**1. What is Iterators and generators**

|  |  |
| --- | --- |
| *Iterators* | *Generators* |
| Iterators are the objects that use the next() method to get the next value of the sequence. | A generator is a function that produces or yields a sequence of values using a yield statement. |
| Classes are used to Implement the iterators. | Functions are used to implement the generator. |
| Every iterator is not a generator. | Every generator is an iterator. |
| Complex implementation of iterator protocols .i.e., iter() and next(). | Generators in Python are simpler to code than do the custom iterator using the yield statement. |
| Iterators in python are less memory efficient. | Generators in Python are more memory efficient. |
| No local variables are used in Iterators. | All the local variables are stored before the yield statement. |

def PowerTwoGen( max=0 ):

   n = 1

   while n < max:

       yield 2 \*\* n

       n += 1

a = PowerTwoGen(6)

# Printing the values stored in a

for i in a:

   print(i)

What is a Python Constructor

A constructor is a special type of method (function) which is used to initialize the instance members of the class.

In C++ or Java, the constructor has the same name as its class, but it treats constructor differently in Python. It is used to create an object.

Constructors can be of two types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Constructor definition is executed when we create the object of this class. Constructors also verify that there are enough resources for the object to perform any start-up task.

class Addition:

    first = 0

    second = 0

    answer = 0

    # parameterized constructor

    def \_\_init\_\_(self, f, s):

        self.first = f

        self.second = s

    def display(self):

        print("First number = " + str(self.first))

        print("Second number = " + str(self.second))

        print("Addition of two numbers = " + str(self.answer))

    def calculate(self):

        self.answer = self.first + self.second

# creating object of the class

# this will invoke parameterized constructor

obj = Addition(1000, 2000)

# perform Addition

obj.calculate()

# display result

obj.display()

**map()** function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

**CODE 3**

|  |
| --- |
| #Add two lists using map and lambda  numbers1 = [1, 2, 3]  numbers2 = [4, 5, 6]  result = map(lambda x, y: x + y, numbers1, numbers2)  print(list(result)) |

Output :[5, 7, 9]

**How do you flatten an image(matrix) in a deep learning architecture**

**Impact of Image Flattening**

Flattening is a technique that is used to convert multi-dimensional arrays into a 1-D array, it is generally used in Deep Learning while feeding the 1-D array information to the classification model.

**What is the need for Flattening of an Image?**

Multi-Dimensional arrays take more amount of memory while 1-D arrays take less memory, which is the most important reason why we flatten the **Image Array**before processing/feeding the information to our model. In most cases, we will be dealing with a dataset which contains a large amount of images thus flattening helps in decreasing the memory as well as reducing the time to train the model.

**Step 1: Importing the necessary libraries**

import numpy as np

import pandas as pd

import cv2 as cv

from google.colab.patches import cv2\_imshow

from skimage import io

from PIL import Image

import matplotlib.pylab as plt

from numpy import array

from sys import getsizeof

#Fetching the url and showing the image using cv2\_imshow

urls=["<https://iiif.lib.ncsu.edu/iiif/0052574/full/800>,/0/default.jpg"]

for url in urls:

  image = io.imread(url)

  cv2\_imshow(image)

  print('\n')

#Getting the multi-dimensional array from the image

array1 = array(image)

#Memory occupied by the multi-dimensional array

size1 = getsizeof(array1)

print(array1)

#Using Flatten function on array 1 to convert the multi-dimensional

# array to 1-D array

array2 = array1.flatten()

#Memory occupied by array 2

size2 = getsizeof(array2)

#displaying the 1-D array

print(array2)

#Print's the two different size's of the array

print(f"Size of Multidimensional Image : {size1}")

print(f"Size of Flattened Image : {size2}")

difference = size1 - size2

#Print's the difference of memory between the size of Multidimensional & 1-D array

print("Size difference in the images: ", difference)

**Difference between semantic segmentation and segmentation**

<https://analyticsindiamag.com/semantic-vs-instance-vs-panoptic-which-image-segmentation-technique-to-choose/#:~:text=Semantic%20segmentation%20associates%20every%20pixel,class%20as%20distinct%20individual%20instances>.

 It segments the visual input in order to process it for tasks such as image classification and object detection.

different kinds of objects in the image

separate out occurrences of each object type

1. semantic segmentation -If you only want to group objects belonging to the same category, say distinguish all cars from all buildings, it is the task of semantic segmentation

2. Instance segmentation -Within each category say, people if you want to distinguish each individual person, that will be the task of instance segmentation

3. panoptic segmentation - Whereas if you want both category-wise as well as instance-wise division, it will be a panoptic segmentation task.

SOLO and SOLOv2 frameworks

1. Any countable entity such as a person, bird, flower, car etc. is termed as a **thing**.

**things** comes under **instance** segmentation since they can be assigned instance-level annotations

1. An uncountable amorphous region of identical texture such as the sky is termed as **stuff**.

**stuff** comes under **semantic** segmentation

Semantic segmentation associates every pixel of an image with a class label such as a person, flower, car and so on. It treats multiple objects of the same class as a single entity. In contrast, instance segmentation treats multiple objects of the same class as distinct individual instances.

panoptic segmentation assigns two labels to each of the pixels of an image – (i)semantic label (ii) instance id. The identically labelled pixels are considered belonging to the same semantic class and instance their id’s distinguish its instances.

## **Confidence scores required only for instance segmentation**

## **Evaluation metrics**

**For semantic segmentation, [IoU](https://arxiv.org/abs/1908.03851" \t "_blank), pixel-level accuracy and mean accuracy are commonly used metrics. These metrics ignore object-level labels while considering only those at pixel-level.**

For instance segmentation, AP (Average Precision) is taken as the standard metric. It requires the assignment of confidence score to each segment for estimation of a precision/recall curve. Confidence scores and hence AP cannot measure the output of semantic segmentation.

PQ (Panoptic Quality) used as a metric for panoptic segmentation equally treats all the classes – be it a thing or stuff. It must be noted that PQ is not a combination of semantic and instance segmentation metrics. SQ (i.e. average IoU of matched segments) and RQ (i.e. F1-Score) are computed for every class and measure segmentation and recognition quality, respectively. PQ is then calculated as (PQ = SQ \* RQ). It thus unifies evaluation over all the classes.