10/04/24

Today’s work

Today at 09.00 we started practicing python for strong basic knowledge.

Python Generators.

A generator is a special type of iterator that generates values on-the-fly using the yield keyword. Unlike regular functions that return a single value, generators can yield multiple values, suspending and resuming their execution state between each yield. This makes them useful for generating sequences of values without storing the entire sequence in memory.

Ex- def squares\_up\_to(limit):

    n = 1

    while n <= limit:

        yield n \*\* 2

        n += 1

# Using the generator

for square in squares\_up\_to(5):

   print(square)

output:

1

4

9

16

25

More eg

import random

def random\_numbers(count):

    for \_ in range(count):

        yield random.randint(1, 100)

# Using the generator

for num in random\_numbers(5):

    print(num)

output:

84

63

12

64

41

After that I made repository named 10-04-2024

Clone it git clone 10-04-2024

cd 10-04-2024

Then make two file 1 Generator.py, 2 Random.py wrote code in both file and saved it

git init

git add .

git commit -m "generator practice"

git push origin main

and with few steps my local file uploaded on my git hub account.

Around 11 to 1 I was in idp zoom meeting session its all about whole process until you reach in usa.

Around 2 I started Advanced Python Modules

Modules would be covered in this module

Collection

Os module and Datetime

Math and program

Python debugger

Timeit

Regular Expressions

Unzipping and zipping modules.

Collection modules:

The collection module in Python provides specialized container datatypes that are alternatives to the general-purpose built-in types (‘dic’,’list’,’set’, and ‘tuple’). These specialized types can be very useful in specific scenarios. Here are some of the main types provided by the collection module:

Namedtuple: is a factory function in Python's collection module that creates tuple subclasses with named fields. It provides a way to create tuple-like objects that have named attributes, making the code more readable and self-documenting.

from collections import namedtuple

# Create a named tuple class Point with fields x and y

Point = namedtuple('Point', ['x', 'y'])

# Create a Point object

p = Point(1, 2)

# Accessing fields using attribute names

print(p.x, p.y)

# Accessing fields using index

print(p[0], p[1])

# Unpacking the named tuple

x, y = p

print(x, y)

# Converting named tuple to dictionary

print(p.\_asdict())

output: 1 2

1 2

1 2

{'x': 1, 'y': 2}

Deque = Is stands for "double-ended queue," and it is a data structure provided by Python's collection module. It is a generalization of stacks and queues and supports fast appends and pops from both ends.

from collections import deque

# Creating a deque

d = deque([1, 2, 3, 4])

# Append to the right

d.append(5)

print(d)

# Append to the left

d.appendleft(0)

print(d)

# Pop from the right

print(d.pop())

# Pop from the left

print(d.popleft())

print(d)

ouput: deque([1, 2, 3, 4, 5])

deque([0, 1, 2, 3, 4, 5])

5

0

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

Chainmap is a class in Python's collection module that provides a way to link multiple mappings together such that they can be treated as a single unit. It is particularly useful when you need to search through multiple dictionaries as a single dictionary.

from collections import ChainMap

# Define two dictionaries

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

# Create a ChainMap with dict1 and dict2

chain\_map = ChainMap(dict1, dict2)

# Accessing keys

print(chain\_map['a'])

print(chain\_map['b'])

print(chain\_map['c'])

# Updating the first dictionary

dict1['a'] = 100

# ChainMap reflects the update

print(chain\_map['a'])

# In case want to access 2nd b value

value\_of\_2nd\_b = chain\_map.maps[1]['b']

print(value\_of\_2nd\_b)

output:

1

2

4

100

3

Counter : It is used to count the occurrences of elements in an iterable or to create a frequency distribution.

from collections import Counter

# Create a Counter from a list

list1 = ['a', 'b', 'c', 'a', 'b', 'a']

counter = Counter(list1)

# Accessing counts

print(counter['a'])

print(counter['b'])

print(counter['c'])

# Accessing all counts as a dictionary

print(dict(counter))

# Update counts

list2 = ['a', 'b', 'c', 'a', 'd']

counter.update(list2)

print(counter)

# Most common elements

print(counter.most\_common(2))

# Convert Counter object back to a list

updated\_list = list(counter.elements())

print(updated\_list)

output:

3

2

1

{'a': 3, 'b': 2, 'c': 1}

Counter({'a': 5, 'b': 3, 'c': 2, 'd': 1})

[('a', 5), ('b', 3)]

['a', 'a', 'a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd']

After this I start uploaded my all today’s stuff on my git hub account

cd 10-04-2024

git init

git add .

git commit -m "namedtuple, deque, chainmap, counter"

git push origin main