class Car:

"""Represent a Car object."""

def \_\_init\_\_(self, name="", fuel=0):

"""Initialise a Car instance.

name: string, reference name for car

fuel: float, one unit of fuel drives one kilometre

"""

self.name = name

self.fuel = fuel

self.odometer = 0

def add\_fuel(self, amount):

"""Add amount to the car's fuel."""

self.fuel += amount

def drive(self, distance):

"""Drive the car a given distance.

Drive given distance if car has enough fuel

or drive until fuel runs out return the distance actually driven.

"""

if distance > self.fuel:

distance = self.fuel

self.fuel = 0

else:

self.fuel -= distance

self.odometer += distance

return distance

def \_\_str\_\_(self):

"""str method to return a string in a given format"""

return "{}, fuel={}, odometer={}".format(self.name, self.fuel,

self.odometer)

from prac\_06.car import car

def main():

"""Demo test code to show how to use car class."""

my\_car = Car("My car", 180)

my\_car.drive(30)

print("fuel =", my\_car.fuel)

print("odo =", my\_car.odometer)

print(my\_car)

print("Car {}, {}".format(my\_car.fuel, my\_car.odometer))

print("Car {self.fuel}, {self.odometer}".format(self=my\_car))

obj1 = Car("limo", 100)

obj1.add\_fuel(20)

print("limo:", obj1.fuel)

obj1.drive(115)

print("distance:", obj1.odometer)

main()

from prac\_06.guitar import Guitar

CURRENT\_YEAR = 2017

def run\_tests():

"""Tests for Guitar class."""

name = "Gibson L-5 CES"

year = 1922

cost = 16035.40

guitar = Guitar(name, year, cost)

other = Guitar("Another Guitar", 2012, 1512.9)

print("{} get\_age() - Expected {}. Got {}".format(guitar.name, 95,

guitar.get\_age()))

print("{} get\_age() - Expected {}. Got {}".format(other.name, 5,

other.get\_age()))

print()

print("{} is\_vintage() - Expected {}. Got {}".format(guitar.name,

True,

guitar.is\_vintage()))

print("{} is\_vintage() - Expected {}. Got {}".format(other.name,

False,

other.is\_vintage()))

if \_\_name\_\_ == '\_\_main\_\_':

run\_tests()

CURRENT\_YEAR = 2017

VINTAGE\_AGE = 50

class Guitar:

"""Guitar class for storing details of a guitar."""

def \_\_init\_\_(self, name="", year=0, cost=0):

"""Initialise a Guitar."""

self.name = name

self.year = year

self.cost = cost

def \_\_str\_\_(self):

"""Return a string representation of a Guitar."""

return "{} ({}) : ${:,.2f}".format(self.name, self.year, self.cost)

def get\_age(self):

"""Get the age of a guitar based on the CURRENT\_YEAR."""

return CURRENT\_YEAR - self.year

def is\_vintage(self):

"""Determine if a Guitar is considered vintage or not based on age."""

return self.get\_age() >= VINTAGE\_AGE

def \_\_lt\_\_(self, other):

"""Less than, used for sorting Guitars - by year released."""

return self.year < other.year

from prac\_06.guitar import Guitar

def main():

"""Guitar program, using Guitar class."""

guitars = []

print("My guitars!")

name = input("Name: ")

while name != "":

year = int(input("Year: "))

cost = float(input("Cost: $"))

guitar\_to\_add = Guitar(name, year, cost)

guitars.append(guitar\_to\_add)

print(guitar\_to\_add, "added.")

name = input("Name: ")

guitars.append(Guitar("Gibson L-5 CES", 1922, 16035.40))

guitars.append(Guitar("Line 6 JTV-59", 2010, 1512.9))

if guitars: # lists, strings and other collections are False when empty, True when non-empty

# In order for sorting to work on Guitar objects,

# at least the \_\_lt\_\_ method must be defined in Guitar class

guitars.sort()

print("These are my guitars:")

for i, guitar in enumerate(guitars):

vintage\_string = ""

if guitar.is\_vintage():

vintage\_string = "(vintage)"

print("Guitar {0}: {1.name:>30} ({1.year}), worth ${1.cost:10,.2f}\

{2}".format(i + 1, guitar, vintage\_string))

else:

print("No guitars :( Quick, go and buy one!")

main()

from prac\_06.programming\_language import ProgrammingLanguage

def main():

ruby = ProgrammingLanguage("Ruby", "Dynamic", True, 1995)

python = ProgrammingLanguage("Python", "Dynamic", True, 1991)

visual\_basic = ProgrammingLanguage("Visual Basic", "Static", False, 1991)

languages = [ruby, python, visual\_basic]

print("The dynamically typed languages are:")

for language in languages:

if language.is\_dynamic():

print(language.name)

main()

class ProgrammingLanguage:

"""Represent information about a programming language."""

def \_\_init\_\_(self, name, typing, reflection, year):

"""Construct a ProgrammingLanguage from the given values."""

self.name = name

self.typing = typing

self.reflection = reflection

self.year = year

def \_\_str\_\_(self):

"""Return string representation of a ProgrammingLanguage."""

return "{}, {} Typing, Reflection={}, First appeared in {}".format(

self.name, self.typing, self.reflection, self.year)

def is\_dynamic(self):

"""Determine if language is dynamically typed."""

return self.typing == "Dynamic"

def run\_tests():

"""Run simple tests/demos on ProgrammingLanguage class."""

ruby = ProgrammingLanguage("Ruby", "Dynamic", True, 1995)

python = ProgrammingLanguage("Python", "Dynamic", True, 1991)

visual\_basic = ProgrammingLanguage("Visual Basic", "Static", False, 1991)

languages = [ruby, python, visual\_basic]

print(python)

print("The dynamically typed languages are:")

for language in languages:

if language.is\_dynamic():

print(language.name)

if \_\_name\_\_ == "\_\_main\_\_":

run\_tests()

<https://github.com/ashishadhana309/CP1404Practical>