# Project Title:

**Electronic Device Dataset**

# Project Description:

**Objectives:**

The purpose of this code is to develop a working system for the management and retrieval of information on different types of electronic devices including a smartphone, a laptop, or a computer in a way that is more flexible and responsive to future changes. Therefore, using object-oriented programming structures this code is intended to carry out the following objectives.

1. Define a Base Class and Subclasses: A base class with the name of the class Electronic Device and add two subclasses to it.
2. Encapsulate Device Information: making sure that all electronic items have similar characteristics under the classes and all should be applicable to all these items.
3. Implement Polymorphism: using a method in the base class that allows the other classes to call the same method..

**Importance and Contribution:**

This project is crucial as it demonstrates the practical application of object-oriented programming concepts in Python, including abstraction, encapsulation, inheritance, and polymorphism. By providing a structured approach to managing electronic device information, the project showcases its relevance in various domains such as inventory management, retail, and consumer electronics analysis. It contributes by offering a reusable and extendable framework for handling device data, highlighting the effective use of Python libraries like Pandas for data processing and object-oriented design principles for creating modular and scalable systems. This project serves as a valuable resource for developers looking to understand and apply core programming concepts in real-world scenarios, making it beneficial for educational purposes and as a foundation for similar projects.

**Four Principles of Object Oriented Programming**

**Inheritance:** Is used to create subclasses (Smartphone, Laptop, Computer) that inherit from the ElectronicDevice base class, inheriting its attributes and methods.

*class Smartphone(ElectronicDevice): # Inheritance: Subclassing*

*class Laptop(ElectronicDevice): # Inheritance: Subclassing*

*class Computer(ElectronicDevice): # Inheritance: Subclassing*

**Polymorphism:** Isdemonstrated by method overriding in the subclasses (Smartphone, Laptop, Computer), where each subclass provides its own implementation of the device\_info() method.

class Smartphone(ElectronicDevice):

def device\_info(self): # Polymorphism: Method overriding

class Laptop(ElectronicDevice):

def device\_info(self): # Polymorphism: Method overriding

class Computer(ElectronicDevice):

def device\_info(self): # Polymorphism: Method overriding

**Abstract:** Is implemented by making ElectronicDevice an abstract base class (ABC) with an abstract method device\_info().Subclasses (Smartphone, Laptop, Computer) provide concrete implementations of the device\_info() method.

*from abc import ABC, abstractmethod*

*class ElectronicDevice(ABC): # Abstraction: Abstract base class*

*@abstractmethod*

*def device\_info(self): # Abstraction: Abstract method*

*class Smartphone(ElectronicDevice):*

*def device\_info(self): # Abstraction: Concrete method implementation in*

*class Laptop(ElectronicDevice):*

*class Computer(ElectronicDevice):*

def device\_info(self): # Abstraction: Concrete method implementation in

**Encapsulation:**

* Encapsulation is achieved by using private attributes (self.\_category, self.\_brand, self.\_model, self.\_year) in the ElectronicDevice class.
* Getter methods (get\_category(), get\_brand(), get\_model(), get\_year()) provide controlled access to these private attributes.

self.\_category = category # Encapsulation: Private attribute

def get\_category(self): # Encapsulation: Getter method

**Code Documentation:**

from abc import ABC, abstractmethod

import pandas as pd

df = pd.read\_csv('ElectronicDevice.csv')

df

# Abstract: Base class for electronic devices

class ElectronicDevice(ABC): # Abstract class

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

self.\_category = category

self.\_brand = brand

self.\_model = model

self.\_year = year

@abstractmethod

def device\_info(self): # Abstract method

"""Get basic information about the device."""

return f"{self.\_category}: {self.\_brand} {self.\_model} ({self.\_year})"

# Encapsulation: Getter methods to access the private attributes

def get\_category(self):

return self.\_category

def get\_brand(self):

return self.\_brand

def get\_model(self):

return self.\_model

def get\_year(self):

return self.\_year

class Smartphone(ElectronicDevice):

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

super().\_\_init\_\_("Smartphone", brand, model, year)

self.\_os = os

def device\_info(self):

"""Get information about the smartphone."""

return f"{super().device\_info()}, OS: {self.\_os}"

def get\_os(self):

return self.\_os

class Laptop(ElectronicDevice):

def \_\_init\_\_(self, brand, model, year, os, screen\_size):

super().\_\_init\_\_("Laptop", brand, model, year)

self.\_os = os

self.\_screen\_size = screen\_size

def device\_info(self):

"""Get information about the laptop."""

return f"{super().device\_info()}, OS: {self.\_os}, Screen Size: {self.\_screen\_size}"

def get\_os(self):

return self.\_os

def get\_screen\_size(self):

return self.\_screen\_size

class Computer(ElectronicDevice):

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

super().\_\_init\_\_("Computer", brand, model, year)

self.\_os = os

def device\_info(self):

"""Get information about the computer."""

return f"{super().device\_info()}, OS: {self.\_os}"

def get\_os(self):

return self.\_os

# Class and Object: Function to create electronic device objects based on user input

def create\_device():

category = input("Enter category (Smartphone, Laptop, Computer), or type 'Done' to stop: ")

if category.lower() == 'done':

return None

model = input("Enter model: ")

# Retrieve brand, year, OS, and screen size from the dataset based on the user's input

device\_data = df[df['Model'] == model]

if device\_data.empty:

print(f"Model '{model}' not found in the dataset.")

return None

brand = device\_data['Brand'].values[0]

year = device\_data['Year'].values[0]

os = device\_data['OS'].values[0]

screen\_size = device\_data['ScreenSize'].values[0] if category.lower() == 'laptop' else None

if category.lower() == "smartphone":

return Smartphone(brand, model, year, os)

elif category.lower() == "laptop":

return Laptop(brand, model, year, os, screen\_size)

elif category.lower() == "computer":

return Computer(brand, model, year, os)

else:

print("Invalid category.")

return None

# Main code to create and display electronic devices

devices = []

while True:

device = create\_device()

if device is None:

break

devices.append(device)

# Display information about each device

for device in devices:

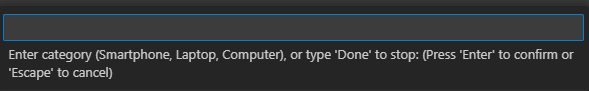
print(device.device\_info())

**My CODE:**

[**https://colab.research.google.com/drive/16N2\_27qkzE\_mkvJZzaiVurOV8Sm5L6Ra?usp=sharing**](https://colab.research.google.com/drive/16N2_27qkzE_mkvJZzaiVurOV8Sm5L6Ra?usp=sharing)

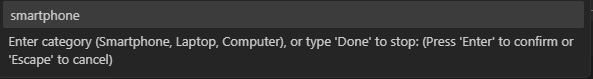
**Usage Guide:**

Step 1:

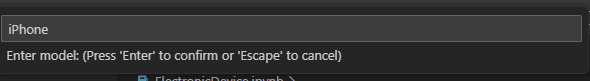
* Start by allowing the user to input based in the Dataset I collected.  
  

Step 2:

* Choose from the given choices Smartphone, Laptop, Computer.
* Then even it start with small later its ok because there is a condition in the code.

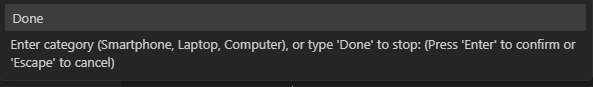


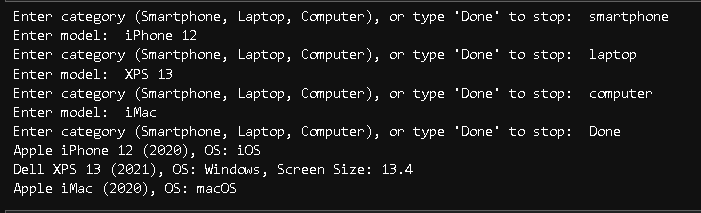
Step 3:

* Next, just type Model that is existing in the dataset.  
  

Step 4:

* Lastly, if ever you’re done or contented just easily type “Done”.



OUTPUT:

**Hardware & Software Used:**

**Hardware:**

* Computer or Cellphone

**Software:**

* **VS Code**

**Discussion:**

This project demonstrates the practical application of object-oriented programming principles, including abstraction, encapsulation, inheritance, and polymorphism, by creating an electronic device information system in Python. Utilizing an abstract base class **ElectronicDevice** and subclasses for **Smartphone**, **Laptop**, and **Computer**, the project encapsulates device attributes and implements polymorphic methods to provide specific device information. It reads data from a CSV file using Pandas, allowing user interaction to create and display device objects. This structured approach to managing electronic device information is relevant for inventory management, retail, and consumer electronics analysis. The project, developed using Visual Studio Code with Jupyter Notebook on a laptop, highlights effective use of Python libraries and object-oriented design principles, serving as a valuable resource for developers to understand and apply core programming concepts in real-world scenarios.

**Reference:**

**Abstract Base Classes in Python:** [**https://docs.python.org/3/library/abc.html**](https://docs.python.org/3/library/abc.html)

**Pandas Library for Data Manipulation:** [**https://pandas.pydata.org/**](https://pandas.pydata.org/)

**Object-Oriented Programming Principles:**

Liskov, B., & Guttag, J. (2000). Program Development in Java: Abstraction, Specification, and Object-Oriented Design. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.