# *Inheritance Lab Challenge*

class MP3:

  def \_\_init\_\_(self, artist, title, album, length, genre):

    self.artist = artist

    self.title = title

    self.album = album

    self.length = length

    self.genre = genre

  def get\_artist(self):

    return f"The artist is {self.artist}"

  def get\_title(self):

    return f"The title is {self.title}"

  def get\_album(self):

    return f"The album is {self.album}"

  def get\_length(self):

    return f"The length is {self.length}"

  def get\_genre(self):

    return f"The genre is {self.genre}"

*# child class*

class Podcast(MP3):

  def \_\_init\_\_(self, name, title, length, genre, date):

    self.name = name

    self.title = title

    self.length = length

    self.genre = genre

    self.date = date

  def get\_name(self):

    return f"The name is {self.name}"

  def get\_date(self):

    return f"The date is {self.date}"

  def get\_length(self):

    minutes = self.length // 60

    seconds = self.length % 60

    return f"The length is {minutes} minutes and {seconds} seconds"

p = Podcast("Planet Money", "Hollywood's Black List", 1460, "economics", "10 July 2020")

print(p.get\_name())

print(p.get\_title())

print(p.get\_length())

print(p.get\_genre())

print(p.get\_date())

**Inheritance Exercise 1**

**Exercise 1**

class CelestialBody:

  def \_\_init\_\_(self, size, mass, composition, name):

*self*.size = size

*self*.mass = mass

*self*.composition = composition

*self*.name = name

*# create the satellite class*

class Satellite(CelestialBody):

  def \_\_init\_\_(self, size, mass, composition, name, host\_planet):

    super().\_\_init\_\_(size, mass, composition, name)

*self*.host\_planet = host\_planet

*# create the planet class*

class Planet(CelestialBody):

  def \_\_init\_\_(self, size, mass, composition, name, host\_star):

    super().\_\_init\_\_(size, mass, composition, name)

*self*.host\_star = host\_star

**Inheritance Exercise 2**

**Exercise 2**

class Book:

  def \_\_init\_\_(self, title, author, pages, genre):

*self*.title = title

*self*.author = author

*self*.genre = genre

*# child class*

class BlogPost(Book):

  def \_\_init\_\_(self, website, title, author, word\_count, genre, page\_views):

    super().\_\_init\_\_(title, author, genre)

*self*.website = website

*self*.word\_count = word\_count

*self*.page\_views = page\_views

**Inheritance Exercise 3**

**Exercise 3**

class Parent1:

  def identify(self):

    return "This method is called from Parent1"

class Parent2:

  def identify(self):

    return "This method is called from Parent2"

*# declare child class here*

class Child(Parent2, Parent1):

  def identify(self):

    return "This method is called from Child"

  def identify2(self):

    return super().identify()

child\_object = Child()

child\_object.identify()

child\_object.identify2()

**Inheritance Exercise 4**

**Exercise 4**

*# DO NOT ALTER THIS CODE*

import sys

strings = [l.split(",") for l in sys.argv[1].split("\*")]

accounts = [[int(n) for n in s] for s in strings]

class Bank:

  def \_\_init\_\_(self, name, customers, accounts):

*self*.name = name

*self*.customers = customers

*self*.accounts = accounts

  def branch\_total(self, accounts):

    total = 0

    for account in accounts:

      total += account

    return total

*# Write your code here*

class RegionalBank(Bank):

  def regional\_total(self):

    total = 0

    for account in accounts:

      total += *self*.branch\_total(account)

    return total

my\_bank = RegionalBank("Bank of America", 9, accounts)

print(my\_bank.regional\_total())

**Inheritance Exercise 5**

**Exercise 5**

*# parent classes*

class Person:

  def \_\_init\_\_(self, name, address):

*self*.name = name

*self*.address = address

  def get\_info(self):

    return f"{*self*.name} lives at {*self*.address}."

class CardHolder:

  def \_\_init\_\_(self, account\_number):

*self*.account\_number = account\_number

*self*.balance = 0

*self*.credit\_limit = 5000

  def process\_sale(self, price):

*self*.balance += price

  def make\_payment(self, amount):

*self*.balance -= amount

*# declare child class here*

class PlatinumClient(Person, CardHolder):

  def \_\_init\_\_(self, name, address, account\_number):

    Person.\_\_init\_\_(*self*, name, address)

    CardHolder.\_\_init\_\_(*self*, account\_number)

*self*.cash\_back = 0.02

*self*.rewards = 0

  def process\_sale(self, price):

*self*.balance += price

*self*.rewards += *self*.cash\_back \* price

platinum = PlatinumClient("Sarah", "101 Main Street", 123364)

platinum.process\_sale(100)

print(platinum.rewards)

print(platinum.balance)

platinum.make\_payment(50)

print(platinum.balance)

print(platinum.get\_info())

**Encapsulation Exercise 1**

**Exercise 1**

class Country:

  def \_\_init\_\_(self, name, capital, population, continent):

    self.\_name = name

    self.\_capital = capital

    self.\_population = population

    self.\_continent = continent

my\_country = Country('France', 'Paris', 67081000, 'Europe')

print(my\_country.\_name)

print(my\_country.\_capital)

print(my\_country.\_population)

print(my\_country.\_continent)

**Encapsulation Exercise 2**

**Exercise 2**

class Artist:

  def \_\_init\_\_(self, name, medium, style, famous\_artwork):

    self.\_name = name

    self.\_medium = medium

    self.\_style = style

    self.\_famous\_artwork = famous\_artwork

my\_artist = Artist('Bill Watterson', 'ink and paper', 'cartoons', 'Calvin and Hobbes')

print(my\_artist.\_name)

print(my\_artist.\_medium)

print(my\_artist.\_style)

print(my\_artist.\_famous\_artwork)

**Encapsulation Exercise 3**

**Exercise 3**

class BankAccount():

  def \_\_init\_\_(self):

    self.\_checking = 0

    self.\_savings = 0

  def get\_checking(self):

    return self.\_checking

  def set\_checking(self, new\_value):

    self.\_checking = new\_value

  def get\_savings(self):

    return self.\_savings

  def set\_savings(self, new\_value):

    self.\_savings = new\_value

my\_account = BankAccount()

my\_account.set\_checking(523.48)

print(my\_account.get\_checking())

my\_account.set\_savings(386.15)

print(my\_account.get\_savings())

**Encapsulation Exercise 4**

**Exercise 4**

class Dancer:

  def \_\_init\_\_(self, name, nationality, style):

    self.\_name = name

    self.\_nationality = nationality

    self.\_style = style

  def get\_name(self):

    return self.\_name

  def set\_name(self, new\_value):

    self.\_name = new\_value

  def get\_nationality(self):

    return self.\_nationality

  def set\_nationality(self, new\_value):

    self.\_nationality = new\_value

  def get\_style(self):

    return self.\_style

  def set\_style(self, new\_value):

    self.\_style = new\_value

  name = property(get\_name, set\_name)

  nationality = property(get\_nationality, set\_nationality)

  style = property(get\_style, set\_style)

my\_dancer = Dancer("Savion Glover", "American", "tap")

print(my\_dancer.name) *#Savion Glover*

print(my\_dancer.nationality)  *#American*

print(my\_dancer.style)  *#tap*

my\_dancer.name = 'Anna Pavlova' *#N/A*

my\_dancer.nationality = 'Russian' *#N/A*

my\_dancer.style = 'ballet'  *#N/A*

print(my\_dancer.name) *#Anna Pavlova*

print(my\_dancer.nationality)  *#Russian*

print(my\_dancer.style)

**Encapsulation Exercise 5**

**Exercise 5**

class Cyclist:

  def \_\_init\_\_(self, name, nationality, nickname):

    self.\_name = name

    self.\_nationality = nationality

    self.\_nickname = nickname

  @property

  def name(self):

    return self.\_name

  @name.setter

  def name(self, new\_value):

    self.\_name = new\_value

  @property

  def nationality(self):

    return self.\_nationality

  @nationality.setter

  def nationality(self, new\_value):

    self.\_nationality = new\_value

  @property

  def nickname(self):

    return self.\_nickname

  @nickname.setter

  def nickname(self, new\_value):

    self.\_nickname = new\_value

my\_cyclist = Cyclist("Greg LeMond", "American", "Le Montstre")

print(my\_cyclist.name)

print(my\_cyclist.nationality)

print(my\_cyclist.nickname)

my\_cyclist.name = "Eddy Merckx"

my\_cyclist.nationality = "Belgian"

my\_cyclist.nickname = "The Cannibal"

print(my\_cyclist.name)

print(my\_cyclist.nationality)

print(my\_cyclist.nickname)

**Polymorphism Exercise 1**

**Exercise 1**

import random

class Lottery:

  def shuffle(self):

    results = []

    for i in range(5):

      results.append(random.randint(1, 20))

    return results

class PowerBall(Lottery):

  def shuffle(self):

    results = []

    for i in range(6):

      results.append(random.randint(1, 99))

    return results

p = PowerBall()

print(p.shuffle())

**Polymorphism Exercise 2**

**Exercise 2**

class Airplane:

  def \_\_init\_\_(self, first\_class, business\_class, coach):

    self.first\_class = first\_class

    self.business\_class = business\_class

    self.coach = coach

  def total(self):

    return self.first\_class + self.business\_class + self.coach

class Train:

  def \_\_init\_\_(self, car1, car2, car3, car4, car5):

    self.car1 = car1

    self.car2 = car2

    self.car3 = car3

    self.car4 = car4

    self.car5 = car5

  def total(self):

    return self.car1 + self.car2 + self.car3 + self.car4 + self.car5

  def passengers(obj):

    print(f'There are {obj.total()} passengers on board.')

t = Train(1,2,3,4,5)

print(t.passengers())

**Polymorphism Exercise 3**

**Exercise 3**

class Characters:

  def \_\_init\_\_(self, phrases):

    self.phrases = phrases

  def total\_characters(self):

    total = 0

    for phrase in self.phrases:

      total += len(phrase)

    return total

  def \_\_gt\_\_(self, obj):

    return self.total\_characters() > obj.total\_characters()

  def \_\_lt\_\_(self, obj):

    return self.total\_characters() < obj.total\_characters()

  def \_\_eq\_\_(self, obj):

    return self.total\_characters() == obj.total\_characters()

sample\_phrases1 = ['cat in the hat', 'green eggs and ham', 'the lorax']

sample\_phrases2 = ['the taming of the shrew', 'hamlet', 'othello']

c1 = Characters(sample\_phrases1)

c2 = Characters(sample\_phrases2)

print(c1 > c2) *# prints 'True'*

print(c1 < c2) *# prints 'False'*

print(c1 == c1) *# prints 'True'*

**Polymorphism Exercise 4**

**Exercise 4**

class Median:

  def calculate\_median(self, n1, n2, n3=None, n4=None, n5=None):

    if n3 is not None and n4 is not None and n5 is not None:

      numbers = [n1, n2, n3, n4, n5]

    elif n3 is not None and n4 is not None:

      numbers = [n1, n2, n3, n4]

    elif n3 is not None:

      numbers = [n1, n2, n3]

    else:

      numbers = [n1, n2]

    numbers.sort()

    median\_index = len(numbers) // 2

    if len(numbers) % 2 == 0:

      median = (numbers[median\_index] + numbers[median\_index - 1]) / 2

    else:

      median = numbers[median\_index]

    return median

m = Median()

print(m.calculate\_median(3,5,1,4,2))

print(m.calculate\_median(8,6,4,2))

print(m.calculate\_median(9,3,7))

print(m.calculate\_median(5,2))

**Polymorphism Exercise 5**

**Exercise 5**

source\_file = '/home/codio/workspace/code/polymorphism/text\_1\_exercise5.txt'

answer\_file = '/home/codio/workspace/code/polymorphism/answer\_exercise5.txt'

class Substitute:

  def \_\_init\_\_(self, source\_file, answer\_file):

    self.source\_file = source\_file

    self.answer\_file = answer\_file

    self.words = None

  def string\_to\_list(self):

    '''Read text file, turn it into a

    2D list of words for each line'''

    words = []

    with open(self.source\_file, 'r') as file\_object:

      lines = file\_object.read().split('\n')

      for line in lines:

        words.append(line.split())

    self.words = words

  def list\_to\_string(self):

    '''Convert 2D list back into a

    string with newline characters'''

    lines = []

    for line in self.words:

      lines.append(' '.join(line))

    string = '\n'.join(lines)

    self.words = string

  def swap\_words(self):

    self.string\_to\_list()

    for line in self.words:

      for i in range(len(line)):

        if (i + 1) % 5 == 0:

          word = line[i]

          line[i] = 'HELLO'

    self.list\_to\_string()

class Stars(Substitute):

  def swap\_words(self):

    self.string\_to\_list()

    for line in self.words:

      for i in range(len(line)):

        if (i + 1) % 3 == 0:

          word = line[i]

          line[i] = '\*' \* len(word)

    self.list\_to\_string()

    file = open(self.answer\_file, 'w')

    file.writelines(self.words)

    file.close()

s = Stars(source\_file, answer\_file)

s.swap\_words()

**Advanced Topics Exercise 1**

**Exercise 1**

import tech

my\_phone = tech.Phone('Pixel 5', 'sage', 128)

my\_laptop = tech.Laptop('MacBook Pro', 15, 256)

print(my\_phone)

print(my\_laptop)

**Advanced Topics Exercise 2**

**Exercise 2**

class Band:

  def \_\_init\_\_(self, name, genre, members):

    self.name = name

    self.genre = genre

    self.members = members

  def \_\_str\_\_(self):

    return f'{self.name} is a {self.genre} band.'

  def \_\_repr\_\_(self):

    return f'Band({self.name}, {self.genre}, {self.members})'

dead = Band('The Grateful Dead', 'rock\'n roll', ['Jerry', 'Bob', 'Mickey', 'Bill', 'Phil', 'Pigpen'])

print(dead)

print(repr(dead))

**Advanced Topics Exercise 3**

**Exercise 3**

class Dog:

  def \_\_init\_\_(self, name, breed):

    self.name = name

    self.breed = breed

dogs = []

dogs.append(Dog('Marceline', 'German Shepherd'))

dogs.append(Dog('Cajun', 'Belgian Malinois'))

dogs.append(Dog('Daisy', 'Border Collie'))

dogs.append(Dog('Rocky', 'Golden Retriever'))

dogs.append(Dog('Bella', 'Irish Setter'))

**Advanced Topics Exercise 4**

**Exercise 4**

**Book.py:**

from library import Library

class Book:

  def \_\_init\_\_(self, title, author, genre):

    self.author = author

    self.title = title

    self.genre = genre

  def \_\_repr\_\_(self):

    return f'Book({self.title}, {self.author}, {self.genre})'

Library.py:

class Library:

  def \_\_init\_\_(self):

    self.books = []

    self.fiction = []

    self.nonfiction = []

  def add\_book(self, book):

    '''Takes a Book object and adds it to self.books'''

    self.books.append(book)

  def search\_title(self, title):

    '''Takes a string and returns a Boolean'''

    has\_book = False

    for book in self.books:

      if title.lower() == book.title.lower():

        has\_book = True

    return has\_book

  def search\_author(self, author):

    '''Takes a string and returns a list of Book objects'''

    author\_books = []

    for book in self.books:

      if book.author.lower() == author.lower():

        author\_books.append(book)

    return author\_books

  def sort\_books(self):

    '''Helper method for sort\_fiction and sort\_nonfiction'''

    self.fiction = self.sort\_fiction()

    self.nonfiction = self.sort\_nonfiction()

  def sort\_fiction(self):

    '''Return list of Book objects where the genre is fiction'''

    fiction\_books = []

    for book in self.books:

      if book.genre.lower() == 'fiction':

        fiction\_books.append(book)

    return fiction\_books

  def sort\_nonfiction(self):

    '''Return list of Book objects where the genre is nonfiction'''

    nonfiction\_books = []

    for book in self.books:

      if book.genre.lower() == 'nonfiction':

        nonfiction\_books.append(book)

    return nonfiction\_books

from book import Book

library = Library()

book1 = Book('Three Musketeers', 'Alexandre Dumas', 'fiction')

book2 = Book('The Count of Monte Cristo', 'Alexandre Dumas', 'fiction')

book3 = Book('Educated', 'Tara Westover', 'nonfiction')

library.add\_book(book1)

library.add\_book(book2)

library.add\_book(book3)

library.sort\_books()

print(library.books)

print(library.fiction)

print(library.nonfiction)

print(library.search\_author('Alexandre Dumas'))

print(library.search\_author('Herman Melville'))

print(library.search\_title('Educated'))

print(library.search\_title('Moby Dick'))

Exercise4.py:

from library import Library

from book import Book

**Advanced Topics Exercise 5**

**Exercise 5**

shopping\_cart.py:

class ShoppingCart:

  def \_\_init\_\_(self):

    self.items = []

    self.total = 0

  def add\_item(self, item):

    self.items.append(item)

    self.calculate\_total()

  def calculate\_total(self):

    self.total = 0

    for item in self.items:

      item.calculate\_subtotal()

      self.total += item.get\_subtotal()

  def get\_total(self):

    return self.total

  def get\_num\_items(self):

    return len(self.items)

  def get\_items(self):

    return self.items

  def \_\_str\_\_(self):

    return f'The cart has {self.get\_num\_items()} items for a total of ${self.total}'

Item.py:

class Item:

  def \_\_init\_\_(self, name, price, quantity):

    self.name = name

    self.price = price

    self.quantity = quantity

    self.subtotal = 0

  def calculate\_subtotal(self):

    self.subtotal = self.quantity \* self.price

  def get\_subtotal(self):

    return self.subtotal

  def \_\_repr\_\_(self):

    return f'Item({self.name}, {self.price}, {self.quantity}, {self.subtotal})'

Exercise5.py:

from item import Item

from shopping\_cart import ShoppingCart

item1 = Item('milk', 1.5, 1)

item2 = Item('apple', 5, 0.75)

item3 = Item('bread', 2, 2.25)

cart = ShoppingCart()

cart.add\_item(item1)

cart.add\_item(item2)

cart.add\_item(item3)

print(cart.get\_total())

print(cart.get\_num\_items())

print(cart)

print(cart.get\_items())