**Title**

Reading Multiple Notes In a TXT File

**Snippet Code**

with open("FileName.txt", "r") as file:  
 # Read the lines from the file  
 lines = file.readlines()

# Iterate through the lines   
 for i in range(0, len(lines), \_\_number of lines each note has\_\_:   
 # Extract the attributes of the note from the lines  
 title = lines[i].strip()  
 text = lines[i + 1].strip()

# Add more attributes if note has more such as links and tags

# Print the note details  
 print(title)  
 print("Text:", text)  
 # Add more print statements if there are more details

**Title**

Save as csv file

**Text**

This snippet allows you to save as a csv file

**Snippet Code**

with open("notes1.csv", "w", newline='') as csvfile:

fieldnames = ['title','text','meta']

writer = csv.DictWriter(csvfile,fieldnames=fieldnames)

writer.writeheader()

for note in notes:

writer.writerow(note)

**Title**

Python List Comprehensions

**Text**

List comprehensions provide a concise way to create lists in Python. They consist of an expression followed by a for clause, and optionally, additional for or if clauses. List comprehensions are a powerful tool for creating lists by applying an expression to each item in an iterable.

**Snippet code**

# Using a list comprehension to create a list of squares  
squares = [x\*\*2 for x in range(10)]

# Using conditions in a list comprehension  
even\_squares = [x\*\*2 for x in range(10) if x % 2 == 0]

# List comprehension with if-else expression  
numbers = [1, 2, 3, 4, 5]  
transformed\_numbers = [x if x % 2 == 0 else x\*2 for x in numbers]

**Title**

Python's Zip Function

**Text**

The zip() function in Python is used to combine multiple iterables (like lists) element-wise into a single iterable.

**Snippet Code**

l1=[1,2,3]

l2 = ['a', 'b','c']

zipp = zip(l1,l2)

zlist = list(zipp)

print(zlist)

the console would return [(1,'a') , (2,'b'), (3,'c')]

#zip() #function

**Title**

Series and DataFrame

**Text**

Pandas has two main data structures, "Series" and "DataFrame." Series is a 1D array, and can include labels. DataFrame is a 2D array, and has a named + ordered column structure; each column can contain a different data type (e.g. string and boolean).

**Snippet Code**

series = pd.Series)array\_1D)

dataframe = pd.DataFrame(array\_2D)

**Title**

Pandas basics

**Text**

Pandas is a data analysis and manipulation tool in python. Pandas has three data structures:deries, dataframe, and index objects.Some of the functionalities include: Reindexing, axis oeprations (drop/add), filter, sorting and ranking, etc.

**Snippet Code**

import pandas as pd

mydataset = {  
 'cars': ["BMW", "Volvo", "Ford"],  
 'passings': [3, 7, 2]  
}

myvar = pd.DataFrame(mydataset)

print(myvar)

links:https://www.w3schools.com/python/pandas/pandas\_getting\_started.asp

Tags: #Pandas, #basics

**Title**

Error Handling in Python Using Try-Except Blocks

**Text**

Error handling is an essential aspect of programming to gracefully handle unexpected situations and prevent program crashes. Python provides the try-except block for handling exceptions.

**Snippet Code**

try:

# Code block that may raise an exception

result = 10 / 0 # ZeroDivisionError

except ZeroDivisionError:

# Code block to handle the exception

print("Error: Division by zero occurred.")

**Title**

Using Python's enumerate() Function for Iterating Over Lists

**Text**

Python's enumerate() function is a useful built-in function for iterating over elements in a list while also keeping track of the index of each element.

**Snippet Code**

# Example: Iterating over a list with enumerate()

fruits = ['apple', 'banana', 'cherry', 'date']

for index, fruit in enumerate(fruits):

print(f"Index {index}: {fruit}")

**Title**

Blackjack Ace Function

**Text**

# Manually selects what value Ace will be for dealer when drawn  
def dealer\_ace(ace\_drawn):  
 """  
 Used only for the dealer. Automatically determines whether Ace is equal to 1 or 11 based on Blackjack conventions.

Arguments:  
 ace\_drawn (string): the Ace the dealer drew  
   
 Returns:  
 ace\_drawn (string): the dealer's current Ace

**Snippet Code**

total = dealer\_draw(card\_deck)

if total >= 10:

ace\_drawn = 1

print("For this draw, Ace equals {}".format(ace\_drawn))

return ace\_drawn

else:

ace\_drawn = 11

print("For this draw, Ace equals {}".format(ace\_drawn))

return ace\_drawn

**Title**

Python Dictionary Comprehension

**Text**

Python Dictionary Comprehension Example

This creates a dictionary with numbers as keys and their squares as values.

**Snippet Code**

squared\_dict = {num: num\*num for num in range(1, 11)}

print(squared\_dict)  
# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

**Title**

String Formatting in Python

**Text**

String Formatting in Python Example

This demonstrates different methods for formatting strings in Python.

**Snippet Code**

name = "Alice"  
age = 30

# Using f-strings (formatted string literals)  
formatted\_string = f"Hello, my name is {name} and I am {age} years old."  
print(formatted\_string)

# Using the .format() method  
formatted\_string = "Hello, my name is {} and I am {} years old.".format(name, age)  
print(formatted\_string)

# Using %-formatting (old-style formatting)  
formatted\_string = "Hello, my name is %s and I am %d years old." % (name, age)

print(formatted\_string)

**Title**

Tkinter Window Rule

**Text**

Everything that has to do w the window object has to go in between these commands.

**Snippet Code**

window = Tk()

window.mainloop()

**Title**

Python Inheritance

**Text**

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

The child class named "Student" will inherit the properties and methods from the parent class named "Person"

**Snippet Code**

class Person:  
 def \_\_init\_\_(self, fname, lname):  
 self.firstname = fname  
 self.lastname = lname

def printname(self):  
 print(self.firstname, self.lastname)

class Student(Person):  
 pass

**Title**

Python Polymorphism

**Text**

The word "Polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

For example, here we have 3 classes: Car, Boat, and Plane, and they all have a method called move()

**Snippet Code**

class Car:

def \_\_init\_\_(self, brand, model):

self.brand = brand

self.model = model

def move(self):

print("Drive!")

class Boat:

def \_\_init\_\_(self, brand, model):

self.brand = brand

self.model = model

def move(self):

print("Sail!")

class Plane:

def \_\_init\_\_(self, brand, model):

self.brand = brand

self.model = model

def move(self):

print("Fly!")

**Title**

Inheritance

**Text**

Inheritance allows us to define a class and inherit all the methods and properties of that class. For example, if I create a parent class, then all the attributes and methods get passed down to the child class.

**Snippet Code**

class Person:  
 def \_\_init\_\_(self, fname, lname):  
 self.firstname = fname  
 self.lastname = lname  
  
 def printname(self):  
 print(self.firstname, self.lastname)  
  
class Student(Person):  
 pass

x = Student("Mike", "Olsen")  
x.printname()

**Title**

Lambda

**Text**

Lambdas is a small anonymous function that can take any number of arguments, but it is constricted to only one expression and one expression only.

**Snippet Code**

This adds ten to the argument a

x = lambda a : a + 10  
print(x(5)

This multiplies a with argument b

x = lambda a, b : a \* b

print(x(5, 6))

Now this add the total result for a, b, and c

x = lambda a, b, c : a + b + c

print(x(5, 6, 2))