': 'Adam',
              'metric': 'Accuracy'}

for key in parameters:
    print(key)
```

```
learning_rate
optimizer
metric
```

Get keys and values

```
parameters = {'learning_rate': 0.1,
              'optimizer': 'Adam',
              'metric': 'Accuracy'}

items = parameters.items()
for key, value in items:
    print(key, value)
```

```
learning_rate 0.1
optimizer Adam
metric Accuracy
```

Dictionary

❖ Get a value by a key

Get value using get() function

```
parameters = {'learning_rate': 0.1,
              'optimizer': 'Adam',
              'metric': 'Accuracy'}

value = parameters.get('learning_rate')
print(value)

print('\nAfter using get() function')
print(parameters)
```

```
0.1

After using get() function
{'learning_rate': 0.1, 'optimizer': 'Adam', 'metric': 'Accuracy'}
```

Get value and delete the corresponding item

```
parameters = {'learning_rate': 0.1,
              'optimizer': 'Adam',
              'metric': 'Accuracy'}

value = parameters.pop('learning_rate')
print(value)

print('\nAfter using pop() function')
print(parameters)
```

```
0.1

After using pop() function
{'optimizer': 'Adam', 'metric': 'Accuracy'}
```

Dictionary

popitem() - lấy ra một phần tử ở cuối dictionary

```
parameters = {'learning_rate': 0.1,
              'optimizer': 'Adam',
              'metric': 'Accuracy'}

item = parameters.popitem()

print(item)
print(parameters)
```

```
('metric', 'Accuracy')
{'learning_rate': 0.1, 'optimizer': 'Adam'}
```

clear() - xóa tất cả các phần tử của một dictionary

```
parameters = {'learning_rate': 0.1,
              'metric': 'Accuracy'}

print('Before using clear() function')
print(parameters)

parameters.clear()

print('\nAfter using clear() function')
print(parameters)
```

```
Before using clear() function
{'learning_rate': 0.1, 'metric': 'Accuracy'}

After using clear() function
{}
```

Use del keyword to delete an item

```
parameters = {'learning_rate': 0.1,
              'metric': 'Accuracy'}
print(parameters)

del parameters['metric']
print(parameters)
```

```
{'learning_rate': 0.1, 'metric': 'Accuracy'}
{'learning_rate': 0.1}
```

Dictionary

❖ Key that does not exist

Try to delete a non-existing item

```
1  parameters = {'learning_rate': 0.1,
2                'metric': 'Accuracy'}
3
4  del parameters['algorithm']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback
<ipython-input-29-e759e1753a28> in <module>
      2                'metric': 'Accuracy'}
      3
----> 4 del parameters['algorithm']

KeyError: 'algorithm'
```

Try to get an item by a non-existing key

```
1  parameters = {'learning_rate': 0.1,
2                'optimizer': 'Adam',
3                'metric': 'Accuracy'}
4
5  value = parameters.pop('algorithm')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback
<ipython-input-30-8310550c04f4> in <module>
      3                'metric': 'Accuracy'}
      4
----> 5 value = parameters.pop('algorithm')

KeyError: 'algorithm'
```

Dictionary

```
dict.setdefault(key[, default_value])
```

setdefault() function

```
1  # setdefault()
2
3  fruits = {'banana': 2}
4  fruits.setdefault('apple', 0)
5
6  print(fruits)
```

```
{'banana': 2, 'apple': 0}
```

```
1  # setdefault()
2
3  fruits = {'banana': 2, 'apple': 4}
4  fruits.setdefault('apple', 0)
5
6  print(fruits)
```

```
{'banana': 2, 'apple': 4}
```

example

```
1  # setdefault()
2
3  fruits = {'banana': 2}
4  fruits.setdefault('apple', 0)
5
6  fruits['apple'] += 10
7  print(fruits)
```

```
{'banana': 2, 'apple': 10}
```

Result ???

```
1  # setdefault()
2
3  fruits = {'banana': 2}
4
5  fruits['apple'] += 10
6  print(fruits)
```

Dictionary

❖ Get a value via a key

Method 1

```
1  # access value via key
2
3  fruits = {'banana': 2, 'apple': 4}
4  print(fruits['apple'])
5  print(fruits['corn'])
```

```
4

---------------------------------------------------------------------------
KeyError                                  Traceback
<ipython-input-21-bda9d1f64a8f> in <module>
      3 fruits = {'banana': 2, 'apple': 4}
      4 print(fruits['apple'])
----> 5 print(fruits['corn'])

KeyError: 'corn'
```

Method 2

```
1  # access value via key
2
3  fruits = {'banana': 2, 'apple': 4}
4  print(fruits.get('apple'))
5  print(fruits.get('corn'))
```

```
4
None
```

Dictionary

Merge two dictionaries

```python
# merge two dicts

fruits = {'banana': 2, 'apple': 4}
cereal = {'rice': 3, 'corn': 7}

result = {**fruits, **cereal}
print(result)
```

```
{'banana': 2, 'apple': 4, 'rice': 3, 'corn': 7}
```

Check if a key exists

```python
# check if a key exists

fruits = {'banana': 2, 'apple': 4}

print('apple' in fruits)
print('corn' in fruits)
```

```
True
False
```

Remove empty items

```python
# remove empty items

fruits = {'banana': 2, 'apple': None}

dict1 = {key:value for (key, value)
                    in fruits.items()
                    if value is not None}
print(dict1)
```

```
{'banana': 2}
```

Dictionary comprehension

```python
# dic comprehension

aDict = {str(i):i for i in range(5)}
print(aDict)
```

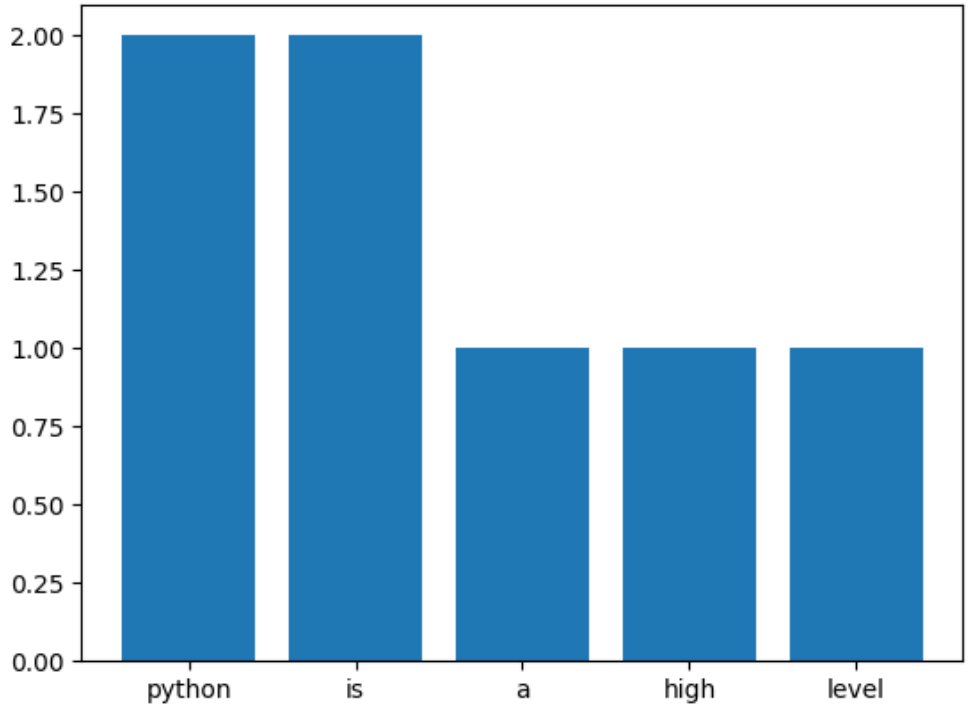
```
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
```

data.txt

```
Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code
readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.  It supports
multiple programming paradigms, including structured, object-oriented and functional programming.
```

Case study:

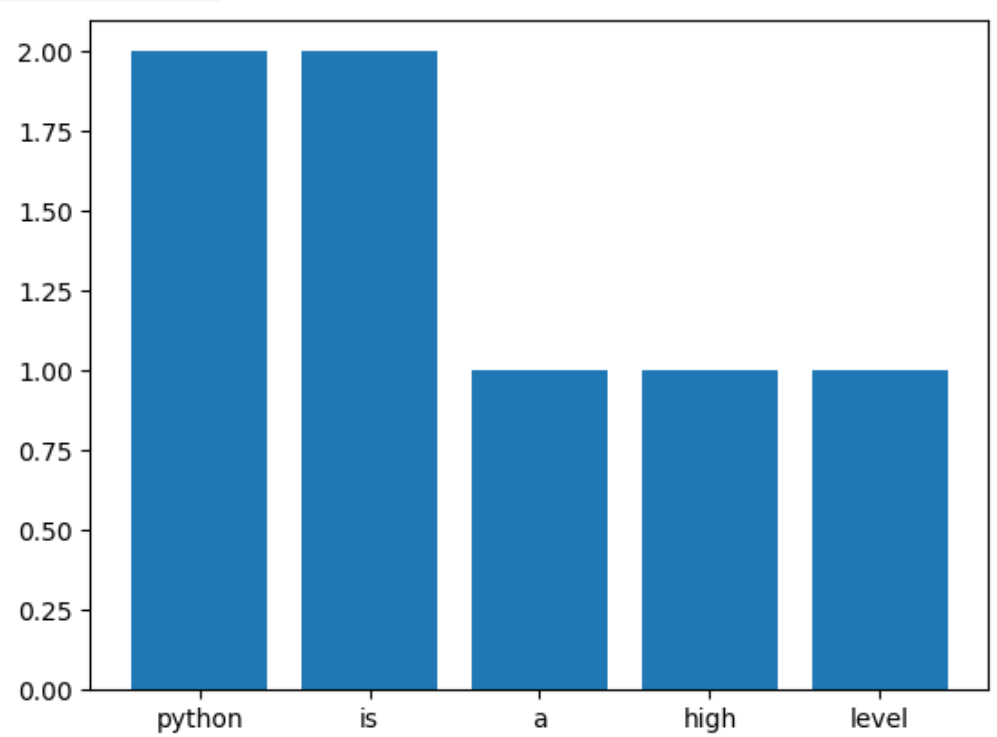
Find frequency of each word in a paragraph



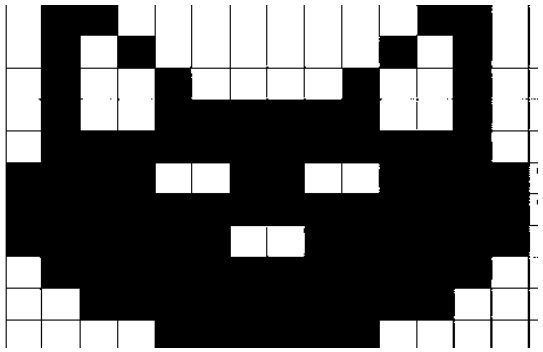

```
{'python': 2, 'is': 2, 'a': 1, 'high': 1, 'level': 1, 'interpreted': 1, 'general': 1, 'purpose': 1, 'programming': 3
```

```python
# kết nối với file
a_file = open('/content/drive/MyDrive/AI2023/data.txt','r')

# read content
data = a_file.read()
print(data)

# Đóng kết nối với file
a_file.close()
```

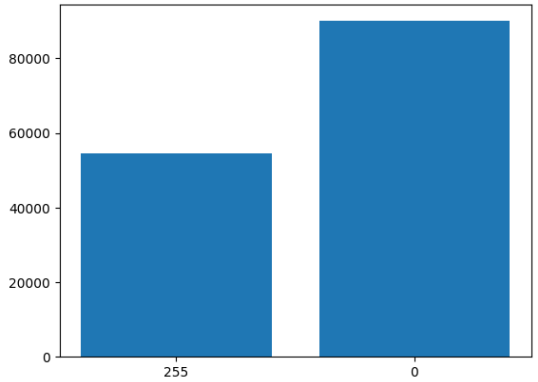
```python
import matplotlib.pyplot as plt

data = data.replace('.', '')
data = data.replace(',', '')
data = data.replace('-', ' ')
data = data.lower()
data = data.split()
dictionary = {}
for word in data:
  dictionary[word] = dictionary.get(word,0) + 1


names = list(dictionary.keys())
values = list(dictionary.values())
plt.bar(range(5), values[:5], tick_label=names[:5])
```



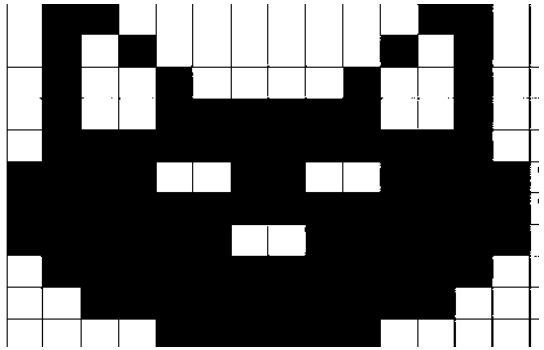
Case study:
(Dictionary in Image Histogram)



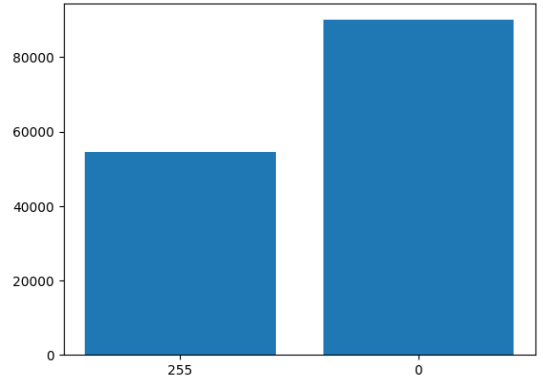
Histogram Algorithm



Case study:
(Dictionary in Image Histogram)



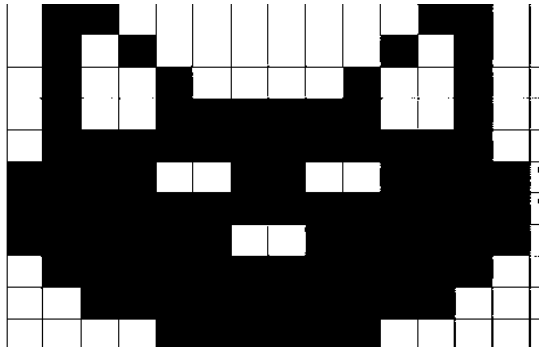
Histogram Algorithm

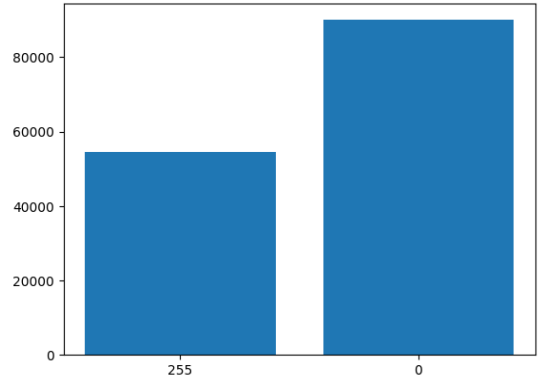


255	0	255	255	255	0	255
255	0	0	255	255	0	255
255	0	0	0	0	0	255
0	0	255	255	255	0	0
0	0	255	255	255	0	0
255	0	255	255	255	0	255
255	255	0	0	0	255	255

```
cat_image = cv2.imread("/content/binary_cat.png",0)
cv2_imshow(cat_image)
```

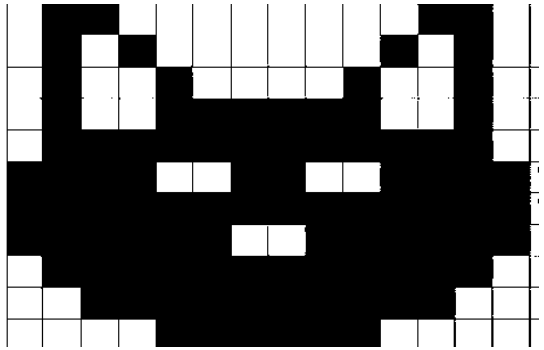
Case study:
(Dictionary in Image Histogram)

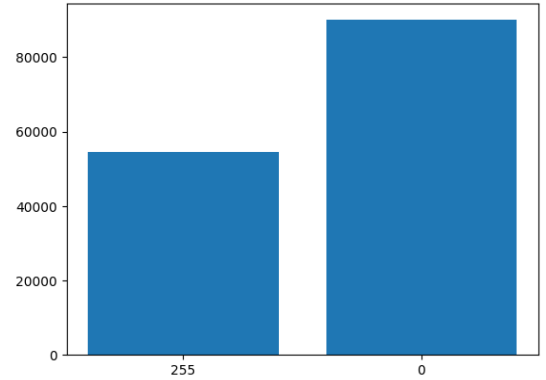





```
counts = dict()
height, width = cat_image.shape
for row in range(height):
  for col in range(width):
      counts[cat_image[row,col]] = counts.get(cat_image[row,col],0) + 1
```

Case study:
(Dictionary in Image Histogram)






```
import matplotlib.pyplot as plt

names = list(counts.keys())
values = list(counts.values())

plt.bar(range(len(counts)), values, tick_label=names)
plt.show()
```

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Further study

STRING	LIST	TUPLE
Immutable	Mutable	Immutable
Ordered/Indexed	Ordered/Indexed	Ordered/Indexed
Allows Duplicate Values	Allows Duplicate Values	Allows Duplicate Values
Empty string = " "	Empty list = [ ]	Empty tuple = ( )
String with single element = " A "	List with single item = [" Apple "]	Tuple with single item = (" Apple ",)
Stores characters or strings	It can store any data type str, list, set, tuple, int and dictionary	It can store any data type str, list, set, tuple, int and dictionary

Summarize

PYTHON TUPLES VS LISTS

TUPLES		LIST
The items are surrounded in parathesis ()	Syntax	The items are surrounded in square brackets []
Tuples are immutable in nature	Mutability	List are mutable in nature
In dictionary, we can create keys using tuples	Usability	In dictionary, we can not use lists as keys

Summarize

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	[] or list()	[5. 7, ‘yes’, 5.7]
Tuple	Yes	No	() or tuple()	(5.7, ‘yes’, 5.7)
Set	No	Yes	{} or set()	{5.7, ‘yes’ }
Dictionary	No	Yes	{} or dict{}	{‘key’: value}

Outline

- Tuple
- Set
- Dictionary
- Summarize
- Further study

Case study (at home): (Set in Text Classification)

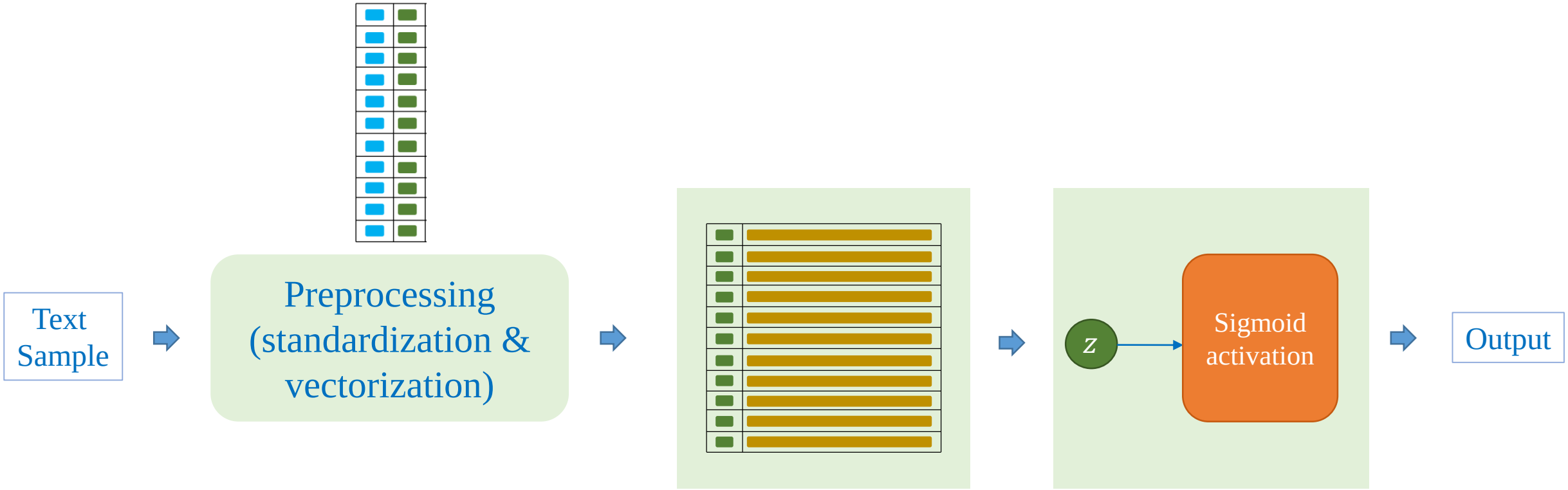
❖ Text classification

- 50,000 movie review for sentiment analysis (data)
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative = 1 – 1

"A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece….."	positive
"This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny anymore, and it's continued its decline further to the complete waste of time it is today…."	negative
"I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer)…."	positive
"BTW Carver gets a very annoying sidekick who makes you wanna shoot him the first three minutes he's on screen."	negative

Case study (at home): (Set in Text Classification)

❖ Text classification



Embedding

- Example corpus

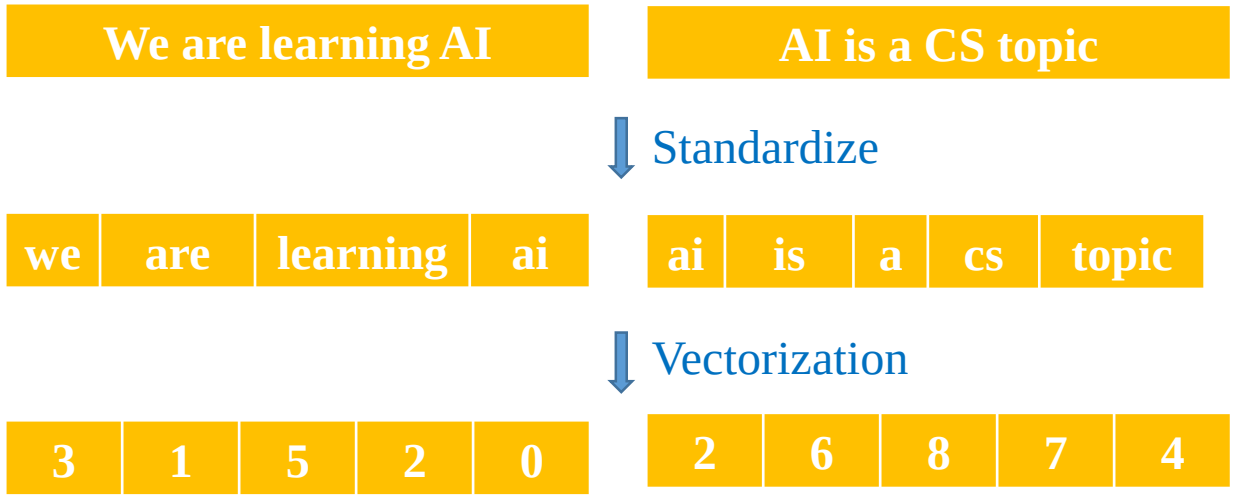
  sample1: ‘We are learning AI’

  sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

index	0	1	2	3	4	5	6	7	8
word	pad	are	ai	we	topic	learning	is	cs	a

(2) Transform text into features



Sorting

```
sorted(iterable, key=None, reverse=False)
```

```python
# create a list
aList = [1, 5, 3, 7, 4]
print(aList)

# sort
sortedList = sorted(aList)
print(sortedList)
```

```
[1, 5, 3, 7, 4]
[1, 3, 4, 5, 7]
```

item=7

```python
# create a list
aList = [1, 5, 3, 7, 4]
print(aList)

# function for sorting
def compare(item):
    return item

# sort
sortedList = sorted(aList, key=compare)
print(sortedList)
```

```
[1, 5, 3, 7, 4]
[1, 3, 4, 5, 7]
```

Sorting

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3)
print(list4)
```

```
[('a', 16), ('b', 15), ('e', 18), ('g', 13), ('h', 11)]
```

```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# function for sorting
def compare(item):
    return item[0]

# create
list3 = list(zip(list1, list2))
print(list3)
```

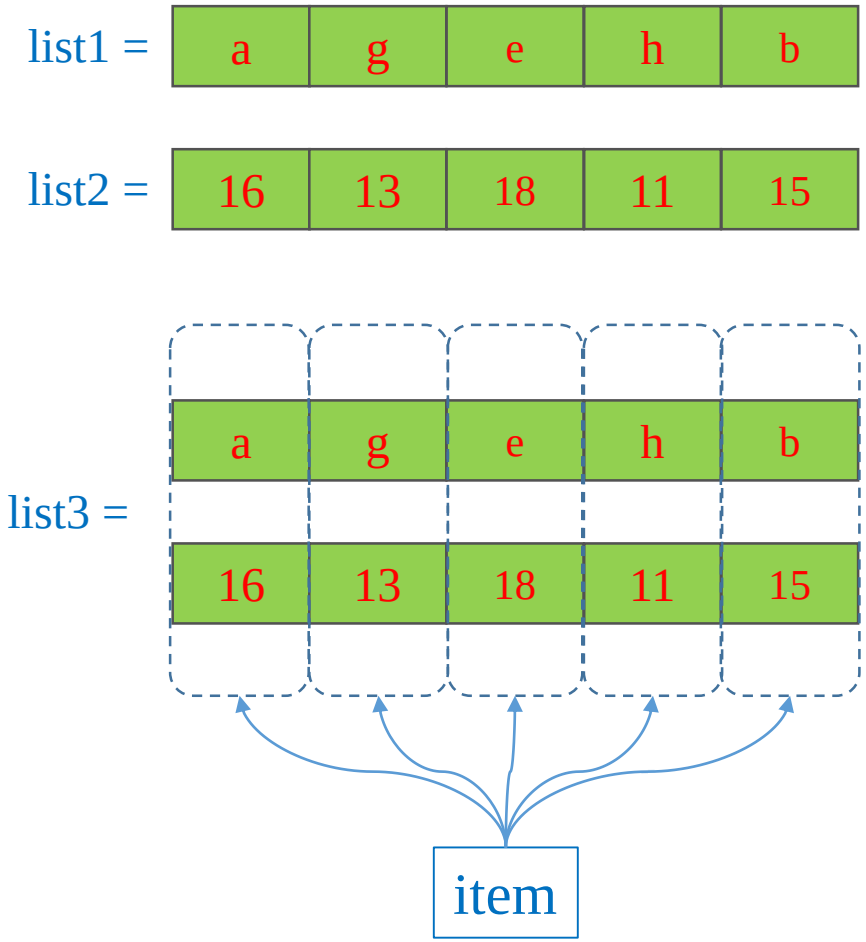
```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3, key=compare)
print(list4)
```

```
[('a', 16), ('b', 15), ('e', 18), ('g', 13), ('h', 11)]
```

AI VIETNAM
All-in-One Course

Sorting




```python
# data
list1 = ['a', 'g', 'e', 'h', 'b']
list2 = [16, 13, 18, 11, 15]

# function for sorting
def compare(item):
    return item[1]

# create
list3 = list(zip(list1, list2))
print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```python
list4 = sorted(list3, key=compare)
print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```

Lambda function

- Take any number of arguments
- Can only have one expression

Syntax

lambda arguments : expression

```
1  # lambda function
2  a_lfunction = lambda v: v + 10
3  print(a_lfunction(5))
```

```
15
```

```
1  # lambda function
2  a_lfunction = lambda v1, v2: v1+v2
3  print(a_lfunction(3, 4))
```

```
7
```

AI VIETNAM
All-in-One Course

Sorting

Using lambda function

```
1  # data
2  list1 = ['a', 'g', 'e', 'h', 'b']
3  list2 = [16, 13, 18, 11, 15]
4
5  # create
6  list3 = list(zip(list1, list2))
7  print(list3)
```

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```
1  list4 = sorted(list3, key=lambda item: item[1])
2  print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```

```
1  # data
2  list1 = ['a', 'g', 'e', 'h', 'b']
3  list2 = [16, 13, 18, 11, 15]
4
5  # function for sorting
6  def compare(item):
7      return item[1]
8
9  # create
10 list3 = list(zip(list1, list2))
11 print(list3)
```

item=('g', 13)

```
[('a', 16), ('g', 13), ('e', 18), ('h', 11), ('b', 15)]
```

```
1  list4 = sorted(list3, key=compare)
2  print(list4)
```

```
[('h', 11), ('g', 13), ('b', 15), ('a', 16), ('e', 18)]
```