CONTEXT MANAGER & COLLECTION LIBRARY

BY-ADITYA RANJAN JHA

20EE10006

PART A- CONTEXT MANAGERS

>>WHY DO WE NEED CONTEXT MANAGER ?

>>HOW DO WE IMPLEMENT CONTEXT MANAGERS ?

>>WHAT MAKES IT A PROFITABLE CHOICE?

>>WHY DO WE NEED CONTEXT MANAGERS?

→IN A PUBLIC LIBRARY BOOKS ARE ISSUED AND THE RECORDS OF THE BOOK ISSUES ARE KEPT IN A REGISTER NOTEBOOK. NOW SUPPOSE IN HOLIDAYS THE DEMAND OF NOVELS INCREASED AND PEOPLE ISSUED NOVELS FOR A WEEK BUT DID NOT RETURN BECAUSE THE LIBRARY DOES NOT CHARGE ANY OVERDUE FINE[IT'S A PUBLIC LIBRARY :)] AND THEY COULD NOT RETURN THE BOOK DUE TO THEIR HECTIC SCHEDULE OR THEY MISPLACED THE BOOK OR WHATEVER SO REASON. NOW IF THE NUMBER OF DEFAULTERS WHO DID NOT CLOSE THEIR BOOK ISSUE ACCOUNTS KEEPS ON INCREASING THE LIBRARY SYSTEM WOULD SLOW DOWN IN ISSUING BOOKS DUE TO LACK OF RESOURCES AND ULTIMATELY IT WOULD COLLAPSE DUE TO LACK OF BOOKS.

→ VISUALIZE THIS EXAMPLE ABOVE AS AN ANALOGY IN PROGRAMMING

Books →Databases and file

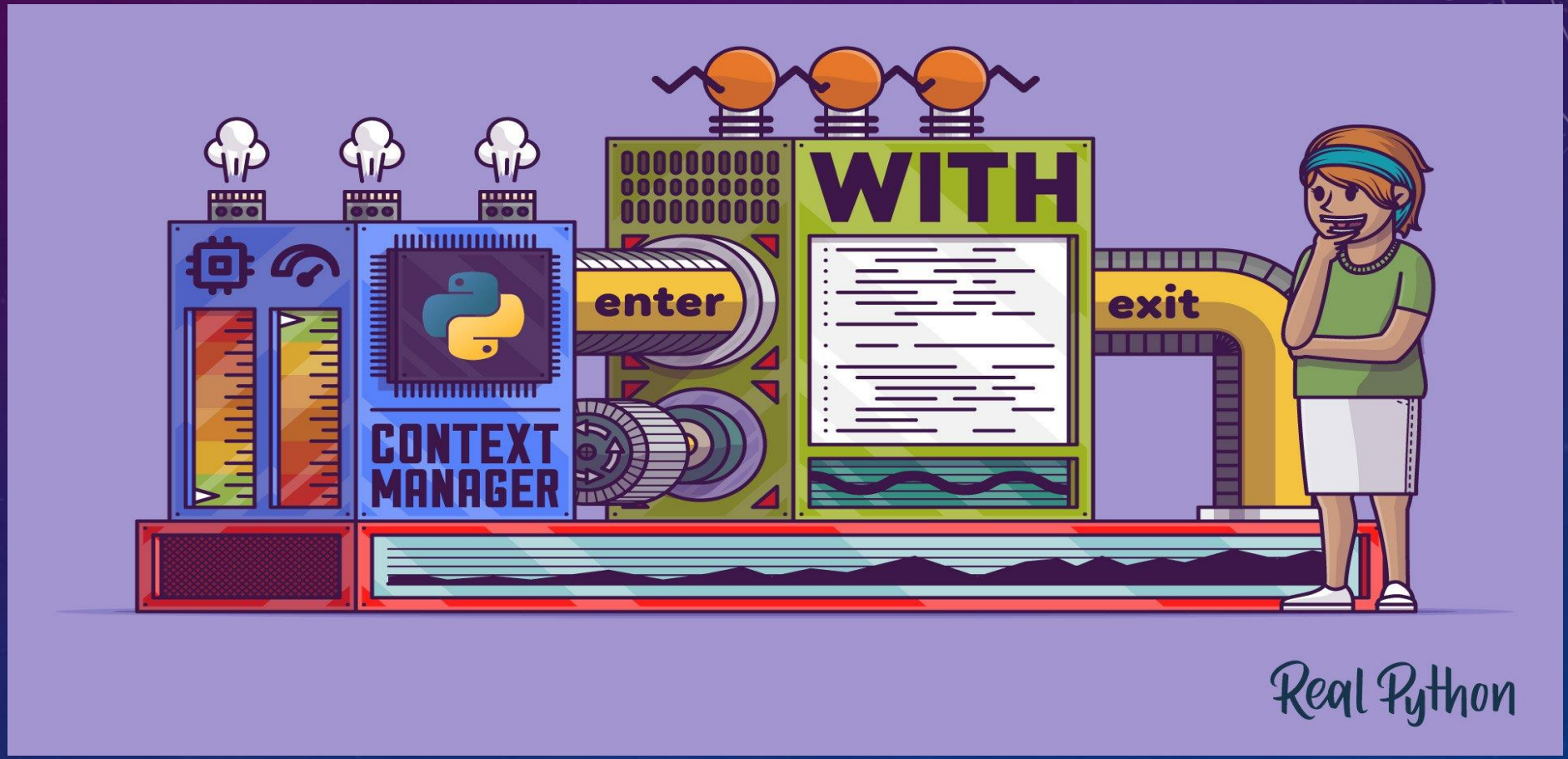
Library → File Management

>>WHY DO WE NEED CONTEXT MANAGERS?

→IN ANY PROGRAMMING LANGUAGE, THE USAGE OF RESOURCES LIKE FILE OPERATIONS OR DATABASE CONNECTIONS IS VERY COMMON. BUT THESE RESOURCES ARE LIMITED IN SUPPLY. THEREFORE, THE MAIN PROBLEM LIES IN MAKING SURE TO RELEASE THESE RESOURCES AFTER USAGE. IF THEY ARE NOT RELEASED THEN IT WILL LEAD TO RESOURCE LEAKAGE AND MAY CAUSE THE SYSTEM TO EITHER SLOW DOWN OR CRASH. IT WOULD BE VERY HELPFUL IF USER HAVE A MECHANISM FOR THE AUTOMATIC SETUP AND TEARDOWN OF RESOURCES

→**WITH PYTHON, IT CAN BE ACHIEVED BY THE USAGE OF CONTEXT MANAGERS WHICH FACILITATE THE PROPER HANDLING OF RESOURCES.**

>>HOW DO WE IMPLEMENT CONTEXT MANAGERS ?



>>How do we implement context managers ?

In Python context managers can be created in two ways:

- Using classes
- Using generators and decorators

>>THE HELLO WORLD EXAMPLE!

helloworld.py

```python
#manually opened the file then closed it after writing
f = open("test.txt","w")
f.write("hello world!")
#oops :( i forgot to close the file
print(f.closed)# showing status of file closure

#used python inbuilt context manager using "with" keyword
with open("cmtest.txt","w") as f1:
    data = f1.write("hello world!")
    #no need to close the file
print(f1.closed)# showing status of file closure
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> pyt
hon helloworld.py
False
True
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

cmtest.txt

```
hello world!
```

test.txt

```
hello world!
```

Spaces: 4 Python

>>IMPLEMENTATION: USING CLASSES

→AS WE SAW IN THE ABOVE HELLO WORLD! CODE THERE IS A INBUILT CONTEXT MANAGER FOR OPENING AND EDITING FILE AND THEN CLOSING THE FILE WHICH CAN BE IMPLEMENTED USING THE "WITH" KEYWORD .BUT WE CAN MAKE OUR OWN CONTEXT MANAGER USING 2 WAYS

→THE FIRST IS WITH THE HELP OF CLASSES

→ THE USER NEED TO ENSURE THAT THE CLASS HAS THE METHODS: __ENTER__() AND __EXIT__().

>>IMPLEMENTATION: USING CLASSES

class_contextmanager.py

```python
class open_file():
    def __init__(self,filename, mode):#will take the parameters to open file
        self.filename=filename
        self.mode = mode
        self.file = open(self.filename,self.mode)
        print("your file has been opened!")
        #file opened


    def __enter__(self):#will give access to user to work)
        return self.file

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.file.close()
        print("File is closed?")
        print(self.file.closed)
        #file closed


with open_file("class.txt", "w") as f:
    print("starting to write in file")
    f.write("Good to go to create ur custom context manager!")
```

class.txt

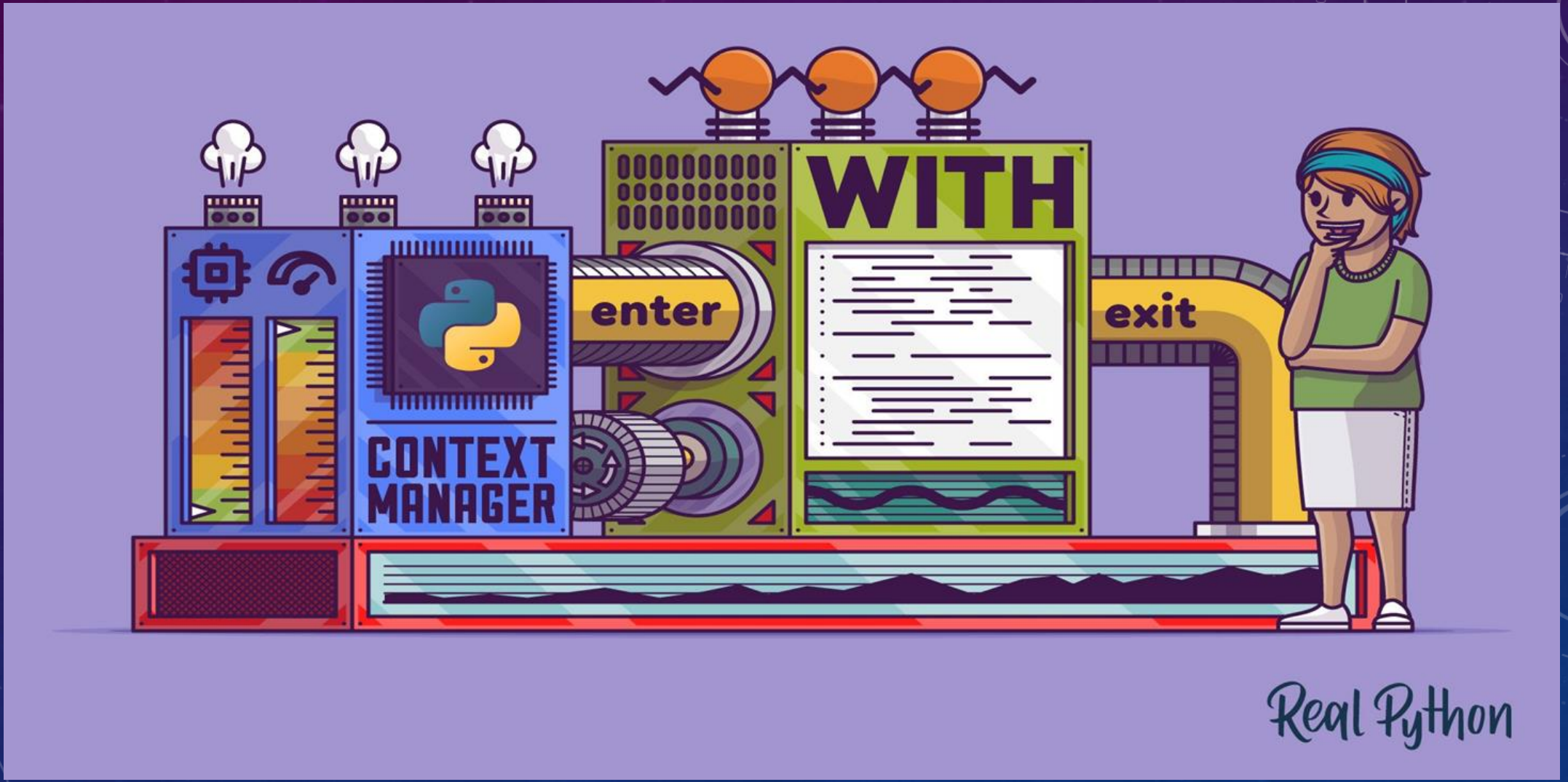
```
Good to go to create ur custom context manager!
```

Windows PowerShell

```
hon class_contextmanager.py
your file has been opened!
starting to write in file
File is closed?
True
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

>>THE PICTURE BELOW SHOWS EXACTLY THE FLOW OF EXECUTION



>>IMPLEMENTATION: USING GENERATORS

→ WE CAN SIMPLY MAKE ANY FUNCTION AS A CONTEXT MANAGER WITH THE HELP OF CONTEXTLIB.CONTEXTMANAGER DECORATOR WITHOUT HAVING TO WRITE A SEPARATE CLASS OR __ENTER__ AND __EXIT__ FUNCTIONS.

>>IMPLEMENTATION: USING GENERATORS

generator_contextmanager.py

```python
from contextlib import contextmanager

@contextmanager #decorator
def open_file(filename,mode):
    try:
        f = open(filename,mode)#file opened
        print("your file has been opened!")

        print("GRANTING U THE FILE TO WRITE")
        yield f
        print("Did u close my file?")


    finally:
        f.close()
        print(f.closed)


with open_file("gentest.txt","w") as f:
    f.write("Yeah generator method to create context manager works fine too")

#NOTE:
# The yield keyword pauses generator function execution and the value of the
# expression following the yield keyword is returned to the generator's
# caller. It can be thought of as a generator-based version of the return
# keyword. A yield , which causes the generator to once again pause and
# return the generator's new value.

# decorator is analogically equivalent to append method of list for a
# function or class. it wraps new defined function as a method to class.
```

Windows PowerShell

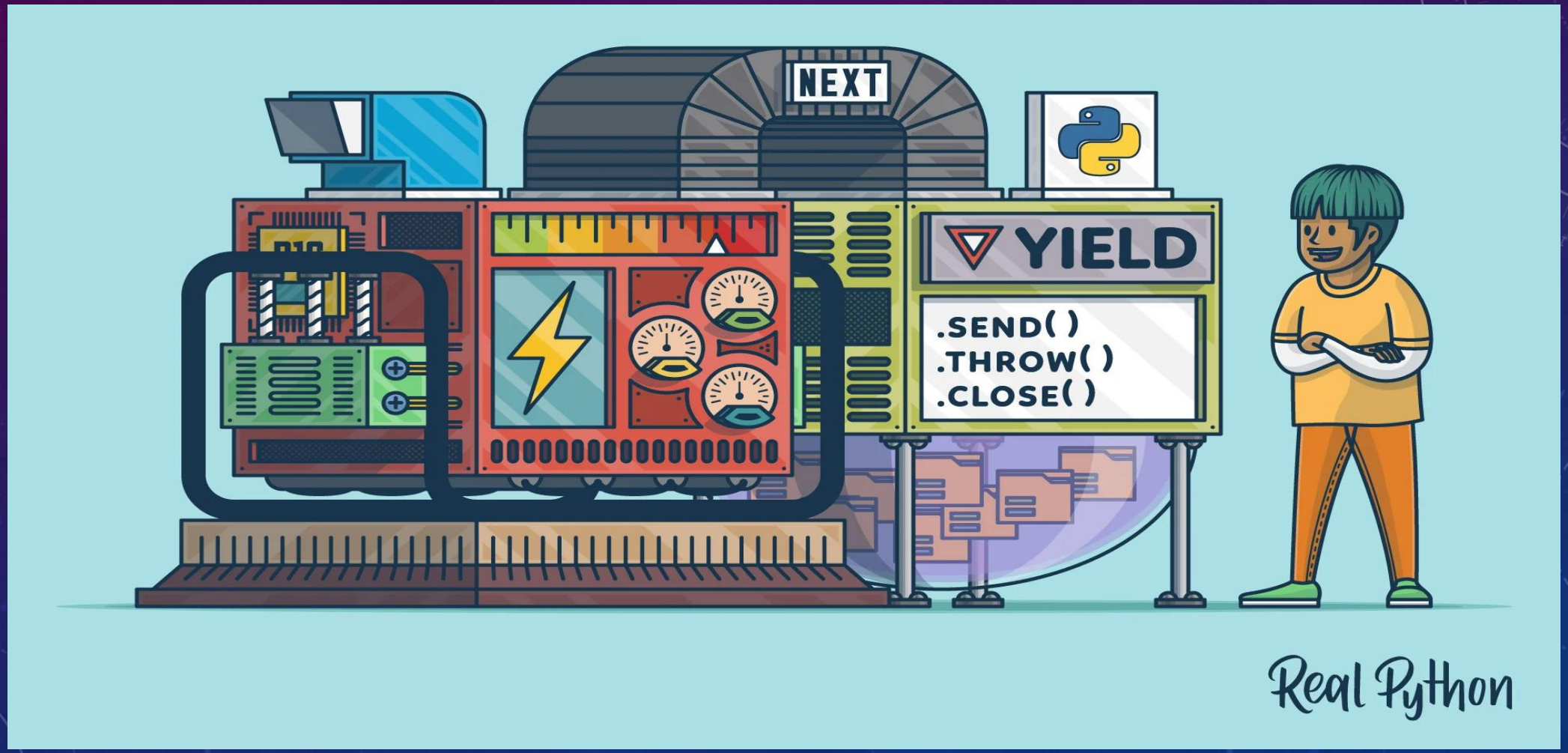
```
Copyright (C) Microsoft Corporation. All rig
hts reserved.

Try the new cross-platform PowerShell https:
//aka.ms/pscore6

PS C:\Users\adiso\OneDrive\Desktop\koss> pyt
hon generator_contextmanager.py
your file has been opened!
GRANTING U THE FILE TO WRITE
Did u close my file?
True
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

gentest.txt

```
Yeah generator
method to
create context
manager works
fine too
```

Spaces: 4 Python

>>THE PICTURE BELOW SHOWS EXACTLY THE FLOW OF EXECUTION FOR GENERATOR



>>CONTEXT MANAGER MERITS

→WE HAVE ALREADY SEEN DURING THE IMPLEMENTATION MOST OF THE BENEFITS LIKE FILE MANAGEMENT AND THE FILES BEING CLOSED EVEN IF THERE IS SOME ERROR WHILE WORKING IN THE FILE. WHAT WE HAVE SEEN IS ABOUT SMALL SCALE DATABASE(FILES FOR US)BUT WHAT HAPPENS WHEN THE NO OF FILES TO BE USED ARE TOO LARGE ?

→ SO HERE IS AN EXAMPLE FROM GEEKSFORGEEKS.ORG

Let's take the example of file management. When a file is opened, a file descriptor is consumed which is a limited resource. Only a certain number of files can be opened by a process at a time. The following program demonstrates it.

```
file_descriptors = []
for x in range(100000):
    file_descriptors.append(open('test.txt', 'w'))
```

Output:

```
Traceback (most recent call last):
  File "context.py", line 3, in
OSError: [Errno 24] Too many open files: 'test.txt'
```

An error message saying that too many files are open. The above example is a case of file descriptor leakage. It happens because there are too many open files and they are not closed. There might be chances where a programmer may forget to close an opened file.

PART B- COLLECTION LIBRARIES

>>INTRODUCTION

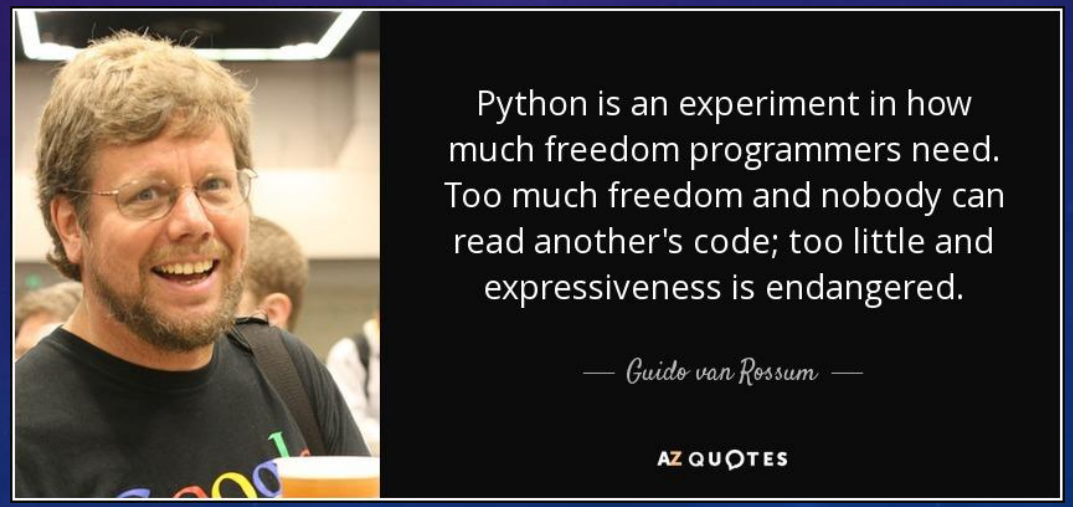
>>MEMBERS OF THE COLLECTION LIBRARIES

>>HOW IS IT AN UPGRADE FROM THE INBUILT CONTAINERS ?

>>FINAL EXAMPLE(TASK)

>>INTRODUCTION TO COLLECTION LIBRARY

→A CONTAINER IS AN OBJECT THAT IS USED TO STORE DIFFERENT OBJECTS AND PROVIDE A WAY TO ACCESS THE CONTAINED OBJECTS AND ITERATE OVER THEM. SOME OF THE BUILT-IN CONTAINERS ARE TUPLE, LIST, DICTIONARY, SET. APART FROM THIS PYTHON ALSO COMES WITH A BUILT-IN MODULE KNOWN AS COLLECTIONS WHICH HAS SPECIALIZED DATA STRUCTURES WHICH BASICALLY COVERS FOR THE SHORTCOMINGS OF THE FOUR DATA TYPES AND REDUCES YOUR NUMBER OF LINES OF EXTRA CODE.



DICTIONARY ↑ →COUNTER

→DICTIONARY IN PYTHON IS AN UNORDERED COLLECTION OF DATA VALUES, USED TO STORE DATA VALUES LIKE A MAP, WHICH UNLIKE OTHER DATA TYPES THAT HOLD ONLY SINGLE VALUE AS AN ELEMENT, DICTIONARY HOLDS KEY:VALUE PAIR IN AN UNORDERED FASHION.

→A COUNTER IS A SUB-CLASS OF THE DICTIONARY. IT IS USED TO KEEP THE COUNT OF THE ELEMENTS IN AN ITERABLE IN THE FORM OF AN UNORDERED DICTIONARY WHERE THE KEY REPRESENTS THE ELEMENT IN THE ITERABLE AND VALUE REPRESENTS THE COUNT OF THAT ELEMENT IN THE ITERABLE.

DICTIONARY → COUNTER

counter.py

```python
from collections import Counter

#An dictionary with integer keys of alphabet A B C
Dict = {1: 'B', 2: 'B', 3: 'B', 4: 'B', 5: 'B', 6: 'A' , 7: 'A' , 8: 'A' ,9: 'C' , 10: 'C'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

print("To find the count of values in above Dictionary")
print("We can create counters in 3 ways")

# With sequence of items
print("1st method")
coun = Counter()
coun.update(['B','B','A','B','C','A','B','B','A','C'])
print(coun)

# with dictionary
print("2nd method")
print(Counter({'A':3, 'B':5, 'C':2}))

# with keyword arguments
print("3rd method")
print(Counter(A=3, B=5, C=2))
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python counter.py

Dictionary with the use of Integer Keys:
{1: 'B', 2: 'B', 3: 'B', 4: 'B', 5: 'B', 6: 'A', 7: 'A', 8: 'A', 9: 'C', 10: 'C'
}
To find the count of values in above Dictionary
We can create counters in 3 ways
1st method
Counter({'B': 5, 'A': 3, 'C': 2})
2nd method
Counter({'B': 5, 'A': 3, 'C': 2})
3rd method
Counter({'B': 5, 'A': 3, 'C': 2})
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

DICTIONARY ↑ →ORDEREDDICT

→We know what a dictionary does (from previous slides)

→An ordereddict is also a sub-class of dictionary but unlike dictionary, it remembers the order in which the keys were inserted. While deleting and re-inserting the same key will push the key to the last to maintain the order of insertion of the key.

→As of the latest version of python dictionary(default container) is also ordered

DICTIONARY → ORDEREDDICT

counter.py | ordereddict.py

```python
from collections import OrderedDict
print("This is a Dict:\n")
d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
print('Before Deleting')
for key, value in d.items():
    print(key, value)
# deleting element
del d['a']
# Re-inserting the same
d['a'] = 1
print('\nAfter re-inserting')
for key, value in d.items():
    print(key, value)

print("\nThis is a ordered Dict:\n")
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3
od['d'] = 4
print('Before Deleting')
for key, value in od.items():
    print(key, value)
# deleting element
od.pop('a')
# Re-inserting the same
od['a'] = 1
print('\nAfter re-inserting')
for key, value in od.items():
    print(key, value)
```

Windows PowerShell

```
This is a Dict:

Before Deleting
a 1
b 2
c 3
d 4

After re-inserting
b 2
c 3
d 4
a 1

This is a ordered Dict:

Before Deleting
a 1
b 2
c 3
d 4

After re-inserting
b 2
c 3
d 4
a 1
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

DICTIONARY ↑ →DEFAULTDICT

→ A DEFAULTDICT IS ALSO A SUB-CLASS TO DICTIONARY. IT IS USED TO PROVIDE SOME DEFAULT VALUES FOR THE KEY THAT DOES NOT EXIST AND NEVER RAISES A KEY ERROR.

DICTIONARY → DEFAULTDICT

defaultdict.py

```python
from collections import defaultdict

# Defining the dict
dd = defaultdict(int)
#printing and empty defaultdict with key values that are not mapped
print("printing and empty defaultdict with key values that are not mapped")

for i in range(4):
    print(dd[i+1])
    # The default value is 0
    # so there is no need to
    # enter the key first

L = [1, 2, 3, 4, 2, 4, 1, 2]

for i in L:
    dd[i] += 1

print(dd)



d ={}
#printing and empty dict with key values that are not mapped
print("printing and empty dict with key values that are not mapped")

for i in range(4):
    print(d[i+1])
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python defaultdict.py
printing and empty defaultdict with key values that are not mapped
0
0
0
0
defaultdict(<class 'int'>, {1: 2, 2: 3, 3: 1, 4: 2})
printing and empty dict with key values that are not mapped
Traceback (most recent call last):
  File "C:\Users\adiso\OneDrive\Desktop\koss\defaultdict.py", line 28
, in <module>
    print(d[i+1])
KeyError: 1
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

N X (DICTIONARY) ↑ →CHAINMAP

→A CHAINMAP ENCAPSULATES MANY DICTIONARIES INTO A SINGLE UNIT AND RETURNS A LIST OF DICTIONARIES.

chainmap.py

```python
import collections

# initializing dictionaries
dic1 = { 'a' : 1, 'b' : 2 }
dic2 = { 'b' : 3, 'c' : 4 }
dic3 = { 'f' : 5 }

# initializing ChainMap
chain = collections.ChainMap(dic1, dic2)

# printing chainMap
print ("All the ChainMap contents are : ")
print (chain)

# using new_child() to add new dictionary
chain1 = chain.new_child(dic3)

# printing chainMap
print ("Displaying new ChainMap : ")
print (chain1)
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python chainmap.py
All the ChainMap contents are :
ChainMap({'a': 1, 'b': 2}, {'b': 3, 'c': 4})
Displaying new ChainMap :
ChainMap({'f': 5}, {'a': 1, 'b': 2}, {'b': 3, 'c': 4})
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

DICTIONARY →USERDICT

→USERDICT IS A DICTIONARY-LIKE CONTAINER THAT ACTS AS A WRAPPER AROUND THE DICTIONARY OBJECTS. THIS CONTAINER IS USED WHEN SOMEONE WANTS TO CREATE THEIR OWN DICTIONARY WITH SOME MODIFIED OR NEW FUNCTIONALITY.

UserDict.py

```python
from collections import UserDict
#plain dictionary
d={'a':1, 'b': 2, 'c': 3}
print("Original Dictionary")
print(d)
del d['a']
print("After deleting")
print(d)

# Creating a Dictionary where deletion is not allowed
class MyDict(UserDict):
    # Function to stop deleltion from dictionary
    def __del__(self):
        raise RuntimeError("Deletion not allowed")

    # Function to stop pop from dictionary
    def pop(self, s = None):
        raise RuntimeError("Deletion not allowed")

    # Function to stop popitem from Dictionary
    def popitem(self, s = None):
        raise RuntimeError("Deletion not allowed")

# Driver's code
ud = MyDict({'a':1, 'b': 2, 'c': 3})

print("Original  User-Dictionary")
print(ud)

ud.pop(1)
print(ud)
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python UserDict.py
Original Dictionary
{'a': 1, 'b': 2, 'c': 3}
After deleting
{'b': 2, 'c': 3}
Original  User-Dictionary
{'a': 1, 'b': 2, 'c': 3}
Traceback (most recent call last):
  File "C:\Users\adiso\OneDrive\Desktop\koss\UserDict.py", line
 30, in <module>
    ud.pop(1)
  File "C:\Users\adiso\OneDrive\Desktop\koss\UserDict.py", line
 18, in pop
    raise RuntimeError("Deletion not allowed")
RuntimeError: Deletion not allowed
Exception ignored in: <function MyDict.__del__ at 0x000002270D4
32040>
Traceback (most recent call last):
  File "C:\Users\adiso\OneDrive\Desktop\koss\UserDict.py", line
 14, in __del__
RuntimeError: Deletion not allowed
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

TUPLES →NAMEDTUPLE

→TUPLE IS A COLLECTION OF PYTHON OBJECTS MUCH LIKE A LIST. THE SEQUENCE OF VALUES STORED IN A TUPLE CAN BE OF ANY TYPE, AND THEY ARE INDEXED BY INTEGERS.

→A NAMEDTUPLE RETURNS A TUPLE OBJECT WITH NAMES FOR EACH POSITION INSTEAD OF INDEXES STARTING FROM ZERO, WHICH THE ORDINARY TUPLES LACK.

→FOR EXAMPLE, CONSIDER A TUPLE NAMES STUDENT WHERE THE FIRST ELEMENT REPRESENTS FNAME, SECOND REPRESENTS LNAME AND THE THIRD ELEMENT REPRESENTS THE DOB. SUPPOSE FOR CALLING FNAME INSTEAD OF REMEMBERING THE INDEX POSITION YOU CAN ACTUALLY CALL THE ELEMENT BY USING THE FNAME ARGUMENT, THEN IT WILL BE REALLY EASY FOR ACCESSING TUPLES ELEMENT. THIS FUNCTIONALITY IS PROVIDED BY THE NAMEDTUPLE.

TUPLES → NAMEDTUPLE

namedtuple.py

```python
from collections import namedtuple
#normal tuple
studenttuple = ('Aditya','16', '20EE10006')

# Declaring namedtuple()
Student_namedtuple = namedtuple('Student',['name','section','rollno'])

# Adding values
S = Student_namedtuple('Aditya','16','20EE10006')

# Access roll no from normal tuple
print ("The Student rollno using tuple is : ",end ="")
print (studenttuple[2])

# Access name from named tuple
print ("The Student name using namedtuple is : ",end ="")
print (S.name)
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python namedtuple.py
The Student rollno using tuple is : 20EE10006
The Student name using namedtuple is : Aditya
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

LIST →DEQUE

→LISTS ARE JUST LIKE DYNAMIC SIZED ARRAYS, DECLARED IN OTHER LANGUAGES (VECTOR IN C++ AND ARRAYLIST IN JAVA). LISTS NEED NOT BE HOMOGENEOUS ALWAYS WHICH MAKES IT A MOST POWERFUL TOOL IN PYTHON. A SINGLE LIST MAY CONTAIN DATATYPES LIKE INTEGERS, STRINGS, AS WELL AS OBJECTS. LISTS ARE MUTABLE, AND HENCE, THEY CAN BE ALTERED EVEN AFTER THEIR CREATION. NEW ELEMENTS CAN BE APPENDED AT THE END.

→DEQUE (DOUBLY ENDED QUEUE) IS THE OPTIMIZED LIST FOR QUICKER APPEND AND POP OPERATIONS FROM BOTH SIDES OF THE CONTAINER. IT PROVIDES O(1) TIME COMPLEXITY FOR APPEND AND POP OPERATIONS AS COMPARED TO LIST WITH O(N) TIME COMPLEXITY.

LIST → DEQUE

deque.py

```python
from collections import deque
lis = [1,2,3]
print ("The deque orignaly : ")
print (lis)

lis.remove(1)
lis.remove(3)
lis.insert(0,0)
lis.append(4)

print ("The list after operation is : ")
print (lis)


# initializing deque
de = deque([1,2,3])
print ("The deque orignaly : ")
print (de)

de.pop()#removes last element
de.popleft()#removes first element
de.append(4) # inserts 4 at the end of deque
de.appendleft(0) # inserts 0 at the beginning of deque

print ("The deque after operation is : ")
print (de)

```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python deque.
py
The deque orignaly :
[1, 2, 3]
The list after operation is :
[0, 2, 4]
The deque orignaly :
deque([1, 2, 3])
The deque after operation is :
deque([0, 2, 4])
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Tab Size: 4 Python

LIST → USERLIST

→USERLIST IS A LIST LIKE CONTAINER THAT ACTS AS A WRAPPER AROUND THE LIST OBJECTS. THIS IS USEFUL WHEN SOMEONE WANTS TO CREATE THEIR OWN LIST WITH SOME MODIFIED OR ADDITIONAL FUNCTIONALITY.

UserList.py

```python
from collections import UserList
# Creating a List where
# deletion is not allowed
class MyList(UserList):
    # Function to stop deleltion from List
    def remove(self, s = None):
        raise RuntimeError("Deletion not allowed")

    # Function to stop pop from List
    def pop(self, s = None):
        raise RuntimeError("Deletion not allowed")

UL = MyList([1, 2, 3, 4])
L = [1, 2, 3, 4]
print("Original List")
print(L)
print("Original User List")
print(UL)
# Inserting to List"
UL.append(5)
L.append(5)

print("After Insertion in list ")
print(L)
print("After Insertion in user list ")
print(UL)
print("Tryin to remove item from list")
# Deliting From List
L.remove(1)
print("Trying to remove item from user list")
UL.remove(1)# Deliting From userList
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python UserList.py
Original List
[1, 2, 3, 4]
Original User List
[1, 2, 3, 4]
After Insertion in list
[1, 2, 3, 4, 5]
After Insertion in user list
[1, 2, 3, 4, 5]
Tryin to remove item from list
Trying to remove item from user list
Traceback (most recent call last):
  File "C:\Users\adiso\OneDrive\Desktop\koss\UserList.py", line 31
, in <module>
    UL.remove(1)# Deliting From userList
  File "C:\Users\adiso\OneDrive\Desktop\koss\UserList.py", line 7,
 in remove
    raise RuntimeError("Deletion not allowed")
RuntimeError: Deletion not allowed
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

Spaces: 4 Python

STRING → USERSTRING

→STRINGS ARE THE ARRAYS OF BYTES REPRESENTING UNICODE CHARACTERS.

→USERSTRING IS A STRING LIKE CONTAINER AND JUST LIKE USERDICT AND USERLIST IT ACTS AS A WRAPPER AROUND STRING OBJECTS. IT IS USED WHEN SOMEONE WANTS TO CREATE THEIR OWN STRINGS WITH SOME MODIFIED OR ADDITIONAL FUNCTIONALITY.

UserList.py | Userstring.py

```python
from collections import UserString


# Creating a custom char match string that is mutable
class Mystring(UserString):
    def append(self, s):# Function to append to string
        self.data += s
    def match_char(self, i, s):# Function to match charecter
        return self.data[i] == s
    def remove(self, s):# Function to rmeove from
        self.data = self.data.replace(s, "")


s1 = Mystring("This is the last container ")
print("Original String:", s1.data)

# Appending to string
s1.append("known as userslists")
print("String After Appending:", s1.data)

# Removing from string
s1.remove("s")
print("String after Removing:", s1.data)
#matching the first charecter
print("Is the first charecter of above string 'T'?")
print(s1.match_char(0,"T"))
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> python Userstring.py
Original String: This is the last container
String After Appending: This is the last container known as usersl
ists
String after Removing: Thi i the lat container known a uerlit
Is the first charecter of above string 'T'?
True
PS C:\Users\adiso\OneDrive\Desktop\koss>
```

FINAL EXAMPLE

task.py

```python
from sample_array import text
from collections import Counter
from collections import defaultdict

def def_value():
    return "Not Present"


count = Counter(text)
default_count = defaultdict(lambda: "Not Present",count)
#second arguement passed is initialization of default dict

#used context manager to write in file
with open("task.txt","w") as f:
    for k in count:
        f.write(k+": "+ str(count[k]) +"\n")
print("Is file closed?")
print(f.closed)
# find count of various words in our default dict
print("Is 'happy'present in the database ?")
print(default_count['happy'])
print("How many 'unstructured' present ?")
print(default_count['unstructured'])
print("How many 'Aditya' present ?")
print(default_count['Aditya'])
```

task.txt

```
It: 2
is: 17
no: 2
doubt: 1
that: 6
today: 8
s: 5
systems: 4
are: 12
processing: 4
huge: 1
amount: 2
of: 30
data: 50
every: 3
day: 1
For: 1
example: 2
Facebook: 1
Hive: 1
```

Windows PowerShell

```
PS C:\Users\adiso\OneDrive\Desktop\koss> pyt
hon task.py
Is file closed?
True
Is 'happy'present in the database ?
Not Present
How many 'unstructured' present ?
4
How many 'Aditya' present ?
Not Present
```

Tab Size: 4 Plain Text

THANKS AND KEEP LEARNING

SOME IMPORTANT LINKS

https://www.geeksforgeeks.org/python-collections-module/

https://docs.python.org/3/library/collections.html

https://www.geeksforgeeks.org/context-manager-in-python/

https://www.geeksforgeeks.org/context-manager-using-contextmanager-decorator/

https://book.pythontips.com/en/latest/context_managers.html

THIS PRESENTATION GIT-HUB REPOSITORY: GITHUB LINK